Photo: Niko Myller

Proceedings

13th Koli Calling International Conference on Computing Education research Koli Calling 2013

November 14 -17, 2013
Koli, Finland

Conference Chairs:
Mikko-Jussi Laakso, University of Turku, Finland
Simon, University of Newcastle, Australia

Sponsored by:
University of Eastern Finland
Aalto University, Finland
University of Turku, Finland
University of Newcastle, Australia

In cooperation with:
ACM/SIGCSE

The Association for Computing Machinery
2 Penn Plaza, Suite 701
New York New York 10121-0701

ACM ISBN: 978-1-4503-2482-3

Koli Calling 2013

Foreword

This volume collects together the papers presented and discussed at the 2013 Koli Calling International Conference on Computing Education Research.

These papers cover a range of different perspectives, approaches and results within a scholarly approach to computing education research, computing education practice, and computers in education. This includes methodological, empirical, curricular and tools-oriented interventions and reflections. The conference accepts empirical and theoretical research papers, system papers, and discussion papers.

We received 40 paper submissions from researchers in 15 countries. Each submission was double-blind reviewed by three members of the program committee, and we accepted 20 papers: 13 research papers, three system papers, and four discussion papers. These proceedings include these 20 papers, the abstracts of eight posters presented at the conference, and the abstract of the keynote address by Matti Lattu, Project Manager of the Matriculation Examination Board of Finland.

Welcome to Koli Calling 2013.

Mikko-Jussi Laakso
Simon

Koli Calling 2013

Program Committee

Moti Ben-Ari	Weizmann Institute of Science, Israel
Anders Berglund	Uppsala University, Sweden
Valentina Dagienė	Vilnius University, Lithuania
Hannu-Matti Järvinen	Tampere University of Technology, Finland
Mike Joy	University of Warwick, UK
Ville Karavirta	Aalto University, Finland
Päivi Kinnunen	University of Eastern Finland
Maria Knobelsdorf	Carl von Ossietzky Universität Oldenburg, Germany
Ari Korhonen	Aalto University, Finland
Mikko-Jussi Laakso	University of Turku, Finland
Lauri Malmi	Aalto University, Finland
Robert McCartney	University of Connecticut, USA
Marian Petre	The Open University, UK
Arnold Pears	Uppsala University, Sweden
Guido Rößling	Darmstadt University of Technology, Germany
Tapio Salakoski	University of Turku, Finland
Carsten Schulte	Freie Universität Berlin, Germany
Judy Sheard	Monash University, Australia
Simon	University of Newcastle, Australia
Jarkko Suhonen	University of Eastern Finland
Erkki Sutinen	University of Eastern Finland
J Ángel Velázquez-Iturbide	Universidad Rey Juan Carlos, Spain

Organising Committee

Ilkka Jormanainen	University of Eastern Finland
Jarkko Suhonen	University of Eastern Finland
Yue Dai	University of Eastern Finland
Calkin Suero Montero	University of Eastern Finland

Koli Caling 2013 – Table of Contents

Transferring the Finnish Matriculation Examination to IT

Matti Lattu
The Matriculation Examination Board of Finland
P.O.Box 50
FI-00581 Helsinki
+358-295-338 200
matti.lattu@ylippilastutkinto.fi

ABSTRACT

This paper describes the goals and current status of the renewal project of the Finnish Matriculation Examination project.

Categories and Subject Descriptors

K3.2 [**Computers and education**]: Computer and Information Science Education – *computer science education*

General Terms

Measurement

Keywords

High-stakes testing, electronic examination

1. PAPER-AND-PENCIL TEST

The Matriculation Examination Board of Finland annually arranges two examinations for students finishing their upper secondary school. The most popular subjects may have as many as 30,000 simultaneously attending students. The exams take place in the 450 schools around Finland. The organisation of the exams (the board) is funded by the government (~33%) and students (~66%).

Currently the exams are carried out in traditional paper-and-pencil style. Most questions take the form of small essays while some utilise multiple-choice questions. The former are evaluated in a two-phase process where the students' teachers and the board censors both review the performance. The multiple-choice questions are examined by OCR. The students are not allowed to use any material other than that given with the questions.

2. PROJECT GUIDELINES

The complete process of organising the exam will gradually start to utilise IT between 2016 and 2019. After the change the students will draw up their answers using some kind of device – probably a laptop or a tablet. Due to financial reasons the students may bring their own devices to the exam. However, we do not expect the nature of the exam to change in the first years. In other words, we have to prevent collaboration and access to the Internet.

The project vision is to arrange the Matriculation Examination Test with only one paper - the diploma.

New possibilities for exams include new question types, more material, section/question-specific timing, testing subject-related ICT skills. Some considerations are:

- Exam for the students: cheap equipment, focus on the exam and not the IT, equal possibilities for all students
- Exam for the schools: easy setup, easy supervision, easy to change defective devices
- Project goals: our system – our IPR, use agile methods (scrum) to minimise risks, open and engaging project communications

3. SYSTEM ARCHITECTURE

The renewed examination system will contain the following subsystems:

- A Test System that interacts with the students. The system must support a variety of question types and devices while a number of schools have issues with low bandwidth. We have requirements for availability, integrity and non-repudiation.
- An Evaluation system carries out the two-step evaluation process.
- Extranet providing self-service for school administrators, teachers and students (e.g. exam results, application processes). This will function as a front-end for a Document Management System.
- A Central Database, "HAL", that holds the production data
- A Data Warehouse or Archive System supporting system integration and academic research.
- An Identity management system holding the data concerning the students, teachers and sensors. It works as the backbone for the other systems (e.g. authentication, authorisation and digital signing). In practice this may be part of the Central Database.

The Test Systems are located in the school localities. As the quality of the internet connections of the schools varies greatly, the connection is used only to deliver students' identities and questions. Also the students' responses and log information is

Koli Calling '13, Nov 14-17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2534902

retrieved to the Central Database. Offline use should be an option to make the exam system as robust as possible.

4. SCHEDULE

The first electronic exams will be organised in autumn 2016. The transition will be carried on in phases where each new exam introduces electronic testing to new subjects. The first subjects are German language, Philosophy and Geography. The last subject to digitalise is mathematics which will change in in spring 2019. The student can choose between old and new arrangements simply by scheduling his/her tests accordingly.

5. CURRENT STATUS

The project has produced the initial process and architecture plans. We are still running the request for information procedure for the Test Systems and the Document Management Systems. The tendering processes for these components will start by the end of 2013. We expect to get the first codelines during the first quarter of 2014.

The upper secondary schools are both interested in and worried about the coming exam arrangements. To explore the setup and procedures with the schools we have built a preliminary test setup, and a number of workshops will be carried out before the end of the year.

The Use of Code Reading in Teaching Programming

Teresa Busjahn, Carsten Schulte
Department of Computer Science
Freie Universität Berlin, Germany
{busjahn,schulte}@inf.fu-berlin.de

ABSTRACT

Programming is an intertwined process of reading and writing. So far, computing education research has often focused on the writing part. This paper takes a further look into the role of reading source code in learning to program. In order to complement the findings from literature, we conducted interviews with programming instructors using the miracle question, on the role of code reading and comprehension. The analysis of these interviews describes this role in terms of the five categories conceptualization, occurrences, and effects of successful code reading, challenges for learners, as well as approaches to facilitate code reading. As a result, we suggest to take a further look into the different reading processes involved in programming, in order to add to the knowledge about programming instruction.

Categories and Subject Descriptors

K3.2 [**Computers & Education**]: Computer and Information Science Education—*computer science education, information systems education.*

General Terms

Experimentation, Human Factors.

Keywords

CS Ed Research, Educational Research, Code Comprehension, Program Comprehension, Code Reading, Teaching Programming

1. INTRODUCTION

"In my class, students do not read code - they write code; which is what they need to learn. It's best to write, to let them discover algorithms, not to let them read", says one computing educator when hearing about the idea of research on code reading (CR) in computing education. By CR we do not mean, that the reader's eyes merely move over the text. It rather denotes the first step in program comprehension, taking in the written elements and understanding them.

The BRACElet group (see e.g. [31]) is researching the relationship between 'reading, tracing, and writing', assuming there are relationships between these skills that explain parts of the ability to write programs. A causal relationship between code reading and writing skills would require to focus on reading and understanding of code before writing it. Yet, if students can learn to write programs without being able to read them, it might not be necessary to put so much effort on that skill [28].

Somewhat differently, Denny, Luxton-Reilly and Simon (2008) mention that "it is commonly believed that code tracing is easier than code writing, but it seems obvious that different skills are needed for each" [6], p. 113.

When taking an abstract, theoretical point of view, of course there's reading involved in programming, as e.g. a programmer must be aware of where to write next, must find bugs in the code, read code to eliminate syntax errors, and so on. Or, taken this argument somewhat further, before a person can write a syntactical statement, she would need to know what or how to write, and therefore have to have read the statement beforehand. This is the argumentation behind the work of the Bracelet group. Still - while there is some connection between reading and writing - it is not clear whether CR is something that has to be learned (in the sense that it needs to be carefully taught) or if it maybe will be picked up effortlessly just while writing. To put it differently: would there be an impact, some visible difference between a learner with a higher level of reading skills and a novice programmer with a lesser level of reading skill? Would the teacher be aware and adapt teaching? Would there be problems in writing code? In understanding code or algorithms? In the speed and accuracy of solving learning tasks?

We believe that the above briefly outlined discussion to a certain degree resembles the discussion on learning programming in the 1960s and early 70s, when programming was thought of in terms of the product, the written program, and so learning programming was thought of in terms of learning the words (to write). So in essence, learning programming was considered as learning a programming language [1]. Now there is a consensus that learning programming is also learning a process.

We can transfer this debate on writing programs in terms of product or process to the debate on reading programs: It might be that it is predominantly thought of in terms of the product like understanding the words written in the text. And so there is presumably an easy to master reading process, in which the text surface is translated in a mental model of the program, - and then something else happens that is focusing on discerning the program execution and the algorithmic idea embedded in the understanding of the program.

However, reading the text surface might be also a complex process in itself.

We believe there are two major aspects involved:

A) While reading and discerning the text surface, not only some translation of the text surface into an understanding of the language constructs takes place. Due to limited capacity of working memory, information processing and filtering already takes place while reading, so that only information considered as relevant during that moment will be processed and included in the process of chunking information into comprehensive forms of elements. So it is much more than just a translation that takes place in reading. Program reading and program comprehension thus cannot be separated, but are highly intertwined. In essence, therefore comprehension problems might be grounded in reading problems.

B) In writing a program it is natural to write several versions and thus refining the program. During programming one thinks about ideas and stepwise refines them. That is: programming is problem solving; and often needs to include cycles of activities. In such a cycle the programmer needs to externalize his current ideas and represent them (writing), so that he then can later on connect different thoughts and reflect on them (reading). This thinking process needs this externalization, and therefore needs reading (as those externalized thoughts and ideas are really externalized in the sense of being deleted from working memory, and later being included again by reading). Thus programming is an intertwined process of writing and reading. And so far, in computing education research, and especially in research with novice programmers and learners, we have focused on writing, and to some degree neglected the side of reading.

However, while it is obvious that reading has a role and importance, it is not really clear that it is an educational problem or an aspect which educators can focus on in order to support learning.

The question is, would there be an impact or visible change in the computing classroom, if the learners would be good readers? Would a teacher recognize a difference and what kinds of problems in teaching and learning programming would be solved?

2. ROLE OF CODE READING IN LITERATURE

2.1 Reading is a common activity during programming

“Source code is, among other things, a text to be read.” [23], p. 3.

Reading is a fundamental and extremely common part of programming. Nevertheless, little is can be found about how programmers actually read in practice. And programmers often take the skill of reading for granted. Reading code corresponds to other forms of reading, but there are specifics to it that arise from the code itself. Rooksby, Martin and Rouncefield (2006) [24] look into reading as done by professional programmers engaged in software development. In their study, they observed many situations in which the programmers read. The authors provide an informal taxonomy of some of this reading, which mostly took place at the screen or in conjunction with working on something at the screen. The first three occasions were ’writing code’, ’debugging’, and ’writing tests’. However, they also identified other situations, e.g. ’searching for information on the internet’, ’reading from textbooks’, and ’documenting’. They found common patterns of reading code. It is

- occasioned: the circumstances in on-going programming work that make the reading of code the relevant next action for programmers to undertake can be located,
- orderly: the ways of ordering the layout of code were expected and reproduced,
- analysable: programmers have consistent and expected ways of making sense of code.

Programmers often read their own code. Programs are usually developed over an extended period. While writing one part of the program, previously written parts must be attended to. Debugging provides similar opportunities to read one’s own code [15].

Programmers often have to maintain software written by others. Raymond points out, “that reading source code is a key activity in software maintenance” [23], p. 3. In order to maintain it, it first has to be understood. Even though there might be design documents, the code is oftentimes the sole source for the comprehension of the programs design. Also, CR is of importance in software review, where the software product under scrutiny is not executed, like in walkthroughs. 50 - 70 % of defects can be found that way [30].

“In the years to come, programmers will increasingly value coding that not only works, but that also can be easily read and understood. To withstand the test of time, coding must pass the litmus test of readability.” [29], p. 89. Likewise, Samaraweera, Shonle & Quarles (2011) observe a focus on writing in the work on program readability. While many guidelines exist for composing programs, only few consider the reader’s expectations. They conducted a survey with Java programmers on how the structure of a program communicates the intentions of the developer. They found that even refactorings that kept the meaning had a measurable effect on what readers believed the programmer’s main intention was. One example originally had a long method with two sections, that were separated by comments. After these two sections were extracted into clearly named methods, many readers were lead to an incorrect description [9]. CR in approaches to teaching programming

“In becoming a programmer you learn, amongst other things, to read code in the way that programmers should, can and do read code.” [24], p. 210. But, what is the way that programmers read code? And, do new programmers learn it? If so, how?

Several studies [18, 31, 16] applying different statistical approaches found significant relationships between tracing code, explaining code, and writing code. Nevertheless, it is not tracing and explaining skills independently, but their combination, that leads to writing skills. Writing code provides many opportunities to improve tracing and explanation skills, which in turn helps to improve writing skills. The skills reinforce each other and develop in parallel. While there is not a strict hierarchy, some minimal competence at both tracing and explaining precedes some minimal competence at systematically writing code. However, some aspects of these assessments results are sensitive to the particular exam questions used. So, Simon et al. (2009) [28] raise some concerns with this kind of assessment of reading and writing skills. Without further ado it is not possible to tell, whether reading and writing questions are really comparable. The same applies for the marking of points for these tasks. The distinction between line-by-line understanding and big-picture understanding does not seem to have a parallel in code-writing questions [17].

2.2 Practice examples for using Code Reading in teaching

Merrienboer and Kramer [20], as well as Selby [27] describe approaches to teaching programming, that focus on CR. Based upon a literature search as well as on an investigation of existing courses and programming textbooks, Merrienboer and Krammer (1987) classified three instructional strategies for the design of introductory computer programming courses in high school. Instructional strategies are general design plans that mainly differ in their control of students' processing load. The Expert approach emphasizes top-down program design. Novices start with a complex but intrinsically motivating programming problem. In the Spiral approach syntax and semantic are acquired emphasizing small incremental steps and building up a program by mastering the basics (language constructs) first. The Reading approach controls processing load by varying the difficulty of the students' task. From the beginning of the course, students are confronted with program reading assignments in the form of non-trivial design problems in combination with their complete or partial solutions. The assigned tasks gradually become more complex during the course, changing from using and analyzing programs, through modifying and extending programs, to designing and coding programs. It is recommended that students begin by understanding relatively complex existing programs and then modify and enhance those programs.

Drawing knowledge from cognitive theory and empirical research directed towards learning to program, they also identified six tactics for teaching programming. Tactics are specific design plans of action that prescribe methods to reach desired learning outcomes under given circumstances. They claim, that an effective programming course should incorporate these tactics. However, some tactics can be effectively applied in courses of one strategy, others are less compatible with the strategy. If more strategies are available and only one is compatible with all tactics, this strategy is clearly superior. If all tactics get equal weight during the evaluation of instructional strategies, the Reading approach surpasses the Expert and Spiral approaches. It is the best strategy to follow in the instructional design of introductory programming courses, as it effectively controls the processing load.

Selby [27] identifies four approaches to teaching introductory programming from literature: code analysis, building blocks, simple units, and full systems. We will outline code analysis:

Students read and understand source code, before they write their own. For that, any programming language, pseudocode or even structured English can be used. The code can be shown on paper or in an appropriate development environment, but it is not absolutely necessary, that learners interact with the code in an environment on a computer. In that case, no tools or environments need to be mastered, students can work on the programming logic straight away. If pseudocode is used, learning programming is also independent of a specific language. Code analysis allows the development of skills involved in explaining and debugging, which forms part of the underlying foundation necessary to facilitate code writing. In addition, it is possible to expose learners to the logic behind simple basic algorithms. There are hints, that especially weaker students profit from this approach.

However, learners might not like to learn programming using paper and pencil instead of the computer. Depending on the learner's age and capabilities, it can prove to be difficult to find appropriate pseudocode instructions (e.g. usage of words or symbols) and the lack of immediate feedback makes this approach problematic for independent learning. Also, there is no verification that the identified pseudocode logic is actually a correct representation of a problem solution. Furthermore, the understanding of pseudocode might not lead to skills, that can be applied to the comprehension of a strict syntax language. Even if students are able to read and understand a program, the are not necessarily able to construct that solution themselves.

In their "Guidelines for Teaching Object Orientation with Java" Kölling and Rosenberg [14] suggest to give students the chance to read code from others from the beginning on, since a lot can be learned from well written code and by copying styles and idioms. They further recommend to use examples with several classes and a considerable amount of methods, so students can practice CR, understand the need for clear code and documentation, etc. Campbell & Bolker (2002) [3] describe a CS1 course, in which students learn Java by immersion. The authors compare it to learning a natural-language by immersing in a foreign country. Thereby students are supposed to not only learn to think in objects, but also to learn the conventions that programmers have developed. So, instead of starting writing programs, students read programs written by experienced programmers, exposing them to good programming practice. Using some explanation, students can supposedly follow the program's logic and understand enough to make minor modifications. After all, modifying a well written program is easier than writing one from scratch. Other benefits of this approach are that the used programs are closer to the "real world" and that real design issues can be addressed sooner. However, the dropout rate did not change compared to previous CS1 courses.

Hilburn, Towhidnejad & Salamah (2011) adopt the principle of reading before writing for case studies in software engineering education. They recommend a teaching approach using Fagan software inspection as an active learning technique, indicating that students should read and study an existing software artifact, before developing one themselves. They claim that software engineering courses with software development projects are often isolated from the rest of the curriculum and do not form a real-world basis. Therefore, though graduates are familiar with the basic theoretical concepts in software development, they are not able to apply these concepts in real-world environments. By means of case studies, inspection is applied simultaneously as a professional practice, that students have to learn, and as a tool to teach about software development. These exercises promote the understanding of the nature, difficulty, and importance of requirements. Their technique could be used in different level courses throughout a computing curriculum and the inspection cases can be used with almost any artifact in the software life-cycle [10].

Spinellis argues, that students should also learn what is readable code, that others can easily decipher. He claims, that "computer programming education often focuses on how to single-handedly develop programs from scratch in a single language and single execution environment, a development style prevalent in the 1950s and 60s. Nowadays, software development is typically a team-based activity and most often involves extending and maintaining existing systems written in a multitude of languages for diverse execution environments. It's now even more important to understand code concepts, forms, structures, and idioms to be able to write code that other programmers can read easily" [29], p. 86.

Deimel (1985) [15] suggests the use of reading questions in exams as an alternative to writing questions. As advantage, especially of multiple-choice questions, he argues that they are easier to grade, more objective, and likely to be more consistent measures of student performance. Deimel and Naveda (1990) [5] even provide an instructor's guide for reading computer programs.

2.3 Code Reading and Program Comprehension

Reading occurs not only in learning programming, but also in debugging, and maintenance. It provides the essential basis for comprehension. The area of program comprehension comprises a vast body of literature, with numerous conflicting models having been proposed [32]. One of the earlier comprehension models developed by Pennington [21, 22] directly draws on research in reading, as it is based on the text comprehension model of Kintsch and van Dijk [13].

Models are typically grounded in experimental studies, mostly involving experienced programmers. The question of how to relate this material to the teaching and learning of programming for novices has proven challenging [26]. One obstacle is that comprehension is an internal cognitive process. In contrast, CR refers to a behavioral process, in principle open for observation by e.g. eye tracking [2]. The challenge to tackle is how to make use of this data to infer higher order comprehension processes.

3. EDUCATORS' PERSPECTIVES ON CODE READING

Interviews were conducted using the miracle question, plus some additional questions. The data gathering was done individually, to the convenience of the participants. Some were done via voice-over-IP. For analysis, the interviews were transcribed and coded. Results are based on this coding process.

3.1 Miracle question

"I am going to ask you a rather strange question [pause]. The strange question is this: [pause] Imagine you will do whatever you need to do the rest of today, such as preparing dinner and watching TV. Then you go to bed. In the middle of the night, a miracle happens and the problem that code reading is difficult for novices is solved! But because this happens while you are sleeping, you miss the miracle. [pause] So, when you wake up tomorrow morning, what might be the small change that will make you notice, that the problem is gone."

The miracle question comes from solution-focused brief therapy. Instead of focusing on problems, it is concerned with the solution. These solutions are developed "by first eliciting a description of what will be different when the problem is resolved" [4], p. 2, and then identifying strategies that worked in previous situations in which at least some characteristics of the described solution were present. Hardly any time is spend to explore the source of the problem. When describing the origins of the problem, language is usually negative and focuses on past-history, furthermore often the permanence of the problem is implied. However, the description of solutions, is normally rather positive, hopeful, and future-focused, while suggesting a transience of the problem. By using the miracle question, clients usually formulate smaller and more manageable goals. The reactions to it can be very different. Nevertheless, given time to think, they name specific things, that would be different, when the problem is solved. Their answers often serve as goals of therapy [4].

Transferring this approach to education can be done by asking educators the miracle question. Their responses can lead to a more detailed description of the behavior they would like their students to show. In turn, this can help to find their previous solutions and approaches. A possible variation of the method is to ask the miracle question to a group of educators and let them work on the miracle description together, or to ask learners what they want to know about a topic [8].

We deliberately ask for code reading instead of understanding, in order to elicit how much teachers are concerned with this part of the comprehension process.

3.2 Key questions

Wanting to complement the answers obtained by the miracle question and to get richer data, we prepared a set of questions for further exploration during the interviews. These questions were asked after the miracle question.

- Where do you see potential to facilitate code reading?
- Which aspects of code reading do you deem important?
- What do you want to know about code reading?
- What else about code reading comes to your mind?

3.3 Participants

Our aim was to gather some current notions of the role of CR in learning programming, with a focus on novices. Our sample size of six interviews is at the lower boundary, as discussed in qualitative research [19]. Our first aim is to figure out, if there is a substantial role of CR at all. For this purpose, the sample size and selection of participants should be sufficient, as we have chosen experienced teachers who are partly also engaged in the computing education community. We chose to focus on the school level, as there are more likely absolute novices, whereas in post-secondary education the students often already have some experiences (even though being enrolled in novice classes). Three of the interviewees are high school teachers from Germany, who in

	Teaching years	Students	Experience
1	9 years	High School, University	high
2	9 years	High School, other teachers	high
3	>30 years	High School, other teachers	high
4	4 years	High School	medium
5	6 years	University	high
6	6 years	High School	high

Table 1: Participants

addition to teaching at high school are also lecturing other teachers or teacher students (10-25% of their time). All of them have presented or published at conferences in computing education.

One other teacher was from UK, another from Iran. In order to get an additional perspective, we also interviewed one lecturer from a Russian university.

3.4 Procedure of Analysis

Basically, the data was analyzed by coding. Therefore the transcribed text is labeled, using a code, and the codes are grouped into categories. Text coding is used in different qualitative and quantitative approaches, e.g. in grounded theory methodology (GTM) [12], in phenomenography [7], and in content analysis [11].

In a way, we used open coding / inductive coding / labeling categories of description – the first step in analysis that is rather common for the above listed approaches. While doing so, we had the impression that we a) cannot or should not aim at giving a result that pretends to be complete or finished (because e.g. we had to few interviews, and there are too many possible additional perspectives to include). And b) we found that many of the aspects mentioned could be captured by using the Block Model (BM) as coding scheme, or rather as basis of a coding scheme. So we changed our approach somewhat and by including this theoretical framework added something which resembles deductive coding in content analysis, or axial coding in GTM. Consequently, we switched back and forth between inductive and deductive reasoning. We then discussed the obtained coding scheme with another experienced CS teacher involved in CSed research, for refinement.[1]

The BM [25] gives a normative account of dimensions and levels of understanding source code. It proposes three distinct dimensions of awareness: text surface, program execution and the function of the code ('function' is a somewhat misleading term in CS, here the term aims at the intention or purpose of a piece of code). These dimensions are to be perceived or discerned for chunks of text that get larger and larger: starting with words or atoms, to blocks, their relations to the whole text is taken into account (see table 2). The participants often used statements that correspond to cells in the BM. Therefore the model provides a suitable frame for coding terms and for grouping perspectives.

Overall, we analyzed the transcribed interviews using a rather conventional approach to coding, where the coding scheme was developed while reading and comparing the transcripts. We then used some deductive coding, and grouped the codes into categories. These categories highlight important aspect of the perceived role of CR, and represent our study results. We believe this result gives valuable insights for further research – but also that at this point it would not be of much use to aim at a full theoretical description or a complete outcome space to present the obtained insights as some complete understanding of the role of CR in secondary school.

[1]Thanks to Andreas Gramm for his time and suggestions.

3.5 Results

In this section we present the results of our analysis. During coding the transcribed interviews, we grouped the codes to five categories, which are presented in the following subsections.

Within each sub-section the category is described based on the codes, and supplemented with citations and general observations made when conducting and analyzing the interviews.

Figure 1 gives an overview of the categories and codes.

The results represent the aggregated understanding of the selected experts.

Many of the codes were mentioned more than once, and by more than one participant. The most frequent codes are highlighted in the figure.

3.5.1 Conceptualization of Code Reading

First we describe the participants personal understanding and conceptualizations of CR. All participants needed to think a while before being able to answer. Several times we heard comments like “hmm”, “difficult question”, “don't really know”, when they tried to explain CR or aspects of CR.

During the interviews, the conceptualization of CR was gradually enriched. In the end, all levels and dimensions of the Block model were explicitly mentioned or referred to. For example, CR was described as following: “Reading code to me means understanding the syntax [...] If students miraculously could read code, they could understand the syntax, the grammar and the vocabulary” (Interview 6). The focus was on the text surface: CR was conceptualized as the process of visually perceiving the source code, and as prerequisite of program comprehension. CR in this perspective is focused on 'deciphering' the words of the language, and translate them into the meaning of the words. This is consistent with our questions about reading, rather than understanding.

Program execution was also frequently mentioned . It was seen as one major challenge to infer from text to execution. Here the execution of blocks, and the flow of execution were focused on. The dimension of function (=intention) was also mentioned, mostly with focus on understanding the algorithmic idea (on the macro level), but also sometimes with focus on understanding the role of a block.

One participant mentioned that understanding the overall intention is a prerequisite for students to being able to read and understand. Sometimes understanding the intention was even seen as ultimate goal.

The participants disclosed both the possibility of bottom-up, and top-down approaches to reading and comprehension. Overall CR was seen as one step in comprehending a program, and rather often means or tools were mentioned to facilitate understanding (see 3.5.5 Approaches to facilitate CR for details). Furthermore it was recognized, that reading

	Text surface	**Program execution (data flow and control flow)**	**Functions (as means or as purpose), goals of the program**
Macro structure	(Understanding the) overall structure of the program text	Understanding the „algorithm“ of the program	Understanding the goal/ the purpose of the program (in its context)
Relations	References between blocks, e.g.: method calls, object creation, accessing data	sequence of method calls „object sequence diagrams“	Understanding how subgoals are related to goals, how function is achieved by subfunctions
Blocks	’Regions of Interests’ (ROI) that syntactically or semantically build a unit	Operation of a block, a method, or a ROI (as sequence of statements)	Function of a block, maybe seen as sub-goal Function of a statement.
Atoms	Language elements	Operation of a statement	Goal only understandable in context
	Structure		*Intention*

Table 2: Block Model

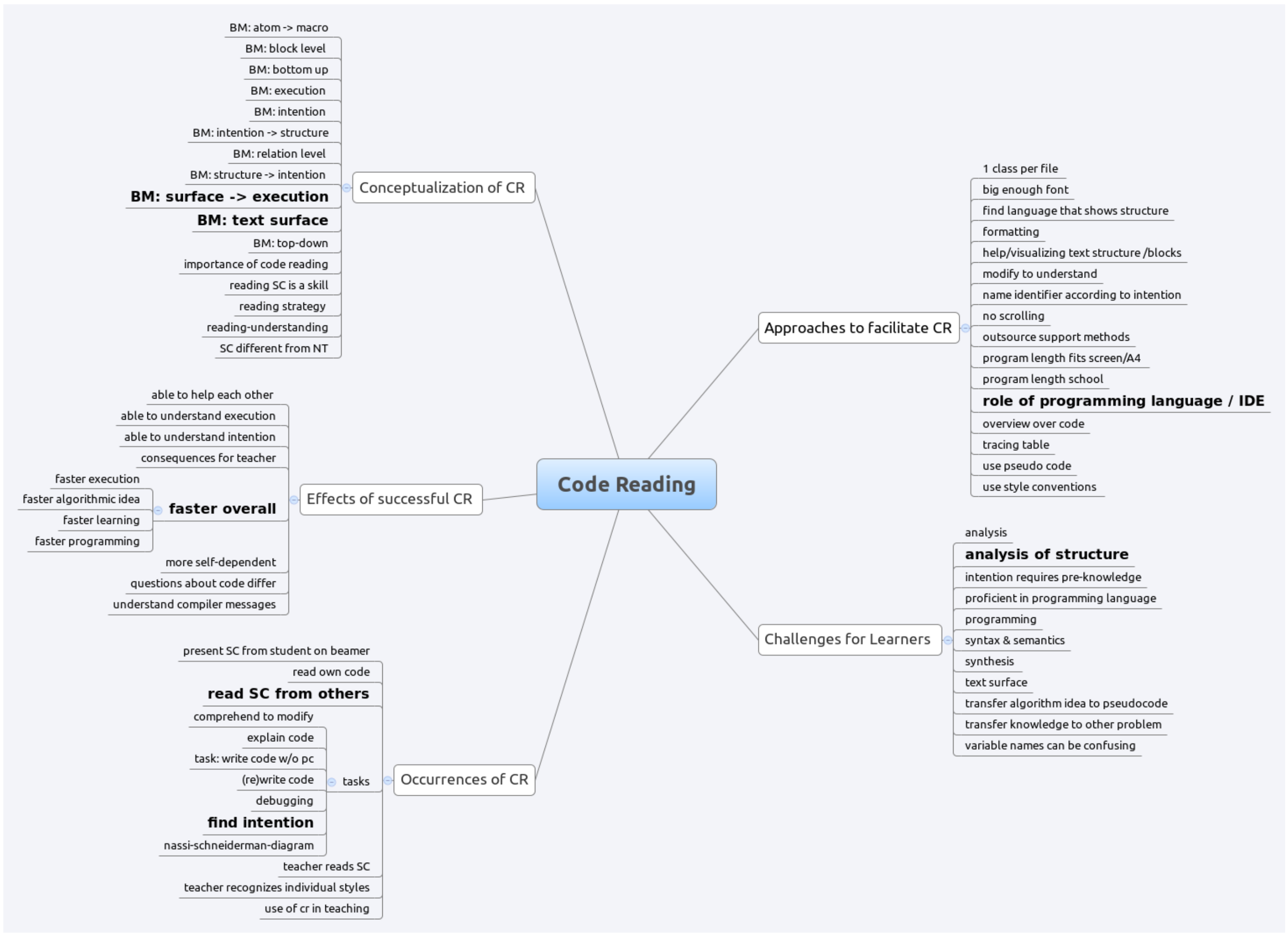


Figure 1: Code categories

source code is different from reading natural-language text, that it is a skill in itself and that there are strategies for it.

3.5.2 *Effects of successful Code Reading*

In accordance with the above described general conceptualization of CR as prerequisite (or first part) of program comprehension, the main effect of mastering this prerequisite is thought of as being able to be faster overall. All tasks related to CR or relying on successful CR will be done quicker; for example: "When e.g. students are presented a program text - what I'm doing frequently in my classes - they would be able to immediately answer what are the results [return values] of the presented functions, what kind of algorithm is in the text, and the would react much faster overall" (interview 1). Successful CR enables student to understand the programs' execution and intention.

Other aspects that were mentioned:

Successful CR would change the level of understanding and reflecting on code. It was assumed that questions and discussions about code would would have a better quality, as there wouldn't be the need to ask questions about the surface features of the text. Ideally, students would be able to read the text and then immediately understand the code's intention.

In addition, it would allow learners engage more often in self-dependent learning, e.g learning of new code constructs through reading code examples.

3.5.3 *Challenges for Learners*

Understanding the code's intention from the text surface was seen as the major challenge in reading. Or rather, it was considered as some challenge in its own, based on reading the text and understanding its parts. Some mentioned that such understanding would only be possible if the reader already possessed pre-knowledge about the program's intention.

One respondent describes the following main challenge: "In order to discern the algorithmic idea I need to combine the individual steps. As a prerequisite for this I need to know the programming language, the meaning of its constructs [...] It is a synthesis, [...] first I need to analyze: what is the structure [...] and then I need to synthesize: what is happening. And [...] to formulate from many small individual steps something big like this algorithm changes the order of the strings, or something like that well, I guess that is abstraction" (interview 1).

Perceiving the hierarchical block structure of the code, the 'analysis of its structure', poses another challenge - albeit the text surface is or might be perceived as a linear sequence of code elements (atoms).

A further challenge mentioned is the transfer from one task or example to other problems.

3.5.4 *Occurrences of Code Reading*

In the classroom, there are several situations were learners need to read code. The most frequently mentioned example refers to reading code of others, e.g. the code of other learners, or examples presented by the teacher.

In addition, a variety of learning tasks and of diagnostic tasks (e.g. in exams) include the need to read code. A prominent example is the task to find out the intention of a given piece of code.

3.5.5 *Approaches to facilitate Code Reading*

Our interviewees stated several means to facilitate CR. They referred to the importance of a suitable programming language, and a suitable programming environment a lot. This is not surprising as the language and the IDE are determining the visual layout and features of the text surface. And, as we have seen in the conceptualization of CR, the text surface is what CR is concerned with.

Support therefore concentrates on changing the code to be read. Concrete suggestions often aimed at facilitating the analysis of structure, e.g. by keeping the code short enough to fit on one page and by using formatting and visualizations to highlight the structure.

4. SUMMARY OF RESULTS

In summary, the interviews shed light on different aspects of CR. These different aspects seem to form a coherent whole. Nearly all educators had to think for a while, before setting to answer the miracle question, as well as some of the following questions. One possible interpretation is that CR was not actively thought about and used in their everyday teaching of programming. However, during answering our interview partners engaged in reflection about the role of CR and could perceive different aspects and roles.

The views of the different persons were rather mutually enriching than contrasting each other. First of all, there seems to be a consensus that reading is the first part of comprehension - reading is conceptualized as perceiving the text. Then participants started to think about comprehension in terms of execution and intention (algorithmic idea). Thus reading is merely concerned with identifying and understanding code concepts on a low level, and on understanding the structure or 'outline' of the text. However during the interviews the more cognitive comprehension processes were sometimes also included in this initial act of reading.

Reading was regarded as important, but seldom taught or used directly. As such, reading is an integral part of learning programming, and is a common activity in the classroom. It occurs during e.g. introducing program constructs like elements of the language or a library / API being used, during programming (for bug fixing and stepwise refining the program, so that it shows the desired behavior), in classroom sessions were code of other learners or examples chosen by the teacher are discussed. It is also used in examinations or tests as a special kind of task (find-out questions).

In addition, two of our respondents even thought about a CR strategy for learners. It was not clear whether this should be taught, or if it was conceptualized as a general reading and comprehension process.

One account was this: Reading starts with analyzing the program's structure: Reading the blocks and thus perceiving the structure is the basis for perceiving the programs flow of execution, and the semantics of the concepts. This first analytic phase is then followed by a phase of synthesis and abstraction, were the algorithmic idea is extracted from this structure. Reading and comprehending is thus rather a bottom-up process (see also the quote in section 3.5.3).

The other account was this: First the reader has to build a map of the source code, a kind of orientation and perception of the semantics. This is needed to be able to navigate through the text. Then - by navigating - the 'territory' so to speak, can be explored in order to answer more detailed questions or to get a deeper understanding of the text. This second phase can also be seen - or is included in the classroom as a kind of interactive experiment, including e.g. changes of the text, executions with different inputs. Overall this process needs to be framed by some overarching notion or idea about the intention of the program. Program reading and comprehending is thus a rather top-down process.

Overall the attempts to describe reading strategies remain preliminary; but there are some aspects of reading strategies that were mentioned by several of the participants:

1. Analysis of the structure of the text surface is important. Readers need to be aware of the block structure of the code.
2. Probably the immediate or first step of program comprehension in terms of making inferences is aiming to comprehend the execution of the program (form text surface to execution).
3. To find out or comprehend the intention of a program without additional help is very difficult.

However, CR is rarely (in our interviews only once) seen explicitly as a learning goal in itself. While it is seen as a natural part of programming classes, it is rarely -brought up in terms of an educational challenge. The question remains whether there is the need or a place for explicitly teaching CR, or if the challenge of CR is best met by choosing appropriate tools to facilitate reading. Our participants suggest the importance of appropriate languages and programming environments, of strategies or support for surveying the code, and help for visualizing the block structure.

While the educators do see problems related to reading and program understanding, they also seem to accept these problems as natural or given part of the learning process.

5. CONCLUSION

In the literature, as well as in our interviews, we found many possible uses of CR, and also many aspects of learning programming where CR is needed. CR is connected to comprehending programs and algorithms, or algorithmic ideas, as well as details, like e.g. semantics of constructs. Additionally there are many possible uses of CR in terms of tasks based on reading in order to facilitate learning to program. But at the same time there is not much knowledge about the reading and comprehension process of learners. If we knew more about this process, suggestions for learning tasks could probably be improved.

Another area is the need for reading and comprehending as part of programming. Again, more knowledge about this part could lead to improved strategies for teaching programming. A possible means to foster learning is to teach reading directly, including reading strategies. However, so far, we do not know much about good reading strategies (neither of experts, nor of learners).

In addition, in the interviews reading was sometimes seen as learning goal in itself. One participant connected reading to automated code generation and the increased role of automatically generated code. Therefore reading should probably be made more explicit as learning goal in itself.

Despite being an essential part of program comprehension, code reading is mostly just implicitly reflected in teaching programming and is worthy of deeper inspection.

6. REFERENCES

[1] Bennedsen, J., and Caspersen, M. E. Revealing the programming process. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education* (St. Louis, Missouri, USA, 2005), ACM, pp. 186–190.

[2] Busjahn, T., Schulte, C., and Busjahn, A. Analysis of code reading to gain more insight in program comprehension. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research* (Koli, Finland, 2011), ACM, pp. 1–9.

[3] Campbell, W., and Bolker, E. Teaching programming by immersion, reading and writing. *Frontiers in Education, 2002. FIE 2002. 32nd Annual 1* (2002), T4G–23.

[4] de Shazer, S., and Dolan, Y. M. *More Than Miracles: The State of the Art of Solution-Focused Brief Therapy.* Haworth Brief Therapy. Haworth Press, New York, 2007.

[5] Deimel, L. E., and Naveda, J. F. Reading computer programs: Instructor's guide to exercises. Tech. rep., DTIC Document, 1990.

[6] Denny, P., Luxton-Reilly, A., and Simon, B. Evaluating a new exam question: Parsons problems. In *Proceeding of the Fourth international Workshop on Computing Education Research* (Sydney, Australia, 2008), ACM, pp. 113–124.

[7] Diehm, R.-A., and Lupton, M. Approaches to learning information literacy: a phenomenographic study. *The Journal of Academic Librarianship 38*, 4 (2012), 217–225.

[8] Diethelm, I., Borowski, C., and Weber, T. Identifying relevant CS contexts using the miracle question. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2010), Koli Calling '10, ACM, pp. 74–75.

[9] Gayani Samaraweera. Programming from the reader's perspective: Toward an expectations approach. Macneil Shonle and John Quarles, Eds., pp. 211–212.

[10] Hilburn, T. B., Towhidnejad, M., and Salamah, S. Read before you write. *Software Engineering Education and Training (CSEE&T), 2011 24th IEEE-CS Conference on*, 371–380.

[11] Hsieh, H.-F., and Shannon, S. E. Three approaches to qualitative content analysis. *Qualitative health research 15*, 9 (2005), 1277–1288.

[12] Kinnunen, P., and Simon, B. Building theory about computing education phenomena: a discussion of grounded theory. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (2010), ACM, pp. 37–42.

[13] Kintsch, W., and Van Dijk, T. A. Toward a model of text comprehension and production. *Psychological review 85*, 5 (1978), 363. 04521.

[14] Kölling, M., and Rosenberg, J. Guidelines for teaching object orientation with java. *SIGCSE Bull. 33*, 3 (2001), 33–36.

[15] Lionel E. Deimel, J. The uses of program reading. *SIGCSE Bull. 17*, 2 (1985), 5–14.

[16] Lister, R., Fidge, C., and Teague, D. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *SIGCSE Bull. 41*, 3 (2009), 161–165.

[17] Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *SIGCSE Bulletin 38*, 3 (2006), 118–122.

[18] Lopez, M., Whalley, J., Robbins, P., and Lister, R. Relationships between reading, tracing and writing skills in introductory programming. In *Proceeding of the Fourth international Workshop on Computing Education Research* (Sydney, Australia, 2008), ACM, pp. 101–112.

[19] Mason, M. Sample size and saturation in PhD studies using qualitative interviews. In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research* (2010), vol. 11. 00135.

[20] Merrienboer, J. J. G., and Krammer, H. P. M. Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science 16*, 3 (Sept. 1987), 251–285.

[21] Pennington, N. Comprehension strategies in programming. In *Empirical studies of programmers: second workshop* (1987), Ablex Publishing Corp., pp. 100–113. 00338.

[22] Pennington, N. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology 19*, 3 (1987), 295–341. 00475.

[23] Raymond, D. R. Reading source code. In *Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research* (Toronto, Ontario, Canada, 1991), IBM Press, pp. 3–16.

[24] Rooksby, J., Martin, D., and Rouncefield, M. Reading as part of computer programming. an ethnomethodological enquiry. In *Proceedings of the 18th Workshop of the Psychology of Programming Interest Group* (2006).

[25] Schulte, C. Block model - an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceeding of the Fourth international Workshop on Computing Education Research* (Sydney, Australia, 2008), ICER '08, ACM, pp. 149–160.

[26] Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., and Paterson, J. H. An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports* (Ankara, Turkey, 2010), ITiCSE-WGR '10, ACM, pp. 65–86.

[27] Selby, C. Four approaches to teaching programming. In *Learning, Media and Technology: a doctoral research conference* (London, 2011).

[28] Simon, Lopez, M., Sutton, K., and Clear, T. Surely we must learn to read before we learn to write! In *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95* (Wellington, New Zealand, 2009), Australian Computer Society, Inc., pp. 165–170.

[29] Spinellis, D. Reading, writing, and code. *Queue 1*, 7 (2003), 84–89.

[30] Uwano, H., Nakamura, M., Monden, A., and Matsumoto, K.-i. Exploiting eye movements for evaluating reviewer's performance in software review. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E90-A*, 10 (2007), 2290–2300.

[31] Venables, A., Tan, G., and Lister, R. A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the fifth international workshop on Computing education research workshop* (Berkeley, CA, USA, 2009), ACM, pp. 117–128.

[32] von Mayrhauser, A., and Vans, A. M. Program understanding - a survey. *Colorado State University Computer Science Technical Report CS-94-120* (1994).

Recording and Analyzing In-Browser Programming Sessions

Juha Helminen[*] , Petri Ihantola, and Ville Karavirta
Department of Computer Science and Engineering
Aalto University
Finland
juha.helminen@aalto.fi, petri.ihantola@aalto.fi, ville@villekaravirta.com

ABSTRACT

In this paper, we report on the analysis of a novel type of automatically recorded detailed programming session data collected on a university-level web programming course. We present a method and an implementation of collecting rich data on how students learning to program edit and execute code and explore its use in examining learners' behavior. The data collection instrument is an in-browser Python programming environment that integrates an editor, an execution environment, and an interactive Python console and is used to deliver programming assignments with automatic feedback. Most importantly, the environment records learners' interaction within it. We have implemented tools for viewing these traces and demonstrate their potential in learning about the programming processes of learners and of benefiting computing education research and the teaching of programming.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Human Factors

Keywords

Computing Education Research, Computer Science Education, Web Based Programming Environment, Programming Assignment, Programming Session, Python

1. INTRODUCTION

With the onset of MOOCs and the general trend towards web-based cloud computing, it is important to study which kinds of programming environments can be implemented in this mode of computing and what new opportunities it may offer in terms of computing education research. A web-based programming environment has, among others, the advantage of presenting learners with a common working environment that can be managed and updated in a rapid and centralized fashion. In MOOCs, this is one possible approach to creating a common classroom environment when participants are geographically dispersed and obviously cannot have access to shared lab equipment and, in fact, may not even always have adequate control over the computer they are using for their studies. Furthermore as we have control over the environment, we are actually, with relative ease, able to gain access to how learners work when solving programming assignments. Indeed, it is interesting and useful to study how and to what extent this could enable us to examine learners' working habits and processes without direct physical intervention such as human or video observation, in real time, and on a large scale. It would be valuable to get insight into how students work, identify struggling students, and give feedback on their process while discouraging bad practices right as they are working on the problem. Following this, in our research we are interested in:

- How to collect accurate data, on a large scale and with minimal effort, on what students actually do when they are programming?
- How to use this data to learn, in quantifiable means, about the difficulties that learners face and the behaviors they, for good or bad, exhibit?

In this work, we contribute to answering the questions listed by

- presenting an implementation of tools for collecting and viewing detailed data about Python programming sessions and
- demonstrating their use in analyzing students activities, such as, examining how they made use of the integrated interactive console and presenting common exceptions encountered by students.

2. RELATED WORK

With today's computing environments it is quite feasible to keep detailed logs of the interaction in software applications. In terms of programming environments, this means, for example, recording how and when the developer edits and executes code. Indeed, there is growing interest in research on analyzing automatically recorded programming sessions as a means of gaining insight into programming processes.

[*]Corresponding author.

Koli Calling '13, November 14–17, Koli, Finland
Copyright 2013 ACM 978-1-4503-2482-3/13/11 ...$15.00.
http://dx.doi.org/10.1145/2526968.2526970

Early Work

There exists already a long history of research dating back to the 80s that has tried to identify the kinds of mistakes students learning to program are prone to make and how their time is spent during development activities. Spohrer and Soloway used an instrumented version of the VAX 750 operating system to record each syntactically correct program compiled on an introductory Pascal programming course [21, 22]. In an in-depth analysis of the first syntactically correct programs for each of the around 50 students in 3 different problems, they concluded that just a few types of bugs accounted for a majority of the mistakes in students' programs, the implication for educators being that they can most effectively improve their students' performance by changing instruction to address and eliminate the high-frequency bugs.

Java Compiler Errors

Decades later, Jackson et al. recorded compiler errors during one semester with their custom-built Java IDE at the US Military Academy [7]. All errors from the on-site systems were collected including data from 583 students and 11 faculty members. They noted similarly that the top ten types of errors represented over half of the total number collected. Furthermore, they reported that there was a discrepancy between the errors they most observed with students and what faculty had believed to be the most common, thus adding to the value of this data. An interesting feature was that any cadet could access a web page with a detailed analysis of their most common errors.

Ahmadzadeh et al. also collected Java compiler errors, but with attached timestamps and source code, using an instrumented version of the Jikes compiler on a CS1 course as students solved 15 exercises in the JCreator environment [1]. They divided the errors into three categories: syntax errors dealt with the grammar of the language, semantic errors with the meaning of the code being inconsistent, e.g. using a non-static global variable inside a static method, and lexical errors with unrecognized tokens. They found the distribution in their data to be 63 % semantic error, 36 % syntax errors, and 1 % lexical errors. They also noted that only 6 of the 226 distinct semantic errors constituted more than half of the error occurrences in each unit of exercises dealing with a single concept. Additionally, they reported a weak negative correlation between time debugging and the mark achieved in the online exams[1]. In another experiment, the students were given a debugging task[2] in the form of a program with both compiler errors and logical mistakes. On closer inspection of the recorded compiler error traces, they concluded that many students with a good understanding of programming[3] still do not acquire the skills to debug programs effectively. Students successful at locating bugs were observed using print statements or a filtering approach of commenting out certain lines. They discovered that the majority of good debuggers are good programmers while less than half of the good programmers are good debuggers. In summarizing their work they suggest that because a good portion of good programmers did not seem to be aware of some bugs, and thus were unable to debug them, they actually lacked an understanding of the actual program implementation and suggest that the ability to read and understand other people's code is an important skill and different from writing one's own.

BlueJ and Java Compilation Behavior

Jadud has studied novice programmers' Java compilation behavior with a more general focus investigating the complete edit-compile cycle of alternating between editing and compiling a program instead of just the errors [8]. He collected data from 63 students on their first university-level programming course working on programming assignments in classroom tutorial sessions. The BlueJ programming environment was modified to, among other metadata, record the time, the source code and, the possible compiler error[4] at every compile. He presents a frequency distribution of compiler errors by type which shows that the minority of different types of compiler errors account for the majority of errors dealt with by students. Indeed, the 5 most common errors accounted for 58 % of all errors generated by students. Furthermore, on closer inspection he notes that the most common error types are handled in less than 30 seconds and require adding or removing only a few characters. Overall, it was common to spend very little time before recompiling after an error. He reports also that instructors observed students often recompiling their programs without attempting to fix or otherwise understand the error they had received. The data seems to corroborate this incidence of students in some manner not believing in the error reported to them since for example 21% of the time for a repeated missing semicolon error the next error was the exact same error with source code unchanged[5]. On considering the students' behavior as individuals Jadud notes that the number of compilations in a single lab session seems to suggest a grouping of students into typical ones, those who compile more often than others, and students whose behavior cannot be adequately described by this one graph. Moreover, some students had compiled up to almost 60 times in a single one-hour lab session, much more than their peers, and he had no explanation for this. Finally Jadud points out that the environment could be modified to guide students past the common errors observed but cautions making this a crutch that students would rely on doing the work for them instead of learning from it.

In another paper, Jadud reports similar observations from students of several first-year programming courses [9]. Furthermore, the paper reports a case study of a single weaker student working on one Java assignment in BlueJ. The session description illustrates the student struggling with syntax errors as the compiler errors are somewhat misleading and cause the student to introduce more and more errors to the code. He ends up spending a significant amount of time dealing with syntax errors instead of the actual program design task. During the session, the student goes on to employ a pattern Jadud calles "remove the error" where the student removes or comments out lines in order to remove and possibly locate an error by way of elimination and maybe later add the code back in. Jadud notes that this strategy was observed frequently but less commonly led to success and

[1]Statistical significance was not discussed.

[2]Students did not have any high-level debugging tools available.

[3]Based on their marks.

[4]BlueJ only provides the student with a single error for a compilation even if many exist in the code.

[5]This might also be due to the student just refreshing the error message that was cleared accidentally as noted by Jadud in [9].

seemed in general to be an indication of being lost and confused. Another common behavioral pattern Jadud describes is that students will move on from a particularly problematic piece of code without addressing the issue. Stronger students will do something else and fix the problem later but weaker students just tend to introduce more errors. Overall, he notes that the students exhibited similar behavioral patterns in struggling with the syntax as described by Perkins et al. in solving programming problems in general [15].

Indeed, in this early work from the 80s Perkins et al. discussed some powerful characterizations of the ways how students approach solving a programming problem [15]. They observed young students, high school students learning BASIC and elementary school students learning LOGO, with an experimenter providing help only when needed in a progression from general strategic prompts to specific advice. Where students were unable to proceed quickly, they observed two general behavioral patterns that they classified as *stoppers* and *movers*. Stoppers will simply stop overwhelmed with the belief that they cannot solve the problem on their own and show unwillingness to explore it any further, perhaps immediately moving on to the next one. Movers, on the other hand, will consistently try one thing after another without ever really seeming to be stuck. At the far end of this, *extreme movers* tend to try new fixes with hardly any reflection or apparent convergence to a solution, and end up abandoning promising ideas prematurely or even going in circles trying the same thing over and over. As Perkins et al. discuss, the students are in their own way disengaging themselves from the task as instead of dealing with their mistakes and the information this might yield, they avoid them by constantly moving on. Moreover, they reported that both of these less optimal behaviors, stoppers and extreme movers, were common.

Furthermore, they discuss that the students often followed an approach they call *tinkering*. Students first write some code and then end up trying to solve the problem by making many successive small edits in the hopes of fixing the program. They describe that tinkerers are movers and with sufficient tracing of their code, or close tracking as the authors call it, to localize the problem and some systematicity to avoid compounding errors, tinkering may lead to success. Indeed, Perkins et al. associate the effectiveness of tinkering to be closely related with the extent of tracing students perform on their code. Often tinkering works poorly because without sufficient understanding of their program tinkerers may assume that small changes will help when, in fact, a change in approach is needed. Some students may also allow these changes to accumulate untested or leave them in place when they have failed, ultimately, producing an incomprehensible tangle of code. Informed by these observations, the authors note that students could be explicitly taught the pitfalls of tinkering and encouraged in such practices as removing failed fixes and considering whether a completely different approach needs to be taken if several tinkers prove unsuccessful.

Going back to Jadud's work, he has also presented an HTML-based visualization of programming sessions in terms of compilation events [9]. The visualization highlights the types of errors with colour-coding, time spent and number of characters changed between compilations, and the locations of the edits and a possible previous compilation error. One can additionally view the associated source code for each compilation via a link. Finally, Jadud describes the error quotient (EQ) that aims to quantify how much a student struggles in a programming session based on the encountered compiler errors. Consecutive erroneous compilations add to the quotient and repeating the same error even more so. Jadud goes on to report a statistically significant, yet be it a weak, correlation between a student's EQ and assignment and exam grades[6]. Overall, Jadud notes that his tools and the EQ allow a teacher to easily view how students are doing when solving the assignments as opposed to simply looking at the end result, and plan interventions when appropriate.

In later work Jadud and Henriksen have published a reimplementation of the BlueJ data collection infrastructure that extends the capabilities by also collecting data about when methods on a class or object are invoked via BlueJ object diagrams [10]. The framework consists of a BlueJ extension and a server. Tabanao et al. have used this framework to collect data on a university CS1 course from a self-selected group of 143 students over 5 lab exercise sessions [23]. They found a statistically significant and moderate (R = -0.54) negative correlation between students' mean EQ scores and midterm exam scores. Consistent with earlier results, they also report similar errors as Jadud and that top ten error types accounted for 76% of all the compiler errors. In view of this, they suggest improving the debugging ability of students by informing them of the common errors encountered by beginning programming students and discussing why they occur and how to solve them. Continuing on this work with similar data, Tabanao et al. have also tried to identify characteristics of high-performing and at-risk students [24]. Based on their midterm exam score students were grouped into high-, average-, and low-performing, at-risk, students. They found statistically significant differences between the groups in the percentage of erroneous compilations (more errors when lower performance), the occurrence of a few of specific types of common errors (more occurrences with lower performance), and the distribution of time spent between compilations (more time with higher performance). Correlation tests revealed that there was a statistically significant relationship between these erroneous compilations, the time between compilations, and EQ, and the midterm exam score. They also built linear regression models to predict the midterm exam score where EQ proved most influential but ultimately the models failed to accurately place students into the at-risk group.

Rodrigo et al. have also collected similar BlueJ data as Tabanao et al. and supplemented this with human-observed affective states [18]. Besides investigating the relationship of affective states with midterm exam scores they found that the number of pairs of consecutive erroneous compilations and those with the same edit location have a statistically significant correlation with the scores, albeit a fairly weak. They failed to build a statistically significant regression model to predict exam score using compilation data. In continuing this work, Rodrigo and Baker generated a linear regression model of a student's frustration based on the average number of consecutive compilations with the same edit location, average number of consecutive pairs with the same error, and the average time between compilations [17]. They achieved statistically significant results in predicting a student's average level of frustration across all labs but the correlation was fairly weak. Per-lab frustration could not be predicted in this way. They envision that if frustration could be identified the

[6]The dataset only included students' work in the labs with no knowledge of how much or how they worked elsewhere.

student could receive a message from the system sympathizing with them and encouraging them to keep trying.

Continuing this line of research, Fenwick et al. have too collected similar Java compilation data in BlueJ, as in work discussed above, using their own extension called ClockIt [14]. In a survey about the value of insights that this type of data may provide, they found that both the large majority of CS students and faculty perceived it would be helpful for introductory CS students to know about the types of compilation errors encountered and how a student's habits compare to his or her peers[7]. A thing of note compared with much of the other work is that students could also view the few types of graphs visualizing their own activity through a web interface. As for observations, the top five errors recorded were in Jadud's top six and make up over half of of all the compiler errors [6]. They also presented similar descriptive statistics of the time between compilations and diagrams that seem to indicate that starting early leads to better grade and that incremental work pays off as well[8]. As students had inadvertently copied the event log file or a project without this history of creating it, they were also able to identify these as potential occurrences of plagiarism.

In recent work, Utting et al. describe a plan to have a similar data collection feature built into future BlueJ versions[9] in order to collect data about students' behaviour on a large scale as opposed to the previous work dealing with closed-lab sessions at a single institution [25]. They plan to include code-edits on a line-by-line basis, compilation events, and other events such as unit test, debugger, and version control use. Furthermore, they intend to anonymize and host this data in an SQL database and allow others access for research purposes while researchers could still also collect identifiable data from their own institutes. Overall, they believe that the large scale will now allow less often used tools like the debugger and rarer error messages to be studied.

Finally, Retina is a system that not only collects Java code snapshots at compilations like in the other work discussed but also makes suggestions to students based on this data via instant messages [13]. The recommendations are based on rules such as suggesting the student work in smaller increments if their rate of errors per compilation is higher than average, or to seek help if they are spending more time on the assignment than expected. Retina's data collection is implemented as both BlueJ and Eclipse extensions as well as a modified javac compiler.

Marmoset and Code Snapshots

Spacco et al. have also collected programming session data but at a finer granularity than compilations [20]. They have used Marmoset, an automated project snapshot and submission system that commits snapshots of a student's code to an individual CVS repository every time the student saves his or her work. The automated recording of student's intermediate work is implemented with an Eclipse plugin. Using this data source, they investigated the accuracy of their bug detectors that use static analysis but, pertaining to this work, they found the CVS-based representation inadequate for exploring this type of data and present the design of a relational database schema for storing it, thus allowing SQL queries on it. The schema is built around the idea of tracking individual lines across the different versions of the code in a file as it evolves. Lines in successive versions of the code are regarded equivalent, i.e. modified instances of the same tracked line, on the basis of ignoring changes in whitespace and comments and the lines having only a small edit distance and the same relative location[10]. The results of unit tests and static analyses are also included with each snapshot in the database. Mierle et al. have also investigated students' code from CVS repositories and implemented a system for storing this data in an SQL database [12]. Furthermore, they attempted to find features that would correlate with final course grades but were unable to find any strong predictors.

In a very recent continuation of the Marmoset work, Spacco et al. report on a dataset collected in a CS2 course consisting of submissions to 6 assignments by 96 students [19]. They visualized the distribution of snapshots against the time of day and found that their students most preferred to work around 4 to 6 pm. Additionally, they visualized the number of snapshots being produced relative to the deadline and found that the majority of work was being done 48 hours before the deadline. Furthermore, they tested for linear regression between the time of the first snapshot and the final score in an assignment. They report that starting early correlates with better scores[11]. Using a heuristic of counting any time between snapshots whose difference in time was less than 20 minutes into a total time of actively working on the assignment, they also found that this time spent coding had a relationship with the final score of an assignment[12]. Something of note too is that they used a submission scheme where the amount of more detailed feedback was limited within a period of time in an attempt to get students working earlier before the deadline and discourage procrastination.

In other very recent work, Balzuweit and Spacco have discussed a prototype web service for visualizing snapshot data like the one Marmoset records [3]. They have reduced the data points into tuples of an identifier, a timestamp, a score, and a label (e.g. an error message), and show a visualization of a student's activity with regard to date (x-axis) and time of day (y-axis) where each data point is additionally colored according to its correctness as per unit tests. They hope to start work on developing standard data formats for storing and analyzing this type of data.

Web-CAT and Submission Data

Web-CAT is another automated grading system that has been made use of in examining students' progress and behavior in programming assignments. In using Web-CAT students are usually allowed to submit their work for assessment an unlimited number of times before the deadline. Edwards et al. have

[7]The survey reporting lacked details such as how large the sample sizes were.

[8]These inferences were not subjected to any statistical evaluation. A single incremental session was defined as a period of time where all successive events were less than 60 minutes apart.

[9]Version 3.1.0 that now includes the data collection features appears to have been published in June 2013 at `http://www.bluej.org/`.

[10]The method for computing the edit distance or the threshold of small are not specified.

[11]The test was statistically significant but the strength of association was very weak ($F(1, 499) = 49.94, p < 0.001, R^2 = 0.09$).

[12]The test was statistically significant but the strength of association was very weak($F(1, 492) = 6.3, p < 0.05, R^2 = 0.01$).

examined five years of programming assignment submission data from their first three programming courses [4]. They partitioned each sequence of submissions to an assignment by a student to two groups, to sequences that resulted in a score above or below 80 % of maximum. Compared with other work, in order to provide greater confidence that the cause of differences in scores has to do with differences in student behavior, rather than some innate ability that particular students possess, they then went on to remove all students from the data that consistently placed in either of the groups. They were then left with 633 students. In analyzing this data, they were able to find statistically significant results suggesting that when students received scores placing them in the high-performing group, they started earlier and finished earlier than on assignments where they received lower scores. They did not appear to spend any more time on their work. Also, around two thirds of the higher scores were received by individuals who started more than a day in advance of the deadline, while around two thirds of the lower scores were received by individuals who started on the last day or later. They suggest a possible explanation to be that when students start earlier, they simply have more opportunities to get help and then go on to perform better.

In other recent work, analyzing a similar large dataset from their locally developed Web Submissions System and data ranging from introductory to advanced assignment work, Falkner and Falkner investigated the relationship of the timeliness of submissions to students' later timeliness of assignment submissions and their average grade from courses [5]. They define a measure of average timeliness that is the average across all assignments where each is either scored 1 for the final submission being on time, i.e. before the deadline, or -1 for being late[13]. Partitioning students into two groups based on their first assignment submission – on time or late – they found that students who submit their first piece of work late seem more at risk of submitting late for the rest of their career and that this behavior also seems to correlate with the grades[14]. They suggest that with the reasonable assumption of a late submission leading to reduced marks or cascading lateness, the timeliness of the final submission of the first assignment can provide an indicator of the future likelihood of under performing and thus allow some resources to be assigned for early intervention.

Data Mining and Hidden Markov Models

Taking a very different analysis approach, Allevato and Edwards have also applied the data mining technique of frequent episode mining to discover frequently occurring patterns in Web-CAT submissions [2]. Data from two assignments by 102 students taking a C++ course after two semesters of learning Java was codified into a time series of events, such as, the student made his or her first submission or first submission that compiled without errors, the number of methods in the code changed, or the cyclomatic complexity changed. In comparing the frequent episodes of events between the students that scored well and those who scored poorly, they found that the weaker students had a frequent pattern of removing entire methods from their code which the other group did not have. They suggest this to be due to deficiencies not only in the students implementation but also in their design which can be more difficult to resolve. Additionally, they compared the frequent episodes occurring early in the course, in the first assignment, and late in the course, in the final assignment, in order to see if they might see some changes but were unable to find indication of a change in habits.

In other recent work, Piech et al. have also used data mining and machine learning techniques to analyze programming session data [16]. They recorded the compilation events in Eclipse on their CS1 course from a self-selected group of students. A Karel the Robot assignment using a Karel language based on Java was analyzed more closely. They clustered the many states of code into more high-level milestones using a metric based on differences in the AST and the API calls of the programs and using these modeled each student's development path using a Hidden Markov Model. They further clustered the individual students' HMMs to create a graphical state machine model of the few different high-level development paths students undertook to solve the assignment. Finally, they showed that the resulting different paths correlated with students' performance and with more power than the score achieved in the assignment. They also report testing the approach on a more complex Java assignment, thus proving its general applicability.

Another line of research that makes use of Hidden Markov Models in capturing the development behavior of students is the work by Kiesmüller et al. [11]. They have studied students' problem solving strategies in the finite state machine -based visual microworld programming environment Kara. Using log data from the system they have implemented an application that is able to recognize four different types of problem solving strategies automatically in real-time.

Summary

There is a steadily growing body of research on investigating learners' programming patterns and behavior from different kinds of automatically captured log data. Previous work has investigated data at many different granularities ranging from snapshots recorded every time the student saves his or her work as in Marmoset and data collected at every compilation as in the many studies surrounding BlueJ, to the submissions of automated grading systems such as Web-CAT. Data collection at the finer granularities has been implemented into modified versions of compilers or as extensions to IDEs such as BlueJ and Eclipse. The great majority of this work has focused on Java. Analyses of this type of data have tried to quantify the occurrences of compiler errors, attempted to distill activity data into usable indicators for high or poor future performance, as well as, tried to identify some higher-level behavioral patterns. To mention few of the results that seem to be gaining evidence from several sources, it seems that a minority of top types of errors accounts for the majority of the errors students learning to program encounter, consecutive erroneous compilations may be a sign of a student struggling, and starting work on assignments early will on average lead to better performance while a lot of the students' work still takes place only close to the deadline. Recent work has applied data mining and machine learning techniques to successfully discover knowledge about students behavior. Overall, the work has been motivated by a need for both discovering early actionable indicators of students struggling that would allow intervention and a desire to test and quantify anecdotal beliefs such as that starting work early and small incremental work will lead to better performance and

[13]Late submissions were allowed but these could not receive the maximum score.

[14]No statistical analyses were performed.

common types of errors students struggle with. Following the findings, there have been suggestions to explicitly teach students about observed difficulties and behavioral patterns that seem less desirable but only a few have implemented such feedback mechanisms of presenting observations on a student's behavior back to them.

With regard to this work, we are not aware of any previous work, besides that upcoming with the new version of BlueJ, that has collected data at the fine granularity of edits that we have, or included interactive console use. Indeed, Jadud has pointed out as possible future work instrumenting the programming environment further to find out what students do when their program is syntactically correct in order to "lift us out of the purely syntactic view of programming that we have" [9]. Similarly, Rodrigo et al. have mentioned as possible future work to attempt developing a finer-grained detector for frustration and other affective states, using more detailed data, such as keystrokes and mouse movements, as well as the coarse-grained compilation data already utilized [17]. In discussing the BlueJ data collection initiative and opting to collect quite detailed data, Utting et al. noted that an important decision is the selection of the data to be captured because that will determine the nature of the research questions that may later be investigated using this data [25]. Finally, our work deals with Python and we are not aware of any previous frequency analyses of Python exceptions during programming sessions.

3. DATA COLLECTION

Assignments and Learners

We collected data from a 5 ECTS Web Software Development course in Fall 2012 at Aalto University. The data analyzed in this study is from a compulsory exercise round on Python. The round had one multiple choice questionnaire on syntax and execution and three small programming assignments.

All the assignments could be submitted an unlimited number of times. If the student got full points in the multiple choice questions, then completing only two of the programming assignments was enough to pass the round and there was no reward for extra points. Almost all students who attempted the programming assignments managed to solve them. Indeed, 150, 149, and 128 students solved the assignments 1, 2, and 3, respectively. All the assignments were graded to give either zero or full points.

The backgrounds of the students varied greatly. About half of the students were master's or bachelor's level CS majors and the rest were from various other engineering disciplines. Whereas some had previous experience with both Python and web programming, some had only little programming experience in general.

Method and Tool

The programming assignments were delivered via a newly implemented web-based programming environment shown in Figure 1. The environment was integrated to the learning management system used on the course. On the left there is the code editor. It is a fully web-based programming editor that uses the CodeMirror library[15]. It provides all the functionality of a basic text editor such as cut-copy-pasting. In addition, it does syntax highlighting and has support for smart indentation that reduces the amount of required typing by, for example, automatically moving the caret to match the indentation of the previous line when entering a newline. On the right there is a console area that provides a fully functional interactive Python console including a command history feature. The console interface uses the jqconsole library[16]. The divider between the two areas can be dragged to assign more space to either of them. The buttons on the left provide functionality for running the code in the editor, submitting it for assessment and feedback, switching to fullscreen mode, and reloading the template code of the assignment. When the code is run, any output or errors of the execution are shown in the console. Also, the console shares the same Python execution environment, so after running the code in the editor, its functions and classes can be accessed – and most importantly interactively tested – in the console. The buttons on the right provide functionality for clearing the console, terminating execution, and resetting the Python environment. The programming environment had never been used on a course before and, thus, none of the students had any previous experience with it.

Python code execution in the environment is done on the client-side, in-browser and is accomplished using the jsrepl library[17] and the C implementation of Python, CPython, compiled into JavaScript via LLVM[18] using the emscripten library[19]. HTML5 web storage feature[20] is used to save and restore the contents of the editor and console including command history of a programming session. HTML5 fullscreen API[21] is used for enabling the environment to occupy the whole screen space not unlike a native application.

The feedback consisted of the execution of unit tests whose code, expected results, and student's program's results were shown. Students passed the assignment when all the tests passed. Figure 1 shows a reproduced snapshot of a student's use of the environment. On the left, the student has written a program that tries to meet the requirements in assignment 2 as described in the previous section. From the console we see that the student has executed the program and then entered a few commands in the console to test and confirm that it functions correctly. Then the student has submitted the solution and received as feedback a description of a failed test.

In this study, the web-based programming environment was used to record a detailed trace of the students' work. Students were not, however, required to use the environment to create the solution, only to submit using it, but when they did, their development activities were logged. This includes all code edits, running the code in the editor, command executions in the console, or submitting and the outcomes of all these. Additionally, using the Page Visibility API[22] and the web document's `focus` events, the trace tries to include data on when the student exited from and returned to the environment. Overall, this approach of a web-based programming environment that integrates the editor and the execution environment allows us in a relatively simple manner

[15]`http://codemirror.net/`

[16]`https://github.com/replit/jq-console`

[17]`http://replit.github.io/jsrepl/`

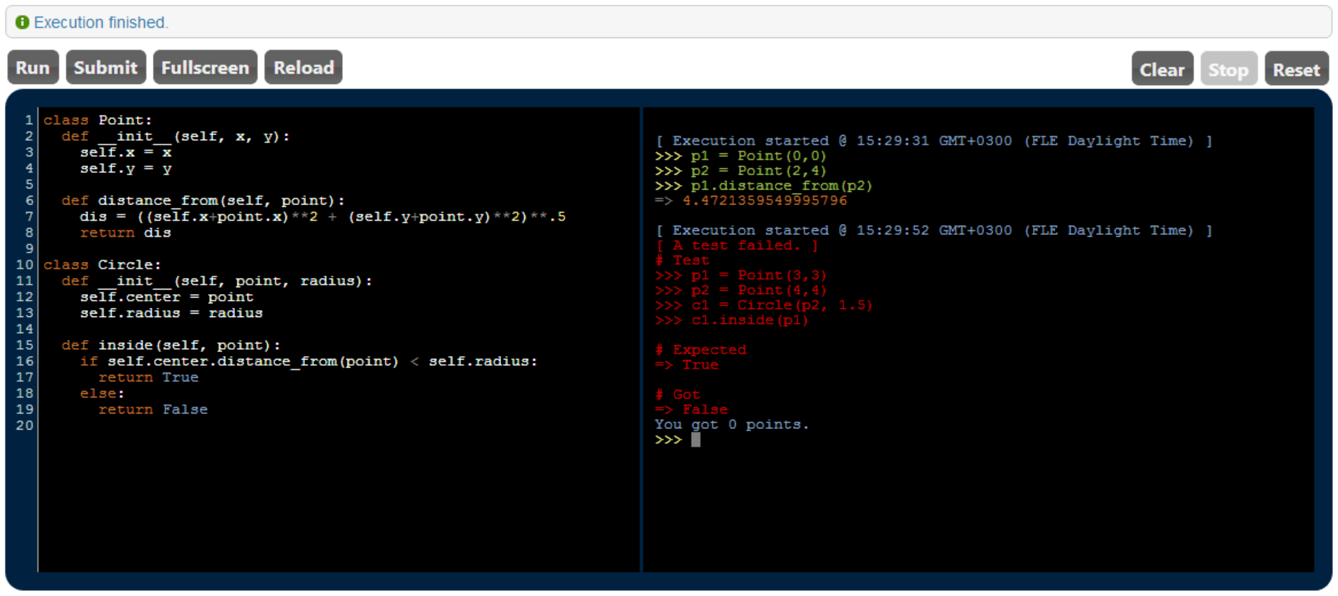
[18]`http://llvm.org/`

[19]`https://github.com/kripken/emscripten/wiki`

[20]`http://www.w3.org/TR/webstorage/`

[21]`http://www.w3.org/TR/fullscreen/`

[22]`http://www.w3.org/TR/page-visibility/`



```python
class Point:
  def __init__(self, x, y):
    self.x = x
    self.y = y

  def distance_from(self, point):
    dis = ((self.x+point.x)**2 + (self.y+point.y)**2)**.5
    return dis

class Circle:
  def __init__(self, point, radius):
    self.center = point
    self.radius = radius

  def inside(self, point):
    if self.center.distance_from(point) < self.radius:
      return True
    else:
      return False
```

```
[ Execution started @ 15:29:31 GMT+0300 (FLE Daylight Time) ]
>>> p1 = Point(0,0)
>>> p2 = Point(2,4)
>>> p1.distance_from(p2)
=> 4.47213595499995796

[ Execution started @ 15:29:52 GMT+0300 (FLE Daylight Time) ]
[ A test failed. ]
# Test
>>> p1 = Point(3,3)
>>> p2 = Point(4,4)
>>> c1 = Circle(p2, 1.5)
>>> c1.inside(p1)

# Expected
=> True

# Got
=> False
You got 0 points.
>>> 
```

Figure 1: The in-browser Python programming environment used in the assignments.

to collect rich data on students' programming processes. In fact, it provides a rather novel data source on students' testing sessions in the interactive console.

4. ANALYSIS AND RESULTS

The traces include a lot of data to process. To support the analysis, we implemented a web-based tool that allows us to explore the traces as they are recorded in the system. Processing of the traces is not done post-hoc but reports are immediately available for any new trace transmitted to the server. Figure 2 shows one of the views. This one focuses on what the code looks like when it is run or when the student pauses for a longer period of time after editing it – 10 seconds was the threshold value here. Other views include ones focusing on students' use of the console, students' use of the RUN button – similar to tracking compilations as in much of the previous work by others – students' edits of the code in detailed steps, and summary statistics of all students' traces in an assignment.

Testing Patterns

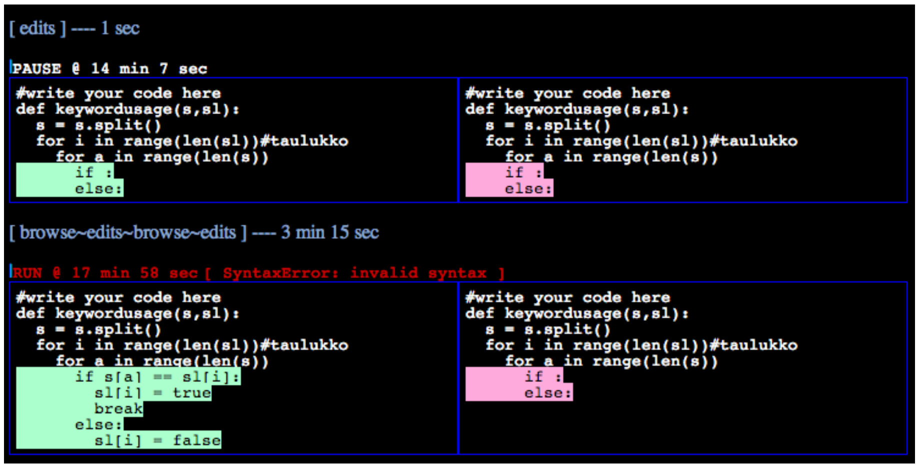
In perusing through the logs of programming sessions we observed two obvious main approaches to testing. Students would either attach test code at the end of their program and run it each time they ran their code or just run their program as is and then exercise its functionality from the console. The latter kind of data has rarely been analyzed before. Thus, we focused our investigative efforts on this and went through the console interaction in all the traces for assignments 2 and 3. The first assignment was left out of the analysis because there initially was an error in the assignment and it was modified mid-course. Also, you have to keep in mind that while we analyzed all the traces, similar to much of the previous work with students opting in to submit their data, our view is not complete because some of the students did also to varying extents work on their solution outside the web environment.

A great majority of the testing focused on the test cases given in the description of the assignments. Some students did nevertheless use test cases they had come up on their own. In an attempt to exclude such cases where the use of their own test input was not intentional but, for example, a typo, we only included in this category cases where the student had repeatedly used their own test cases (more than once). After submitting but failing to pass (which happened rarely) students would move on to using the test where their code failed during assessment. Finally, about a third of the students did not enter a single command to the console. More detailed statistics on the types of test cases used are found in Table 1. Each individual student's console testing behavior between assignments 2 and 3 was quite consistent in that they almost always exhibited the same patterns in the two assignments. In assignment 3, the test cases used to test students' solutions and give points were exactly the same as those given in the description of the problem. However, some students did much more thorough testing on their own in the console. An example of such a console session is shown in Figure 3.

Table 1: Students' use of the console for testing. The three latter categories are not mutually exclusive.

	Assign. 2	Assign. 3
Did not use	49 (32%)	52 (34%)
All the test from the assignment	82 (53%)	74 (48%)
All the tests from feedback	8 (5%)	- (-)
Own tests	26 (17%)	19 (12%)

In addition to examining how their program functions, some students would also use the console for more general exploration of language constructs and features before incorporating related code to their solution. In assignment 2, many students tried out the mathematical functions `sqrt` and `pow` that were needed in the assignment. An example of such behavior is shown in the console session in Figure 4. In



```
[ edits ] ---- 1 sec

PAUSE @ 14 min 7 sec
```

```
#write your code here
def keywordusage(s,sl):
  s = s.split()
  for i in range(len(sl))#taulukko
    for a in range(len(s))
      if :
      else:
```

```
#write your code here
def keywordusage(s,sl):
  s = s.split()
  for i in range(len(sl))#taulukko
    for a in range(len(s))
      if :
      else:
```

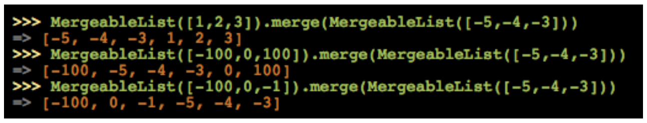
```
[ browse~edits~browse~edits ] ---- 3 min 15 sec

RUN @ 17 min 58 sec [ SyntaxError: invalid syntax ]
```

```
#write your code here
def keywordusage(s,sl):
  s = s.split()
  for i in range(len(sl))#taulukko
    for a in range(len(s))
      if s[a] == sl[i]:
        sl[i] = true
        break
      else:
        sl[i] = false
```

```
#write your code here
def keywordusage(s,sl):
  s = s.split()
  for i in range(len(sl))#taulukko
    for a in range(len(s))
        if :
        else:
```

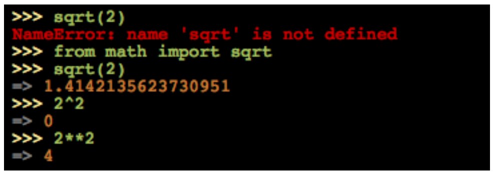
Figure 2: The view in the analysis tool showing a condensed reproduction of a student's programming session. Time progresses downwards and on the left we see the new versions of the code and on the right the previous state of it. In this line-based difference visualization, green lines signify additions (on the left) and red lines are deletions (on the right).



```
>>> MergeableList([1,2,3]).merge(MergeableList([-5,-4,-3]))
=> [-5, -4, -3, 1, 2, 3]
>>> MergeableList([-100,0,100]).merge(MergeableList([-5,-4,-3]))
=> [-100, -5, -4, -3, 0, 100]
>>> MergeableList([-100,0,-1]).merge(MergeableList([-5,-4,-3]))
=> [-100, 0, -1, -5, -4, -3]
```

Figure 3: Example of a student's console session of thorough testing.

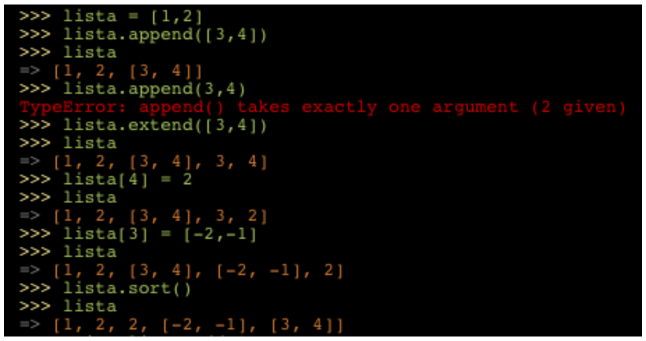
total 15 students (10%) were found having done this kind of exploration.



```
>>> sqrt(2)
NameError: name 'sqrt' is not defined
>>> from math import sqrt
>>> sqrt(2)
=> 1.4142135623730951
>>> 2^2
=> 0
>>> 2**2
=> 4
```

Figure 4: Example of a student console session exploring math.

In assignment 3, it was typical for students to explore functionality of Python lists. An example is shown in Figure 5. A total of 19 students (12%) tested list functionality in the console. Furthermore, many (11) students tried the built-in function `sorted` which returns a sorted clone of a given list.



```
>>> lista = [1,2]
>>> lista.append([3,4])
>>> lista
=> [1, 2, [3, 4]]
>>> lista.append(3,4)
TypeError: append() takes exactly one argument (2 given)
>>> lista.extend([3,4])
>>> lista
=> [1, 2, [3, 4], 3, 4]
>>> lista[4] = 2
>>> lista
=> [1, 2, [3, 4], 3, 2]
>>> lista[3] = [-2,-1]
>>> lista
=> [1, 2, [3, 4], [-2, -1], 2]
>>> lista.sort()
>>> lista
=> [1, 2, 2, [-2, -1], [3, 4]]
```

Figure 5: Example of a student's console session of exploring the functionality of Python lists.

Table 2: Types of errors in students' code in the assignments.

Error	Assign. 1	Assign. 2	Assign. 3
SyntaxError	394 (35%)	195 (17%)	202 (23%)
NameError	274 (24%)	484 (42%)	218 (25%)
IndentationError	173 (15%)	66 (6%)	62 (7%)
TypeError	141 (12%)	252 (22%)	223 (25%)
AttributeError	66 (6%)	135 (12%)	118 (13%)
IndexError	40 (4%)	1 (0%)	25 (3%)
UnboundLocalError	26 (2%)	1 (0%)	15 (2%)
ValueError	21 (2%)	9 (1%)	0 (0%)
ImportError	0 (0%)	15 (1%)	3 (0%)
Other	1 (0%)	7 (1%)	12 (1%)

Execution Errors

The number of errors in students' executions in both running the code in the editor and running commands in the console are summarized in Table 2. As can be seen, the most common error depends on the type of the assignment. In assignment 1, there are much more indentation errors than in the later assignments. We assume this to be explained by students not being familiar with Python. Still, syntax errors along with indentation errors that may be regarded as such too were quite common overall even though Python is often praised for its relatively simple and readable syntax.

In assignment 2, the high number of NameErrors is mostly explained by students trying to call `sqrt` function from `math` package without properly importing it. Furthermore, a lot of the TypeErrors stem from students not knowing how to specify arguments when defining functions of a class. Missing `self` argument in the function definition was a typical mistake, which resulted in TypeError as the number of arguments does not match what is defined. In Python, the `self` argument is implicitly passed when calling an instance function, but needs to be explicitly specified in the method signature.

In assignment 3, the most common TypeError was caused by overwriting the built-in `sorted` function by assigning an object into a variable with the same name, and then trying to use the sorted function in the code. This error was, however, probably mostly our fault, as the example test cases initially used sorted as a variable name.

Students' Perceptions

The web-based programming environment had not been used on a course before. In order to gauge students' attitudes towards this kind of web environment and get some measure of how large a portion of students' development activities the recorded traces include, we issued a short web questionnaire to the students. Students were given some points on the course for answering it. The students were asked whether they had used any other applications in editing and running code when solving the assignments, what was good and what was bad about the environment, and finally whether it should be used on the course in the future. 116 students filled the questionnaire. As 151 students submitted assignment 1, this gives a response rate of around 77 %.

A little less than half of the respondents (44 %) reported having only used the web environment for editing or execution, about a fifth had sometimes used other tools, and about a third reported having edited and run code elsewhere often or almost always. The most commonly reported choices were Eclipse IDE[23] and Notepad++[24] for editing, and command line Python interpreter and Eclipse with PyDev[25] extension for running. Still, relatively few had used any integrated development environments and apparently done without more advanced tools like visual debuggers. A great majority of the students, 79 %, mostly or strongly agreed with the statement that the web-based programming environment should be used in the course assignments in the future (mostly 35 %, strongly 44 %). Overall, students thus seem to have generally felt positively about the system.

On closer inspection of this data, we found no correlation between the use of our tool and students' performance on the course. For the students who answered the questionnaire, we compared their reported use of the in-browser editor and execution environment to their final grade (i.e. exercise + exam + project work in groups of three), sum of points from all exercises, the sum of points from Python exercises divided by the number of submissions per student (i.e. average points), and finally the number of resubmissions needed for the Python multiple choice questions. In the corresponding eight cases, the Spearman's rank correlation was low ($\rho < 0.2$) and not significant ($p > 0.05$).

103 students left feedback in the two free text questions about the pros and cons of the environment. We went through all the feedback and placed the comments into categories of common themes that emerged. 32 students gave general positive comments about the system with opinions ranging from "worked as intended" to "I can't think of anything that was bad about the environment. I think the web-based programming environment was brilliant!". About as many students mentioned the lack of setup as a specific advantage of the environment going as far as saying that "This made Python really look very simple and fun". 25 students commented on the user interface being clear and easy to use. Quite a few students also responded having liked how editing, executing, and submitting code was so conveniently integrated.

The most common complaint dealt with being used to a different programming environment which was mentioned by 28 students. Another common issue had been the editor and the interface in general being too small when integrated to the learning management system used on the course. The system could however still be switched to a fullscreen mode but students found this feature lacking and inconvenient because in the Firefox browser after switching to another window the mode would have to be reactivated again and again (but not in Google Chrome). On the other hand, it would also appear that not all students realized that the divider's position between the editor and console was adjustable. Quite a few students (16) also mentioned about the environment being "a bit sluggish" or "laggy". Indeed, the in-browser Python execution environment is noticeably slower than running a natively compiled Python interpreter directly in your operating system instead of the browser. Furthermore, some students missed having an explicit way of saving their code and mentioned even having been afraid of losing their progress if accidentally closing their browser. This would not have been the case unless browser's web storage had specifically been disabled. However, there was no clear mention of this feature being there. To mention just one of the less frequent issues, a few students did not like the color theme of having a black background.

5. DISCUSSION

Overall, students used automatic feedback rather sparingly and preferred to test their code themselves. This is rather surprising because the previous experience is that an unlimited availability of automatic feedback tends to encourage some trial-and-error behavior using the automatic system as a tester. Most of the testing was still based on code given in the assignment. Still, it is interesting that so many students used the console in their testing. It is difficult to say whether the availability of the console as an integral part of the system had an effect on students' willingness to exercise their code before submitting but this may very well be the case. On the other hand, it could be that just the availability of the test cases in the descriptions of the problems was the sole underlying factor in the students' behavior. Maybe it would be beneficial to provide students with some tests in the first few programming assignments and then omit the tests with the assumption that then they would have got used to the idea and would be more likely to continue testing their code but now with tests they have designed on their own.

The group of students who did not use the console may not have benefited much from our browser-based environment. The answers to the feedback questionnaire also revealed that students used a wide range of editors and environments to write and execute their code. This could be at least partially explained by the course being intended for 3rd year students. Especially computer science students are likely to have their favorite code editor and execution environment selected by the time they take this course. Novice students might benefit from this kind of environment more. Indeed, many of the free text answers in the feedback highlighted the ease of the environment due to not needing to install any tools.

The web-based programming environment had never before been used on a course and it still lacked some features that some students would have liked. Its interface did confuse

[23]http://www.eclipse.org/

[24]http://notepad-plus-plus.org/

[25]http://pydev.org/

some too. For example, we could infer from the traces that there were occasions where students expected the interpreter to function differently. How it did work is that whenever a student would run their code, the Python environment was first cleared and then the new code was evaluated and a message saying "Initializing Python environment" would inform about this too. However, it appears some students expected that if they had, for example, executed an assignment in the console that this definition would stick and they could make use of it across different runs. It could of course work like this but we felt that this would too easily lead to hard-to-interpret errors resulting from some old definitions that stuck in the execution environment. As another example, some students had often copied their code in the editor and then pasted it into the console in order to evaluate it into the Python environment. However, they could have just as well clicked the run-button above the editor since the console shares the same execution environment.

6. CONCLUSIONS

In this paper, we have reviewed and investigated the collection and use of programming session data in supporting educational research about programming. We have presented a web-based Python programming environment that integrates an editor, an execution environment, and an interactive Python console. The environment includes functionality for collecting and analyzing fine-grained data on how learners edit and execute code when solving programming assignments. We used the tool to examine traces of university students solving three tasks assigned as part of coursework. We reported some observations from this data, such as, how students used automatic feedback rather sparingly and made active use of the console for both testing their code and, less frequently, for exploring language features and libraries. Our results also confirm the findings of previous work that a minority of error types account for the majority of error occurrences. As a method to both replicate and extend the many previous studies dealing with more coarse-grained data, the programming environment and the access to and views of the traces can help propel future research into students' difficulties in programming assignments. Ultimately, added understanding of how the traces reflect students' progress and difficulties could enable us to provide automatic feedback and guidance based on students' observed behavior in the environment.

7. REFERENCES

[1] M. Ahmadzadeh, D. Elliman, and C. Higgins. An Analysis of Patterns of Debugging among Novice Computer Science Students. *ACM SIGCSE Bulletin*, 37(3):84–88, 2005.

[2] A. Allevato and S. H. Edwards. Discovering Patterns in Student Activity on Programming Assignments. In *2010 ASEE Southeastern Section Annual Conference and Meeting*, 2010.

[3] E. Balzuweit and J. Spacco. SnapViz: Visualizing Programming Assignment Snapshots. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 350–350, 2013.

[4] S. H. Edwards, J. Snyder, M. A. Pérez-Quiñones, A. Allevato, D. Kim, and B. Tretola. Comparing Effective and Ineffective Behaviors of Student Programmers. In *Proceedings of the fifth international workshop on Computing education research workshop*, pages 3–14, 2009.

[5] N. J. Falkner and K. E. Falkner. A Fast Measure for Identifying At-Risk Students in Computer Science. In *Proceedings of the ninth annual international conference on International computing education research*, pages 55–62, 2012.

[6] J. Fenwick Jr., C. Norris, F. Barry, J. Rountree, C. Spicer, and S. Cheek. Another Look at the Behaviors of Novice Programmers. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, pages 296–300, 2009.

[7] J. Jackson, M. Cobb, and C. Carver. Identifying Top Java Errors for Novice Programmers. In *Proceedings of the 35th Annual Conference on Frontiers in Education*, 2005.

[8] M. Jadud. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education*, 15(1):25–40, 2005.

[9] M. Jadud. Methods and Tools for Exploring Novice Compilation Behaviour. In *Proceedings of the Second International Workshop on Computing Education Research*, pages 73–84, 2006.

[10] M. Jadud and P. Henriksen. Flexible, Reusable Tools for Studying Novice Programmers. In *Proceedings of the fifth international workshop on Computing education research workshop*, pages 37–42, 2009.

[11] U. Kiesmüller, S. Sossalla, T. Brinda, and K. Riedhammer. Online Identification of Learner Problem Solving Strategies Using Pattern Recognition Methods. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 274–278, 2010.

[12] K. Mierle, K. Laven, S. Roweis, and G. Wilson. Mining Student CVS Repositories for Performance Indicators. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[13] C. Murphy, G. Kaiser, K. Loveland, and S. Hasan. Retina: Helping Students and Instructors based on Observed Programming Activities. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, pages 178–182, 2009.

[14] C. Norris, F. Barry, J. B. Fenwick Jr, K. Reid, and J. Rountree. ClockIt: Collecting Quantitative Data on How Beginning Software Developers Really Work. *ACM SIGCSE Bulletin*, 40(3):37–41, 2008.

[15] D. N. Perkins, C. Hancock, R. Hobbs, F. Martin, and R. Simmons. Conditions of Learning in Novice Programmers. *Journal of Educational Computing Research*, 2(1):37–55, 1986.

[16] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein. Modeling How Students Learn to Program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 153–160, 2012.

[17] M. Rodrigo and R. Baker. Coarse-Grained Detection of Student Frustration in an Introductory Programming Course. In *Proceedings of the fifth international workshop on Computing education research workshop*, pages 75–80, 2009.

[18] M. M. T. Rodrigo, R. S. Baker, M. C. Jadud, A. C. M. Amarra, T. Dy, M. B. V. Espejo-Lahoz, S. A. L. Lim, S. A. Pascua, J. O. Sugay, and E. S. Tabanao. Affective and Behavioral Predictors of Novice Programmer Achievement. *ACM SIGCSE Bulletin*, 41(3):156–160, 2009.

[19] J. Spacco, D. Fossati, J. Stamper, and K. Rivers. Towards Improving Programming Habits to Create Better Computer Science Course Outcomes. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 243–248, 2013.

[20] J. Spacco, J. Strecker, D. Hovemeyer, and W. Pugh. Software Repository Mining with Marmoset: An Automated Programming Project Snapshot and Testing System. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[21] J. C. Spohrer and E. Soloway. Novice Mistakes: Are the Folk Wisdoms Correct? *Communications of the ACM*, 29(7):624–632, 1986.

[22] J. G. Spohrer and E. Soloway. Analyzing the high frequency bugs in novice programs. *Empirical Studies of Programmers*, 1986.

[23] E. Tabanao, M. Rodrigo, and M. Jadud. Identifying At-Risk Novice Programmers through the Analysis of Online Protocols. In *Philippine Computing Society Congress 2008*, 2008.

[24] E. Tabanao, M. Rodrigo, and M. Jadud. Predicting At-Risk Novice Java Programmers Through the Analysis of Online Protocols. In *Proceedings of the seventh international workshop on Computing education research*, pages 85–92, 2011.

[25] I. Utting, N. Brown, M. Kölling, D. McCall, and P. Stevens. Web-Scale Data Gathering with BlueJ. In *Proceedings of the ninth annual international conference on International computing education research*, pages 1–4, 2012.

Academic Integrity: Differences between Computing Assessments and Essays

Simon
University of Newcastle, Australia
simon@newcastle.edu.au

Beth Cook
University of Newcastle, Australia
beth.cook@newcastle.edu.au

Judy Sheard
Monash University, Australia
judy.sheard@monash.edu

Angela Carbone
Monash University, Australia
angela.carbone@monash.edu

Chris Johnson
Australian National University
chris.johnson@anu.edu.au

ABSTRACT

There appears to be a reasonably common understanding about plagiarism and collusion in essays and other assessment items written in prose text. However, most assessment items in computing are not based in prose. There are computer programs, databases, spreadsheets, and web designs, to name but a few. It is far from clear that the same sort of consensus about plagiarism and collusion applies when dealing with such assessment items; and indeed it is not clear that computing academics have the same core beliefs about originality of authorship as apply in the world of prose. We have conducted focus groups at three Australian universities to investigate what academics and students in computing think constitute breaches of academic integrity in non-text-based assessment items; how they regard such breaches; and how academics discourage such breaches, detect them, and deal with those that are found. We find a general belief that non-text-based computing assessments differ in this regard from text-based assessments, that the boundaries between acceptable and unacceptable practice are harder to define than they are for text assessments, and that there is a case for applying different standards to these two different types of assessment. We conclude by discussing what we can learn from these findings.

Categories and Subject Descriptors

K3.2 [**Computers and education**]: Computer and Information Science Education – *computer science education*

General Terms

Measurement

Keywords

Academic integrity, computing education, non-text-based assessment

1. INTRODUCTION

Plagiarism and collusion are two major manifestations of academic dishonesty. Plagiarism occurs when a student uses the work of others without appropriate acknowledgement. Collusion is somewhat similar, but is distinguished by the fact that the 'others' are typically the student's own colleagues: collusion is essentially the sharing of work among students, whether the submissions be based on the work of one student or on a collaborative effort. With plagiarism, work purporting to be that of the student or group has too much in common with work that is typically in the public domain. With collusion, work purporting to be that of the student or group has too much in common with the work of other students or groups, often in the same class. Just how much is too much tends to depend on the context [37]. It is also difficult to be clear about what constitutes collusion in circumstances where students are encouraged to work together except when doing assessable work [5].

There appears to be broad agreement [14, 31, 34] that:

- Plagiarism and collusion are not good for the student, because students who plagiarise or collude are failing to practice the academic skill of assimilating the ideas of others and using them as the basis of one's own ideas. Rather, they simply echo the ideas of others with no evidence of assimilation or even of understanding. These practices are also seen as diminishing the students' employment prospects: employers do not want to see their reputations put at risk by what they see as a form of irresponsible behaviour.
- Plagiarism and collusion are not fair on other students: when students choose to work hard for their marks – or, indeed, not to work hard and to accept a lower mark – they feel aggrieved when other students attain good marks for submitting somebody else's work.
- Plagiarism and collusion are not good for the institution, as graduates who have side-stepped the learning process may not perform well in the workplace, reflecting poorly on the institution and on the discipline.
- Plagiarism and collusion are not good for the education system as a whole because they suggest that the system is willing to produce graduates who have succeeded not by independent thought and analysis but by finding the work of others that has some bearing on the subject at hand and presenting it as their own.

The literature of academic integrity leans heavily towards plagiarism of prose text. This focus is sometimes explicit, but more often implicit. Books on avoiding plagiarism [5, 19, 29] focus almost exclusively on text-based plagiarism, with references to such concepts as paper mills, literature, using other people's words, translating foreign articles, and so on. Suggested ways of avoiding plagiarism include learning to paraphrase and learning how to synthesise the words of multiple authors. Referencing guides explain how to place directly copied text into quotation marks and reference it appropriately. All of this is done with little apparent recognition that text is not the only form of work that can be plagiarised.

In relation to prose text, students are generally less likely than academics to recognise certain practices as plagiarism or collusion [4, 12, 33]. Moreover, many students remain confused about definitions of plagiarism and collusion and expectations regarding academic integrity [17, 26].

While there is much work on educating students to avoid plagiarism and produce work that is clearly their own, detection of similarity remains a cornerstone of practice in academic integrity [5, 9, 11, 39]. So long as there are some students who are willing to plagiarise, and so long as this is seen as inappropriate, some academics will apply techniques to help them determine whether the work they are assessing is plagiarised. Within the realm of text-based assessment there are many standard tools to assist in the detection of similarity, tools such as Turnitin and AcademicPlagiarism.

Yet the higher education system includes numerous academic disciplines in which prose text is not the principal medium of assessment. Art students and design students are required to produce images as part of their assessment; mathematics students are required to construct mathematical proofs and derivations; music students are required to write musical compositions; computing students are required to write computer programs and to construct databases and spreadsheets; architecture students are required to produce plans and drawings. These forms of assessment are all dramatically different from prose text. Furthermore, none of these forms of assessment are amenable to the similarity detection of the standard tools such as Turnitin. Turnitin cannot tell whether two computer programs, two databases, two images, or two musical compositions have too much in common to be considered as distinct pieces of work.

Furthermore, while there appears to be broad agreement on the nature and inappropriateness of text plagiarism, academics and students in the non-text-based areas do not necessarily regard the use of others' work in the same light as do academics and students in text-based areas. In a world where re-mixes, mash-ups, re-use of computer code and other combinations of existing work are increasingly accepted and valued as legitimate professional and creative practice, some authors question the value of insisting that the work produced by students must be substantially or entirely original [18, 21].

Further still, the types of academic dishonesty tend to vary from field to field. It is possible that design students lacking inspiration will tend to base their work on an image found somewhere in the public domain, whereas students struggling with computer programming might be likely to borrow and appropriate the work of their more capable colleagues, or to work in inappropriately large groups to produce a joint solution to the problem at hand.

2. BACKGROUND

Some types of assessment items in computing involve text, but of a type that is not amenable to the standard techniques and tools used for similarity detection in prose text. Computer programs, for example, generally have a large text component; but that component is written in a computer programming language, not in a natural language such as English, and the same similarity criteria do not apply. Formal structures such as those found in computer programs are not amenable to detection on the basis of the percentage of textual similarity [13, 14, 18]. Students quickly learn that two computer programs look quite different textually if they have different variable names, different comments, and different spacing, even though they are to all intents and purposes the same program. For this reason, students copying one another's computer programs tend to change the variable names, comments, and spacing, in the hope of evading detection [3, 23, 25]. However, the similarity between the programs is actually evidenced by their logical structure, and such copying tends to be readily detected.

Other items for assessment in computing include databases and spreadsheets. Although these items have some text within them, they cannot in any way be described as text documents, they cannot meaningfully be converted into reader-friendly text documents, and again the points of similarity between two such items are far more likely to be in their structure than in their textual content.

Because copying and collusion are rife among students of computer programming [3, 7, 10, 13, 15, 36, 37, 41], a large number of similarity detectors have been created [1, 3, 8, 23, 30, 40]. Unfortunately, many of these detectors work with programs in just one programming language, and programming is taught in many different programming languages. There are some similarity detectors for multiple programming languages [3, 32], but their adoption appears to be far less wide than that of, say, Turnitin in the realm of prose assessment [5]. In other areas of computing, such as spreadsheets and databases, we have found no similarity detectors. Similarities between submissions are detected by eye, if at all.

There are computing academics who do not check for plagiarism and others who pay no heed to inappropriately similar submissions even when they notice them [7, 10, 18]. Anecdotally, they suggest that as the students are likely to be working collaboratively when they graduate and find employment, it is not inappropriate to do the same when they are studying. Others fiercely seek out similar submissions, insisting that the mark given to an individual should be for work carried out by the individual. This range of diverse academic opinions and behaviours has not been explored in any systematic way. Computing degree programs are almost all professionally accredited, and relate their development of student behaviour to the computing industry; but the expectations of good professional practice in acknowledging the work of others are not reported, and the codes of ethics and professional practice in computing do not cover this.

We aim to find answers to a number of specific questions:

- How do academics and students perceive academic integrity in regard to non-text-based computing assessment items?
- Are there assessments for which academics and/or students think that every answer is unique, so copying is acceptable so long as one personalises the copy?

- Are there areas in which academics and/or students think that there is only one correct answer, so copying cannot be detected?
- What steps do academics and students take to ensure that academic standards are adhered to?
- What do academics do to detect similarities that might suggest academic misconduct?
- How serious is academic misconduct considered to be and how is it dealt with?
- To what extent do computing academics believe that university policies for academic integrity are adequate for non-text-based assessments?

3. RESEARCH APPROACH

We conducted focus groups of computing students and academics at three Australian universities in late 2012 and early 2013. Three focus groups were made up of 12 students studying information technology, business systems, commerce and educational technology. Three focus groups of academics comprised 18 teaching staff. Participation in the focus groups was voluntary, with participants responding to posters or email advertising the research or to direct approaches by researchers. There is no assurance that the perceptions of participants are representative of other students and academics at these or other institutions. The staff and student focus groups were held separately, and no teachers were present at the student focus groups, as their presence might have constrained the students' responses.

The purpose of the focus groups was to inform the design of a broad survey to be conducted subsequently. The focus groups were based on an indicative set of questions, with the facilitators encouraged to ask exploratory follow-up questions when the discussion suggested so doing.

The focus groups were recorded and transcribed, and the transcripts were corrected and shown to the participants in case they might feel there were any egregious errors in the transcriptions.

Researchers used a directed or deductive content analysis methodology, using the extant literature to identify key concepts that were used to develop initial coding categories [20, 28]. Further categories were developed from the data using an inductive process. This approach combines the benefits of 1) using insights from the literature to validate or challenge previous findings; and 2) maintaining the flexibility to incorporate new insights directly from the data.

The initial analysis included listening to the tapes and reading the transcripts. The transcripts were then coded using the initial categories. Data outside these categories were identified and subsequently examined to determine whether they suggested new categories. Finally, the categories were reduced to themes, and the responses listed under each theme were analysed for concordance or conflicting views.

4. FINDINGS

The analysis identified five major themes: the level of understanding of academic integrity issues; perceptions of the importance of academic integrity; the complexity of computing compared with text-based situations; the processes of detecting and dealing with breaches; and, at an emotional level, the impact on relationships within institutions. This section explores these themes in relation to the research questions.

Based on the premise that most universities strive to inform their students about the requirements of academic integrity, but that the information provided is highly skewed towards the written word, the focus groups began with a discussion on academic integrity in essays and similar assessments.

4.1 Perceptions of academic integrity in regard to essays

Discussion around plagiarism and collusion relating to essays revealed considerable variation in the level of understanding of participants. Staff and students agreed that taking other people's words or ideas without acknowledgement constituted plagiarism. This includes 'copy and paste' and paraphrasing without referencing. However, a number of students thought that they needed to reference only when using a direct quote, including one who stated:

> 'If you're adding your own ideas in it I think it makes it yours because you're not directly like using their ideas. You're writing it differently.'

Plagiarism was viewed as a serious matter. Although the students might not share the accepted understanding of what plagiarism is, they know that it's not a good practice.

> 'It defeats the purpose of being at university.'
>
> 'Well it's not proving your own ability so it is extremely important.'

Among the students, few were familiar with the concept of collusion. Once it was defined for them, they had difficulty imagining what collusion might mean in the context of an essay-type assessment. One group concluded that it was not really possible to collude in writing an essay unless the students used exactly the same words. They felt that it was acceptable to show an essay to a friend and get them to correct any mistakes, including correcting the ideas and pointing out that the student had misinterpreted the question.

> 'Cause even if they do help, like they tell you that your thing is wrong and you suddenly have a change of heart and you agree with them. When you rewrite it it's in your own words. It's not in their words. So it's not copying.'

4.2 Perceptions of academic integrity in regard to computing assessment items

Perceptions about plagiarism and collusion in computing were shaped by opinions about seriousness, levels of understanding, the additional complexity (including requirements that varied by assessment), and fuzzy boundaries.

While computing assessments are many and varied, including databases, spreadsheets, design diagrams, and more, most of the discussion in the focus groups clearly revolved around computer programming assessments.

Students and academics stated that sources need to be acknowledged.

> 'Someone else's code that you've got to acknowledge where it's come from, and both document that in the code and acknowledge it in any formal, prose-form documentation that they produce.' (Student)

'...if there is some or all of their code that is not authored by them it has to be acknowledged.' (Academic)

The focus groups revealed that both plagiarism and collusion are issues, and that in the area of computing, collusion may be more of an issue than plagiarism, a view that supports the findings of Culwin et al [10]. Students spoke extensively about assisting friends with code if they were stuck, pointing them in the right direction, and comparable practices. They also discussed using libraries and message boards as a normal practice in computing.

Plagiarism in computing

As with essays, there was general agreement that plagiarism is copying or using someone else's ideas or work without acknowledging it. Some participants made no clear distinction between plagiarism and collusion: for example, some specific actions mentioned as forms of plagiarism were paying someone else to write code and taking someone else's USB device and copying their work from it.

However there were some issues that participants thought led to differences between plagiarism in essays and plagiarism in computing. First, there was the tradition of learning from the community, in which programmers adapt and learn from the code of other programmers. Second, students felt uncertain about whether they are permitted to reuse code they have previously developed for another purpose. There are certainly some programming courses in which code reuse is accepted as good practice [13, 22]. Third, there was the difference that while a reference in an essay is obvious when reading the essay, there is no way to make a reference in a program obvious when running the program, and reference guides provide scant information on how to reference code [13]. Of course the reference can be included in the code, but that will seldom be seen by users of the program. One student commented:

'…there's no formal way or clear way to do that. We've talked about writing comments in code to say where these come from but that's certainly not a standard way of doing that.'

Differences such as this led to a general feeling that it is much more difficult to establish the boundaries than it is with text-based assessments.

The fact that most university policies and plagiarism modules are largely silent on non-text-based assessments such as computer code has contributed to a situation where students are unclear about what practices constitute plagiarism [22]. Students who participated in the focus groups stated that they had received little specific education or guidance in relation to computer code. Therefore, they relied on what some referred to as their own ethics for guidance:

'We're not saying what we do is right, we're saying that's how we do it. We run it based on our ethics'

'We're just going on ethics and they haven't expressly told us how much of our code we can't reuse from someone else or where we can't get it from'

Possible consequences of such approaches were evident in one of the student focus groups where there was a lengthy discussion about individual students' abilities to determine intuitively whether practices constituted plagiarism or collusion.

'And in this it always comes back to ethics. If I feel like I've done something that's my work and … I don't feel like I've plagiarised it, then the chance is I haven't and even if I have then it's probably, like I could probably prove that I haven't, sort of thing. Like because of the amount of steps that you go through to do work and like no program has like one version of a program.'

Among students there was a feeling that it was acceptable to use other people's ideas but not to copy code.

'If you start copying their code, how they got their idea to work, then I think you've crossed the boundary.'

'It's still for me fuzzy 'cause there's an emphasis on peer learning as well. It's hard to know where help stops and plagiarism starts, I think.'

However, opinions varied significantly between individual students. Some felt that they could only be said to have copied code if they copied whole programs or whole websites, and there was a great deal of discussion about the acceptability of taking smaller pieces of code, and about how much of one's own work needed to be added to a piece of code in order to consider it one's own.

'…I just Google and I implement and I feel like because I had to implement it and stuff like that and make it work within my code and add things to it, it becomes mine.'

'Plagiarising code: it's just copying the exact code and pasting it into your own work.'

Confirming previous research [7, 13, 15, 22], the focus groups revealed that the academics' perceptions of plagiarism were generally somewhat stricter and more consistent than the students' perceptions For example, one academic described as a myth the widely held view of students that it was not plagiarism if they changed the code by 10 or 20 percent.

Notwithstanding this, the academics conceded that the boundaries remained blurred.

'…there's a difference between using something as a reference and actually copying something outright… I think it's just very hard to define.'

While previous research has established that students are likely to have a more liberal view than academics of the acceptability of certain practices [13, 15, 36], at least one study found some scenarios that students identified as unacceptable whereas teachers agreed they were acceptable [38]. In a similar vein, one academic in our focus groups was concerned that students are sometimes mistaken in their belief that something is plagiarism.

'I've actually had the reverse experience of plagiarism, of trying to get students to understand that it is actually quite legitimate to use existing code and further develop that, rather than having to start from scratch… there is always some of this, that I can't use someone else's work because that would be plagiarism.'

Collusion in computing

The focus groups revealed that there are different perceptions of what constitutes collusion between the two groups and also within the groups. This was further complicated by the fact that what constitutes collusion varies from one assessment to another, based on the assessment specifications.

The issue for academics revolved around the educational objectives of the particular assignment, although it was generally agreed that students were probably not entirely clear on this point and that academics could do a better job of explaining why they wanted students to complete the work individually.

While academics often permitted some level of collaboration, there was considerable variation in what was acceptable. Some academics said that students could discuss the assignment but could not collaborate on the development of code.

> 'I encourage students to collaborate and brainstorm right down to the point where they've discussed details about the assignment but they're writing their own code…if it's about helping a friend to work out how methods are used, then do it without the context of the assignment.'

One academic indicated that he would be prepared to allow students to work together on an individual assessment as long as they declared it.

> 'I was trying to think of a model where we didn't make it illegal for them to work together, because as far as I'm concerned if four of them work together and they turn in a good assignment, well, they've learnt something. It doesn't worry me.'

However, without further safeguards such as interviewing students and ensuring that they know how the code works and can explain how it was developed, this approach would not necessarily ensure that all students had contributed to the assignment and met the learning objectives.

Students generally had a more liberal view of the level of support that was permissible. In some instances this included obtaining assistance with coding from fellow students and posting code on message boards to seek assistance. These methods were seen as legitimate so long as the answers pointed them in the right direction rather than supplying the code.

> '…no one ever says "here's the code to make it work". They always say "Have a look at this, it's because IE does this or Chrome does this to display it".'

Some students adopted a more nuanced stance, distinguishing between the legitimacy of these two approaches.

> 'No, the difference is if you post your question online it will be their solution. And you're copying their solution. But when your friend is checking, he'll tell what's wrong and you have to change it yourself, so that's different.'

A small minority of students thought that collusion occurred only in extreme cases such as when two assignments were exactly the same. One student stated that in situations where students could not get code to work they had no alternative other than to get help from somewhere:

> 'Like if you're stuck on like code or something, and like you just don't know how to fix it, if you don't seek help you can't finish the assignment. You're going to have to get help.'

Student misconceptions about what constitutes plagiarism and collusion, combined with the difficulty that both academics and students mentioned in determining the boundaries of acceptable and unacceptable practices, point to the need to clearly define acceptable academic practice and educate students.

4.3 Is unreferenced copying legitimate so long as one personalises the copy?

In general, students were more likely to agree with this proposition than academics, although both groups acknowledged that the boundaries are blurred. Both academics and students expressed the view that it was permissible to copy commonplace or trivial pieces of code without referencing. Similarly, students sometimes felt that it was not necessary to reference some code if they made considerable changes to it.

> '…like more complex but like still generic… Even stuff like that I just Google and I implement and I don't like, I feel like because I had to implement it and stuff like that and make it work within my code and add things to it, it becomes mine. Whereas if it was like the variables were set and I just copied a chunk, pasted a chunk and then left it, it would be closer to plagiarism.'

To tease out where the boundaries are, students were presented with a scenario where a student copied methods from the web without referencing them. Students generally felt that this was an acceptable practice, and one student justified it in these terms.

> 'You have to change the code and make it fit in the context of your assignment so I wouldn't think it would be plagiarism.'

4.4 Is there sometimes only one correct answer, so copying cannot be detected?

Academics agreed that students do think they can get away with copying, particularly in relation to computer code. However, they had differing views on why students had this perception. The first position was that students think there is only one correct solution so all the assignments will be identical

> '…the ones who do the totally copying seem to assume that "there is only one right answer so nobody will know I've copied".'

An opposing view was that students think teachers simply run the code to see if it works, and are not aware that their work would be compared to that of other students.

> 'Well I wondered if they don't realise some of them that we look at their code. I think maybe they think we only run it. Because the cheating is so bad when we catch it, you know I'm insulted that they think I'm so stupid.'

A third view was that there was a combination of these two reasons.

> 'They'll at least know from the unit tiff, they think all we do is run the unit tiff and we never look at their code and they also think… a lot of them think that if we do happen to look at their code, well everybody is going to have the same code anyway.'

One group of computing students stated that on the rare occasions where the question would lead to everyone having similar or identical code, teachers would ensure that the task was completed in class under supervision so there was no real opportunity for students to collude or copy one another's code. They contrasted this with individualised assessments where it was unlikely that they would have similar code, so high levels of code similarity would alert academics to the possibility of copying.

‘…with most of the open assessments like with make it up, there’s infinite answers. So if they were so specific, and because there’s infinite answers, if they were within a range, like, this range, you can tell.’

However the ability of teachers to pick up similarity in such situations was seen to be severely constrained if there were large classes.

‘…if there’s a large class of say 100 students doing one assessment, just an idea, how would they remember after marking 100 assessments, which ones are similar? And you can’t just put it in the computer.’

4.5 What do academics and students do to ensure standards are adhered to?

Academics outlined numerous strategies they employ to ensure that students adhere to academic integrity standards. The major strategies involve education, monitoring work in class, and viewing work in progress, and are well aligned with recognised practice as summarised, for example, by Carroll [5].

In addition to the general information given to students by the university, such as academic integrity modules, some academics outlined methods they use to ensure that students understand what plagiarism and collusion are and the parameters of acceptable practice for the degree, the course, and specific assignments. Frequently, supplementary written information is provided in course outlines and/or specifications for each assessment. Some academics also provide specific courses, lectures or tutorials on academic integrity.

‘I present my lecture in the first or second week. I make it very explicit about what they cannot do. Then if a student gets caught they know they’ve been caught.’

A common practice is the provision of specifications with each assignment so that students understand what is permissible for the particular assessment task in line with the learning objectives.

‘a very specific indication in the assignment specification as to the expectation… that it is their own work, and that anything that’s not their own work is clearly referenced as such.’

Academics also employ a number of other strategies to reduce opportunities for breaching academic integrity, strategies such as designing individualised assignments, viewing work in progress, and awarding marks for work in progress to encourage students to apply sufficient effort in this area.

‘…a week after I give out an assignment they have to hand in some pseudocode.’

Similarly, monitoring progress in class made it easier for staff to keep track of individual students and reduce the likelihood of cheating.

‘If you actually want students to generate code you don’t send them away to do it, you have them sit down and generate code in front of you.’

Some academics permitted collaboration within the specification of assessment tasks, with safeguards to ensure that learning objectives are met. One approach permitted students to collaborate and hand in one assignment, and then each student was interviewed to ensure that they understood the work. Another academic outlined a similar approach:

‘The way I do it is by producing assignments, the collaborative parts were small portions, so that even if they do collaborate then I have these lab tests or quizzes or other things and an exam where I can check that they’ve actually learnt what they are meant to learn.’

Students expressed the view that they received little appropriate instruction, either from academic integrity modules or similar tasks to be completed by all students, or from special lectures, tutorials, or assessment guidelines from computing academics. This perception was in stark contrast to the extensive discussion by academics detailing all their attempts to help students understand and adhere to academic integrity standards.

The discussion of the steps students themselves take to maintain academic integrity standards was succinct and centred upon referencing when they copied code and not sharing code with other students. Students also acknowledged that their opportunities to plagiarise or collude were reduced by the practices adopted by academics, as outlined above, such as being given unique assignment tasks; submitting work in progress as well as the final assessment item; working on assessments in class; being required to demonstrate how they developed their work; and attending interviews to explain how the code worked.

4.6 How do academics detect similarities that might suggest academic misconduct?

In line with much of the academic integrity literature [2, 6, 27, 29], the consensus among academics was that prevention is more important than detection, although it is necessary to take action when breaches occur.

‘If it is difficult to detect plagiarism in these non-text things then the solution is not to find better ways of detecting it, but to avoid the problem.’

Academics indicated that it was more time-consuming and more difficult to detect plagiarism with non-text-based items such as computer code than was the case for text. While some academics said it was easy to detect copied code, others said it could be detected only ‘with difficulty’. It was mentioned more than once that detection was easier for small student cohorts where one person is more likely to mark all of the assignments.

There are a number of strategies that can be used to detect breaches of academic integrity. In the first instance academics rely heavily on knowledge of the abilities of their individual students. Breaches were frequently identified when students submitted work that was better than expected.

‘If I get an assignment that’s better than I would expect from that student, then I would be looking at it closely.’

A second method involved the use of technological solutions, typically involving detection tools such as Google, Google Image, Plaggie, JPlag, MOSS, and others.

Some academics indicated that they used techniques such as looking for white space in the program that should not be there. Similarly, variable names that did not seem appropriate could be a pointer to copied code in which students had simply changed the names of variables in an attempt to avoid detection.

‘I still use the detect tool because it looks at the words and comments as well, it looks at identifiers and it doesn’t attempt to look at program structure. And it sticks out like a signature, particularly if they’ve taken stuff on the web and

> they haven't understood it and they haven't changed the identifiers…it is a fairly subtle signature sort of tool.'

However, detection tools are less effective with computer code than with text due to the more constrained nature of computer code, and the consequent increased probability of matching segments in truly independent programs. Compared to the situation with text, using the percentage of similarity to detect plagiarism is not as straightforward in the case of code for many reasons, including the formal structure of the language [7, 8, 25] and the fact that students have similar levels of experience and are using the same textbook [25].

One student participant in a focus group recounted a situation where researchers asked students from different classes to submit their assignments and then analysed the code for similarities. There was a 90 per cent match between this particular student's code and that of another student even though the students had never met or associated in any way. While anecdotal, this raises the possibility of code detection tools suggesting that students have copied code even when they have worked independently.

In some instances staff indicated that other students provided information on suspected breaches by their fellow students.

> '…we actually got emails from some students saying that other students were asking questions on some sort of cheat sites, and they were literally the questions that were being put to them on the assignment.'

Testing a student's knowledge of code (as mentioned in the previous section) is also a potential detection method as well as a reasonably effective deterrent.

4.7 How serious is academic misconduct and how is it dealt with?

Focus groups were asked how serious an issue plagiarism and collusion are. Due to some confusion over the wording, some focus groups interpreted this in a normative sense and others responded in relation to prevalence. Breaches of academic integrity were seen as a serious issue since 1) they involve the unethical practice of taking credit for work that was not completed by the individual student; 2) they have the potential to devalue qualifications from particular universities, or universities in general, if graduates enter the workforce without the necessary skills; and 3) they will eventually have negative consequences for the individuals involved since they have not developed the skills they need. Student comments included:

> '…so people will think [name of university] standards aren't very high and … it devalues the value of our degree'

> '…people will find out you're not that good anyway.'

Academics' assessments of prevalence varied and were very general. However, breaches of academic integrity were generally thought to greatly exceed the number of detected cases.

> 'I always have a guilty feeling that it's more prevalent that I'm aware of and so I feel like that I'm being a bit naïve and dumb. It's the tip of the iceberg here.'

In relation to the issue of how breaches of academic integrity are dealt with there was a broad difference of knowledge between students and academics. Students appeared to be genuinely concerned about the prospect of committing an inadvertent breach of academic integrity and were aware that the consequences could include expulsion from the university. One student stated that the reason for not cheating was fear of being caught and punished. However, students were not generally aware of situations where breaches had been detected or of the consequences for the students involved. While this could be because few instances of breaches were being detected, some students surmised that it was likely to be due to privacy protections at the university.

On the other hand, academics recounted specific instances of detecting breaches and how they were dealt with at their universities. Some academics indicated that students were first spoken to by the lecturer as part of the investigative process. If students admitted to cheating the consequences might include a warning or a zero mark for the assignment and/or a note in their personal file. The case was escalated if the student denied the breach and was unable to provide a satisfactory explanation.

> '…it's only the ones where they're in dispute that tend to escalate to higher authority.'

In other universities there was no flexibility since university policies stipulated that academics must immediately report any suspicions to a nominated academic conduct officer.

The prevailing view of academics was that breaches are difficult to prove and that pursuing breaches was time-consuming and resource-intensive. Some stated that they received inadequate support from their university, both in terms of resources and because the university was sometimes too lenient with students.

> '…you're not supported higher up in the system and that happens a lot at our institution.'

4.8 Is the university policy adequate for non-text-based assessment items?

Academics were unanimous in the view that university policies related predominantly to text-based situations and there is not enough information on what is expected in non-text-based situations. Academics identified the need for further development to incorporate non-text assessments.

> '… I think when it comes to interpreting the policy with regard to non-text it doesn't really stack up… our students … are required to do an academic integrity module … and it really doesn't address at all things like images and computer programs and databases, and diagrams, and mathematical proofs, and all the rest of it.'

However, some academics also noted that if there was a rigid policy developed for non-text-based assessments it could leave them without the flexibility they now enjoy.

Due to the inadequacy of whole-of-institution approaches that concentrate on text, some other initiatives have been developed for non-text-based assessments and practice as mentioned previously: special lectures / tutorials for students to make sure they are aware of how academic integrity relates to non-text-based assessments; and specifications for each assignment outlining what can and cannot be done.

Academics stressed that there is a need for consistency in policies pertaining to non-text-based assessment items, but this is complicated by the fact that there are different requirements related to educational objectives between faculties, degrees, courses, and even within courses. Therefore, it is necessary to have very clear guidelines at all levels and to ensure that students understand what is required of them and why – what the learning objectives are that these requirements contribute to.

> '…how do we inform students that this difference in approach is expected? That in one course we've got "you are expected to implement your own, bar very simple, almost year one things". But in another you are expected to make optimum reuse of existing things?'

Both academics and students expressed the view that there is a case for applying different standards for non-text-based computing assessments. One academic commented:

> 'I think that policy should give just general guidelines, it depends on the discipline. Like Computer Science and the Law school in terms of plagiarism are different.'

Similarly, students pointed to what they considered to be significant differences for non-text situations.

> '…when you're just getting chunks of code, you're not really copying, you're just getting concepts behind it and interpreting it to yourself.'

Moreover, some students felt that guidelines would need to vary by the type of non-text-based assessment involved, which would result in different guidelines for programming, web design, and images, for example. They also thought that establishing rules would be problematic due to the rapidly changing nature of computing.

> '…that's why it's just going to be so grey and it's got to be a process. It can't be a rule book.'

Another issue that emerged with both academics and students was the tension between academic expectations and the standards that apply in the workforce. One student noted that, in contrast to the academic situation,

> '…in the real world in IT stuff there's like exceptions and there's different rulings based on different things which makes them a bit more complex.'

One student expressed the more extreme view that there was nothing wrong with collusion because it was the normal way of working in professional world.

> 'I don't understand collusion myself because like basically collusion is like working together, right, and 99% of the world is made up of people working together to get things done. So in my eyes I'm not really for rules against.'

In the focus groups with academics there was a difference of opinion as to whether there should be two distinct standards (academic and commercial) or whether a common standard should be developed. Academics also debated whether students should be educated to understand commercial standards while at university in order to prepare them for the workforce. Comments from academics included:

> '…to respect real practice in our discipline and to make education match what people are going to find in the workplace later, which has to do with reuse, teamwork, and so on.'

> '…in their professional life….it is perfectly legitimate for them to go to Google and look on the web and look at other companies that have solved similar problems.'

5. OTHER ISSUES ARISING

Other issues that emerged from the data related predominantly to deficits in students' understanding of the importance of academic integrity. Academics discussed a number of issues that were almost entirely neglected in the student focus groups. First, academics stressed that one prominent reason for developing and implementing standards was to ensure that the learning objectives of the course were met. If students were conscious of the learning objectives of assignments they would be more likely to be engaged and honour the assessment specifications.

> 'So that they can engage with the kind of educational objectives, rather than misunderstanding them and then behaving in perverse ways because they think we're assessing them on basic learning stuff.'

While the majority view was that communicating the educational objectives of assessments could be improved, there was a perception that practices varied considerably between universities, and one academic thought that current practices at their university were adequate.

> 'At my university we have a legal obligation to do so. We have to. We have an academic integrity statement that we have to make at the beginning of the course, and before every assignment; and then on top of that we like to – it's not legally required, but we like to articulate the learning goals of the assignment as well, to make it clear for the students.'

A second area that academics identified was facilitating a greater understanding of why students should cite sources. This included principles of scholarship:

> '… the provenance of your work, why is this a reliable thing to use, what's the integrity of this, so there's traceability…

In addition, a number of academics mentioned that from a pragmatic point of view students should be made aware of the fact that they would obtain higher marks by citing their sources.

The third issue that academics raised related to cultural differences between international and domestic students. Cultural differences have been identified in studies of computing students' perceptions of plagiarism [22]. Some instances related by academics in the focus groups included international students working in groups despite being instructed to work alone or copying solutions when they were expected to develop their own solutions. In another instance, Chinese students justified copying solutions from a lecturer by saying:

> '… what are we supposed to do, because in China, in Asian culture … when there is a mentor, what are we taught, I should follow that. If the teacher says do it this way we should just follow the teacher.'

This suggests that there is a need to ensure that international students are educated to understand the requirements of studying in an Australian university.

Finally, the academic focus groups postulated reasons why students breach academic integrity guidelines. Major reasons included: looking for shortcuts; not being able to see the relevance of the work; high expectations of teachers; time pressures; and the myth that it is not plagiarism if they change it by 10 or 20 per cent. Previous studies involving computing students have identified similar reasons for cheating [10, 13, 36, 37, 41].

6. CONCLUSION AND FURTHER WORK

The focus groups revealed that plagiarism and collusion are viewed as serious issues by academics and students, but many students lack a holistic appreciation of the importance of academic integrity. Perhaps as a consequence, some students revealed a fear of inadvertently breaching the academic integrity rules. These issues also impact on the relationship between students and the academics who police academic integrity.

The overwhelming view of the academics and students who participated in the focus groups was that there is a substantial difference between computing assessments and text-based assessments, and that the boundaries of acceptable and unacceptable practices are much more difficult to define for computing assessments than for essays.

The focus groups confirmed that academic integrity policies in these institutions remain heavily prose-based and guidelines for non-text-based assessments are underdeveloped. Both the academics and the students supported the view that there is a case for different standards of academic integrity for non-text-based assessments, but also stressed the need for consistency in policies pertaining to such assessment items.

Academics understood the need for clarity and consistency in guidelines. However, policy development is complicated by differences in perceptions of academic integrity, both between academics and students and within each group, as well as different requirements related to educational objectives between faculties, degrees, courses, and even within courses. Furthermore, if specific non-text-based policies were devised, they would need to be carefully incorporated into the existing policies, which relate mainly to prose-based items.

Initiatives that academics have developed to fill the current void and inform students of requirements for non-text-based assessments include delivering special lectures or tutorials on academic integrity as it relates to these assessments, and issuing detailed academic integrity guidelines in the specifications for each assignment. Despite these initiatives, student awareness remains low, echoing findings of knowledge or perception asymmetries from other research in computing [13, 15, 35]. The chasm between academics' attempts to inform and prevailing student understandings warrants further research to determine why these attempts to inform students are currently ineffective, and to develop improved strategies.

Previous research has emphasised the need for a holistic approach to academic integrity that incorporates education, clearly defined policies which are adhered to, and attempts to inculcate antipathy to breaches of academic integrity [14, 15, 16, 34]. Joyce et al [24: 195] state:

> 'we must ensure that we give equal emphasis to the necessary ingredients: assessment design, education of staff and students, detection tools, academic integrity policies, and disciplinary processes.'

Greening et al [16] advocate integrating ethics education into computing courses rather than teaching it in stand-alone units. Innovations such as these would allow a discussion of ethics in relation to the real situations likely to be faced by students in their academic and professional lives.

A possible direction for future research that would contribute to this holistic model involves developing communication strategies that enunciate why students are expected to conform to academic integrity requirements. An emphasis on the learning objectives of assessment tasks might engender a greater commitment from students to engage with and abide by the policy.

In relation to breaches of academic integrity, the majority of participants agreed that collusion was more prevalent than plagiarism amongst computing students.

Academics use a number of methods to detect plagiarism and collusion, including similarity detection software and inspecting code for similarities.

Policies for dealing with breaches varied between institutions. Some academics have a level of discretion when suspected breaches are detected, while others are required to escalate breaches immediately.

While these focus groups involved staff and students at three universities, we are now conducting a survey across a far larger number of institutions within Australia. This survey will ensure a more representative perspective, and will give us a feel for the prevalence of the beliefs that we have identified.

At the heart of this work is the question of whether the same academic integrity guidelines should apply to non-text-based computing assessment items as to essays. The focus groups suggest that there might be a case for different standards, and we shall look with interest to see whether the survey confirms this belief.

If the feeling is that the same guidelines should apply, but that this is not currently happening, we would hope to develop tools to help academics and students in computing understand and apply the academic integrity guidelines. On the other hand, if the feeling is that the guidelines were developed for essays and similar assessment items and are not appropriate for non-text-based assessment items, we would aim to develop more appropriate academic integrity guidelines and to argue for their adoption.

7. REFERENCES

[1] Ahtiainen, A, S Surakka, and M Rahikainen (2006). Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. Sixth Baltic Sea conference on Computing Education Research (Koli Calling 2006), Koli, Finland 141-142.

[2] Alam, LS (2004). Is plagiarism more prevalent in some forms of assessment than others? 21st ASCILITE Conference, 48-57.

[3] Arwin, C and SMM Tahaghoghi (2006). Plagiarism detection across programming languages. 29th Australasian Computer Science Conference, 277-286).

[4] Brimble, M and P Stevenson-Clarke (2005). Perceptions of the prevalence and seriousness of academic dishonesty in Australian universities. The Australian Educational Researcher 32(3): 19-44.

[5] Carroll, J (2002). A Handbook for Deterring Plagiarism in Higher Education. Oxford Centre for Staff and Learning Development, Oxford, UK.

[6] Christe, B (2003). Designing online courses to discourage dishonesty. Educause Quarterly 2003(4): 54-58.

[7] Chuda, D, P Navrat, B Kovacova and P Humay (2012) The issue of (software) plagiarism: a student view. IEEE Transactions on Education 55(1): 22-28.

[8] Cosma, G and M Joy (2012). An approach to source-code plagiarism detection and investigation using latent semantic analysis. IEEE Transactions on Computers, 61(3): 379-394.

[9] Crisp, GT (2007). Staff attitudes to dealing with plagiarism issues: perspectives from one Australian university. International Journal for Educational Integrity 3(1): 3-15.

[10] Culwin, F, A MacLeod, and T Lancaster (2001). Source Code Plagiarism in UK HE Computing Schools: Issues, Attitudes and Tools. Joint Information Systems Committee.

[11] Curtis, GJ and R Popal (2011). An examination of factors related to plagiarism and a five-year follow-up of plagiarism at an Australian university. International Journal for Educational Integrity 7(1): 30-42.

[12] de Lambert, K, N Ellen, and L Taylor (2006). Chalkface challenges: a study of academic dishonesty amongst students in New Zealand tertiary institutions. Assessment & Evaluation in Higher Education 31(5): 485-503.

[13] Dennis, L (2004). Student attitudes to plagiarism and collusion within computer science. International Plagiarism Conference 2004, viewed 28 March 2011, available at: http://www.plagiarismadvice.org/research-papers.

[14] Dick, M, J Sheard, C Bareiss, J Carter, D Joyce, T Harding, and C Laxer (2003). Addressing student cheating: definitions and solutions. SIGCSE Bulletin 35(2): 172-184.

[15] Dick, M., J Sheard, and S Markham (2001). Is it okay to cheat? The views of postgraduate students. ACM SIGCSE Bulletin 33(3): 61-64.

[16] Greening, T, J Kay, and B Kummerfeld (2004). Integrating ethical content into computing curricula. Sixth Australasian Conference on Computing Education, 91-99.

[17] Gullifer, J and GA Tyson (2010). Exploring university students' perception of plagiarism: a focus group study. Studies in Higher Education 35(4): 463-481.

[18] Hamilton, M, SMM Tahaghoghi, and C Walker (2004). Educating students about plagiarism avoidance - A computer science perspective. International Conference on Computers in Education, 1275-1284.

[19] Harris, R (2001). The Plagiarism Handbook. Pyrczak Publishing, Los Angeles, USA.

[20] Hsieh, H-F and SE Shannon (2005). Three Approaches to Qualitative Content Analysis. Qualitative Health Research 15(9): 1277-1288.

[21] Johnson-Eilola, J, and SA Selber (2007). Plagiarism, originality, assemblage. Computers and Composition 24(4): 375-403.

[22] Joy, M, G Cosma, J Y-K Yau, and J Sinclair (2011). Source code plagiarism – a student perspective. IEEE Transactions on Education 54(1) 125-132.

[23] Joy, M, and M Luck (1999). Plagiarism in programming assignments. IEEE Transactions on Education 42(2): 129-133.

[24] Joyce, D (2007). Academic integrity and plagiarism: Australasian perspectives. Computer Science Education 17(3), 187-200.

[25] Mann, S, and Z Frew (2006). Similarity and originality in code: plagiarism and normal variation in student assignments. Eighth Australasian Computing Education Conference, 143-150.

[26] McCabe, D L, KD Butterfield, and LK Treviño (2006). Academic dishonesty in graduate business programs: prevalence, causes, and proposed action. Academy of Management Learning & Education 5(3): 294–305.

[27] McCabe, DL, LK Treviño, and KD Butterfield (2001). Cheating in academic institutions: a decade of research. Ethics & Behavior 11(3): 219-232.

[28] Moretti, F, L van Vliet, J Bensing, G Deledda, M Mazzi, M Rimondini, C Zinnermann and I Fletcher (2011). A standardized approach to qualitative content analysis of focus group discussions from different countries. Patient Education and Counselling 82: 420-428.

[29] Neville, C (2010). The Complete Guide to Referencing and Avoiding Plagiarism. Open University Press, Maidenhead, UK

[30] Ohno, A, and H Murao (2008). A quantification of students coding style utilizing HMM-based coding models for in-class source code plagiarism detection. Third International Conference on Innovative Computing, Information and Control, ICICIC '08, 553-556. doi: 10.1109/ICICIC.2008.614.

[31] Popyack, JL, N Herrmann, P Zoski, B Char, C Cera, and RN Lass (2003). Academic dishonesty in a high-tech environment. SIGCSE 2003, 357-358.

[32] Prechelt, L, G Malpohl, and M Philippsen (2002). Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science 8(11): 1016-1038.

[33] Razera, D, H Verhagen, TC Pargman, and R Ramberg (2010). Plagiarism awareness, perception and attitudes among students and teachers in Swedish higher education – a case study. Fourth International Plagiarism Conference, Newcastle upon Tyne, UK.

[34] Riedesel, CP, AL Clear, GW Cross, JM Hughes, Simon, and HM Walker (2012). Academic integrity policies in a computing education context. ITiCSE Working Groups 2012.

[35] Sheard, J and M Dick (2011). Computing student practices of cheating and plagiarism: a decade of change. ITiCSE'11, Darmstadt, Germany, 233-237.

[36] Sheard, J and M Dick (2012). Directions and dimensions in managing cheating and plagiarism of IT Students. Fourteenth Australasian Computing Education Conference (ACE2012), Melbourne, Australia, 177-185.

[37] Sheard, J, S Markham, and M Dick (2003). Investigating differences in cheating behaviours of IT undergraduate and graduate students: the maturity and motivation factors. Higher Education Research and Development 22(1): 91-108.

[38] Stepp, M and B Simon (2010). Introductory computing students' conceptions of illegal student-student collaboration. SIGCSE 2010, Milwaukee, USA, 295-299.

[39] Sutherland-Smith, W and R Carr (2005). Turnitin.com: teachers' perspectives of anti-plagiarism software in raising issues of educational integrity. Journal of University Teaching & Learning Practice 2(3), accessed at http://ro.uow.edu.au/jutlp/vol2/iss3/10.

[40] Vandeventer, J and B Barbour (2012). CodeWave: a real-time, collaborative IDE for enhanced learning in computer science. SIGCSE 2012, 75-80.

[41] Vogts, D (2009). Plagiarising of source code by novice programmers a "cry for help"? 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, 141-149.

Computer Science Students Making Games: A Study on Skill Gaps and Requirement

Jussi Kasurinen
Lappeenranta University of Technology
P.O. Box 20
FI-53851 Lappeenranta, FINLAND
+358 400 213 864
jussi.kasurinen@lut.fi

Saeed Mirzaeifar
Lappeenranta University of Technology
P.O. Box 20
FI-53851 Lappeenranta, FINLAND
saeed.mirzaeifar@lut.fi

Uolevi Nikula
Lappeenranta University of Technology
P.O. Box 20
FI-53851 Lappeenranta, FINLAND
+358 40 559 1374
uolevi.nikula@lut.fi

ABSTRACT
Computer science (CS) is a field of practical and scientific approach on computation and applications. Consequently, the CS students should be able to adjust to develop different types of software applications. However, even though video games are one type of software, they also impose additional requirements for the developers. In this paper we present the results of our qualitative studies on how prepared CS students are to function as game developers. The paper assesses the knowledge gaps between students majoring in computer science and game developer needs in two ways; a longitudinal study on a game development course and a focused case study on developing a game. Based on our results there are differences in communication and planning approaches between the CS students and game developers, and skill needs for game development content on a traditional computer science course curricula.

Categories and Subject Descriptors
K.3.2 [**Computing Milieux**]: Computers and Education – *Computer science education.*

General Terms
Experimentation, Human Factors

Keywords
Knowledge gap, game development, computer science, qualitative study

1. INTRODUCTION
Computer science is a field of practical and scientific study on computation, and applications made to be used with computer systems [5]. In this sense, computer science students should be proficient programmers and software developers. However, the video game industry is also considered a software business [12], and even if it has several similar activities as software industry, it also has separate issues of its own [4].

In this study we observe computer science students in game development activities to assess and analyze how competent they are in game development activities. Besides technical competence, we also assess the working methods and test if there are any critical knowledge needs in game development that the computer science curricula does not address. In practice we had two research questions, *"how typical computer science program prepares students to work in the game development tasks"* and *"what topics should be included to a computer science program to better prepare for game industry needs"*.

To approach these research questions we collected student data from two sources. The first source was a game development project course held twice in 2012 and 2013, with 43 master's level students in computer science along with a comparison group of 23 professional game developers. The second data collection method was an in-depth case study within our research group, done with final year master's thesis student and post-doctoral researcher working as a game development team. This study is also related to our other empirical studies on game and software development. In this study we apply our previous knowledge on game and software organizations concerning development tools [14], development methods [13] and technologies [18].

The rest of the paper is structured as follows: In Section 2 studies relevant to this paper are discussed and related research is introduced. In Section 3, the research approach is described and the main results are introduced in Section 4. Section 5 discusses the implications and applicability of the results, and the paper is ended with conclusions in Section 6.

2. RELATED RESEARCH
The common generalization of video game development is that it is a specialized field of software development [12]. However, there are studies in which the general knowledge requirements and connections between software development and game development have been discussed.

A study by McGill [15] discusses the skill needs from the viewpoint of games industry, and assesses the knowledge gaps between industry and academia. The conducted survey assessed the prioritization of different technologies and process aspects by comparing the business needs against academic strengths. Based on the survey, academia mostly met the requirements for programming languages and frameworks such as C#, Python, Java, .NET, ASP or Flash, with only minor changes needed to fulfill the industry expectations. Also based on the McGill survey

[15], industry does not rank development process knowledge, such as software process management, user interface design or mathematical modeling as high but they are more interested in pure technical knowledge and experience.

Another view into game development as a software work is presented by Blow [4]. The development process and knowledge needs for developing a game product have become more complicated since early 2000's, mainly because of increased computing power allows more advanced design. In addition, game development has requirements which are unusual in other business domains; for example development of artificial intelligence, 3D objects and sound design. However, there are studies for example by Kanode and Haddad [12] and Petrillo et al. [17] which indicate that the game development projects are still similar to software development, and have - if not the same - then at least similar process problems and challenges.

In education, the applicability of games in software engineering and computer science education has been studied by Smith et al. [20]. Their implementation of a game development course applied agile practices. In the Smith course, largest problems in the actual development work were related to the communication within the development team. However, similarly to the Claypool and Claypool [6] study, the students in computer science program were capable of designing and implementing game products without any major obstacles. In fact, the final course outcome for Claypool and Claypool course [6] was considered a major improvement against more traditional capstone course. Similar observations have been also made on other software-related topics such as design patterns [7] and software architecture [23].

Table 1: CS program Bachelor-level course structure

Study Credits	Course name or topic
22	University-level mathematics (over 7 courses)
14	University-level physics (over 4 courses)
12	Fundamentals of Computer Science (over 3 courses)
10	Fundamentals of Programming (over 2 courses)
7	Fundamentals of Data Communications
5	Data Structures and Algorithms
5	Operating Systems
5	Object-Oriented Programming
10	Software Engineering (over 2 courses)
5	Databases
13	Communication, technical writing and presentation skills (over 5 courses)
7	User Interfaces and Interface Design
7	Web-based Application Technologies
7	Computer Security
5	Low-level System Programming
5	Fundamentals of Intelligent Computing
10	Bachelor's degree
20	Selected Minor topic
11	Freely selected courses

The earlier studies support the conception that computer science programs offer the required knowledge and skills to develop games. In our current Computer Science (CS) program education is offered in two levels, Bachelor and Master-levels. The students in Bachelor's degree are in one "Computer Science and Communications Software" program, whereas Master's level students can choose from two majors which are "Software Engineering" and "Intelligent Computing". The structure of the Bachelor's degree program is presented in Table 1. The contents of both the degree programs have been validated against the ACM/IEEE Computer Science curricula [11] and accredited by ASIIN in 2012 [1]. The learning objectives of the Bachelor's program are listed in Table 2.

In the Master's level program, the students select a major from communication software, intelligent computing, or software engineering. Majority of the Master's degree program is composed of selected major topic courses (72 credits out of 120), more minor topic courses (20 credits), shared courses between Master's programs (18 credits) and freely selected additional courses (10 credits). Overall, the program content-wise expands and elaborates on the Bachelor's degree program, and is something that could be characterized as a "typical" computer science program. Based on the related research, it can be expected that there should be no problems on the technical aspects of computer science students and game development. Implication from earlier studies [for example 6, 7, 23] and ASIIN certification objectives indicate that the Master's level students should be able to design, construct and do basic quality assurance work on game software without any major obstacles or problems. However, even if the games are seen as software constructs [4, 12], they still have some content such as graphics and audio which is quite unusual for a software product, so there is a valid reason for assessing the possible skill gaps of computer science students in game development work.

Table 2: CS program Bachelor-level objectives by ASIIN

	Objective: The graduates
1	Understand basic principles of scientific thinking and working.
2	Have basic skills in mathematics and natural sciences.
3	Have basic skills in computer science and programming.
4	Can solve problems with self-made computer programs.
5	Can describe and solve problems using software engineering techniques and methods.
6	Can solve data communication problems using various communication networks and different communication patterns.
7	Know the basic principles of intelligent computing.
8	Are capable of independent study and are ready for life-long learning.
9	Can participate in software projects using the acquired knowledge and technical skills.
10	Can work as a part of project team communicating in writing and verbally both in English and own language.

3. RESEARCH METHOD

The results presented in this paper are based on two sources of information; course reports from our game development course held at 2012 and 2013, and from a case study on development of a game demo conducted at our research laboratory. The general view on data sources is available as Figure 1 and the general structure of the game development courses in Table 3.

3.1 Game development courses

The knowledge gaps and the way the computer science students develop games was first observed in two implementations of a game development course. The objective of this course was to design and develop a game within 48 continuous hours, loosely based on a theme announced at the start of each year's development event. The event started on Friday afternoon, and ended on Sunday. The development event was co-located with a local Finnish Game Jam event, which itself is part of the Global Game Jam [9] international game development challenge. All teams working on the game development course participated also in the Game Jam.

Before the development event, an orientation lecture was given on the general game development approaches, including a list of software tools and environments which could be useful for the development teams. After the development event, a report on the development and seminar presentation were required. A summary of the development course is available in Table 3. For the report, our research team designed a questionnaire, to which each student group was expected to answer. These questions included topics such as how the game was tested, what was the most important objective for the development, and what the team would do differently if they had to redo the developed game. For 2013 course we also included some further questions on tool selection and contents of the design documentation. The full sets of questions, including the supplementary questions used on the 2013 course, are available at http://www2.it.lut.fi/projects/SOCES.

For the game development course in 2012 20 Master's level students from Computer Science major enrolled, and for the 2013 course 23 Master's level students enrolled. In addition to the Master's level students, additional participants from local university of applied sciences and local game industry participated, totaling the number of participants to 38 in 2012 and

Table 3: Game development course contents

Course item	
Orientation day	-Lecture on game design and development by an industry expert (2* 45 min) -Lecture on general characteristics of game development, software work and game industry (2* 45 min) -Lecture on game development tools (45 min)
Development event	Continuous 48h development event during weekend.
Seminar day	20 min presentation and talk on the developed game, used technologies and development methods, feedback on results and methods.
Course reporting	15 item written questionnaire (25 in 2013) and report on development event.

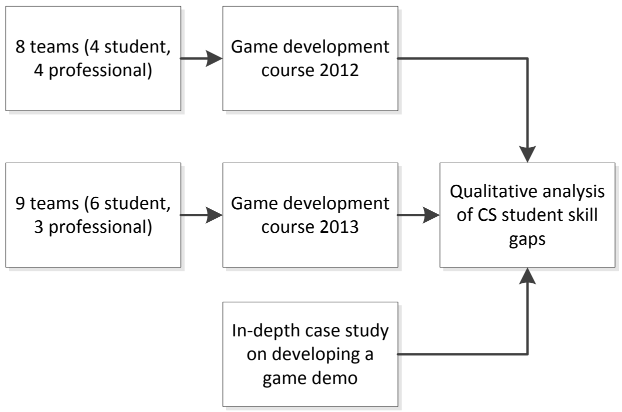

8 teams (4 student, 4 professional)
Game development course 2012
9 teams (6 student, 3 professional)
Game development course 2013
Qualitative analysis of CS student skill gaps
In-depth case study on developing a game demo

Figure 1: Data sources on the qualitative study

54 in 2013. The participants formed in total 20 game project teams (9 in 2012, 11 in 2013), which for the analysis were divided into two groups based on the team member backgrounds. The first group, student teams, were teams where the students from our Computer Science program were a majority. The second group, professional teams, were groups in which the majority of participants were professional game developers, having worked on at least one development project leading to a published product, or were working in a game developing company. Team members which would have fulfilled both criteria – both student and a professional game developer – were classified as professional game developers. From our data set, the 2012 course had four student teams and four professional teams, and 2013 course had 6 student teams and 3 professional teams. Additionally, three teams (1 in 2012 and 2 in 2013) were not included in the analysis as they did not fulfill either of the classification criteria having no majority group or having majority of other participants than CS students or game developing professionals.

The course reports were analyzed based on the Grounded Theory [8] method by Strauss-Corbin [21]. Grounded Theory method is said to be well-suited for analysis work for identifying phenomenon, which has multiple factors in human interaction, organizational aspects and practical work, especially if the studied phenomena is not well-known or strictly definable [21]. For this reason we concluded that this approach was appropriate for analyzing the collected information and observations made on the development approaches the student and professional teams used.

Besides report, the participating teams were observed in work at the development event by the researchers, and for the course reports, teams were requested to give at least a few sentence long answers per question item. This enabled our research team to conduct codification, open coding, and selective coding [21] activities to the data to enable us to compare different teams (inter-team differences) and types of groups (inter-group differences) against each other. In open coding, the different observations made on the teams were collected, and at the same time axially coded, organized, to categories such as "problems", "tool-related issues" or "design method". Since some of the course questionnaire items were strongly related, some of the items were joined during the selective coding to form larger categories. For example, questions regarding *development method* and *considerations on applicability of structured processes* were merged, in the same vein as *observations on tool selection* *principles* and *team background*. The actual codification and analysis work was done using ATLAS.ti software tool [2] and Excel.

In selective coding, we systematically compared and reviewed the collected data and analyzed how different teams functioned to form a bigger picture on how student teams fared against professional teams. Overall, after the codification process we had 259 codified observations in 16 different categories. For this study, the quotations are translated to English by the authors when necessary, since several reports were written in Finnish.

Based on the responses to the questionnaire items and actions during the development event, our research team was able to identify areas in which the inter-group differences between students and professionals were large and consistent enough to differentiate Computer Science students from experienced game developers. Based on these areas, the observations were categorized (observable in Table 4) and main findings were derived (listed in section 4.2) to explain and discuss how these two groups worked differently.

3.2 In-depth case study

Besides the collected course reports which focused on differences between the computer science students and professional game developers, we studied the compatibility of the local Computer Science education and game development needs with an additional in-depth action case study. The objective of the in-depth case study on game development was to assess the knowledge requirements for a game developer by recruiting a last year Master's thesis student to develop a game for the research team. Based on the results, the concept was to assess the encountered problems and knowledge needs of a Computer Science student during the development in general. For the case study, the developed game had the following requirements:

- Full 3D environment populated with 3D objects
- 3D characters with animation
- Sound effects for every interaction
- Several playable stages or levels
- Enemy characters with artificial intelligence, interaction with the player character
- Made with development tools popular with the game industry, selection based on [14]

In general, these requirements were created to reflect the requirements of a real, commercial grade game demo. Besides the Master's thesis student working as a game developer, one post-doctoral researcher acted as a project manager, steering the development and ensuring that the requirements were satisfactory fulfilled by having weekly progress reviews and test sessions. The usability aspects and general results were also tested with random test users in one open door event. Overall, six months was reserved for the design, development, testing and reporting work, from which four months were used in the game development activities and two months in reporting the findings and finalizing the Master's thesis.

The student working as a game developer was selected from a pool of applicants for the open vacancy at the project. The suitable candidate was selected based on the previous course records and work history. The selected candidate had good course history (4,5 average on scale of 0-5 with 5 being best grade) and more than a year of experience as a software developer for non-game-related software company.

4. RESULTS

In this section we present the results of our two analyses. First, we discuss the categorized observations and main findings made from the game development course based on the Grounded Theory analysis. Secondly, we discuss the in-depth case study and finally, in the implications the combined results are presented.

4.1 Observations from courses

Based on the 17 game course reports and observations made during the development event, we were able to identify seven categories, which focus on the differences between the student groups and the professional developers. Based on these categorized observations, we were able to derive four main findings, which summarize and explain the differences between the students and the professionals. The categorized observations are summarized in Table 4, with student teams being the teams S1 to S10, and professional teams being the teams P1 to P7.

The category *Amount of design* describes the amount of design work the group did before starting implementation. The "+" means that the team created a comprehensive game design document, with the game features, technical design and overall gameplay. The "0" means that there was a document, which summarized at least some of the game design, and "-" that the group did only some minor drafts or minutiae before starting the implementation work.

The category *Source of input* describes the way the team tested their game design or collected feedback for their game. *Internal* means that the team only used internal sources or did not do any significant cooperation with people outside the development team. *External* implies that the team collected feedback from test users outside their own team, or actively discussed their design decisions with peers during design work.

The category *Applied tools* describes the main development tool the team used when making their game. *Game engine* means that the team applied development tools designed for game development, such as Unity [22] or Scirra [19]. *Programming language + framework* means that team used a programming language with a suitable framework to enable game development, such as C++ with Box2D or Python with PyGame. *Self made* means that team decided to develop the game directly with some programming language, constructing all the required libraries and functions themselves.

The category *Most important objective* describes the most important result the developer team was aiming at in their work. *Technically working* means that the game demo should be technically proficient, with all the intended functionalities working as designed. *Game experience* means that the focus on development was on the game content, presentation and playability. With *Game design* the final product could be missing some technical details, or during the development the technical aspects were redesigned to meet the schedule or ease complexity.

The category *How close to original idea* denotes how close to the original idea for game the team got in their final product when comparing the first design and final game demo. "+" denotes that the team was basically able to implement their original idea fully or that they only had to do minor revisions to their design. "0" denotes that the team had to drop some features or large part of the designed content to meet the deadlines. "-" denotes that the team had to either redesign their entire game during the development, or drop major features or most of the intended content.

The category *Root cause of difficulties* describes the most important thing the team felt was responsible for their development problems. *Tools* category includes problems and restrictions caused by the selected development tools. *Talent* denotes that the team felt that they were missing some key talent in their group, for example in programming, graphics or audio experience. *Programming* means that the team felt that the most restricting part was the programming work, finding bugs in the game or implementing features the way the team wanted. One team also named *Project management*, which denotes that the team had problems coordinating their effort since part of their team was working off-site due to unexpected circumstances.

Finally, the category *What would do differently?* describes the most important change the team would do to their approach, if they were to do the same or similar tasks again with the same team. *Tools* means that the team would use different tools, or would try to incorporate their working style more closely to the features provided by the selected tools. *Planning* indicates that the team would do more design and preparation work before starting the implementation to familiarize themselves with the libraries, algorithms or generally, make more detailed game design document before starting the implementation.

4.2 Findings based on game course

Based on the reports and observations made on the student and professional teams during the game development course, we were able to define four main factors which differentiate the students and the professionals from each other. These were the following: student teams tend to focus on the technical aspects of the game development, students do not use external knowledge to a large degree and that overall professional teams communicate more on

Table 4: Observations from game course development teams; student teams in white (S1-10), professional teams in grey (P1-7)

Teams	Amount of design	Source of input	Applied tools	Most important objective	How close to original idea	Root cause of difficulties	What would do differently?
S1	+	Internal	Game engine	Technically working	0	Tools	
S2	+	Internal	Programming language+framework	Technically working	+	Programming	
S3	0	External	Programming language+framework	Game experience	0	Tools	
S4	0	Internal	Game engine	Technically working	-	Talent	Tools
S5	+	Internal	Programming language+framework	Technically working	0	Talent	Planning
S6	0	Internal	Programming language+framework	Technically working	0	Programming	Tools
S7	+	External	Game engine	Game experience	+	Tools	
S8	+	Internal	Self-made	Game experience	+	Programming	Tools
S9	-	Internal	Self-made	Technically working	-	Programming	Tools
S10	+	External	Programming language+framework	Game experience	-		Planning
P1	+	Internal	Game engine	Technically working	+	Programming	
P2	+	External	Game engine	Game experience	0	Programming	Planning
P3	0	External	Programming language+framework	Game experience	+		
P4	+	Internal	Programming language+framework	Game experience	+	Project management	Planning
P5	+	Internal	Game engine	Technically working	0	Talent	Planning
P6	0	External	Game engine	Technically working	0	Programming	Tools
P7	+	External	Game engine	Game experience	0	Programming	Planning

their ideas and concepts. Next we shall go through all these findings in more detail.

4.2.1 Professional teams focus on game mechanics, student teams on technical aspects

The difference between professional teams and student teams in the most important development objective was very indicative of the design mindset the teams had. Majority of the professional developer teams considered as their most important objective to get the game experience as good as possible, whereas student teams were making a system that would fulfill the design requirements.

"The main goal was to make a game that is easy and fun to play." – Team P3

"Our plan was to do something original, nightmarish, Alice in Wonderland-type fantasy game." – Team P4

"At first the team decided to create the game with a single level, with a single goal, and with only the minimum number of required features such as rotation. After that more advanced features would be added." – Team S2

"For us the most important objective was developing the controls of the game… graphics of the game was secondary issue." – Team S4

In two of the professional teams where the aim was not in the game experience, P1 and P5, the reason for aiming towards technically working game was that the team did not have actual programming experience, and the team consisted of only game designers and artists. Also, Team P5 recognized this as their biggest problem.

"When you take into account our previous backgrounds with game programming, [name] being only one who had programmed as a game developer, [a tool] provided us relatively good basis to make games." – Team P1

"Most stuff was easy to implement but hard part was getting them to work correctly." – Team P5

4.2.2 Professional teams seek outside opinions

One difference on the way the teams developed their designs and tested their games was on the use of outsider information. Similarly as with the focus on game mechanics, professional teams wanted more external opinions by either discussing their designs with non-group members, or using non-group members as their testers.

"The game demo was tested by handing out the prototype to a group of other participants." – Team P3

"We tried the game ourselves and also asked other people from different groups to try our game. After those experiences we adjusted the game a little." – Team P6

"Every now and then via network we sent [the current build] to fellow developers and a couple other interested parties, from where we also got some testing feedback." –Team P7

In several cases (S2, S5, S8, S9) the student team tested their game only by doing unit testing or play testing themselves. Most student teams considered that the 48 hour deadline was too restrictive to actually focus on usability or game experience aspects. Instead of play testing with non-group members, some of the student teams such as S5 and S7 focused on fine-tuning the technical solutions of their product.

"We tested our game by playing it. Every time we added new features, we tested them in game. In final stages we also checked if there were any memory leaks and tried to fix them." – Team S5

"The fact that one of our mobile testing devices was of older technology with less computing capabilities, helped us identify performance bottlenecks in less powerful devices. We managed to overcome those with profiling tests and code optimization where needed." – Team S7

4.2.3 Professional teams and students would improve different things for the second project

The most common solution to the problems with student teams was seen either in the technology or tools. For example, teams S2, S4, S5 and S7 had problems with the selected technology, with either the intended controlling scheme being uncooperative, or the external system libraries having problems related to the intended usage.

"The performance of [external library] is a greatly restricting feature. Liquid simulation itself should be completely recreated as well as the engine behind the whole game." – Team S2

"We probably wouldn't base the collision engine on a matrix. Some better method for version management might also be in place." – Team S5

"If we [were to make another game], we will definitely use development tools, with which we have some experience. Maybe we will try to make some Linux or multiple platform game using SDL + OpenGL." – Team S4

"Some other programming language or tool could be used straight from the beginning to allow better distribution and user base coverage." – Team S 8

4.2.4 Professional teams used domain-specific tools, student teams programming languages with frameworks

The teams were also divided based on the tool selection principles they applied. The student teams were more eager to work directly with a programming language, whereas professional teams applied more sophisticated tools. This approach also went beyond the reasonable expectations of experience with the game-development specific tools; some student teams actually rejected the idea of full game engine to gain "more control" over the technical aspects:

"The main disadvantage with advanced game engines is that, even though usually designed to be simple to use, often they require quite a bit of time to be grasped to the level where you could fluently create the kind of results you could achieve with a traditional graphics library and programming language combination. Also, programming your own game engine from a scratch gives you total control over the game." – Team S8

"We preferred to use traditional tools and libraries because we wanted to start game programming from the basics." – Team S5

4.3 Observations from the case study

The in-depth case study on game development knowledge needs was conducted between the first and second implementation of the game development course. The aim of this in-depth case study was to assess the skills gaps of our computer science students and our degree structure, when presented with the task of creating a reasonably modern video game. As a result, one last year Master's thesis student spent four months working on a game development project, with one post-doctoral researcher working as a project manager, steering the development work and ensuring that the requirements were met in each category.

Based on the case study, most of the development time was spent on the development of 3D art and learning the development tools. Third most time consuming task was quality assurance – testing – to ensure that all of the implemented features worked and that the requirements set for the project were fulfilled. On Table 5 is a summary of all the working hours spent on different activities.

The game development took 655 hours, approximately 18 weeks' worth of effort (See Table 5). Half of the effort (game mechanics, content design, 3D artwork, and audio work, total of 330 hours) was spent on tasks that were not covered directly in any of the student's previous courses. Rest of the tasks such as planning, project management, development and testing work, were comparable or at least partially covered in the topics taught at some point during the studies. Overall, during the game development we identified following problems:

- Unfamiliar tools: The tools used in the game development project were significantly different from "normal" software development tools, so earlier experience on software development with their IDE tools was not applicable.
- Too ambitious project plan: The original project schedule and plan were too ambitious to be implementable as is, and was tuned down to match deadlines for example by cutting activities like acquisition of outsourced assets and user testing to a minimum.
- Skill gaps in development: In programming work, the previously acquired knowledge related to collision detection, artificial intelligence and 3D vector mathematics provided to be insufficient.
- New knowledge requirements: Earlier courses provide no support on game development-focused topics such as sound engineering or working with 3D models.

4.4 Implications

The conducted two studies gave our research group data on how capable our computer science students are as game developers. Similarly as in the related research studies [for example 6, 20 etc.] the students were able to develop games as course project work. However, there were some differences between the approaches student groups took on development work when compared with professional teams, and in addition, some knowledge gaps were identified in the in-depth case study.

Based on the game development course findings, the students focused heavily on technical details and followed the original design, whereas professional teams made changes to the design if they encountered too many issues. In fact, two student teams (S4 and S9) spent most of their development time tackling technical difficulties imposed on them by their original plan and idea. Similarly to making changes to the design, professional teams were more willing to modify their work based on external feedback, and used external feedback to a larger extent. In testing work, many professional teams (P2, P3, P6, P7) applied user testing to try out their ideas whereas only one student team did that. Supporting the notion of students focusing on technical issues, two student teams (S5 and S7) focused on finding performance issues with memory management and algorithms.

Table 5: Working hours spent on different game development activities

Task	Hours	Description
Planning	30h	Project management, game design
Game mechanics	38h	Game design, 2D design, texture work
Content design	42h	Level design, construction of maps
3D artwork	215h	Development of own 3D objects, modification of freely available objects
Development	165h	Development work with the game engine.
Audio work	35h	All audio and sound effect related tasks
Quality assurance	130h	Testing

On the tool side, student teams preferred programming languages with some suitable frameworks, although there are explanations for this behavior. Even though the game engine-based IDE tools were introduced to students, based on the in-depth case study it can be argued that the students did not have enough experience to use them effectively or at least they chose to stay with the more familiar tools. Some student teams (S5, S8) chose to use programming languages to gain more control over the entire system, and several student teams (S4, S6, S8, S9) considered that they should change their tools, but not necessarily towards game-oriented development tools.

As for the research questions, it seems that there are gaps in the acquired skills our Computer Science students have. Based on our observations *our current computer science program does not have enough support for more game-oriented topics such as 3D mathematics, game-related algorithms or audio work. In addition, when developing games the students do not communicate as much as professional developers and their project effort tends to focus on technical aspects.* In general, the students seemed to focus too much on their first design and get stuck on technical issues caused by that design, whereas game developers made changes based on feedback on the practical issues of the game development work. Additionally, based on our in-depth case study, after completing most of our Master's program studies, our student had acquired about half of the skills (measured in working hours) required for developing games with modern tools. In general, four major shortages in the current Computer Science program and program recommendations were observed: 1) the tools for professional game development differ significantly from "normal" IDE tools, 2) the current curricula does not offer enough experience on some topics such as artificial intelligence, 3) the current curricula is missing game development-related topics such as 3D modeling and finally, 4) approximately half of the game development effort is spent on activities which are not covered in the "traditional" Computer Science curricula.

In summary, to develop the Computer Science program towards the viewpoints of game development, following topics should be addressed in more detail: game-related mathematics and algorithms such as artificial intelligence and vector mathematics, 3D object design, audio work, and communication and team working skills, especially during design and testing.

5. DISCUSSION

The results of this study indicate that there are skill gaps and course needs which could bridge the differences between what is taught in our Master's level Computer Science program and what skills game developers need in general. Our program can be characterized as "typical" Computer Science program, and when compared to general recommendations such as ACM/IEEE CS curricula [11] it has no obvious needs or shortcomings, but still our reports indicate that to accommodate game development to our curricula, we should expand our course offerings. Considering the type of observed skill gaps, it would seem probable that these same skill gaps in general exist in Computer Science curricula, if no action to address game development needs is taken. This finding is not a big surprise but supports the need to develop own curricula for game developers as has been done in England [10]. In their definition project planning, programming and testing, which are part of typical Computer Science curricula, cover less than half of the recommended topics. Besides them, focus on topics such as game design, graphical skills, audio work and business studies are recommended for IT students aiming towards game industry.

It should also be remembered that the Computer Science students in this study were introduced to game development tools during the orientation lecture. Some of the student groups made the decision to apply basic programming language with framework mostly because they wanted more control over the low-level technical aspects. Similarly the technical mindset was present in the testing practices; student teams focused on internal testing work and technical aspects instead of playability or testing how "fun" the game design was. Overall, when compared to professionals the student teams were not very keen to adopt ideas originating outside their project team at any phase.

There are threats that should be acknowledged when addressing the validity of qualitative research approaches [24]. For example, reliability and validity in qualitative research are not the same as in quantitative research such as surveys, so they should be explained in more detail to put the results and observations presented in the study into a context. For example, Onwuegbuzie and Leech list several threats in qualitative studies and approaches to qualitative data analysis that can affect the results [16]. According to their study, the most common and severe threat is personal bias meaning that the researchers apply their personal opinions, knowledge, and believes in the data analysis, disregarding the patterns and observations which do not fit or support their own theories. Obviously this can affect the data collection and analysis, and in worst case scenario make the study unreliable [16]. In this study these risks have been taken into account when planning and implementing the study with several actions. First of all, the study questionnaires were designed by four researchers to avoid personal bias on the questionnaires; the data collection was conducted via email to prevent affecting the report contents or leading the interviewed game development groups. Finally, the analysis was validated by three researchers to assure that the reported findings represent the collected data.

Assessing the in-depth case study results is a separate issue. First of all, one case study obviously is not enough for conclusive study, but the results are indicative for development needs. The Master's thesis student conducting the case study was not a graduate from our Bachelor's program, but from another university, so the Bachelor's level curriculum does not really apply. However, the student had completed Bachelor's degree on Computer Science, so the domain was the same, even if there were small differences in detailed course structure. In addition, the student was on final stages of our Master's level program, and also had a high average grade (4,5 on scale 0-5 where 5 is the best grade), with work experience from software industry from a large international non-game company. Based on this background, it can be assumed that the student had representative computer science skills, even exceeding those of an average Master's thesis student. In any case, considering the general nature of the observations, the observations are still valid as recommendations when designing game-related content to the course curricula.

Qualitative studies are always limited to the environment where the study was conducted. Beyond this environment, the study results should be regarded as recommendations or suggestions which can be applied in other context to provide understanding of their practices [3]. In our case, the in-depth case study and qualitative analysis results indicate that our current course contents do not cover game development in all areas, and the situation could be improved by including the identified skill gaps in the voluntary course offerings.

6. CONCLUSIONS

The Computer Science studies should offer enough knowledge to enable the graduates to acclimate themselves to any software industry they want to. In this sense, the game industry is no different from other software development fields, since the students are able to produce games when requested to do so. However, the game development work has large bodies of content which are not included in our Master's Thesis curricula, or in the ACM recommendations for Computer Science program. In this paper we presented results from our study on the knowledge and skill gaps between "a typical" Computer Science program and game development. The observations were collected from two implementations of our game development course and one in-depth case study, analyzed with the Grounded Theory approach.

The results indicate that the observed Computer Science students had skill gaps in game industry-related content. Students tend to work towards technology-oriented objectives, and do not use external sources of information in their design and testing activities. Computer Science students were also much less likely to do comprehensive design work before implementation or change their designs if they encountered problems.

Our case study on developing games revealed that there are topics and skill gaps which should be addressed in Computer Science curricula if the objective is to assist graduates into the game industry. We observed that there are topics such as game-focused algorithms, sound engineering and 3D object development which are insufficiently covered or completely missing from our course contents. In the study, approximately half of the work effort was spent on the topics which were not covered by the Computer Science curricula. The design and development of animated three-dimensional objects alone took one third of the project effort.

For future work, these findings are the basis for recommendations in the development of Computer Science degree structure, which also supports game development. The content recommendations identified in this study are used in the design of additional courses to enable students to more easily adapt to the game industry and to give content recommendations for our existing courses. For scientific future work, we will observe how the changes affect the students participating in the game development courses to assess and identify the best methods of teaching game development.

7. ACKNOWLEDGMENTS

This study was supported by the European Union Regional Development Fund projects number A31814, "Kaakon Peliklusteri" administered by the Council of Southern Karelia and number A32139 "Game Cluster" administered by the Council of Päijät-Häme, and by the organizations funding the related research projects. We would also like to thank all the interviewed students and representatives of the participating organizations.

8. REFERENCES

[1] ASIIN e.V., 2013. "ASIIN General Criteria for the Accreditation of Degree Programmes", referenced 28.3.2013, available at http://www.asiin-ev.de/

[2] ATLAS.ti: The Qualitative Data Analysis Tool, http://www.atlasti.com/, referenced 19.6.2013.

[3] Birks, M. and Mills, J. "Grounded Theory – A practical guide", SAGE Publications Ltd, 2011. ISBN 978-1-84860-992-1

[4] Blow, J., 2004. Game Development: Harder Than You Think, *Queue*, vol. 1, nro. 10, ss. 28–37, February 2004.

[5] Brookshear, J. Glenn, 2005. *Computer Science: An overview*, Ninth edition, Pearson International, Addison-Wesley, ISBN 0-321-43445-5.

[6] Claypool, K. and Claypool, M., 2005. Teaching software engineering through game design. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* (ITiCSE '05). ACM, New York, NY, USA, 123-127. DOI=10.1145/1067445.1067482

[7] Gestwicki, P. and Sun., F-S., 2008. Teaching Design Patterns Through Computer Game Development. *J. Educ. Resour. Comput.* 8, 1, Article 2 (March 2008), 22 pages. DOI=10.1145/1348713.1348715

[8] Glaser, B. and Strauss, A.L., 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine.

[9] Global Game Jam, http://globalgamejam.org/, referenced 19.6.2013.

[10] Ip, B. 2012. Fitting the needs of an industry: An examination of games design, development, and art courses in the UK. ACM Trans. Comput. Educ. 12, 2, Article 6 (April 2012), 35 pages

[11] Joint Task Force on Computing Curricula, 2001. Computing Curricula 2001, Computer Science. IEEE Computer Society Association for Computing Machinery, Final Report, 15.12.2001. Available at http://www.acm.org/education/curricula-recommendations

[12] Kanode, C.M. and Haddad, H.M. (2009) "Software Engineering Challenges in Game Development", Proc. 2009 Sixth International Conference on Information Technology: New Generations, 27.-29.4., Las Vegas, USA. DOI: 10.1109/ITNG.2009.74

[13] Kasurinen, J., Laine, R. and Smolander, K., 2013. "How applicable is ISO/IEC 29110 in Game Software Development?", Proc. 14th Int. Conf. on Product-Focused Software Development and Process Improvement (Profes), 12.-14.6.2013, Paphos, Cyprus.

[14] Kasurinen, J., Strandén, J. and Smolander K., 2013. "What do Game Developers Expect from Development and Design Tools?", Proc. 17th International Conference on Evaluation and Assessment in Software Engineering (EASE), 14.-16.04.2013, Porto de Galinhas, Brazil.

[15] McGill, M., 2009. "Defining the expectation gap: a comparison of industry needs and existing game development curriculum", In Proceedings of the 4th International Conference on Foundations of Digital Games (FDG '09). ACM, New York, NY, USA, 129-136. DOI=10.1145/1536513.1536542

[16] Onwuegbuzie, A.J. and Leech, N.L. (2007). "Validity and Qualitative Research: An Oxymoron?", Quality and Quantity, Vol 41(2), pages 233-249. DOI: 10.1007/s11135-006-9000-3.

[17] Petrillo, F., Pimenta, M., Trindade, F. and Dietrich, C. (2008) "Houston, we have a problem…: A survey of Actual Problems in Computer Games Development", Proceedings of SAC'08, 16.-20.3.2008, Fortaleza, Brazil.

[18] Riungu-Kalliosaari, L., Kasurinen, J. and Smolander, K., 2013. "Cloud Services and Cloud Gaming in Game Development", accepted for publication in Proc. the IADIS Game and Entertainment Technologies 2013 (GET 2013), 22.-24.7.2013, Prague, Czech Republic.

[19] Scirra 2; Create Games with Construct 2, http://www.scirra.com/, referenced 19.6.2013.

[20] Smith, T., Cooper, K.M.L. and Longstreet, C.S., 2011. Software engineering senior design course: experiences with agile game development in a capstone project. In *Proceedings of the 1st International Workshop on Games and Software Engineering* (GAS '11). ACM, New York, NY, USA, 9-12. DOI=10.1145/1984674.1984679

[21] Strauss, A. and Corbin J., 1990. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. SAGE Publications, Newbury Park, CA, USA.

[22] Unity – Game Engine, http://unity3d.com, referenced 19.6.2013.

[23] Wang, A.I., 2011. Extensive Evaluation of Using a Game Project in a Software Architecture Course. *Trans. Comput. Educ.* 11, 1, Article 5 (February 2011), 28 pages. DOI=1921607.1921612

[24] Whittemore, R., Chase, S.K. and Mandle, C.L., 2001. Validity in Qualitative Research, *Qual Health Res*, July 2001, 11: 522-537, doi:10.1177/104973201129119299

Alternate Reality Games for Computer Science Education

Lasse Hakulinen
Department of Computer Science and Engineering
Aalto University
Finland
lasse.hakulinen@aalto.fi

ABSTRACT

Alternate reality games (ARG) are games that often blur the boundaries of reality and fiction. They use many different types of media to deliver an interactive narrative to the players and include puzzles that are part of a bigger quest that the players are trying to solve. They are not widely used in education and there is limited amount of research done considering their benefits to learning. However, especially some commercial entertainment ARGs have managed to engage people in collaborative problem solving very well. Therefore, benefits and issues of using ARGs in education, and especially in computer science education (CSE), are discussed in this paper. Alternate reality games could potentially be used to teach various computer science concepts, to enable student networking, and to promote computer science programs. A case study was conducted in order to research the potential of using alternate reality games in computer science education. The research of the case study is still in progress, but the preliminary results are promising. Therefore, we want to raise discussion of using alternate reality games in computer science education.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Human Factors

Keywords

Alternate reality games, serious games, game-based learning, collaborative learning, computer science education

1. INTRODUCTION

Alternate reality games (ARG) are games that deliver an interactive narrative to the players using different types of media. They often consist of puzzles and challenges that are part of a bigger quest that the players are trying to solve. The puzzles can be very challenging and require a wide range of skills and therefore they often call for collaboration among players. Typically, ARG participants are actively engaged in the game and solve difficult tasks collaboratively or individually in order to make progress in the game.

Alternate reality games can be seen as a way to combine voluntary learning, problem-solving, collaboration and even peer-learning. This offers an opportunity for educators to use ARGs to engage students in learning tasks. There are some examples of ARGs that have been developed for educational purposes but clearly the opportunity is not fully exploited. Moreover, research done on the educational possibilities of ARGs is limited and the need for further studies is indicated in the literature.

Alternate reality games have some similarities to other approaches used in computer science education (CSE) and they could be a motivating way for some students to learn and practice computer science skills. Ideally, ARGs could be used to enrich computer science education and to offer participants different learning experiences than other teaching methods can offer. However, integrating them to teaching is not a trivial task and more experience and research is needed about effective practices of using ARGs in computing education.

This paper presents an overview of alternate reality games and discusses their use in education. Moreover, their suitability and potential to computer science education is discussed. We also introduce a work-in-progress case study of a computer science ARG and some preliminary results of the research.

2. RELATED WORK

Serious games are games that have also other objectives than pure entertainment [22] and they are being used in various ways in computer science (CS) education. Programming assignments are a natural way to involve games in CS education and there are many examples of using games in programming courses. Leutenegger and Edgington [17] describe the use of 2D game development as a unifying theme in introductory programming and they state that game programming motivates most new programmers. Decker et al. [7] used the implementation of a game as a capstone assignment for CS1 course and Kurkovsky [15] describes the use of mobile game development projects in CS1 and CS2. Moreover, Kurkovsky argues that mobile games are simpler than computer games and therefore they are better suited

Koli Calling '13, November 14 - 17 2013, Koli, Finland
Copyright 2013 ACM 978-1-4503-2482-3/13/11 ...$15.00
http://dx.doi.org/10.1145/2526968.2526973.

for introductory programming.

Kazimoglu et al. [13] argue that current serious games that are specifically developed for learning programming do not consider a deep gameplay for developing computational thinking skills. To address this problem, they describe a serious game called *Program your robot*. The game enables students to practice introductory programming constructs within an environment that supports the acquisition of computational thinking skills, such as: algorithm building, debugging, and simulation.

Wallace et al. [32] grouped the use of games in CS education based on how the games facilitate the learning process. They report that most commonly games are used in a way where students need to implement computer games or write programs related to the game. Scenarios where students learn by playing a game, as opposite to implementing it, are far less common. Nevertheless, there are examples of using serious games in CSE in a way where students learn by playing a game and not by implementing it. Shabanah et al. [25] describe games that can be used to teach algorithms. In the games, players need to simulate the functionality of an algorithm in order to proceed to the next level. Schäfer et al. [24] describe a game-based multitouch learning environment for practicing mathematical logic for CS education.

Serious games are often associated with computer games but there are examples of non-computer games used in CS education as well. Hamey [11] used a game with envelopes, papers and key tokens when teaching secure communication protocols. Hakulinen [10] describes the use of card games to teach sorting algorithms and Bell et al. [3] introduce computer science concepts by using games and other activities in *Computer Science Unplugged*.

2.1 Alternate Reality Games

Alternate reality games (ARG) is a relatively new genre of games. Typically, they involve online puzzles as well as location based real world challenges that the players are solving together. ARGs use different media creatively and interact with the players letting the game evolve in real time. A popular ARG community called Unfiction describes ARGs as: *"interactive fusion of creative writing, puzzle-solving, and team-building, with a dose of role playing thrown in"* [30].

Alternate reality games can be very different by nature and they lack a concise definition. However, many descriptions of ARGs emphasize their strong relation to the real world. McGonigal [21] describes ARGs as *antiescapist* games and suggests that they are not played to escape real life, but rather to get more out of it. Kim et al. [14] say that ARGs are designed to blur the distinction between a player's experience in the game world and the real world outside the game. Szulborski [29] suggests that a successful ARG immerses the world of the game into the everyday life of the player instead of immersing the player into an artificial world of the game. These descriptions are supported by a central idea in ARGs called the *This Is Not A Game* (TINAG) aesthetic [20]. TINAG emphasizes the fact that the game does not represent itself as a game but balances between real life and fiction.

An alternate reality game can be seen as a collaborative effort to solve a mystery or a quest in a setting designed by the organizer of the game, called a *puppetmaster*. The tasks needed to make progress in the game can vary between different ARGs but typically involve puzzle-like assignments that may require collaboration and deep knowledge on the subject. ARGs also involve fictional game characters that the players are able to interact with and impact the narrative of the game. In order to blur the border between the game and real life, ARGs often use media that is already common for the players, for example: email, social media, phone calls, and blogs, instead of having dedicated communication channels implemented for the game only. Hence, alternate reality games can be perceived as games that integrate to players' real lives letting the players to become part of the game.

One of the first successful ARGs was a game called *The Beast* that was created to market the movie *Artificial Intelligence* [14]. The Beast succeeded very well in enticing players to collaborative puzzle-solving and had an estimated three million unique visitors to the game website. Another well known ARG, *I love bees*, was developed as a viral marketing campaign for the *Halo 2* video game [14].

Swedish Television (SVT) succeeded to blur the border between reality, fiction, and games exceptionally well with their interactive drama *The Truth About Marika* [9]. The idea of the game was based on an accusation that SVT had stolen the story of the TV series from a young woman's blog who was searching her missing friend. Viewers of the show were lured to solve assignments and post their progress online in order to solve the "truth about Marika". Waern and Denward [31] researched the game in order to find out the different participant interpretations of the production. They found out that as much as 30% of the people who answered the survey reported that they thought that the story was real. This was a surprise for the game developers as the most desired answer for the question would have been "I pretended that it was real". However, only 24% players chose that answer. It might seem surprising that the most desired answer would have been "I pretended that it was real". However, it can be seen as a central characteristic for alternate reality games and the TINAG aesthetics. McGonigal [19] calls it the *Pinocchio effect*, meaning that the players choose to pretend that the story is true even thought they actually know it is fiction.

2.1.1 ARGs in education

Alternate reality games have been mainly used for marketing and promotion of commercial products but they have been used in education as well. Beer and Holmner [2] report on using an alternate reality game as a capstone course in a multimedia post-graduate degree. In the final year of the studies, students have an opportunity to design an alternate reality game where they need to apply the knowledge and technologies they have learned in the previous years of their studies.

Connolly et al. [5] developed an ARG called *Tower of Babel* for language learning. There were 328 students from 28 schools across 17 European countries participating in the game that lasted for 10 days. They report that, in general, students' attitudes towards the game were very positive and they found it to be a useful way to motivate students in learning a foreign language.

Whitton [34] describes the *Alternate Reality Games for Orientation, Socialisation and Induction* (ARGOSI) project that was aimed at university students to help them in the beginning of the studies. The game combined a series of collaborative challenges within an unfolding storyline. The game was meant to be a mechanism for students to make friends, orientate themselves to the new area, and to learn basic information literacy skills. The game had a total of 173 players, 23 of whom were active. The number of active players was lower than they had expected. However, despite the limited amount of research data, they were able to conclude that the ARGOSI project was successful at creating and piloting a workable ARG. They also state that, for some students, the game was an effective and appropriate medium for meeting the learning outcomes set for it.

3. SUITABILITY FOR EDUCATION

Several authors have highlighted the lack of empirical evidence on using games in education and the limited amount of studies that analyze the games that have been used in education [5, 4, 26]. The issue of not having empirical evidence about learning during a game is even greater with alternate reality games. The whole game genre is new and educational ARGs are a marginal part of the genre and therefore there are not many studies done about the effects of educational alternate reality games.

When thinking of including an ARG to education, it is important to acknowledge the purpose of the game and the potential target audience. Whitton [33] states that ARGs generally appeal to a small proportion of the population, but those who do become involved, typically are extremely engaged in the game. Whitton's statement suggests that ARGs could be effective as an alternative way of learning for those who become engaged in the game. However, if the majority of students are not engaged in the game, it is not suitable for a compulsory assignment.

Moseley [23] presents several features that ARGs offer and which would be beneficial in educational contexts to increase engagement, problem solving skills and communities of practice within the subject:

- ARGs involve problem solving at varying levels and enable students to pick their own starting level and work up from there.
- The game has a feel of progress and it rewards successful participation (e.g. in form of a leaderboard or a grand prize).
- ARGs involve a narrative that can be used to give a sense of purpose for the tasks/puzzles and to increase players' engagement in the game.
- Players' actions can have an influence on the outcomes of the game.
- ARGs can deliver new problems/events regularly in order to keep players engaged in the game.
- There is a potential for a large and active community which can also be self-supporting, especially if the subject of the game is not too narrow.
- ARGs can be based on simple and existing technologies/media that does not need a big effort to take into use.

Learner's motivation towards the learned subject is always important. Alternate reality games are often completely voluntary and finding the initial clue of the game, the *rabbit hole*, can sometimes be an achievement in itself. If there are no external rewards involved in playing the ARG, it is especially important to make the game motivating to keep the players engaged in the game and to avoid them from dropping out. Whitton [33] reports six possibly motivational elements in ARGs that were identified in interviews done in the ARGOSI project: community, competition, completion, creativity, narrative, and puzzle-solving. Example implementations of the motivational elements are shown in Table 1. Malone and Lepper [18] define a taxonomy for intrinsic motivation for learning and they identify *challenge, fantasy* and *curiosity* as elements that promote intrinsic motivation. Davies et al. [6] suggest the following guidelines for ARGs to promote intrinsic motivation by including the elements of challenge, fantasy and curiosity:

1. The player must be able to tangibly affect the outcome of the game.
2. There must be an overriding goal/challenge as well as sub-goals and challenges to the player with positive and negative utcomes based on their actions.
3. The game must require mental or physical skill.
4. The outcome must be uncertain at the outset.
5. The ARG must require the player to develop strategies in order to win or succeed.
6. The ARG must offer multiple paths to success.
7. Players must be able to ultimately overcome most obstacles in the game.

Connolly [5] states that collaboration among players forms a key role in ARGs as players must often work together when solving puzzles in order to successfully complete the game. The collaborative nature of ARGs can be seen as an opportunity to networking but also to peer-learning among the players. Furthermore, Lee [16] points out that in ARGs, players act as themselves and ARGs rely on knowledge that the players possess in real life. She sees that as an advantage over traditional types of computer games where players help an avatar to "learn" skills.

3.1 ARGs and Computer Science Education

Alternate reality games are not a commonly used method in computer science education but there are similarities to other methods that are more widely used. Programming competitions in various forms offer opportunities to practice programming and problem solving and different online puzzle resources have a wide range of exercises requiring CS skills. Moreover, Massive Open Online Courses (MOOCs) and open learning environments such as Codecademy[1] and Khan Academy[2] offer many opportunities to learn computer science. All these approaches have some similarities with alternate reality games, but none of them cover all the features that a computer science ARG could have.

Programming competition tasks have many similarities with the individual puzzles in a computer science ARG and it seems possible to use programming competition tasks in

[1]http://www.codecademy.com/
[2]https://www.khanacademy.org/

Table 1: Possibly motivational elements of ARGs [33].

Element	Example implementation(s)
Community	Collaborative activities, communication tools
Competition	Prizes, leaderboard
Completion	Overview of complete structure, pieces needing filled in
Creativity	Creative challenges that involve making artefacts
Narrative	Ongoing storyline that contains a mystery
Puzzle-solving	Challenges based on puzzle-solving

a computer science ARG. However, there are also significant differences in the overall setting of an ARG and a programming contest. Alternate reality games try to increase the engagement by creating a compelling story and a sense of bigger purpose rather than providing several non-related tasks. Furthermore, in ARGs, realizing the actual task is often part of solving the puzzle.

There are some concepts in computer science that are beneficial considering educational alternate reality games. Cryptography and hidden messages are already often used in ARGs whether it is included in the objectives of the game or not. The fact that encrypting and hiding messages is a typical feature in ARGs builds a convenient setting to include learning goals related to cryptography that can be easily included in the storyline of the game.

Alternate reality games typically include different puzzles that need to be solved in order to make progress in the game. This offers an opportunity to make various puzzles involving algorithmic thinking, programming and problems from all areas of computers science. The puzzles can be similar to tasks used in CS courses or programming competitions but presented in a way that fits to the storyline of an ARG. Furthermore, the flexible design of alternate reality games makes it easy to include programming and non-programming puzzles in an ARG depending on the learning goals of the game.

3.2 Issues

Building a successful alternate reality game for educational purposes is not an easy task. There are examples of ARGs that have failed to meet the goals set to them and sometimes the game has been stopped because of lack of players before it was played to the completion [28]. Whitton [35] reports on difficulties in engaging students in the ARGOSI project. Student interviews revealed some reasons for the low participation rate. Several students would have been more interested in participating in the game if they had realized that it would benefit their studies. Whitton also reports that the steps needed to get started in the game were not clear to all students and some students would have wanted an extrinsic motivation to participate.

One of the issues of ARGs in educational sense is that the common collaboration can decrease the efforts of an individual to learn. Karau and Kipling [12] call the tendency to expend less effort when working collectively than when working individually as *social loafing.* Stenros et al. [27] found out that several players reported about social loafing in the *Conspiracy for good* ARG. They interviewed players after the game and some of them felt that because there were so many other players solving the puzzles quicker than them, there were no point in trying to push oneself to solve them.

The most successful ARGs, in terms of number of players, have been done for promotion and marketing purposes. When thinking of educational ARGs, we should learn from successful entertainment ARGs about keeping players engaged in the game and building a large community of active players. However, depending on the educational goals of an ARG, there might be differences in ARGs that are made for education or for purely entertainment purposes. Whitton [34] states the following four ways in which ARGs for education are necessarily different from ARGs for entertainment:

1. The ARG aesthetic of *this is not a game* may not be appropriate in the context of education as students needed more support in knowing how to get started and more motivation for completing the activity.

2. Most students require a clear purpose for taking part in a game like this, whether it is linked to assessment, there is a prize or simply a clear link to being able to help them with their studies. The fact that something is a game does not appear to be a sufficient motivator for many busy students.

3. There is a tension between the niche nature of ARGs and the inclusivity strived for in formal education. There are also issues of how to make a game accessible without spoiling it for other players.

4. In games where students are asked to meet and work with others (who can not necessarily be verified as bona fide students) there are issues of online safety and duty of care by the institution.

4. CASE: "STOP TOILWORN DIAMOND"

Stop Toilworn Diamond was an alternate reality game that was held in 2013 as part of research done in Aalto University. The game lasted for 10 weeks and it was not part of any curriculum. Furthermore, the organizer of the game was not revealed to the players. Therefore, participating in the game was totally voluntary and the players were not offered any external rewards such as study credits or prizes for participating. The game was not limited to students of Aalto University, but was aimed at computer science students as well as anyone who wanted to participate.

The game was held in order to study the potential of using alternate reality games in computer science education. The research about the game is still in progress and the results are not discussed in this paper in detail. However, the preliminary results suggest that the game managed to be a learning platform for some participants. In the feedback collected after the game, participants reported that they were able to learn new things or refreshen their knowledge about various computer science concepts, such as: Boolean algebra, programming, steganography, and graphs. Furthermore, spontaneous discussions about the game in an online discussion forum support the conception of participants learning during the game. Moreover, the game was able to reach and engage also people that are not studying or working in a field related to computer science even though the content was strongly related to computer science.

The background story of the ARG encouraged players to solve puzzles in order to decode messages from the future that warned about a society-paralyzing LOLCat Apocalypse. The puzzles considered various topics of computer science but the game was not marketed as anything related to computer science education. The game was mainly played online, but it had some location based tasks that required finding hidden cards from the campus area of Aalto University.

Participants were able to communicate with the game characters with blog comments, email, and Twitter messages. There were two main sides in the game:

1. Beth Swillower was the central character who posted new puzzles to her blog. She worked for a company called Avecira Solutions, but revealed all information in her blog and asked for help in revealing the content of the mysterious messages she was able to get.
2. Avecira Solutions was claimed to be responsible for the catastrophe that would paralyze the world. The CEO of the company (Richard Exaltego) was actively seeking new talented problem solvers and persuated them to solve puzzles for him.

4.1 Puzzles and Challenges

The game included multiple puzzles of different levels of difficulty. By solving the puzzles, participants were able to reveal messages from the future and contribute to the ultimate goal of the game to stop the project Toilworn Diamond. Some of the easier puzzles included converting binary numbers to ASCII characters and some of the more difficult puzzles required finding a solution to traveling salesman problem with a given graph. Next, few example puzzles and challenges from the game are presented.

4.1.1 Example puzzle: QR codes

This puzzle was included in a series of puzzles that involved QR codes. The input of the puzzle is two pictures that are shown in Figure 1. The correct result of the puzzle is a QR code that contains an url for the next puzzle. The two pictures given as input are created from the original QR code. "Picture 1" (Figure 1a) is created by merging two adjacent pixels that are on the same row in the original QR code. If the adjacent pixels are white, the corresponding pixel in Picture 1 is also white. If the adjacent pixels are black, also the corresponding pixel in Picture 1 is black. In case the two adjacent pixels are not the same color, the corresponding pixel in Picture 1 is gray. "Picture 2" (Figure 1b) is generated similarly but the two adjacent pixels are taken from a column, not a row.

Creating the correct solution from the two pictures given as input requires problem solving and programming skills as well as some understanding of QR codes. The resulting QR code can be reformatted by reading one pixel at a time from both images. However, in case there are gray pixels in both

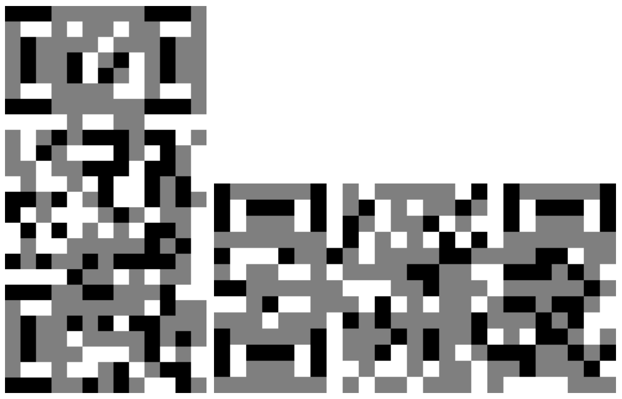

(a) Picture 1 (b) Picture 2

Figure 1: Input pictures for the QR puzzle

pictures, colors of the original pixels can not be determined. Nevertheless, the QR code has some tolerance for error, and trying different possible combinations will give the correct message in a reasonable time with these input images.

4.1.2 Location based clues

The game contained also some location based tasks. The players needed to find two cards hidden in campus area in order to reveal a message that was encoded in them. One of the cards is shown in Figure 2. Each character of the message was represented as a binary number based on the ASCII value of the character. The binary numbers could be read from the cards by putting one card on top of the other and looking at the positions of the common holes.

4.1.3 Face modelling challenge

In addition to the puzzles that needed to be solved, Avecira Solutions organized a "Face modelling challenge" in order to get people to help with their project Toilworn Diamond. The idea of the challenge was to offer ARG players an interesting open ended task that they could work on while there were not any unsolved puzzles posted at the time. However, fitting the challenge to the storyline did not work very well as the players did not have clear motivation to participate and help Avecira Solutions by submitting a solution to the challenge.

The idea of the challenge was to create a face by using 100 ellipses. The model picture of the face was given as well as a simple example solution to the challenge with the following hint: *"This is created with some kind of "Let's climb the hill" or "Hill climbing" method. Unfortunately, the person who did this didn't leave us any other information."*.

The players asked for help from a supposedly fired employee of Avecira Solutions (Hubert Acker) who was able to provide them some base code for the solution. However, the players needed to implement the actual algorithm to decide the locations and parameters of the ellipses. The face challenge got one submission and it is shown in Figure 3.

5. DISCUSSION

Alternate reality games have shown the potential to be engaging and to be able to motivate people to collaborative problem solving. On the other hand, some ARGs have had problems engaging enough people to participate in the game,

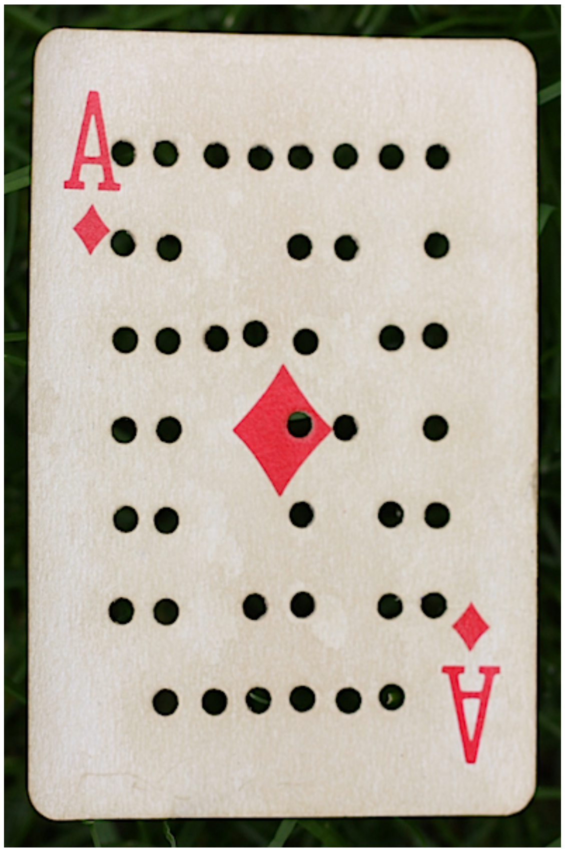


Figure 2: Cards that were hidden around campus contained a message coded with holes.

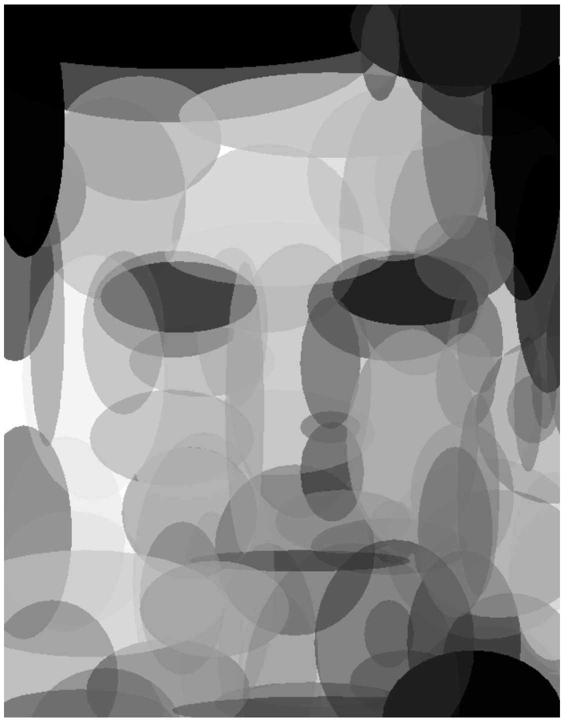


Figure 3: An ARG player's submission to the face modelling challenge.

or they have been stopped completely because of lack of players. The preliminary results of a computer science ARG called *Stop Toilworn Diamond* introduced in this paper indicate that alternate reality games could be used to teach various aspects of computer science and to engage players with CS tasks even without offering external rewards for participating.

Getting people to engage in a computer science ARG is not a trivial task. However, it is a method that could potentially have different advantages than other approaches used in computer science education. If done successfully, it could be used to offer new types of learning experiences and to widen the scope of how serious games are used to support learning computer science. However, the number of research done on educational ARGs is limited and therefore new studies are needed to understand the possibilities and pitfalls of using alternate reality games in CSE.

Ali and Shubra [1] report that enrollment in computer science programs has been facing a steady decline and that it falls short in meeting employment demands. Alternate reality games could offer a novel way to attack the issue. The preliminary results of the case study described in this paper indicate that a computer science ARG can attract people to participate that are from outside the scope of computer science education. Computer science programs could potentially be promoted by creating appealing games that involve puzzles from wide range of computer science topics. Ideally, this kind of games could break some preconceptions of computer science and showcase different CS concepts in an appealing way.

Dena [8] describes how the theory of *tiering* can be used in alternate reality games to provide different content to different audiences. She explains typical tier types in ARGs and how players can experience the same game in various different ways and levels of engagement. Alternate reality games could be used to offer participants CS tasks in different levels of abstraction and difficulty. This could enable people with different backgrounds and skills to participate together in the game and learn from each other.

Kazimoglu [13] argues that current serious games that are developed specifically for learning programming do not consider a deep game-play for developing computational thinking skills. ARGs could address the shortcoming stated by Kazimoglu. The flexible design of ARGs enables game creators to include all kind of puzzles to the game while still having the overall motivational elements described in Chapter 3. Puzzles in a CS ARG are not limited to programming tasks and they could be designed to emphasize computational thinking skills.

One characteristics of an ARG is that the puzzles and the story are often discussed online bringing people together from all over the world. Creating a CS ARG that would be marketed in different universities around the world could potentially be used to enable student networking. The game could also encourage collaboration between students in different universities by hiding some puzzle pieces in different campus areas and requiring the participants to gather and share information in order to solve puzzles.

In addition to the collaboration between students in different universities, a CS ARG could be used to enhance networking in a broader sense. By creating content that appeals to people with different competence levels of computer science, the game could attract people from the industry as well as people from high schools. Furthermore, by expanding the content of the ARG beyond computer science, the game could promote cross disciplinary networking.

Another way to use ARGs could be to let students design and implement the puzzles required for the game. Providing an opportunity to create puzzles where students can apply knowledge from their earlier studies could be a motivating way to refreshen and deepen understanding in the subjects. Moreover, creating the puzzles could be more easily integrated to a course, and possibly assessed, than just playing an ARG.

Balancing with the educational objectives and the free nature of ARGs can be problematic when thinking of ways to integrate ARGs in education. Whitton [34] listed differences between educational ARGs and entertainment ARGs that were described in Chapter 3.2. She states that the *This Is Not A Game* aesthetic may not be appropriate in educational context as students need more support in knowing how to get started and motivation to complete the game. In many educational ARGs, the TINAG aesthetic is not fully included in the game. It might be that the game is officially launched as a part of a course and it might be a mandatory part of students' studies. The situation is opposite in many successful entertainment ARGs. McGonigal, who has been designing several successful ARGs, describes her own interpretation on the TINAG aesthetic with a tiger cage metaphor in the following way: *"Perhaps the central goal of successful immersive game design is to communicate to players that a cage is in place, while making it as easy and likely as possible for the players to pretend that they don't see the cage* [19]".

While the need to offer support and motivation to participate in the game is present and emerged from the research on the ARGOSI project [34], it is important to think what are the best ways to deliver the support and still remain the aspects that might be engaging to the players. As McGonigal's tiger cage metaphor suggests, the goal is not to mislead players, but to make it easy to play along by creating a believable setting. One problem with the mysterious nature of ARGs is that it is hard to know beforehand if the content is interesting to a certain player or not. Compromising the TINAG aesthetic is one option but not the only one. Having clues early in the game about the general topic is one way to entice potential players to the game. Moreover, making it apparent that the game is aimed at certain group of people (i.e. computer science students) could also be used as a hint about the topic.

Holding to the TINAG aesthetic and the voluntary participation makes it difficult to assess the performance of a single player. Puzzle solving activities in ARGs are often collaborative what might make it impossible to assess individual player's contribution. Therefore, ARGs may not work well as a compulsory assignment that needs to be assessed.

In addition to the educational benefits to the players, computer science ARGs could also offer opportunities to research computer science education. Having a global game with a large community of players and multiple puzzles covering different topics of computer science could provide interesting data to researchers in the field of computing education.

Some benefits and possible issues of using alternate reality games in computer science education have been discussed in this paper. It is apparent that creating a successful ARG for CSE is not a trivial task as there are examples of educational ARGs that have not met the goals set to them. However, the preliminary results from a CS ARG case study introduced in this paper indicate that there is potential to use ARGs in CSE and to attract people outside the computer science community. Therefore, the following questions are raised to be discussed in the computing education community:

- Should ARGs be used in computer science education? *(Yes / No / Yes, but...)*
- In which role should ARGs be used in computer science education? *(As part of a course / as extra activity / students creating the content for an ARG)*
- What could be the benefits of a CS ARG? *(Learning / networking / promoting CS)*
- What value could ARGs bring to CSE research?

6. REFERENCES

[1] A. Ali and C. Shubra. Efforts to reverse the trend of enrollment decline in computer science programs. *The Journal of Issues in Informing Science and Information Technology*, 7:209–225, 2010.

[2] K. d. Beer and M. Holmner. The design of an alternate reality game as capstone course in a multimedia post-graduate degree. In *Proceedings of the 34th IATUL Conference*, pages 32–40. Purdue University, 2013.

[3] T. Bell, I. Witten, and M. Fellows. *Computer Science Unplugged: Off-line Activities and Games for All Ages.* Citeseer, 1998.

[4] T. M. Connolly, M. Stansfield, and T. Hainey. An application of games-based learning within software engineering. *British Journal of Educational Technology*, 38(3):416–428, 2007.

[5] T. M. Connolly, M. Stansfield, and T. Hainey. An alternate reality game for language learning: Arguing for multilingual motivation. *Computers & Education*, 57(1):1389 – 1415, 2011.

[6] R. Davies, R. Krizova, and D. Weiss. emapps. com: games and mobile technology in learning. *Innovative Approaches for Learning and Knowledge Sharing*, pages 103–110, 2006.

[7] A. Decker, S. Haydanek, and C. Egert. When objects collide: abstractions over common physics problems for capstone projects in cs1. *Journal of Computer Sciences in Colleges*, 21(2):12–18, Dec. 2005.

[8] C. Dena. Emerging participatory culture practices player-created tiers in alternate reality games. *Convergence: The International Journal of Research into New Media Technologies*, 14(1):41–57, 2008.

[9] M. Denward and A. Waern. Broadcast culture meets role-playing culture. In *The Book of Solmukohta 2008: Playground Worlds: Creating and Evaluating Experiences of Role-Playing Games.* Ropecon ry, 2008.

[10] L. Hakulinen. Using serious games in computer science education. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, Koli Calling '11, pages 83–88, New York, NY, USA, 2011. ACM.

[11] L. Hamey. Teaching secure communication protocols using a game representation. In *Proceedings of the fifth Australasian conference on Computing education*, pages 187–196. Citeseer, 2003.

[12] S. J. Karau and K. D. Williams. Social loafing: A meta-analytic review and theoretical integration. *Journal of personality and social psychology*, 65(4):681, 1993.

[13] C. Kazimoglu, M. Kiernan, L. Bacon, and L. Mackinnon. A serious game for developing computational thinking and learning introductory computer programming. *Procedia - Social and Behavioral Sciences*, 47:1991 – 1999, 2012.

[14] J. Y. Kim, J. P. Allen, and E. Lee. Alternate reality gaming. *Communications of the ACM*, 51(2):36–42, Feb. 2008.

[15] S. Kurkovsky. Mobile game development: improving student engagement and motivation in introductory computing courses. *Computer Science Education*, 23(2):138–157, 2013.

[16] T. Lee. This is not a game: Alternate reality gaming and its potential for learning. http://archive.futurelab.org.uk/resources/publications-reports-articles/web-articles/Web-Article477. [Online; accessed 18-July-2013].

[17] S. Leutenegger and J. Edgington. A games first approach to teaching introductory programming. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, SIGCSE '07, pages 115–118, New York, NY, USA, 2007. ACM.

[18] T. Malone and M. Lepper. Making learning fun: A taxonomy of intrinsic motivations for learning. *Aptitude learning and instruction*, 3(3):223–253, 1987.

[19] J. McGonigal. A real little game: The performance of belief in pervasive play. *Level Up*, 2003.

[20] J. McGonigal. This is not a game: Immersive aesthetics and collective play. In *Melbourne DAC 2003 Streamingworlds Conference Proceedings*. Citeseer, 2003.

[21] J. McGonigal. *Reality is broken: Why games make us better and how they can change the world*. Penguin Pr, 2011.

[22] D. R. Michael and S. L. Chen. *Serious Games: Games That Educate, Train, and Inform.* Muska & Lipman/Premier-Trade, 2005.

[23] A. Moseley. An alternative reality for higher education? lessons to be learned from online reality games. In *ALT-C 2008*, 2008.

[24] A. Schäfer, J. Holz, T. Leonhardt, U. Schroeder, P. Brauner, and M. Ziefle. From boring to scoring - a collaborative serious game for learning and practicing mathematical logic for computer science education. *Computer Science Education*, 23(2):87–111, 2013.

[25] S. Shabanah, J. Chen, H. Wechsler, D. Carr, and E. Wegman. Designing computer games to teach algorithms. In *2010 Seventh International Conference on Information Technology*, pages 1119–1126. IEEE, 2010.

[26] K. Squire. Cultural framing of computer/video games. *Game studies*, 2(1):90, 2002.

[27] J. Stenros, J. Holopainen, A. Waern, M. Montola, and E. Ollila. Narrative friction in alternate reality games: Design insights from conspiracy for good. In *Proceedings of DiGRA 2011 Conference: Think Design Play, Utrecht, The Netherlands. DiGRA*, 2011.

[28] E. Sylvan, J. Larsen, J. Asbell-Clarke, and T. Edwards. The canary's not dead, it's just resting: The productive failure of a science-based augmented-reality game. In *Proceedings of Games+Learning+Society 8.0*, GLS 8.0, pages 31–37. ETC Press, 2012.

[29] D. Szulborski. *This is not a game: A guide to alternate reality gaming*. New-Fiction Publishing, 2005.

[30] Unfiction, Alternate Reality Gaming. http://www.unfiction.com/history/. [Online; accessed 10-July-2013].

[31] A. Waern and M. Denward. On the edge of reality: Reality fiction in 'sanningen om marika'. *Breaking New Ground: Innovation in Games, Play, Practice and Theory. Proceedings of DiGRA*, 2009.

[32] S. Wallace, R. McCartney, and I. Russell. Games and machine learning: a powerful combination in an artificial intelligence course. *Computer Science Education*, 20(1):17–36, 2010.

[33] N. Whitton. Alternate reality games for developing student autonomy and peer learning. In *Proceedings of the LICK 2008 Symposium*, pages 32–40. Napier University / TESEP, 2008.

[34] N. Whitton. Alternate Reality Games for Orientation, Socialisation and Induction (ARGOSI), 2009.

[35] N. Whitton. Encouraging engagement in game-based learning. *International Journal of Game-Based Learning (IJGBL)*, 1(1):75–84, 2011.

How to Study Programming on Mobile Touch Devices – Interactive Python Code Exercises

Petri Ihantola, Juha Helminen, and Ville Karavirta
Department of Computer Science and Engineering
Aalto University
Finland
petri.ihantola@aalto.fi, juha.helminen@aalto.fi, ville@villekaravirta.com

ABSTRACT

Scaffolded learning tasks where programs are constructed from predefined code fragments by dragging and dropping them (i.e. Parsons problems) are well suited to mobile touch devices, but quite limited in their applicability. They do not adequately cater for different approaches to constructing a program. After studying solutions to automatically assessed programming exercises, we found out that many different solutions are composed of a relatively small set of mutually similar code lines. Thus, they can be constructed by using the drag-and-drop approach if only it was possible to edit some small parts of the predefined fragments. Based on this, we have designed and implemented a new exercise type for mobile devices that builds on Parsons problems and falls somewhere between their strict scaffolding and full-blown coding exercises. In these exercises, we can gradually fade the scaffolding and allow programs to be constructed more freely so as not to restrict thinking and limit creativity too much while still making sure we are able to deploy them to small-screen mobile devices. In addition to the new concept and the related implementation, we discuss other possibilities of how programming could be practiced on mobile devices.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education—*Computer-assisted instruction (CAI), Distance learning*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Human Factors

Keywords

Programming, Mobile Touch Devices, Python, Teaching, Learning, Mobile Learning, mLearning, Parsons Puzzle, Parsons Problem

ACM 978-1-4503-2482-3/13/11 $15.00
http://dx.doi.org/10.1145/2526968.2526974

1. INTRODUCTION

Today, mobile devices are ubiquitous and provide an attractive platform for consuming all kinds of interactive material on-the-go. Indeed, these devices also have great potential as an environment for learning, regardless of time and place. The challenge is how to effectively create or adapt content to meet the requirements and overcome the restrictions of this quite a different learning space. Learning sessions may be very short in time and sporadic in frequency. Thus, the learning applications must be able to deliver brief self-contained learning tasks.

How can mobile devices be used to learn programming? Such devices can be used for reading tutorials – perhaps accompanied with some visualizations – but based on previous research it seems likely that a more active role than mere viewing would lead to better results [4]. Indeed, some programming-related online tutorials utilize in-browser programming environments where users can experiment without installing anything extra on their computers (e.g. Runestone Interactive Python [11] and Learnpython[1]). Although excellent in desktop environments, the usability of these is lacking on mobile touch devices where the screen space is limited and where typing is more challenging as there is no physical keyboard.

If typing code with touch devices is challenging, then how can you practice programming? For Windows Phone users (and as an HTML5 app for many other platforms) there is TouchDevelop [14] – a programming language and environment designed explicitly for these devices and where much of the code is created by tapping through menus. A different, but related approach would be to use environments where programs are constructed visually by dragging and dropping blocks of code. Scratch [9] is a well-known example of this. So-called Parsons problems (or puzzles), program construction exercises where the task is to select and arrange code fragments in order to compose a program that meets some set requirements, are another example [12]. Here, a code fragment is typically a single line of code but may comprise multiple lines as well. If there is no extra code and blocks cannot be defined on-the-fly, the problem reduces to that of reordering the code. Most blocks languages as well as TouchDevelop support only some special programming languages designed together with the system whereas Parsons problems can be used to practice any programming language (e.g. Java or Python).

The existing exercises that are available on mobile touch devices and deal with programming topics, like the program

[1]`http://www.learnpython.org/`

construction of Parsons problems, are fairly limited in terms of what kinds of learning tasks and goals are feasible. For example, the existing implementations of Parsons problems do not allow any modifications to the given code fragments and thus strongly limit the way programs can be built, and assignments designed. Indeed, it would be useful to have something that falls between text as input and the immutable blocks of pre-written code – something that is more akin to the core task of designing and creating programs instead of merely reading and answering questions about them, or attempting to piece one together somewhat like a puzzle. Influenced by observations from an analysis of students' submissions to programming exercises, we designed a new type of exercise aimed at mobile touch devices. Our approach bears resemblance to existing work on block-oriented graphical programming environments but intentionally skews closer to code as text instead of visual elements. This is in order for the practice to better benefit and the skills acquired to better translate to the actual task of writing code in a widely-used general-purpose programming language (Python).

In this paper, we present the design and implementation of a new type of exercise as an extension to the MobileParsons [6] learning application for practicing Python programming skills on mobile touch devices. Self-study is facilitated by the availability of immediate automatic feedback. Furthermore, we also discuss existing solutions and other alternatives to (learning) programming on mobile devices.

2. DESIGN AND IMPLEMENTATION

Different solutions to a programming assignment can be the result of using different language constructs. On the other hand, very similar lines may also lend themselves to different approaches, as illustrated in Figure 1. The types of lines in the figure can be used to construct many different kinds of attempts at solving the problem of finding the largest value, both correct and incorrect. More precisely, the solutions in the figure can be constructed from four line types – the signature line (`def find_largest(a,b,c):`), if statements (`if VAR CMP-OP VAR:`), else statements and return statements (`return VAR`), where VAR is either a, b, or c and CMP-OP is <, <=, >=, or >.

To get a better understanding of this phenomenon, we examined whether students' solutions to small programming exercises could indeed be constructed by selecting lines from a relatively small pool of lines – the pool of lines being different for each exercise. To investigate this, we analyzed all submissions to, in total, 11 small automatically assessed introductory level Python programming exercises from two different programming courses. The courses were the first introductory programming course (CS1) and Web Software Development (WSD) at Aalto University, Finland. CS1 and WSD are both large courses with ca. 600 and 150 students, respectively. In selecting the exercises we considered that we would like at least some of the solutions to fit on the screen of a typical touch device when using the MobileParsons environment. This implies programs of around 10 lines for handheld devices and 20–25 for tablets.

We only considered the last correct submission for each student and searched for sets of common codelines from which as many students' solutions as possible could be constructed with the idea in mind that then these lines could be the building blocks in the respective Parsons problem. For this, we went through students' solutions searching for similar lines. We ignored all empty lines and comments, and lumped together all variable names and string literals. Indeed, with Parsons problems, we would be able to use some canonicalized names for all the occurring variables. Moreover, string literals could also all be equated because they did not affect the adherence to the given specifications in any of the exercises but were either debugging code or strings whose actual phrasing did not matter in terms of the functionality. After canonicalizing all the lines, we got the set of all lines that occurred in students' solutions – one set for each exercise.

```
def find_largest(a,b,c):
  if a >= b:
    if a >= c:
      return a
  if b >= c:
    return b
  else:
    return c
```

```
def find_largest(a,b,c):
  if b >= a:
    if b >= c:
      return b
  if a >= c:
    return a
  else:
    return c
```

```
def find_largest(a,b,c):
  if b < a:
    if c < a:
      return a
  if c > b:
    return c
  else:
    return b
```

Figure 1: Different solutions, all based on nested if-else constructs, to the programming assignment where the task is to find the largest among the three arguments given.

In some cases, we were able to identify groups of often used lines that even after the initial canonicalization were only slightly different from each other. If in addition to positioning lines, a user was allowed to reuse and edit some small parts of the lines, a smaller set of lines could allow a much larger number of different types of solutions to be constructed. After that, we processed all the lines so that all identifiers, boolean literals (i.e. `True`, `False`), number literals (e.g. `0.1`, `1`, and `3`), mathematical operators (i.e. `+`, `-`, `*`, `/`, and `//`), comparison operators (i.e. `<`, `<=`, `==`, `>=`, `>`, and `!=`), and logical binary operators (i.e. `and` and `or`) were lumped together under an umbrella term like `COMP-OP` for a comparison operator.

After the canonicalization, we applied a heuristic search to find subsets of (editable) lines from where one could select lines to fully cover as many students' solutions as possible.

For each exercise, we selected the set of lines whose relative expressive power was best by finding the one where the number of lines divided by the number of solutions covered was at its minimum. These are reported in Table 1. *Set size* column in the table describes how many lines were needed to cover n solutions, where n is given in column *coverage (c-all)*. Although the number of fully covered solutions (line coverage c-all in the table) is relatively small, it is better than where we can get with traditional Parsons problems (line coverage c-id-str in the table). What this means is that having around 11 partly editable lines would allow 46 percent of students' solutions to assignment CS1 12 to be constructed while this number of non-editable lines could cover only 15 percent of the solutions.

Table 1: The number of solutions that could be fully covered by N prototype lines for both canonicalizations. N is selected so that the fraction of set size and coverage with all canonicalizations is at its minimum.

	set size (`c-all`)	coverage (`c-all`)	coverage (`c-id-str`)
CS1 11	10	170 (36%)	140 (29%)
CS1 12	11	209 (46%)	67 (15%)
CS1 21	23	79 (17%)	5 (1%)
CS1 22	26	219 (47%)	54 (12%)
CS1 23	27	206 (49%)	21 (5%)
CS1 31	21	188 (40%)	49 (10%)
CS1 32	19	241 (52%)	86 (19%)
CS1 33	24	83 (20%)	20 (5%)
WSD1	10	46 (31%)	46 (31%)
WSD2	16	27 (18%)	21 (14%)
WSD3	12	14 (11%)	13 (10%)

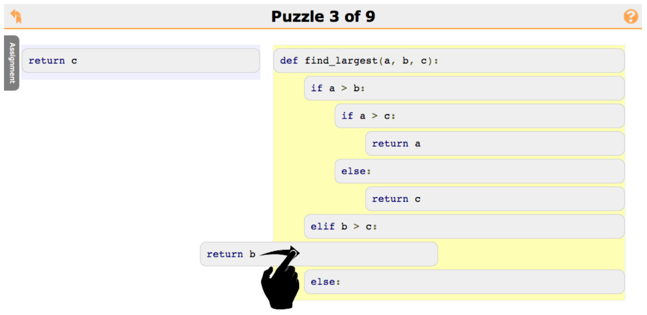
Implementation

Influenced by our observations, we decided to implement a new type of Python exercise that allows limited editing of lines by providing parts where you can toggle (i.e. click or tap) between alternatives. We implemented this on top of js-parsons [5], or more precisely its mobile fork called MobileParsons [6].

In the original MobileParsons, students are to use some of the code given on the left side (in landscape mode) to build a program on the right without forgetting appropriate indentation which is significant in Python. Lines are positioned by drag-and-dropping. An example of an exercise in MobileParsons is shown in Figure 2. The assignment description is available in a tab on the left. Similarly, after requesting feedback, another tab appears on the right which shows the previously requested feedback.

In the underlying js-parsons, there are two different types of feedback: line-based and execution-based [2]. The type of feedback is defined when the exercise is created. Line-based feedback indicates the lines that should be repositioned. For this feedback to be possible, there needs to only exist a single correct arrangement of the given lines. In execution-based feedback, the code is executed and run against predefined tests. Feedback is given to the student as failed/passed assertions. Python code is executed in-browser using the Skulpt library[2], a JavaScript implementation of Python. Feedback is given whenever requested by the student. However, in order

[2]`http://www.skulpt.org/`

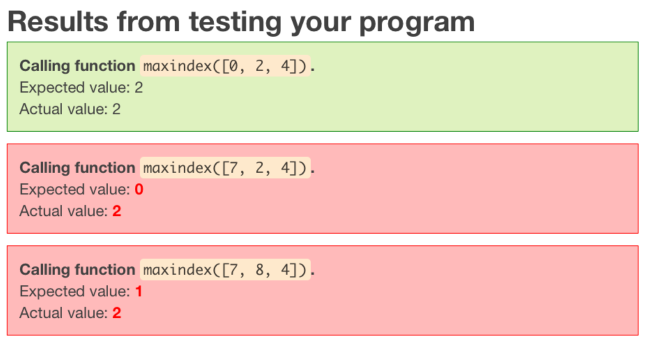


```
return c

def find_largest(a, b, c):
    if a > b:
        if a > c:
            return a
        else:
            return c
    elif b > c:
return b
    else:
```

Figure 2: Example of an assignment in MobileParsons. Line `return b` is being inserted into the program.

to discourage trial-and-error behavior, feedback is automatically disabled for a period of time if used too frequently. Previously, no execution-based feedback was available in MobileParsons but this feature was added as part of the extension we implemented. Figure 3 shows what this feedback looks like.



Results from testing your program

Calling function `maxindex([0, 2, 4])`.
Expected value: 2
Actual value: 2

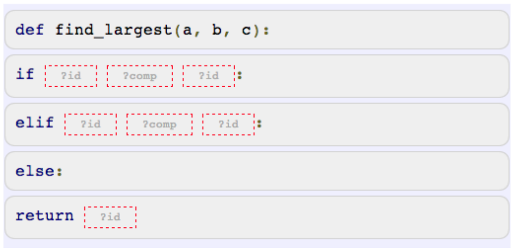
Calling function `maxindex([7, 2, 4])`.
Expected value: 0
Actual value: 2

Calling function `maxindex([7, 8, 4])`.
Expected value: 1
Actual value: 2

Figure 3: Example of execution-based feedback in MobileParsons.

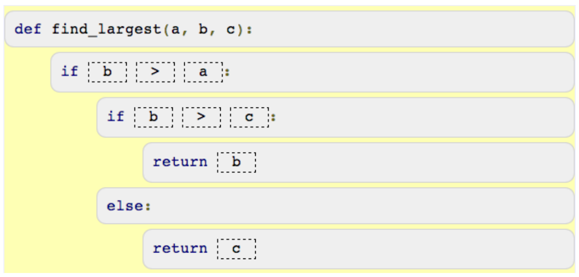
The main difference between MobileParsons and the new type of exercise is that in the latter, the given code fragments can include parts where the learner needs to select a piece of code from a set of options in order to complete the codeline. An example of a possible set of editable lines for the previously described task of finding the largest item of the three arguments (Figures 1 and 2) is shown in Figure 4. Initially, the blocks show only the type of the missing content. This is in order to help learners think about the possible solutions before starting to change the pieces. We did not want to make one of the options show by default, as we felt that would too easily fixate learners on that value. However, when an exercise is created, a default value can be specified if need be. Learners interact with the changeable parts of the code by tapping (or clicking) them. That changes the piece to the next option. If the prototype lines can be duplicated, this allows constructing larger solutions with the given selection of prototype lines. For example, the partial solution shown in Figure 5 could be constructed from the lines in Figure 4. A complete example of a partially solved exercise in the mobile application is shown in Figure 6.

The options for a changeable piece of code can be specified



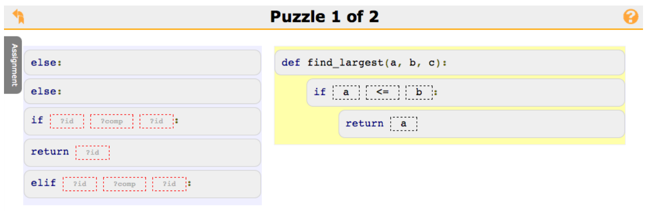
```
def find_largest(a, b, c):
if ?id ?comp ?id:
elif ?id ?comp ?id:
else:
return ?id
```

Figure 4: The prototype lines sufficient to solve the `find_largest` task in Figure 1. Whereas the task in Figure 1 has only one correct ordering of the lines, these prototypes allow constructing multiple correct solutions.



```
def find_largest(a, b, c):
    if b > a:
        if b > c:
            return b
        else:
            return c
```

Figure 5: A partial solution to the example exercise.



```
else:
else:
if ?id ?comp ?id:
return ?id
elif ?id ?comp ?id:
```

```
def find_largest(a, b, c):
    if a <= b:
        return a
```

Figure 6: A complete example of an exercise with toggleable parts in MobileParsons.

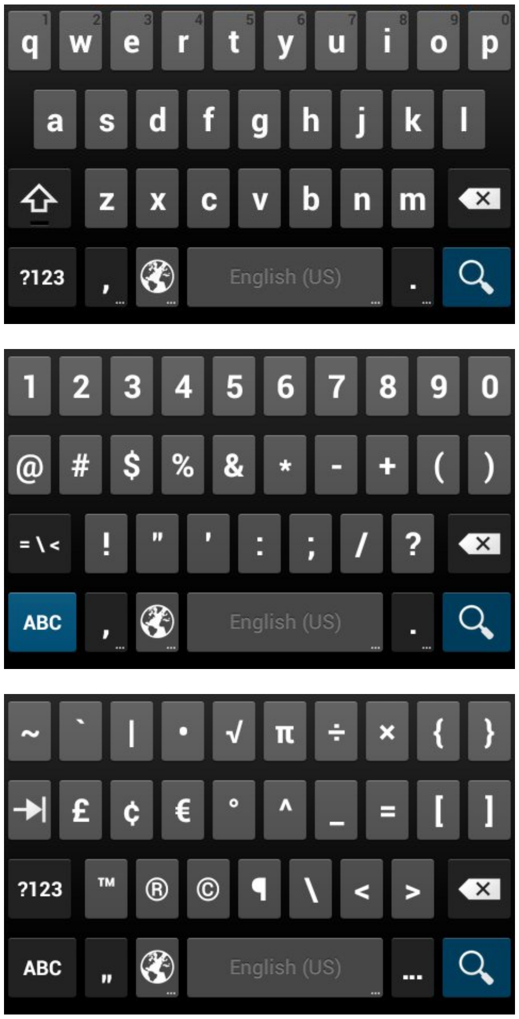


Figure 7: Android default keyboard layout with three different layout windows. Typing, for example, `foo(a[1])` requires switching between all of these layout screens.

as a list when creating the exercise. Alternatively, only the type of the piece can be specified in which case the options are determined by this. The supported types are boolean (True, False), comparison operator ($<, >, <=, >=, ==, != $), math operator ($+, -, *, /$), logical binary operator (and, or), and numeric range. The numeric range is specified as in the form field type specification of HTML5[3], that is, with minimum and maximum values as well as a step used to increase the value.

3. RELATED TOOLS

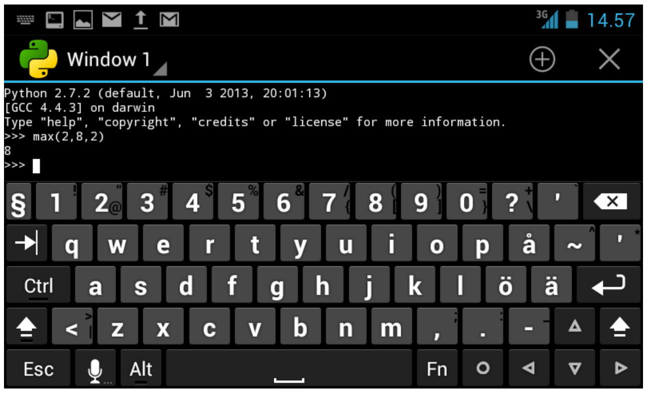
In addition to many mobile applications providing textbook-like static content on programming topics, some also provide interactive elements such as multiple choice questions, Parsons problems, and IDEs designed especially for handheld devices. In this section, we present examples of these. The selected examples are skewed towards the Android and iOS platforms but, in general, there are similar tools available on other platforms as well.

3.1 Tools for Composing Programs on Touch Devices

One reason why typing source code with the on-screen keyboards of touch devices is difficult, is that special characters such as curly braces and brackets, often used in programming, are hard to access. Thus, several programmer-friendly soft keyboards have been created. The obvious trade-off is between having a smaller number of larger keys so that they are easier to hit and having a larger number of smaller keys in order to avoid switching between multiple keyboard screens while typing. The difference is highlighted in Figures 7 and 8 where the former presents the default keyboard layouts on Android and the latter is the layout provided by an app called Hacker's Keyboard[4]. This keyboard is similar to traditional

[3]http://drafts.htmlwg.org/html/master/forms.html

[4]https://play.google.com/store/apps/details?id=org.pocketworkstation.pckeyboard



```
Python 2.7.2 (default, Jun  3 2013, 20:01:13)
[GCC 4.4.3] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> max(2,8,2)
8
>>> 
```

Figure 8: Hacker's Keyboard soft keyboard used in QPython shell on Android.

keyboards and allows typing many special characters by first tapping shift and then some other key. Another example of a keyboard for typing programs – that does not even require tapping two keys to get the special characters – is shown in Figure 9 from the Textastic[5] text editor for iPad/iPhone. In Textastic, tapping on the keyboard buttons on top of the traditional keyboard adds the character in the middle of the button, whereas swiping the button towards its corners adds the character displayed in that corner of the button.

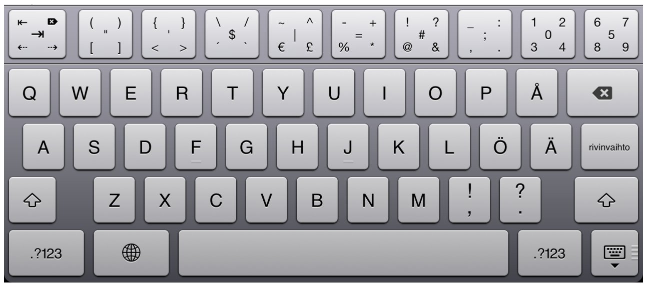
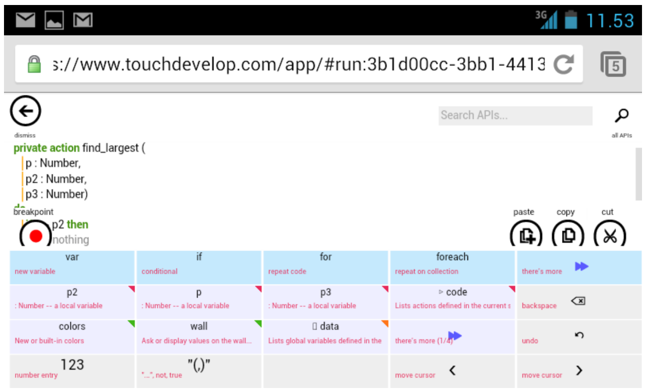


Figure 9: Keyboard in the Textastic application on an iPad.

Features like *predictive typing* [8] and *auto-complete* make typing software on mobile touch devices even easier but they still do not utilize new interaction methods enabled by these devices. Despite recent interest (e.g. [3, 10, 13]), most publicly available mobile programming tools lack *natural interaction* [15] mechanisms (e.g. use of gestures). Although TouchDevelop (Figure 10), makes creating small programs on mobile devices easier, as they are are created by tapping through dynamic menus on the screen, it does not support, let us say, Python.

3.2 Tools for Practicing Programming on Touch Devices

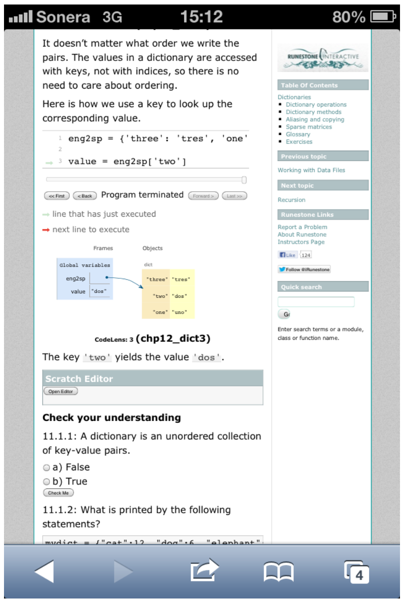
On the other hand, when practicing programming, being able to execute and experiment with the code is equally important as typing code. Correspondingly, there exist many mobile IDEs, interpreters and compilers for mobile devices. QPython[6], just like many other similar tools, provides an editor and an interactive console separately as illustrated in Figure 8.



```
private action find_largest (
  p : Number,
  p2 : Number,
  p3 : Number)
  p2 then
  nothing
```

Figure 10: A screenshot from HTML5 version of TouchDevelop on Android.

However, this and other applications like it are designed for programming in general – not for learning or teaching it. Aside from native applications, there is still also the Runestone Interactive Python [11] that is an interactive web-based book for learning Python. As shown in Figure 11, the interactive content works on mobile devices as well. Learnpython is another similar website. It can be used with tablets but the usability with phone-sized devices is limited.



It doesn't matter what order we write the pairs. The values in a dictionary are accessed with keys, not with indices, so there is no need to care about ordering.

Here is how we use a key to look up the corresponding value.

```
eng2sp = {'three': 'tres', 'one'

value = eng2sp['two']
```

Program terminated

line that has just executed

next line to execute

CodeLens: 3 (chp12_dict3)

The key `'two'` yields the value `'dos'`.

Scratch Editor

Check your understanding

11.1.1: A dictionary is an unordered collection of key-value pairs.

a) False

b) True

11.1.2: What is printed by the following statements?

```
mydict = {"cat":12, "dog":6, "elephant"
```

Figure 11: Example of textual content and a program visualization in Interactive Python on an iPhone.

Although we have built our extension to an environment designed for Parsons problems, the work we have done can

[5]`https://itunes.apple.com/app/id383577124`

[6]`https://play.google.com/store/apps/details?id=com.hipipal.qpyplus`

also be compared to blocks languages such as the pioneering BLOX [1] or later Scratch[7], Snap[8], Blockly, and others. Some of the latest blocks language environments are browser-based and thus can be used even on tablets. The screen layout of all the environments we have tried, however, is such that the tools are in practice unusable with phone-sized devices (Figure 12 shows an example of Snap on a tablet device). In blocks languages, the arrangement of code fragments is supported visually by the jigsaw metaphor where pieces can be linked together only in certain ways. Languages often allow small edits to the pieces just like switching through options in our implementation. Most blocks languages, being fully expressive programming languages, support more versatile editing of blocks than what is currently possible in our implementation. For example, it may be possible to insert an editable block inside some other blocks.

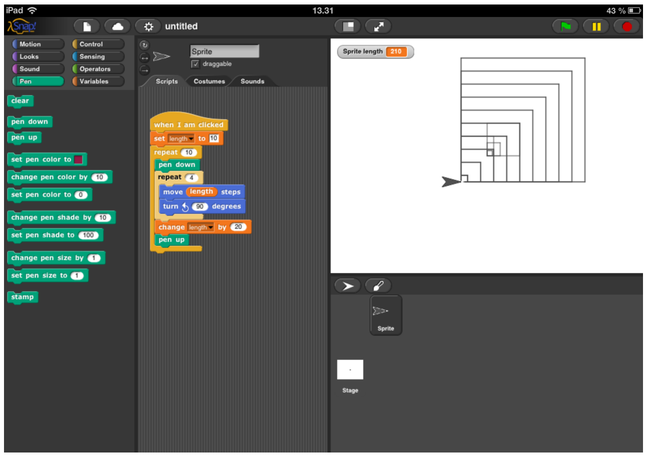

```
when I am clicked
set length to 10
repeat 10
  pen down
  repeat 4
    move length steps
    turn 90 degrees
  change length by 20
  pen up
```

Figure 12: Example of a program in Snap.

Our work falls somewhere between simple Parsons problem systems and more complex blocks languages. Both Parsons problems and blocks languages have been used to lower the barrier to programming [7, 12]. Our primary goal, however, has been to bring Python programming tasks with automatic feedback to mobile touch devices. Nonetheless, the system we have developed can also be used with browsers on desktop machines. Closest to what we had on our mind, and what was already available on a mobile platform, was the SingPath[9] mobile environment for simple Parsons problems. An example of a problem in SingPath is shown in Figure 13. It should be noted that, unlike js-parsons and MobileParsons, SingPath automatically indents the added lines correctly.

In addition to the program construction environments, there exist some quiz applications that deal with programming, for example, the iOS applications Java Quiz[10] and Quiz&Learn Python[11]. Java Quiz asks questions both about program behavior and different constructs of the language.

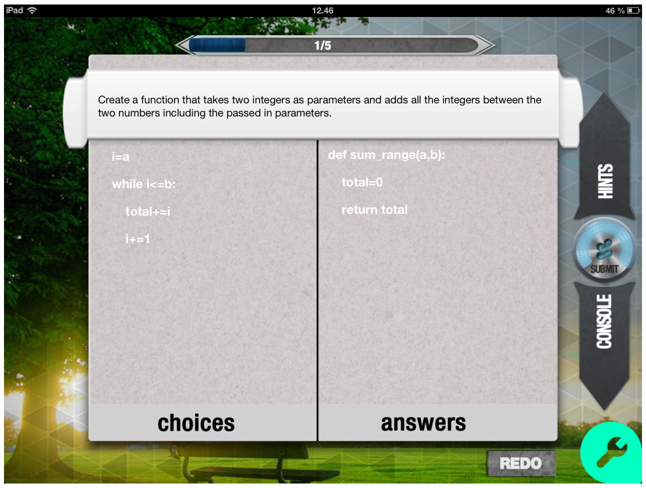
[7]http://scratch.mit.edu/

[8]http://snap.berkeley.edu/

[9]https://itunes.apple.com/us/app/singpath-mobile/id567470737

[10]https://itunes.apple.com/app/java-quiz/id464249097

[11]http://www.bythemark.com/products/quiz-learn-python/


Create a function that takes two integers as parameters and adds all the integers between the two numbers including the passed in parameters.

choices:
```
i=a
while i<=b:
    total+=i
    i+=1
```
answers:
```
def sum_range(a,b):
    total=0
    return total
```

Figure 13: Example of a problem in SingPath.

Questions in Quiz&Learn Python are all about the behavior of short programs. In addition to showing the correct answers, it gives the learner the possibility to step through the program line-by-line and inspect its behavior.

4. DISCUSSION

There are several aspects of the proposed type of exercise as well as the whole concept of learning programming on mobile devices that we would like to see discussed by computing education researchers and practitioners. In the following, we raise these questions, as well as provide our own opinions as a starting point for the discussion.

> Is learning programming on mobile touch devices worth pursuing? If yes, what kinds of tools are available (or missing) to practice programming on mobile touch devices? How should mobile learning tools be integrated into courses? Is it enough for them to be available for self-study?

Despite some pioneering work to edit source code through gestures [13], existing programming tools for mobile touch devices seem to be based on static material, multiple choice questionnaires, visual (blocks) languages, Parsons problems, or exporting IDE concepts and environments directly from desktop environments to the mobile context. Visual editing of textual source, where our work aims at, seems to be missing. In addition, combinations of static and interactive content are also missing. Some browser-based solutions like learnpython can be used with tablets but are quite difficult to deal with on phone-sized devices. This kind of combination of easy to read tutorials accompanied with small exercises is something that we would like to see on mobile devices in the future. However, because of sporadic and brief usage sessions, editing the source code should be faster than where typing with soft keyboards appears to get us.

> Does the proposed type of exercise provide benefits compared to existing systems?

Although our code exercises may allow some freedom while also making use of interaction methods natural to touch devices – dragging, dropping,and tapping – the approach still significantly restricts learners' program implementation options. Some of this can be educationally justified. For example, when teaching if-else constructs, it is better to guide students towards constructing program like in Figure 1, instead of producing quick one-liners as in Figure 14. When teaching some specific concepts, we would like to provide building blocks exactly with those in mind. However, in some cases the blocks languages approach of being able to insert blocks inside other elements would help us to produce exercises that would better match the different solutions students would create without the restrictions of our system.

```
def find_largest(a, b, c):
    return max(a, b, c)
```

Figure 14: An alternative solution to the programming assignment where the task is to find the largest the three arguments given.

Indeed, we made the design choice of requiring Python as the language in our mobile code exercises. While Scratch, for example, would be an interesting approach to learning programming concepts on mobile devices, in order to eliminate the extra cognitive load of switching between languages, we preferred an environment that is based on the same language used in our programming courses anyhow. For our needs, the blocks language environment would thus have to support constructing programs in Python. We chose not to make use of the jigsaw metaphor known from blocks languages. On the one hand, we wanted to minimize the effort of grasping additional visual metaphors beyond actual Python code and programs and, on the other hand, this approach allows students to create syntactical errors. Adopting the jigsaw approach to our work would prevent students from doing and learning from these mistakes. Instead, we give similar automated feedback that a Python interpreter would when a student tries to execute the erroneous code.

Finally, in our analysis of students' submissions, we focused on the correct solutions in order to examine the variance in correct implementations. We could also investigate typical errors by analyzing all submissions and then consider adding respective erroneous lines in the exercises. Such lines in Parsons problems are called *distractors*. Allowing students to modify lines, as we have, will in practice create many distractors and therefore make exercises more difficult. Indeed, if the goal is to allow different correct solutions, instead of merely making the exercises more difficult, it should be carefully considered which modifications are allowed. Informed decision by the instructor is always a choice. Another is mining the submissions of automatically assessed programming exercises, as we did in order to ensure the proposed exercise approach is meaningful. However, in future research, it might be better to exclude the solutions of lesser quality instead of using all the submissions. Thus, although original Parsons problems force the students to reproduce the teacher's answer, more research is needed to understand how toggleable Parsons implemented in this work affect and how the toggleable values in different exercises should be selected to best support learning.

Further research is needed to verify our assumption that controlled scaffolding, that is the ability to fully tailor from which kinds of building blocks students construct their programs, would help teachers to create good exercises. This might even be useful on blocks languages, where on Blockly it is also implemented.

5. CONCLUSIONS

In this paper, we have discussed different existing approaches to providing learning content for programming on mobile touch devices and raised some questions about possible future directions in this area of research. To address the needs for programming exercises on mobile touch devices and the shortcomings of existing learning environments on such platforms, we have designed and implemented a new type of exercise dealing with programming. It has been implemented as an extension to the existing MobileParsons application that provides exercises where students drag and drop given code fragments to construct a program. The key idea is to try and allow more choice in how programs are constructed by allowing some choice in and modifications to the given code through toggleable elements in the code fragments, such as, designating a part of a line as a comparison operator which would then allow switching through the different alternatives. The exercises may be designed both to less restrict the choice of implementation for the solution and to be more complex and challenging. A core feature, which enables effective self-study, is the availability of immediate automatic feedback. This feedback is based on running unit tests on the constructed program as previously only available in the desktop equivalent of MobileParsons.

6. REFERENCES

[1] E. P. Glinert. Towards 'second generation' interactive, graphical programming environments. In *Proceedings of the 1986 IEEE Computer Society Workshop on Visual Languages (VL'86)*, pages 61–70, 1986.

[2] J. Helminen, P. Ihantola, V. Karavirta, and S. Alaoutinen. How do students solve parsons programming problems? – execution-based vs. line-based feedback. In *Learning and Teaching in Computing and Engineering (LaTiCE), 2013*, pages 55–61, 2013.

[3] M. Hesenius, C. Orozco Medina, and D. Herzberg. Touching factor: Software development on tablets. In T. Gschwind, F. Paoli, V. Gruhn, and M. Book, editors, *Software Composition*, volume 7306 of *Lecture Notes in Computer Science*, pages 148–161. Springer Berlin Heidelberg, 2012.

[4] C. Hundhausen. A meta-study of software visualization effectiveness. *Journal of Visual Languages and Computing*, 13:259–290, 1996.

[5] P. Ihantola and V. Karavirta. Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education: Innovations in Practice*, 10:1–14, 2011.

[6] V. Karavirta, J. Helminen, and P. Ihantola. A mobile learning application for parsons problems with automatic feedback. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 11–18, New York, NY, USA, 2012. ACM.

[7] C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming

environments and languages for novice programmers. *ACM Comput. Surv.*, 37(2):83–137, June 2005.

[8] I. S. MacKenzie and R. W. Soukoreff. Text entry for mobile computing: Models and methods,theory and practice. *Human–Computer Interaction*, 17(2-3):147–198, 2002.

[9] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *Trans. Comput. Educ.*, 10(4):16:1–16:15, Nov. 2010.

[10] S. McDirmid. Coding at the speed of touch. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, ONWARD '11, pages 61–76, New York, NY, USA, 2011. ACM.

[11] B. N. Miller and D. L. Ranum. Beyond PDF and ePub: toward an interactive textbook. In *ITiCSE'12: Proceedings of the 17th annual joint conference on Innovation and technology in computer science education*, pages 150–155, 2012.

[12] D. Parsons and P. Haden. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *ACE '06: Proceedings of the 8th Austalian conference on Computing education*, pages 157–163, 2006.

[13] F. Raab, C. Wolff, and F. Echtler. Refactorpad: editing source code on touchscreens. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '13, pages 223–228, New York, NY, USA, 2013. ACM.

[14] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich. Touchdevelop: programming cloud-connected mobile devices via touchscreen. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software (ONWARD '11)*, pages 49–60, 2011.

[15] D. Wigdor and D. Wixon. *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.

Getting to know computer science freshmen

Päivi Kinnunen
Department of Computer Science and Engineering
School of Science
Aalto University, Finland
paivi.kinnunen@aalto.fi

Maija Marttila-Kontio
School of Computing
University of Eastern Finland
Finland
maija.marttila@uef.fi

Erkki Pesonen
School of Computing
University of Eastern Finland
Finland
erkki.t.pesonen@uef.fi

ABSTRACT

This paper reports an ongoing study on CS freshmen at two universities in Finland. The goal of this report is to describe what kind of students decide to study computer science; what are students' programming background, expectations and perceptions regarding their studies and future work life. We collected data from 190 students from two universities at the beginning of the first study year. The results draw a picture of students' who have mainly positive and trusting perceptions of the IT field and who have high expectations of their success at their studies. We will discuss the results, especially the students' heterogeneous programming background and some gender differences, in more detail and consider how we could and should take those into consideration in our curriculum and teaching.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education

General Terms

Human Factors

Keywords

Computer science majors, freshmen, previous programming experience

1. INTRODUCTION

As programming teachers we have observed that part of the students have serious difficulties in learning programming, which lead to high dropout rate at CS courses during the first study year. Previous studies highlight that we are not alone with our problems [see e.g. 11, 12, 15, 19]. Our aim is to develop our curriculum and teaching systematically. Some of the big research questions we had in mind when we started our currently ongoing research project included: How to best scaffold first year computer science students to acquire the knowledge and skills they need at the end of the first study year in order to be able to successfully continue studies? Which factors are salient in computer science students' success/struggles during the first years of university studies? Our questions were motivated both by our own observations as teachers and literature regarding the difficulties students seem to encounter during the first years of university studies.

The big research questions we started with are rather extensive and most likely do not have a single and simple answers. This paper reports on our first step of this ongoing study. We start our journey by dividing the big question into smaller parts. In this paper we concentrate on getting to know our students better; who are they, what influenced their choice of major, and what interests them. We hope that knowing who our students are guides us on how we should acknowledge students' differing background and interests in our curriculum and teaching practices.

The literature gives us some ideas of what to ask and what to expect our students to answer. For instance, a short overview of literature on occupational choices reveals several intervening factors. Lent et al. [6] lists factors like interest, exposure to work-related activities, work conditions, ability considerations, experiences at leisure time, and finally also career counseling as playing a part in career choices in general. Other researchers have studied particularly engineering and computer science students' occupational choices. While the results corroborate in many way more general occupational choice theories, a more detailed insight into engineering and computer science students reveals new factors. Schulte and Knobelsdorf [13] studied computing experiences of CS students and psychology students in order to determine factors that make some students to choose CS as their future career and others to shy away. The factors that seemed to have affected the career choice were interest in computers accompanied with expansive and active learning habits (in computing context), which lead to positive experiences with computers. The results of Lang [5] and Paloheimo [10] add yet another factors. Studies concentrate on female students' choice to study computer science or some engineering field. The results reveal that aspects like personal connection to and encouragement by a relative, friend working in the field, or a teacher affected students' choice considerably. In addition, happenstances also played a role in the decision.

As a teacher and curriculum developer it is interesting to know about factors relating to why freshmen chose computer science major. Another background factor that we might be able to take into account in our teaching is students' previous programming experience. We have designed this study foremost to provide us such knowledge of students that we might take into account in our teaching. Below we have formulated following research questions, which we aim to answer in this paper are:

1. What kinds of students decide to study computer science at our universities?

1.1 What kind of previous programming experience do students have?

Koli Calling '13, November 14 - 17 2013, Koli, Finland
Copyright 2013 ACM 978-1-4503-2482-3/13/11<$15.00.

1.2 What kind of perceptions of and interests towards information technology jobs do students have?

1.3 Which factors had an influence in students choosing computer science as a major?

1.4 What kind of expectations do students have concerning their ability to succeed in their studies?

2. METHODOLOGY

The aim of the study is to describe on a general level who our students are. We were interested in getting to know the whole cohort (over hundred students) and therefore we decided that sending out web based questionnaire would be the best, and efficient, way to get the information we are interested in at this point of our larger research project. This method also provided us with the type of data that answered our research questions and guided us further in our ongoing research project.

The authors of this paper constructed a questionnaire (Appendix) where we inquired, for instance, students' academic history, previous programming experience, and respondents' perceptions of their ability to study successfully, and who/what influenced their choice of major. The questions were motivated by the literature on occupational choice and our own observations and interest concerning the freshmen students. Majority of the questions were closed ended questions with Likert-scale or multiple answering options.

We collected data in fall 2011 and 2012 at the University of Eastern Finland and in fall 2012 we collected additional data from Aalto University. The reason to include the second university to our data pool related to our future research interests concerning factors relating to students' success/struggles. It is important to be able to make a difference between factors that might be local to one university and more general factors.

Both of the universities in our study are large research-intensive universities (15 000 – 20 000 students), which grant master's degrees and doctoral degrees in various fields. Students choose their major as they apply for the university. The questionnaire was aimed at computer science freshmen who are aiming at Master of Science (computer science) degree. The two universities provide similar kinds of Masters of computer science degrees. Both universities' yearly intake is around 100 students who are mainly admitted based on entrance examination and/or matriculation examination grades. In the previous years the admission levels have differed between the two universities while Aalto University has been more selective.

We sent students an email containing the link to the questionnaire at the beginning of fall semester in fall 2011 and 2012. A reminder followed the original invitation few weeks later. In addition, the teacher who was teaching the first course in the fall reminded students to fill in the questionnaire. The analysis of the data contains mainly descriptive statistics since at this phase of our research project we are interested in a broad overview of who our students are.

3. RESULTS

At the University of Eastern Finland (hereafter referred as UEF) 120 students started their computer science studies in 2011. 63% (n=77) of them answered the questionnaire. In 2012 there were 103 starting students and the response rate was 56% (n=48). Since there were no big differences in student population between the two years, in the following summaries we have combined the data sets from UEF. At Aalto University (hereafter referred as Aalto) 101 students started their studies in fall 2012. 64% (n=65) of the students answered the questionnaire. Altogether we got 190 answers to the questionnaires from the two universities.

The distribution of male and female students was very similar between the respondents of the two universities. 85-86% of the respondents were males and only 14-15% were females. The students at Aalto were in average a year younger than at the UEF (UEF: mean=21,9, min=19, max=43, median = 21, mode= 20, std.deviation= 3,6. Aalto: mean=20,5, min=18, max=33, median = 20, mode=19, std.deviation= 2,5). The fact that students at UEF were on average slightly older was also reflected in students' previous academic histories. The older students had had time to study in other secondary and tertiary education institutions before starting their computer science studies (Table 1).

Table 1 Students' previous academic history

	UEF		**Aalto**	
Previous academic history	**%**	**n**	**%**	**n**
Only high school	68%	85	75%	49
Studies/degree at the vocational school	9%	11	3%	2
Studies/degree at the university of applied sciences	12%	15	0%	0
Studies/degree at the university	7%	9	22%	14

3.1 Programming background

At UEF nearly 40% and at Aalto University over 60% of students had some prior programming experiences (Table 2). The clear majority of students at both universities had gained their experience through hobbies. Quite many had also some prior studies on programming. Few students had gained their programming experience at work. Java, Python, and C++, were the three most used languages among students. The two student populations differ in the degree of programming related experience. It seems students at Aalto have been experimenting with slightly more languages and writing longer programs before starting their studies. At both universities fewer female freshmen had previous programming experience than their male peers.

3.2 Students' perceptions of and interest towards computer science

The respondents in general had very positive perceptions of computer science/IT field as a workplace and their own possibility to be successful in this field. Students totally or a lot agreed with the following statements:

- There are good employment opportunities at IT (UEF 84% (n=104), Aalto 89% (n=58))
- IT has high wages (UEF 67% (n=81), Aalto 86% (n=56))
- IT offers interesting duties (UEF 77% (n=97), Aalto 82% (n=53)).
- I will be successful in IT field (UEF 66% (n=82), Aalto 83% (n=54))

Table 2 Students' previous programming experience

		UEF		Aalto	
		%	n	%	n
Has previous programming experience	All	38%	48	62%	40
	Males	44%	46	68%	38
	Females	11%	2	22%	2
Where have you gained the programming experience*	Hobbies	73%	35	78%	31
	Studies	63%	30	43%	17
	IT work experience	13%	6	20%	8
Which programming languages you have used before*	Java	50%	24	58%	23
	C++	33%	16	45%	18
	Python	31%	15	70%	28
	C	23%	11	30%	12
	Basic	15%	7	15%	6
	JavaScript	15%	7	20%	8
	PHP	15%	7	28%	11
How many programming languages you have experience with? *	1	44%	21	15%	6
	2-3	44%	21	48%	19
	≥ 4	8%	4	38%	16
	No answer	2%	2	-	-
How long was your longest program*	< 100 lines	58%	28	18%	7
	< 500 lines	18%	9	30%	12
	< 1000 lines	10%	5	18%	7
	> 1000 lines	14%	7	35%	14

Note. Percentages are calculated from the number of those respondents who have previous programming experience

However, a closer look at the data revealed some interesting differences between male and female respondents at UEF: only 53% (n=10) of female students answered that IT offers interesting duties and from the same group only 21% (n=4) agreed that they will be successful in IT field. However, at Aalto there was no such difference between male and female students' perceptions.

When we asked students about their favourite future work titles the answers scattered relatively evenly (Table 3) manager being the most popular work title.

Table 3 Students' favorite future work titles

	UEF		Aalto	
	%	n	%	n
Manager	55%	69	62%	40
Software Designer	51%	64	46%	30
Programmer	46%	58	57%	47
IT-educator or Consult	31%	39	43%	28

Most of the students (UEF 74% (n=92), Aalto 89% (n= 58)) said that IT interested them very much or a great deal. We also inquired in more detail which aspects of computer science interests students at the moment. Students showed very much or a great deal interest especially towards programming, games and software design. The results also indicate that male and female students interests differ in that female are less interested in games and computer hardware/technology than male students. Differences between universities show that students in Aalto are more interested in programming, games and software design than students in UEF. On the other hand students in UEF are more interested in computer systems than students in Aalto. Generally female students are more interested in software design than male students. (Table 6, Appendix)

3.3 Factors influencing students choosing computer science as a major

Relatives and friends had influenced part of the students when they chose CS as their study program (UEF 21% (n=26), Aalto 16% (n=10), but study counsellors had only a very small effect if any. In open answers students also highlighted the importance of own interest towards computer science (UEF 15% (n=19), Aalto 18% (n=12)) and prior hobbies in their decision (UEF 15% (n=19), Aalto 12% (n=8)). Some also explicitly mentioned that a family member or an acquaintance, who already works at the field, influenced their choice to some degree.

Based on the responses CS was most students' first choice and they did not choose CS just because they wanted to get accepted to the university (Table 4). Happenstance seemed to play only a minor role in choosing the major. However, when we looked at the results in more detail we found out that there were differences both between the two universities and between male and females. The students at Aalto seemed to be quite certain they had got in to study the major that was their first choice. The different entrance requirements also showed in students' responses; 47% of UEF students thought it was easy to get admitted to study CS but only 9% of students in Aalto hold the same thought (Aalto has had

Table 4 Students' perceptions of choosing CS as a major

		UEF		Aalto	
		%	n	%	n
CS was my 1st choice major	All	57%	72	82%	53
	Males	66%	69	82%	46
	Females	16%	3	78%	7
	No previous programming experience*	47%	36	72%	18
	Previous programming experience*	75%	36	88%	35
I just wanted to get accepted to the university	All	20%	25	5%	3
	Males	17%	18	5%	3
	Females	37%	7	0%	0
	No previous programming experience*	25%	19	4%	1
	Previous programming experience*	13%	6	5%	2
Me choosing CS was due to a happenstance	All	12%	15	21%	13
	Males	8%	9	20%	11
	Females	31%	6	33%	2
	No previous programming experience*	14%	11	28%	7
	Previous programming experience*	8%	4	15%	6
It was easy to get admitted to study CS	All	47%	58	9%	6
	Males	47%	49	9%	5
	Females	47%	9	11%	1
	No previous programming experience*	53%	41	12%	3
	Previous programming experience*	33%	16	8%	3

Note. Percentages refer to the respondents who totally or a lot agreed with the statement.

*Percentages are calculated from the number of students who had/did not have previous programming experience.

higher entrance requirements than UEF). This fact reflects most likely also in differences between the universities when we asked students whether they chose CS majors just because they wanted get accepted as a student at a university. This was quite clear in UEF where 25% of those who had no programming experience just wanted to get to the university and 53% of the same group thought it was easy to get admitted to study CS.

There were some differences also between genders. At UEF only 16% (n=3) of females agreed that CS was their first choice major and 37% of females agreed that they just wanted get accepted to a university. However, we did not find similar differences between male and female students at Aalto.

3.4 Students' expectations concerning their studies

Students had rather strong belief that they will succeed in their studies and graduate with the degree in computer science (Table 5) Factors that might hinder students from graduating were: getting interested in some other field or declined motivation in general. Few students (UEF 7% (n=9), Aalto 6% n=4) were also uncertain about their ability to deal with the extent and difficulty of the studies). Receiving a good job offer before graduating was also mentioned as a possible hindrance for graduation. Students who had programming experience had the strongest belief in graduating as masters of computer science (UEF: 67% vs. 51%. Aalto: 88% vs. 60%). In UEF there was a similar trend in the belief in the success of the studies (71% vs. 48%), but in Aalto there was no difference. Once again the women from UEF were an exception: 21% (n=4) of them were uncertain about their success in studies. Female students in both universities were uncertain on their graduation: in UEF only 21% and in Aalto 44% thought they will graduate.

4. DISCUSSION

The response rate was reasonable high, especially since answering the questionnaire was voluntary and we did not offer any incentives. We got answers from over half of the starting students and therefore we are confident that the results represent the freshmen to some degree. However, since the number of students in some sub-groups (e.g. female students) is very low we need to be careful when interpreting the results.

The results have helped us to get to know our freshmen better and thus this research has fulfilled one of its' goals. The results also provide us with the essential background knowledge we need in our future research. Next we are going to highlight some aspects regarding our results and discuss whether and how we should take them into account in our teaching and curriculum design.

Table 5 Students' beliefs on their success in studies

		UEF		Aalto	
		%	n	%	n
I will succeed in my studies	All	57%	71	80%	52
	Males	64%	67	80%	45
	Females	21%	4	78%	7
	No previous programming experience*	48%	37	80%	20
	Previous programming experience*	71%	34	80%	32
I will graduate as masters of computer science	All	57%	71	77%	50
	Males	71%	67	82%	46
	Females	21%	4	44%	4
	No previous programming experience*	51%	39	60%	15
	Previous programming experience*	67%	32	88%	35

Note. Percentages refer to the respondents who totally or a lot agreed with the statement.

*Percentages are calculated from the number of students who had/did not have previous programming experience.

4.1 Previous programming experience

Quite many students have had programming as a hobby or they had studied programming in another school/university or gained their experience through work. Small minority had considerable previous programming experience writing long programs and having experience with several languages. Many of the students also had experience with the same programming languages (Python and Java), which we use in our courses. Previous programming experience gives students a good starting point to their studies: they know at least to some degree what programming is.

A pragmatic question follows from this result: How should we take students previous programming (and work) experience into account in the teaching? Would these students need some special short cuts in their studies? On the other hand, students with previous programming experience are not a homogeneous group. The degree of experience varies a lot. Therefore, one model (e.g. standard short-cuts) does not fit all students. Or if we think the other way around: should we pay special attention to those students, who do not have programming experience?

Another interesting future research question is whether these students who have programmed before succeed in their studies better than their peers who do not have previous programming experience? This question is certainly worthwhile investigating in our future studies.

4.2 Occupational choices and interest towards computer science

The results partly corroborate the literature on factors influencing occupational choices [5, 10, 13]. The role of relatives, friends in general, and acquaintances who already work in IT field came up in our results as well as own interest towards the field and computer related hobbies. However, the role of study counsellors did not seem to have any weight in students' choices. These results call for further discussion on a closer cooperation with high schools, for instance in a form of alumni visits and/or cooperation with study counsellors and science teachers.

Students also had very positive image of the IT work (e.g., provides interesting duties, good salary) and their own abilities to be successful in the field. It is likely that these positive perceptions also played a role while students were choosing the major. Finally, it is reassuring to note that a majority of the students are studying a major that was their first choice. Most of the students also thought that they would succeed in their studies and graduate as Master of Science. However, we cannot ignore the result that we still have a group of students, who at the very beginning of freshman year doubt whether they will succeed in their studies and whether they will ever graduate. We certainly need to take a closer look at this group of students in our future studies.

One group of students that stands out as being "low expectations": female students at UEF. Many of female students were uncertain about the success in their studies and in IT field after studies. They had chosen IT mostly because they wanted to get to university and CS offered an easy way to that. Because the number of female students is small in this special group, it might be possible to arrange some special activities to enhance students' motivation and efficacy beliefs regarding successful studies and success in IT related career. However, a more qualitative approach in the future is needed in order for us to get a proper insight into this group of students so that we would be able to provide the type of support and encouragement that would best benefit these women.

4.3 What we learned from our results

As teachers we have always been aware that our students have different backgrounds and experiences. This study gives those previous "gut feelings" an empirical bases and gives us a sense of proportions of, for instance, how much previous programming experience students have or how interested in IT and computer science students actually are. We use the results for two kinds of purposes in the future: to reconsider our teaching and curriculum, and to guide us in our future studies.

During this study the CS courses at the UEF, for example, mostly followed the ACM's Computer Curriculum [ACM2001]. The first year curriculum for CS majors included three programming courses: Introduction to Programming CS0.5 (3 ECTS[1]), CS1 (4 ECTS), and CS2 (5 ECTS) courses. The programming language in the CS0.5 course was Python. In CS1 and CS2 courses we used Java. The basic structure of a programming course consists of

[1] 1 ECTS = 26,5 hours of work for a student

lectures (theory), labs (practice), and an assignment (theory and practice). Our curriculum is advertised suitable for students without any programming background. In practice, however, the drop-out rates in our programming courses are relatively high especially among novice programmers. At the same time, we have experienced that advanced students become easily frustrated without proper programming challenges.

So how can we use the discovered information to increase our CS students' ability and motivation to learn to program, and hence increase the passing rates in programming courses? Since we are dealing with a very heterogeneous group of programming students the previous question can be asked considering different experience levels, both genders separately, and different aspects on motivation and self-efficacy. Several publications have highlighted that the heterogeneous of students' qualifications and background is challenging for teachers (how to teach and how to motivate) [see, e.g. 4, 1, 2].

Though the results of this study show that the students have high motivation in the beginning of their studies our experience shows that it doesn't last for a long time. Many students seem to lose their motivation before the end of autumn semester. Why does this happen and what could be done?

There are numerous studies proposing solutions for enhancing students' motivation, for example, by using robots [8], using a "games first approach" [7], or proposing more motivating programming goals and context [3, 4]. While game programming can be seen as a motivating programming context we need to take into account that based on the results, our female CS students do not find game programming interesting. Wilson [18] presents same kind of results presenting that (when learning to program) females prefer real-world assignments to games, while males prefer game programming. To motivate our students, we could offer programming tasks that are more connected to a student's personal life, such as, hobbies. Game programming can be offered as an option but it should not be used as the only motivational context.

Finally, we would like to ask how to raise self-efficacy of the students who are feeling uncertain about their graduation in CS. Self-efficacy problems are normally connected to low programming experience (see, for example, [14]). In our case, self-efficacy problems especially involve female programmers. As we again turn to literature, Simon and Hanks [16] propose pair programming as a way to improve student's self-confidence. Mendes and Al-Fakhri in [9], and Williams et al. in [17] offer similar suggestions.

In an ideal world where we do not need to think about resources, one solution for a heterogeneous group of programming students is to divide students into two or three groups by their programming background. This way, advanced students can have programming tasks better suitable for their skills, while beginners can get intensive and more individual instructions to fulfil the course learning objectives. Furthermore we should give more time for beginners to internalize new, quite abstract things. Naturally, this will cause timing issues with other courses and students which, again, requires more resources if put into practice. The critical question is: to whom and by what criteria the restricted resources should be targeted?

In our future studies we will follow up on our students and see how they have proceeded in their studies. We are planning on adding also more qualitative approaches to our study to get a deeper insight into study processes and how different kinds of pedagogical interventions may affect it. This "getting to know our students" research has provided us relevant information, which we can use, e.g., in finding relevant subgroups of students to follow. We will also continue collecting data from two different universities. The current results suggest that student populations are slightly different at different universities.

5. CONCLUSION

The goal of this study was to get to know our computer science freshmen better and thus help us to stay in tune with the students. The results portray a picture of heterogeneous student group where different subgroups of students differ from each other by their previous programming experience, self-efficacy beliefs, and what interests them. We have discussed what some of these results could mean in relation to how and what we should teach to our students to scaffold all students' learning.

We would like to end the article by listing some very practical questions our results posed to us.

- How and to whom we should allocate the limited teaching resources?
- Which student groups would benefit from what kind of support? Some of the student groups we have in mind include:
    - Female students, especially the ones with low expectations concerning successful studying and graduation.
    - Students with/without previous programming experience.
    - Students with prior academic studies/work experience.
    - Students who have high motivation in the beginning but start to lose it during the studies. How could we identify these students and help them to keep their high motivation?

6. REFERENCES

[1] Andersson, R., and Bendix, L. 2005. "Towards a Set of eXtreme Teaching Practices." Presented at Proceedings of the 5th Baltic Sea conference on Computing education research (Koli Calling 2005), Koli National Park, Finland.

[2] Herrmann, N., Popyack, J., L. , Char, B., Zoski, P., Cera, C., D. , Lass, R., N. , and Nanjappa, A. 2003. "Redesigning introductory computer programming using multi-level online modules for a mixed audience" Proceedings of the 34th SIGCSE technical symposium on Computer science education. Reno, Navada, USA.

[3] Kelleher, C. and Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. 37, 2 (June 2005), 83-137.

[4] Lahtinen, E., Ala-Mutka, K., and Järvinen, H-M. 2005. A study of the difficulties of novice programmers. SIGCSE Bull. 37, 3 (June 2005), 14-18.

[5] Lang, C. 2010. Happenstance and compromise: a gendered analysis of students' computing degree course selection. Computer Science Education. 20 (4), 317 - 345.

[6] Lent, R. W., Brown, S.T., Talleyrand, R., McPartland, E.B., Davis, T., Chopra, S.B., Alexander, M. S. Suthakaran, V., Chai, C-C. (2002). Career Choice Barriers, Supports, and

Coping Strategies: College Students' Experiences.Journal of Vocational Behavior 60, 61–72.

[7] Leutenegger, S. and Edgington, J. 2007. A games first approach to teaching introductory programming. In Proceedings of the 38th SIGCSE technical symposium on Computer science education (SIGCSE '07).

[8] McGill. M. M. 2012. Learning to Program with Personal Robots: Influences on Student Motivation. Trans. Comput. Educ. 12, 1, Article 4 (March 2012), 32 pages.

[9] Mendes, E., Al-Fakhri, L. B., and Luxton-Reilly, A. 2005."Investigating pair-programming in a 2nd-year software development and design computer science course." in ACM SIGCSE Bulletin, pp. 296-300, 2005.

[10] Paloheimo, A., Pohjonen, K., Putila, P. (2011). 'Pathways to Male-Dominated Engineering Programs', Proceedings of 2011 ASEE Annual Conference, Vancouver, B.C., Canada, June 26-29, 2011.

[11] Pears, A., Seidman, S. Malmi, L., Mannila, L. Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. 2007. A survey of literature on the teaching of introductory programming. SIGCSE Bull. 39, 4 (December 2007), 204-223.

[12] Piteira, M. and Costa, C. 2012. Computer programming and novice programmers. InProceedings of the Workshop on Information Systems and Design of Communication (ISDOC '12). ACM, New York, NY, USA, 51-53.

[13] Schulte, C., Knobelsdorf, M. (2007). Attitudes towards Computer Science - Computing Experiences as a Starting Point and Barrier to Computer Science. In: ICER '07: Proceedings of the 3rd Workshop on International Computing Education Research, ACM, 2007, 27-38.

[14] Sethuraman, S. and Dee Medley, M. 2009. Age and self-efficacy in programming. J. Comput. Small Coll. 25, 2 (December 2009), 122-128.

[15] Shuhidan, S., Hamilton, M. and D'Souza, D. 2009. A taxonomic study of novice programming summative assessment. In Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95 (ACE '09), Margaret Hamilton and Tony Clear (Eds.), Vol. 95. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 147-156.

[16] Simon, B. and Hanks, B. 2008. First-year students' impressions of pair programming in CS1.J. Educ. Resour. Comput. 7, 4, Article 5 (January 2008), 28 pages.

[17] Williams, L., McDowell, C., Nagappan, N, Fernald, J., and Werner, L. 2003. "Building pair programming knowledge through a family of experiments," Proceedings 2003 International Symposium on Empirical Software Engineering. ISESE 2003. IEEE Computer Society 2003, pp. 143-52, 2003.

[18] Wilson, B. C. 2006. Gender differences in types of assignments preferred: Implications for computer science instruction. J. Ed. Comput. Res. 34, 3, 245–255.

[19] Yadin, A. 2011. Reducing the dropout rate in an introductory programming course. ACM Inroads 2, 4 (December 2011), 71-76.

7. APPENDIX

Questionnaire

Background questions

1. Campus

2. Student number

3. Year of birth

4. Gender

5. Previous studies (high school/vocational degree/vocational high school studies/vocational high school diploma/university studies/university degree)

6. Define in your own words: What is programming?

7. Do you have previous experience in programming? (yes/no)

8. What kind of previous experience? (hobby/studies/work)

9. Which programming languages have you used?(you can choose many)(Java/Python/ C++/C/Basic/JavaScript/PHP/Other – What?)

10.How many lines does your longest program have? (<100/<500/<1000/>1000)

Interest towards IT

11. I'm interested on information technology (a lot/quite a lot/somewhat/a little/ not at all)

12. Especially I'm interested in: (you can choose many) (computer technology/programming/ computer systems/software design/user-interfaces/games/theory)

13. In addition, I am interested in … (open ended question)

14. I think I could be successful in IT (totally agree/agree a lot/agree somewhat/agree little/don't agree)

15. IT offers good possibilities to get a job (totally agree/agree a lot/agree somewhat/agree little/don't agree)

16. IT has high wages (totally agree/agree a lot/agree somewhat/agree little/don't agree)

17. IT offers interesting jobs (totally agree/agree a lot/agree somewhat/agree little/don't agree)

To what degree the following factors influenced your choice major?

18. My careers counsellor suggested me IT (very significant/quite significant/somewhat significant/little significant/not significant)

19. My relative or friend suggested IT for me (very significant/quite significant/somewhat significant/little significant/not significant)

20. Someone else suggested IT for me (very significant/quite significant/somewhat significant/little significant/not significant)

21. I became interested on IT thanks to the image of IT in media (very significant/quite significant/somewhat significant/little significant/not significant)

22. Something else got me interested in the IT field, please elaborate (open ended question)

23. Choosing IT was just an impulse (very significant/quite significant/somewhat significant/little significant/not significant)

24. It was easy to get to IT studies (very significant/quite significant/somewhat significant/little significant/not significant)

25. I could not get the place I wanted, but I wanted to the university (very significant/quite significant/somewhat significant/little significant/not significant)

26. Other reason, please elaborate?

27. I was accepted to study IT which was my favourite (totally agree/agree a lot/agree somewhat/agree little/don't agree)

Questions relating to future

28. I'm convinced the I will graduate from IT (totally agree/agree a lot/agree somewhat/agree little/don't agree)

29. If you did not agree on the previous question what might prevent you from graduating?

30. What is your favourite job? (you can choose many) (programmer/it-educator/consult/designer/manager/something else)

31. If you answered something else, what would it be?

32. I'm quite sure I will succeed in my IT studies (totally agree/agree a lot/agree somewhat/agree little/don't agree)

33. Open comments

Table 6 Interesting aspects of computer science

		UEF		**Aalto**	
		%	**n**	**%**	**n**
Programming	All	45%	56	83%	54
	Males	46%	49	80%	45
	Females	37%	7	100%	9
Games	All	32%	40	62%	40
	Males	36%	38	66%	37
	Females	11%	2	33%	3
Software design	All	23%	26	49%	32
	Males	27%	20	46%	26
	Females	32%	6	67%	6
Computer hardware/technology	All	34%	44	43%	28
	Males	40%	42	46%	26
	Females	11%	2	22%	2
Computer systems	All	41%	51	32%	21
	Males	41%	43	34%	19
	Females	42%	8	22%	2
User-interface	All	14%	18	32%	21
	Males	14%	15	32%	18
	Females	16%	3	33%	3
Theory	All	14%	13	23%	15
	Males	10%	11	21%	12
	Females	11%	2	33%	3

Use of concept maps to analyze students' understanding of the I/O subsystem

Edurne Larraza-Mendiluze
University of the Basque Country (UPV/EHU)
Department of Computer Architecture and Technology
Manuel Lardizabal Pasealekua 1
20018 Donostia-San Sebastian
+34 943 015159
edurne.larraza@ehu.es

Nestor Garay-Vitoria
University of the Basque Country (UPV/EHU)
Department of Computer Architecture and Technology
Manuel Lardizabal Pasealekua 1
20018 Donostia-San Sebastian
+34 943 015080
nestor.garay@ehu.es

ABSTRACT

The Input/Output topic is mandatory in the Computer Architecture branch of the computing curricula. However, in our experience it is a rather complex topic for the students to understand.

This paper presents the process followed to analyze the concept maps built by the students at the end of the "Computer Structure" subject, in the second semester of the degree course.

The analysis process is explained because we had to adapt the software used in other disciplines to our needs. Merging all the concept maps showed that some relationships were missing, some were very messy and others were very strong. Based on the results achieved, some points need to be reinforced during the learning process.

Categories and Subject Descriptors

B.4.0 [**Input/Output and data communications**]: General; K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

Measurement, Documentation, Experimentation, Human Factors, Theory, Verification.

Keywords

Input/Output topic, Concept maps, Evaluation.

1. INTRODUCTION

Learning the functioning principles of the computer Input/Output (I/O) subsystem of a computer is an important part of computing degrees; both for computer engineers and computer scientists [1, 2]. The I/O topic covers a lot of concepts that need to be explained. Unfortunately, there is no consensus in the academic community about the importance of these concepts, the best way to explain them, the type of problems to be solved and the exercises to be carried out, the amount of time to be devoted to each concept, etc. In most cases, several classical references are used to support their lectures.

The importance of the concepts involved varies between references. Based on a list of the five most widely used textbooks in computer organization and architecture [3], and after a study of the syllabuses of 36 universities ranked among the first 100 in six different university rankings, we selected four textbooks. Stallings [4] describes the I/O modules in a general way, including their structure and function. The possibility of having memory-mapped registers or specific I/O instructions are considered, and the existing synchronization methods are also presented (polling, interrupts and DMA). Exercises deal with performance. Patterson and Hennessy [5] place greater emphasis on storage, dependability and performance. Hamacher et al. [6], focus more on low-level programming of interrupt service routines, and [7] pays attention to data-path and signals.

In the university where the authors of this paper work, the approach is a combination of general description and low-level programming, while in other universities the approach may focus more on performance, or other details depending on the textbook taken for the computer I/O subsystem topic.

However, this paper is not related to how the computer I/O subsystem topic is taught, but how it is acquired by the students. We would like to identify exactly where the problem lies when students are trying to understand the computer I/O subsystem. Thus, the question we would like to empirically answer in this paper is: How do students understand the I/O topic?

This paper is intended to present the work we have carried out in order to answer the above question, continuing the work presented by Larraza-Mendiluze and Garay-Vitoria [8]. For that purpose, a research has been carried out assessing students' level of understanding after being taught the computer I/O subsystem topic. The research will also show the strengths and weaknesses of the educational process giving us the opportunity to suggest modifications to be made in the educational process employed in the classroom.

The paper is divided into four further sections. The next section will review the bibliography on this topic. Section three will detail the investigation itself, the participants we selected to participate in the investigation and the steps followed to obtain the information. Section four will report the data obtained during the study and the information extracted from those data, while last section presents the conclusions and outlines future work.

2. RELATED WORK

The literature studying the issues in computer architecture education can be found at the Workshop on Computer Architecture Education [9], and in other conferences and journals dealing with computing education, such as ICER, Koli Calling, ITICSE [10, 11, 12], and IEEE Transactions on Education [13] or ACM Inroads [14]. The computer I/O subsystem topic has not been very widely treated. There are several references that include the I/O unit within computer architecture, either in simulators such as those presented in [15] and [16] or integrated into new teaching approaches [17].

The different approaches to treating I/O topics found in the textbooks is also reflected in the different universities. It is possible to analyze teaching guides to validate this. Other approaches to identifying the differences between what is taught in different centers include analyzing exam questions and problem-solving exercises, and obtaining the relevant concepts that are proposed by teachers in the corresponding universities. In order to obtain this information, as stated in [18], the collaboration of the teaching groups is essential.

In this step of the investigation, we want to find out how the students view the I/O topic at the end of the semester, using the teaching/learning methodology described by Larraza-Mendiluze et al. [19]. This will enable us to identify misconceptions that need to be addressed during subsequent teaching/learning processes.

A person's knowledge can be assessed by using different elicitation techniques [20]. Concept maps, one of these techniques, are defined by Novak and Cañas [21] as graphical tools for organizing and representing knowledge, including concepts and their relationships. As Sanders et al. [22] point out, concept maps have been used for different purposes, such as: helping students to learn; measuring changes in students understanding; and, what really interests us, obtaining a static picture of what students know. Every technique requires the training of both the person who is going to elicit the knowledge and the person from whom the knowledge is going to be elicited. Since we are working with first year students, it is important not to overload them with extra work or take too much of their time. We chose this technique because it is also used to help in students' learning, and therefore the effort put into learning concept mapping would later be rewarded.

A concept map shows individual knowledge, but we wanted to analyze the whole class. Therefore, software tools for electronic concept mapping from the learner's perspective [23] were not suitable for this research. Software that could be used to analyze all the concept maps together was not easy to find. During the search, most of the solutions we found were valid only for individual concept map assessment. However, McLinden [24] showed us the similarities between concept maps and social networks. In social network analysis there are many tools used to find similar patterns between users. We tried two of these software tools; UCINET [25], and PAJEK [26]. Due to the similarities with the text documents generated by the concept map editor we were using (CM-ED [27]), PAJEK [26] was selected as the social network analysis software for this project. [28, 29] were used in order to learn how to use PAJEK [26].

3. METHOD

3.1 Subjects

The research was carried out in a first year, second semester course called "Computer Structure"; more specifically, during the second half of the course, where the computer I/O subsystem topic is taught. Although 78 students were enrolled, the experiment was carried out with only the 39 students that actively completed the whole course. This number might be interpreted being low, but it must be borne in mind that it is a first year course. Many students could decide to abandon these studies during the first semester and never come to second semester courses. Many other students could decide not to follow a course during that semester in order to be able to fulfill the requirements of another course. Some other students may decide to take the course exams, but without actively following it (this is an option that the university gives them). Figure 1 shows how only 50% of all the students enrolled responded to our research. It is indeed 70% of the students that after that year continued the computing studies, and 100% of the students that actively followed the course.

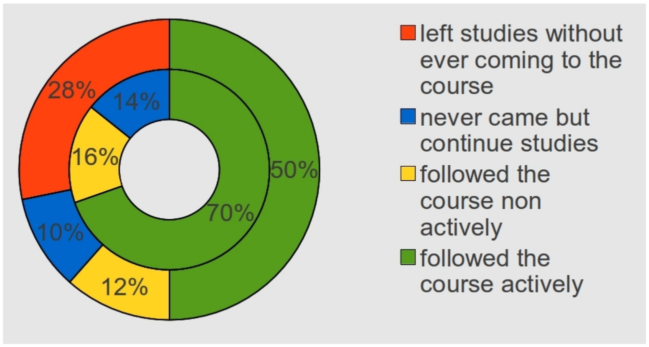


Figure 1: Graph showing the different ways in which students take the course. The outer circle refers to enrolled students while the inner circle refers to the students that did not leave the computing studies after the first year.

3.2 Procedure

3.2.1 Training the students to build concept maps

It is very important to train students in the use of a new tool (in this case, the concept maps). Even the way in which they are trained can influence the final result [30]. Therefore, the training phase was very carefully planned. Since what we wanted to obtain was the picture of the topic the students had at the end of the semester, the concept mapping training period was the same as the period devoted to teaching/learning the topic.

During the presentation of the topic, the importance we were going to give to the concept maps was emphasized.

Then, as an example, the evaluation method for this half (the computer I/O subsystem topic) of the "Computer Structure" course was explained using a concept map (see Figure 2). We wanted the students to be able to build a concept map at the end of the semester and, we therefore made the delivery of the concept maps mandatory. However, we did not want the students to learn the concept maps by rote. This was the tricky part. We clearly stated that concept maps would not be part of the examination. Moreover, the percentage of the score given to the concept maps delivered was very low, 3.5%. This approach carried the risk that the students would not take the concept maps seriously and would build them up without paying attention. In order to avoid this, we explained that concept maps built without any care would be taken as not delivered.

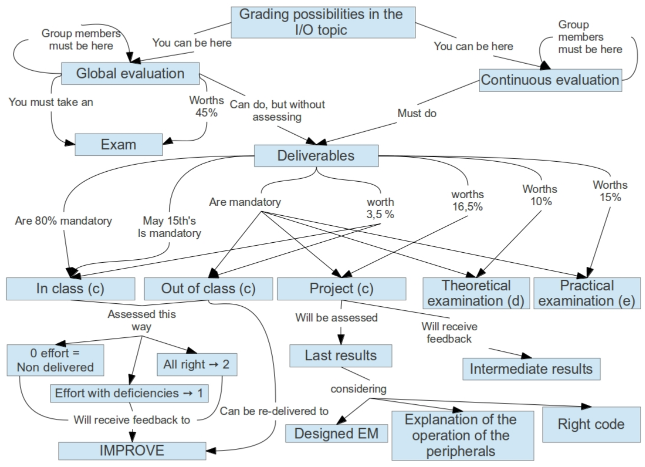


Figure 2: Concept map handed out to the students to show the different evaluation options.

During the seven weeks that the computer I/O subsystem topic lasted, the students were asked to deliver four different concept maps answering the following questions:

1. What is the Von Neumann structure? (Out of the given concepts).

2. What is needed for I/O to occur, and how does the synchronization work?

3. Complete the previous concept map with what you now know.

4. Complete the previous concept map answering the question: How does DMA work?

All the students received feedback after completing each concept map. We did not want the construction of the concept maps and our intervention to have too much effect on the final result. Indeed, it would have been better to use another topic during the training phase, but we had too many time constraints. Therefore, we decided that the feedback would be almost exclusively on the construction of the map. The only feedback statements related to the content itself were as follows:

1. You need to expand the topic further.

2. A concept such as "controller" is too broad, Please be more specific.

3. This is just a classification. You need to try to focus more on the description and the operation level. For example, you said there are two synchronization methods, but what do you need synchronization for?

The main feedback given to the students concerning the construction of the map was due to the lack of linking phrases (see Figure 3), the construction of block diagrams instead of concept maps (see Figure 4), or the use of concepts that were not concepts but whole phrases (even paragraphs taken from the course notes); in other words, a schema that looks like a concept map (see Figure 5).

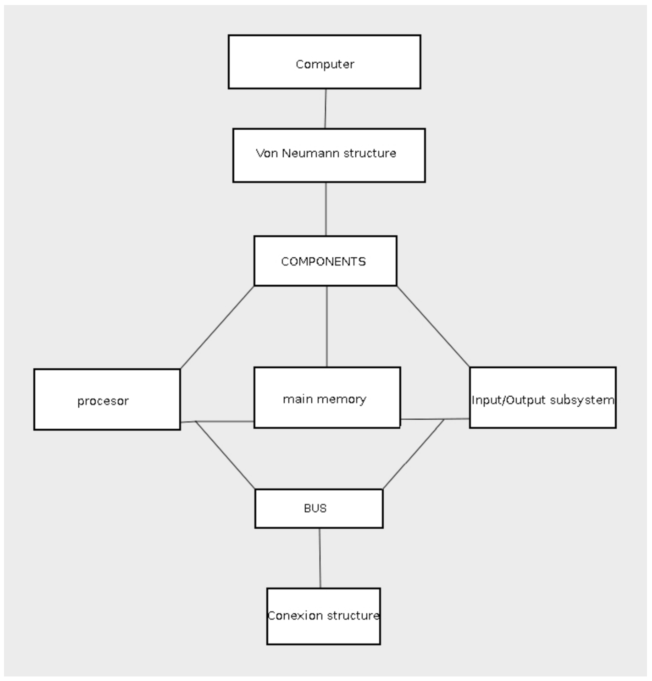


Figure 3: A student's concept map to answer the question "What is the Von Neumann structure", without any linking phrase. Translated into English from the original in Basque.

For the last concept map the students built on their own, we had to tell them to please try to show their own knowledge, because most of them were building the concept maps while reading the course notes. It is obvious when the concept maps are built that way, because they contain too many specific details.

3.2.2 The last concept map

Finally, the day after the exam, all the students were called to the classroom. They did not know they were going to be asked to build a concept map. They were given twenty selected concepts and they were told they could add more concepts if needed. The students were placed in exam conditions, in order to avoid copying, and they built their concept maps on paper.

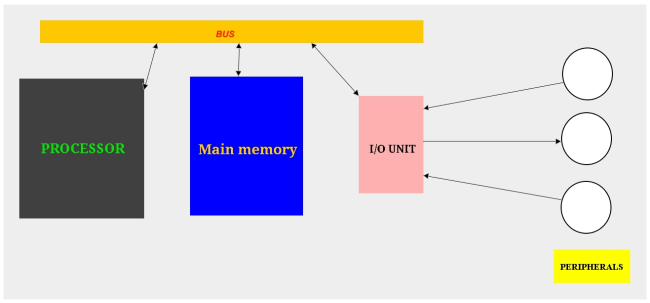


Figure 4: A student's trial concept map to answer the question "What is the Von Neumann structure" that turned out to be a block diagram. Translated into English from the original in Basque.

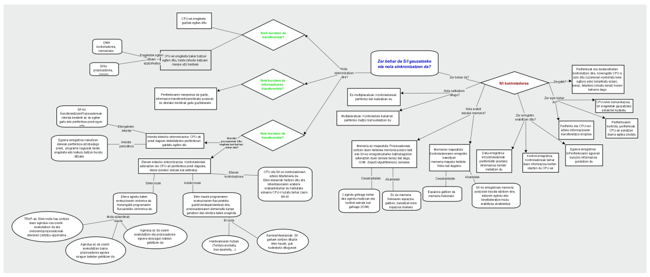


Figure 5: Although impossible to read, this is a good example of schema that looks like a concept map.

The twenty concepts given to the students to build the last concept map were taken from the course notes, according to the weighting they had during the course (e.g. we talked much more about interrupts and different kinds of interrupts than about DMA, and therefore the percentage of the concepts about interrupts is bigger than the percentage of concepts about DMA). We also considered that the concepts selected appeared widely used text books in the area [4, 5, 6, 7], although some times the terms used changed. The selected concepts can be seen in Table 1, in a random order.

3.2.3 *Students' feedback*

The students were not very happy about having to build the concept maps. Some of them told us this personally and it was also reflected in a satisfaction survey they filled out. To the question "Was concept mapping helpful in your learning process?", the students answered in a Likert scale (1: strongly disagree to 5: strongly agree) as can be seen in Figure 6. However, the data obtained with the students' concept maps was essential for this research. Moreover, more students passed the I/O subject (i.e. obtained a score higher than 5 out of 10) than in previous years, and the drop-out ratio (without considering the students who left studies the computing studies) was quite low, as can be seen in Figure 7. Although this was not the aim of this research, this data might indicate that concept maps were helpful for the students. Further analysis is needed to support this idea.

Table 1: The 20 concepts given to the students to build the last concept map.

Interrupt Service Routine	DMA controller
Enable/Disable interrupts	Polled I/O
Read control instructions	I/O register
Interrupt Controller	DMA controller
Interrupt Controller	DMA
Interrupt identification	Peripheral
Interrupt-driven I/O	CPU
Nested interrupts	Subroutine
Interrupt priority	DMA controller
I/O instructions	I/O controller

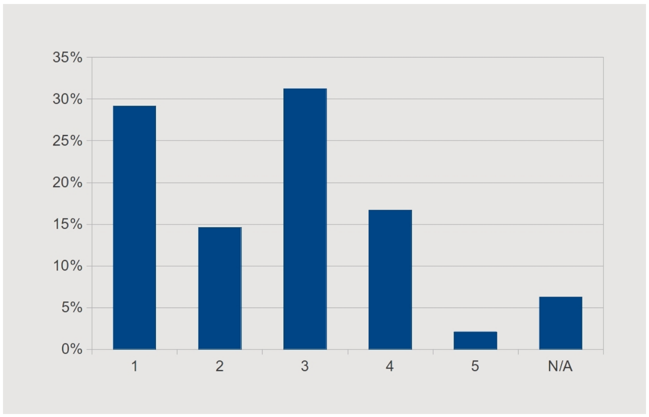


Figure 6: Answers to the question "Was concept mapping helpful in your learning process?" in a Likert scale.

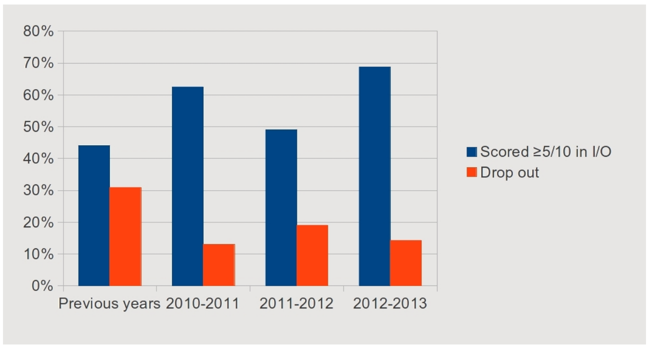


Figure 7: I/O pass and drop-out rates.

3.2.4 *Analysis of the concept maps*

Step 1.

All the concept maps built by the students in their last deliverable (the one under exam conditions) were digitalized with CM-ED [27] software. One big difference between concept maps and social networks is that while in concept maps it is possible to use n-ary relationships, social networks apparently only accept binary relationships. Therefore, some of the relationships had to be changed during this first step. Mainly two kinds of changes where made, as can be seen in Figure 8.

Step 2.
Once all the concept maps had been digitalized, the essential information needed to be extracted from the XML document that defines the concept map and converts it to PAJEK format (nodes or concepts vs. vertices, links vs. arcs(directed) or edges (non directed)).

Step 3.
Merge all the concept maps. The students had the possibility to add concepts that were not on the list (See Table 1). Therefore, it was very important to bear in mind that not all the concept maps had the same amount of concepts and that the same concept would always be numbered the same.

Step 4.
Use PAJEK to improve the readability of the graph.

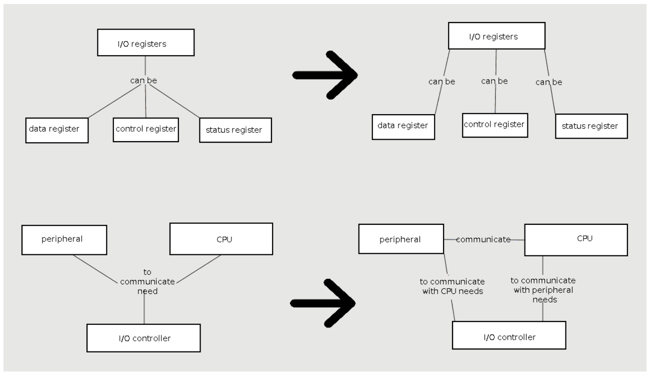


Figure 8: Conversions needed to adapt the concept maps for social network analysis.

4. RESULTS

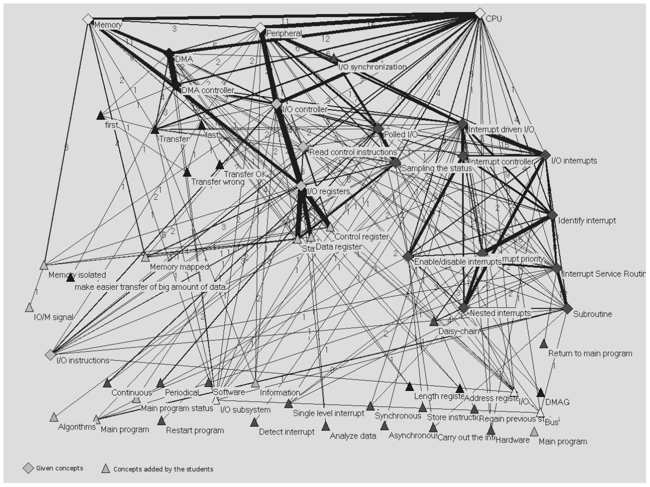


Figure 9: Graph showing all the relationships in the students' concept maps.

When merging all the concept maps in a graph, showing the strength of the concept relationships, we obtained the graph shown in Figure 9. As can be appreciated, in most of the weak relationships at least one of the linked concepts has been added by the students.

The graph in Figure 9 shows a lot of very weak relationships that hamper reading. Therefore we decided to prune the graph by removing the relationships used by three or less students. We stopped there because if we removed the relationships used by four students at least one of the concepts of the given list would be left an orphan. The resulting graph is shown in Figure 10, where most of the weak connections have been pruned out and only 7 of the 39 concepts added by the students remain connected.

4.1 Connections between subtopics

The concepts given to build the concept map can be divided into four subtopics; the system itself (*CPU, memory, and peripheral*), represented in white; the I/O controller (*I/O controller, I/O registers, read control instructions, I/O instructions*), represented in light grey; synchronization (*polled I/O, interrupt-driven I/O, sampling the status, I/O interrupts, interrupt controller, identify interrupt, enable/disable interrupts, interrupt priority, Interrupt Service Routine, nested interrupts, subroutine*), represented in dark grey; and DMA (*DMA, DMA controller*), represented in black. The concepts added by the students were also classified into these subtopics, but there was the need to create a new subtopic for some of them that were too general (e.g. program). The newly created subtopic was given the name "global".

We wanted to determine whether the students were able to correctly relate these subtopics. We shrunk the subtopics in order to be able to depict in a graph (see Figure 11) all the connections between subtopics. The graph in Figure 11 shows that the "System" subtopic is strongly connected to the other subtopics, but the "I/O controller", "Synchronization", and "DMA" subtopics are either not connected at all or are weakly connected. For example, the CPU in the "System" subtopic and the I/O controller in the "I/O controller" subtopic need synchronization in order to connect, but there are only 17 links between these two subtopics. The DMA controller in the "DMA" subtopic is in fact an I/O controller, but there are only 7 links between these two subtopics. Finally, the DMA controller needs to synchronize with the CPU in order to allow data transfer between memory and a peripheral, but there is no link between the "DMA" and the "Synchronization" subtopics.

The following subsections are going to look more closely at these connections between subtopics. That for, figures 12-13 expand these connections.

4.1.1 Links between the "I/O controller" and the "synchronization" subtopics

Figure 12 shows that although there are 17 links between the "I/O controller" and the "Synchronization" subtopics these links are divided into the two synchronization methods presented in the subject; *Polled I/O* and *interrupt-driven I/O*. Only 4 students (12%) linked the *I/O controller* to the *interrupt-driven I/O*, while 13 (39.4%) see the relationship the *I/O controller* has with *sampling the status* of the peripheral. These students also see that this has been done via *I/O registers*, and 8 students (24.2%) know that the *I/O register* used for that purpose is the *status register*.

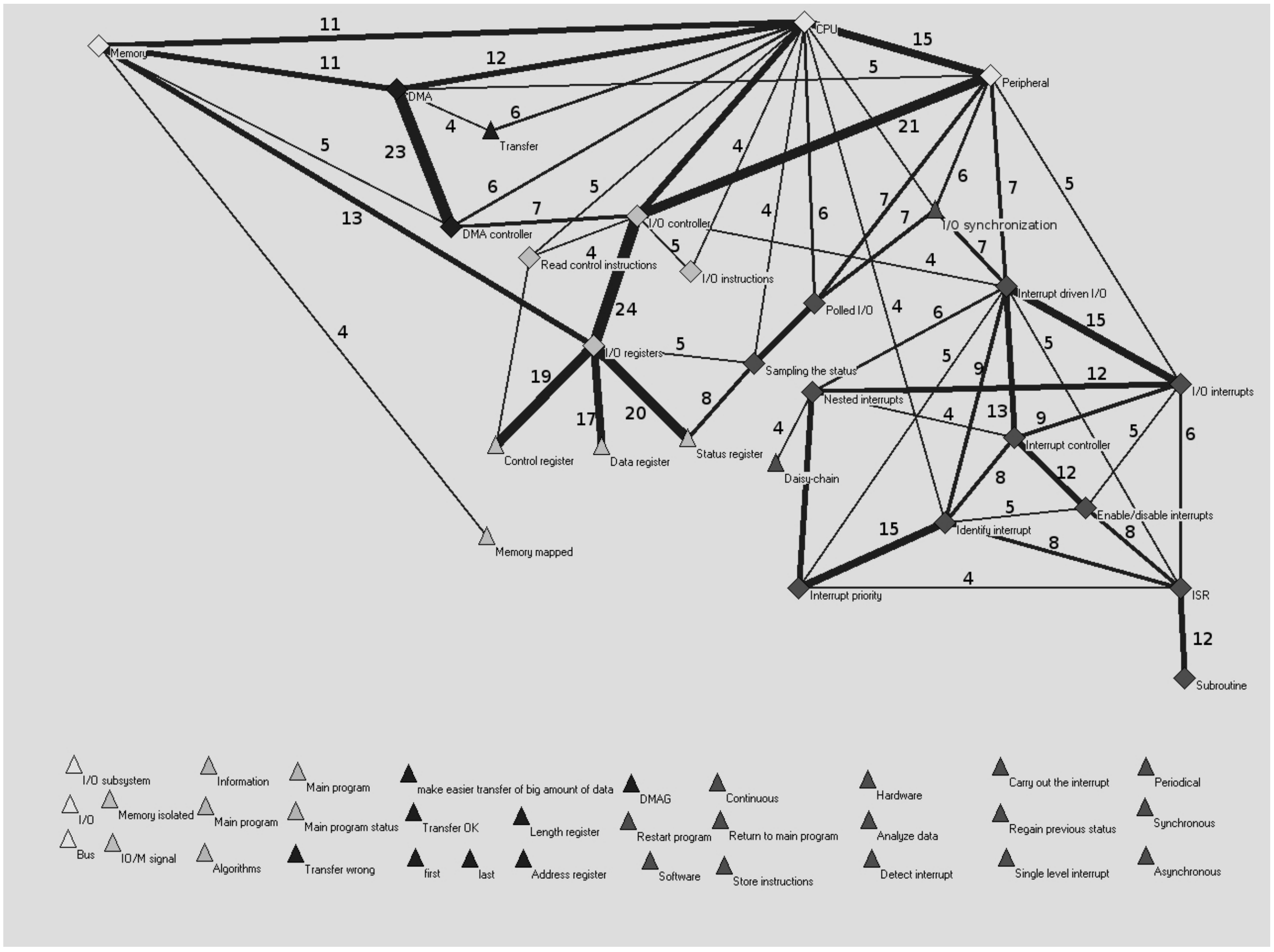


Figure 10: Graph showing the concept map links used by more than three students.

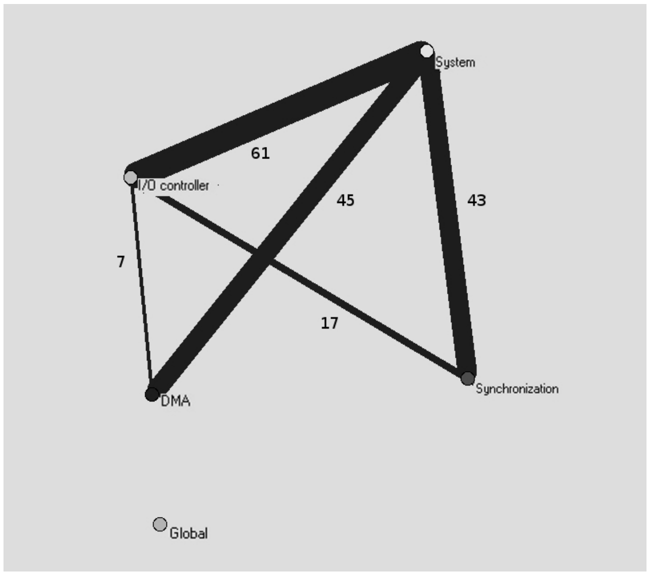


Figure 11: Graph showing the links between subtopics.

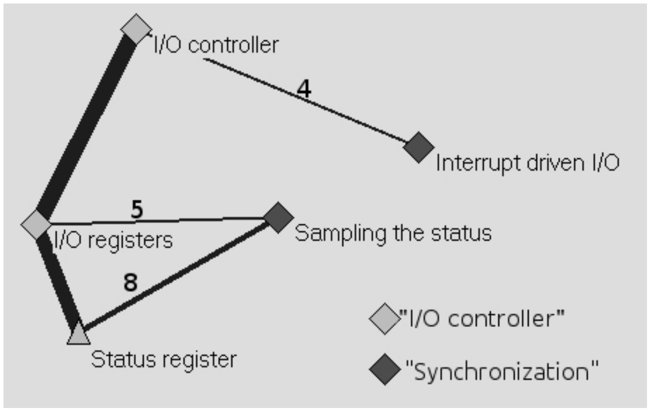


Figure 12: Graph showing the links between the "I/O controller" and the "synchronization" subtopics.

4.1.2 Links between the "I/O controller" and the "DMA" subtopics

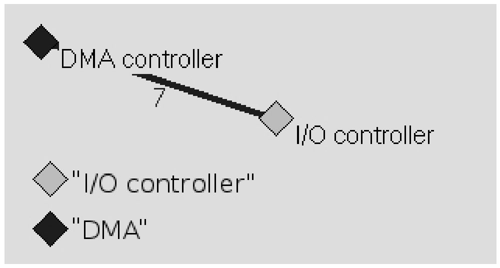


Figure 13: Graph showing the links between the "I/O controller" and the "DMA" subtopics.

Figure 13 shows all the links connecting the "I/O controller" and the "DMA" subtopics. 7 students (21.2%) know that the *DMA controller* is in fact an *I/O controller*, but no more than 3 students mention the specific *I/O registers* of the *DMA controller*.

4.1.3 Links between the "DMA" and "synchronization" subtopics

Nobody linked the "DMA" subtopic with the "synchronization" subtopic. The *DMA controller* needs to synchronize with the CPU once the transfer is finished. This synchronization is usually done by *interrupt-driven I/O* because of the ability of the *DMA controller* to generate *I/O interrupts*. However the students' concept maps show that this mental model [31] has not been created in the students' minds.

In the following subsections, we are going to look more closely at each of the subtopics.

4.2 Connections inside the "I/O controller" subtopic

Figure 14 shows the relationships inside the "I/O controller" subtopic and the links from this subtopic to the others.

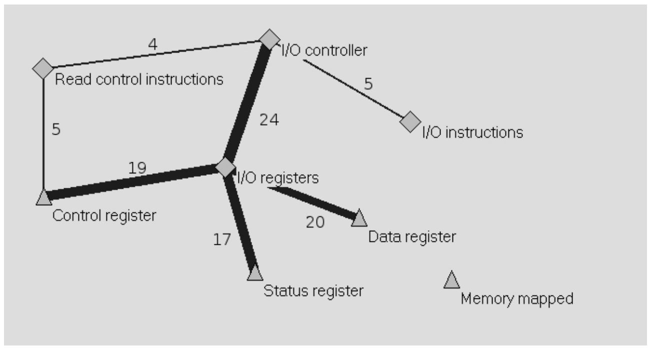


Figure 14: Graph showing the links inside the "I/O controller" subtopic.

Inside the subtopic we find four very strong connections. 24 students used the link between the *I/O controller* and the *I/O registers* in their concept maps (almost 73% of the students). Also, although these concepts were not in the concept list given, 19 (57.5%), 17 (51.5%), and 20 (60.6%) students linked the *I/O registers* to the *control register*, *data register* and *status register* respectively.

While most of the students correctly link the *I/O controller* to its *I/O registers* and even specify that these are the *control, data, and status registers*, only 6 (18%) said that the *control instructions are read* from the *control register*, and 4 (12%) that the *I/O controller* is what performs this operation.

Moreover, at least 4 (12%) students used the concept *memory mapped* but did not link it to any of the concepts in its own subtopic, they only linked it to the concept *memory* in the "System" subtopic.

4.3 Connections inside the "Synchronization" subtopic

Figure 15 shows the relationships inside the "Synchronization" subtopic. Two *I/O synchronization* methods clearly appear in this graph; *polled I/O* and *interrupt-driven I/O*. Figure 15 shows only 7 links (21.2%) to *Polled I/O* and *interrupt-driven I/O*. This is because the graph only shows direct links inside the subtopic. However, when we also consider the links from the "System" subtopic (see Figure 10), the total number of links to *Polled I/O* is 20 (60.6%), and to *interrupt-driven I/O* is 18 (54.5%). Therefore, more than half of the students have distinguished the two synchronization methods.

Polled I/O was linked by 14 students (42.4%) to *Sampling the status*, which, remember, was linked to *I/O registers* by 5 students (15.1%) and to the *status register* by 8 students (24.2%); i.e. 13 students in total (39.4%). There are no other significant links to or from these nodes.

The net formed around the *interrupt-driven I/O* concept is much bigger. It has 10 nodes in total, where only one of them was inserted by the students; *Daisy chain*. This makes it difficult to quantitatively analyze, just the direct links between concepts, because two concepts could very well be connected by indirect links that are not considered in this case. However, considering the strongest links (those used by at least 8 students (24.2%)), the graph can be read as follows: *interrupt-driven I/O* needs *I/O interrupts*. The interrupts can be *nested interrupts*. Both types of interrupts (nested and simple) are controlled by *interrupt controllers*. *Interrupt controllers* can *enable/disable interrupts* and *identify interrupts*, and according to *interrupt priority*, execute *ISRs*. *ISRs* are in fact *subroutines*.

This reading seems good, but, a deeper analysis of the linking words is needed, because, for example, the previous sentence states that *ISRs* are *subroutines*, but the *CPU* executes them directly without a *subroutine* call. Is that what the linking words imply or is it something else? The qualitative analysis needed to be able to say that is outside of the scope of this paper.

4.4 Connections inside the"DMA" subtopic

Figure 16 shows the relationships inside the "DMA" subtopic, where 23 students (69.7%) link *DMA* to *DMA controller*. A concept added by the students appears in Figure 16 (*transfer*). DMA is indeed used for large and continuous transfers of data from a peripheral to the CPU, but only 4 students (12%) used this link.

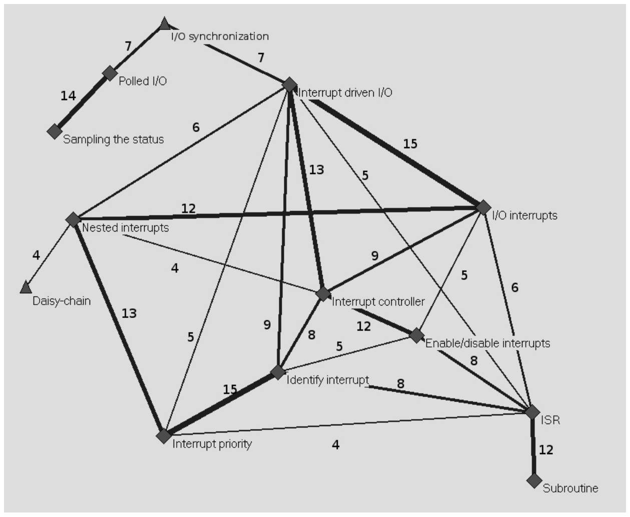


Figure 15: Graph showing the links inside the "Synchronization" subtopic.

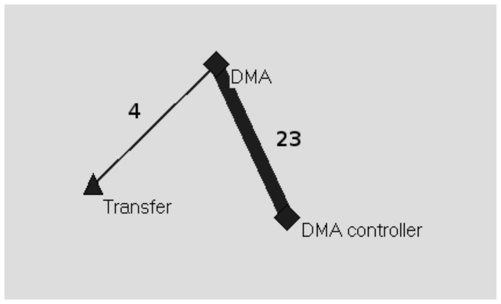


Figure 16: Graph showing the links inside the "DMA" subtopic.

5. CONCLUSIONS AND FUTURE WORK

In this research we used concept maps to identify strengths and weaknesses in students' understanding of the computer I/O subsystem. Tools from social network analysis were used for the quantitative analysis of the merged concept maps, and the results are shown.

The results show that in the students' minds there are three largely unconnected subtopics: the "I/O controller", the "Synchronization", and the "DMA". This might be because the lectures indeed separate the three subtopics. First the "I/O controller" is introduced. Once students know how to read and write information in the *I/O registers* the two synchronization methods are explained and the students use them in their projects [19], and finally the students get an introduction to the "DMA".

Based on the results of this research we would propose asking students explicit questions while they are developing their project, in order for them to understand all the different cases in which they are using the *I/O registers* (when polling, when programming a peripheral, in the interrupt service routine, etc.).

Moreover, and although *DMA* is not included in the project, a problem-solving paper exercise could be designed for each project so that the students can see the programming of the *DMA controller*; i.e. writing in the *DMA controller's I/O* *registers*, and the programming of the *ISR* for the *DMA*.

In future studies, we would like to:

- enrich the results, analyzing the links from a qualitative perspective, by looking at the link words in order to be able to say whether these links are correct or not.
- further analyze data in order to be able to say why certain links are left out by the students.
- automate the process for analyzing a big set of concept maps, so that the process can be easily replicable.
- analyze whether the information generated from the concept maps and the analysis obtained from it is reliable and whether it can be triangulated with data collected by other means (exam scripts, interviews, etc.)
- extend the analysis to other groups, universities, countries, in order to see whether or not the same problems occur.
- extend the analysis procedure to other subjects.

6. ACKNOWLEDGMENTS

First of all, the authors want to thank the students who gave us a significant amount of their time, since without these data the research would have been impossible. This research work has been partly funded under grant UFI11/45 of the UPV/EHU, under project TIN2010-15549 of the Spanish Ministry, and by the grant IT395-10 of the Department of Education, Universities and Research of the Basque Government. We also would like to thank the reviewers for their valuable comments and suggestions to improve the quality of the paper.

References

[1] The Joint Task Force on Computing Curricula (IEEE Computer Society and Association for Computing Machinery). *Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering.* IEEE Computer Society Press, Los Alamitos, CA, USA, 2004. URL `http://www.acm.org/education/education/curric_vols/CE-Final-Report.pdf`. [Retrieved 06/28/2013].

[2] The Joint Task Force on Computing Curricula (IEEE Computer Society and Association for Computing Machinery). *Computer Science Curriculum 2008: An Interim Revision of CS 2001.* IEEE Computer Society Press, Los Alamitos, CA, USA, 2004. URL `http://www.acm.org/education/curricula/ComputerScience2008.pdf`. [Retrieved 06/28/2013].

[3] L. Cassel, M. Holliday, D. Kumar, J. Impagliazzo, K. Bolding, M. Pearson, J. Davies, G.S. Wolffe, and W. Yurcik. Distributed expertise for teaching computer organization & architecture. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, ITiCSE-WGR '00, pages 111–126, New York, NY, USA, 2001. ACM. doi: 10.1145/571968.571973. URL `http://doi.acm.org/10.1145/571968.571973`.

[4] W. Stallings. *Computer Organization and Architecture: Designing for Performance.* Prentice Hall Press, 9th edition, 2012. ISBN 978-0132936330.

[5] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design, 4th Ed, D. A. Patterson and J. L. Hennessy.pdf*, volume 4th. Morgan Kaufmann, 2009. ISBN 0123744938.

[6] C. Hamacher, Z. Vranesic, S. Zaky, and N. Manjikian. *Computer organization and embedded systems.* McGraw-Hill Science/Engineering/Math, 6th edition, 2011. ISBN 9780073380650.

[7] M. Morris Mano. *Computer System Architecture.* Prentice Hall PTR, Englewood Cliffs, NJ, USA, 3rd edition, 1993. ISBN 0131755633.

[8] E. Larraza-Mendiluze and N. Garay-Vitoria. A comparison between lecturers' and students' concept maps related to the input/output topic in computer architecture. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 57–66, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1795-5. doi: 10.1145/2401796.2401803.

[9] Workshop on Computer Architecture Education. `http://www4.ncsu.edu/~efg/wcaes.html`. Retrieved 07/11/2013.

[10] ICER Conference. `http://icer.hosting.acm.org/`. Retrieved 07/09/2013.

[11] Koli Calling international conference on computing education research. `http://cs.joensuu.fi/kolistelut/`. Retrieved 07/09/2013.

[12] ITICSE Conferences. `http://www.sigcse.org/events/iticse`. Retrieved 07/09/2013.

[13] IEEE Transactions on Education. URL `http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=13`. [Retrieved 09/23/2013].

[14] ACM Inroads. URL `http://inroads.acm.org`. [Retrieved 09/23/2013].

[15] J.L. Donaldson, R.M. Salter, and R.E. Punch. DLSys: A Toolkit for Design and Simulation of Computer System Architecture. In *Proceedings of the 2011 workshop on Computer architecture education*, WCAE'11, pages 0–6, 2011. URL `http://www.ncsu.edu/wcae/HPCA2011/p9-donaldson.pdf`. [Retrieved 07/03/2013].

[16] N. Fujieda, T. Miyoshi, and K. Kise. SimMips–A MIPS system simulator. In *Proceedings of 2009 workshop on computer architecture education*, WCAE'09, pages 32–39, New York, NY, USA, 2009. ACM Press.

[17] U. Ramachandran and W.D. Leahy Jr. An integrated approach to teaching computer systems architecture. In *Proceedings of the 2007 workshop on Computer architecture education*, WCAE'07, pages 38–43, New York, NY, USA, 2007. ACM Press.

[18] E. Larraza-Mendiluze and N. Garay-Vitoria. The Learning Outcomes of the Exam Question in the Input/Output Topic in Computer Architecture. In *Learning and Teaching in Computing and Engineering (LaTiCE), 2013*, pages 212–215, 2013. doi: 10.1109/LaTiCE.2013.13.

[19] E. Larraza-Mendiluze, N. Garay-Vitoria, J.I. Martin, J. Muguerza, T. Ruiz-Vazquez, I. Soraluze, J.F. Lukas, and K. Santiago. Game-Console-Based Projects for Learning the Computer Input/Output Subsystem. *IEEE Transactions on Education*. URL `http://dx.doi.org/10.1023/A:1003028902215`. [Accepted for publication. Retrieved 06/28/2013].

[20] N.J. Cooke. Varieties of knowledge elicitation techniques. *International Journal of Human-Computer Studies*, 41(6):801 – 849, 1994. ISSN 1071-5819. doi: http://dx.doi.org/10.1006/ijhc.1994.1083. URL `http://www.sciencedirect.com/science/article/pii/S1071581984710834`.

[21] J.D. Novak and A.J. Cañas. The theory Underlying Concept Maps and How to Construct Them. *Technical Report IHMC CmapTools 2006-01 Rev 01-2008, Florida Institute for Human and Machine Cognition*, 2008. URL `http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf`. [Retrieved 06/28/2013].

[22] K. Sanders, J. Boustedt, A. Eckerdal, R. McCartney, J. E. Moström, L. Thomas, and C. Zander. Student understanding of object-oriented programming as expressed in concept maps. *ACM SIGCSE Bulletin*, 40 (1):332, Feb 2008. doi: 10.1145/1352322.1352251.

[23] A. Mühling and P. Hubwieser. Towards software-supported large scale assessment of knowledge development. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 145–146, New York, NY, USA, 2012. ACM Press. ISBN 978-1-4503-1795-5. doi: 10.1145/2401796.2401818.

[24] D. McLinden. Concept maps as network data: Analysis of a concept map using the methods of social network analysis. *Evaluation and Program Planning*, 36(1):40 – 48, 2013. ISSN 0149-7189. doi: 10.1016/j.evalprogplan.2012.05.001.

[25] L. Freeman, M. Everett, and S. Borgatti. UCINET software. URL `https://sites.google.com/site/ucinetsoftware/home`. [Retrieved 07/03/2013].

[26] A. Vlado. Networks / Pajek: Program for Large Network Analysis. URL `http://vlado.fmf.uni-lj.si/pub/networks/pajek/`. [Retrieved 07/03/2013].

[27] GaLan Group. CM-ED (Concept Maps EDitor). URL `http://galan.ehu.es/Galan/node/34`. [Retrieved 07/03/2013].

[28] W. de Nooy, A. Mrvar, and V. Batagelj. *Exploratory Social Network Analysis with Pajek*. Structural Analysis in the Social Sciences. Cambridge University Press, 2005. ISBN 9780521602624. URL `http://books.google.es/books?id=beRRM_GH1YkC`. [Retrieved 07/03/2013].

[29] V. Batagelj and A. Mrvar. Pajek, Program for Analysis and Visualization of Large Networks, Reference Manual. URL `http://vlado.fmf.uni-lj.si/pub/networks/pajek/doc/pajekman.pdf`. [Retrieved 07/03/2013].

[30] E. Santhanan, C. Leach, and C. Dawson. Concept mapping: How should it be introduced, and is there evidence for long term benefit? *Higher Education*, 35 (3):317–328, 1998. URL `http://dx.doi.org/10.1023/A:1003028902215`. [Online; accessed 28/06/2013].

[31] D. Gentner and A.L. Stevens. *Mental Models*. Cognitive Science - Lawrence Erlbaum Associates. Lawrence Erlbaum Associates, 1983. ISBN 9780898592429. URL `http://books.google.es/books?id=QFIOSvbieOcC`. [Retrieved 07/03/2013].

Exploring Misconceptions of Operating Systems in an Online Course

Sonia Pamplona
Computer Science Department
Universidad a Distancia de Madrid
Carretera de La Coruña, Km. 38,500
28400 Collado Villalba, Madrid, Spain
sonia.pamplona@udima.es

Nelson Medinilla
Computer Languages and Systems and Software Engineering Department
Universidad Politécnica de Madrid
28660 Boadilla del Monte, Madrid, Spain
nelson@fi.upm.es

Pamela Flores
Computer Languages and Systems and Software Engineering Department
Universidad Politécnica de Madrid
28660 Boadilla del Monte, Madrid, Spain
pamela.flores@fi.upm.es

ABSTRACT

Operating Systems is a difficult subject to learn; however, little is known about these difficulties, as they have not been studied or determined by the relevant literature. The objective of this article is to specify the most difficult concepts for understanding the subject and misconceptions that students have regarding these concepts. The study was conducted through an online university over the course of an entire semester. The research data comes from the evaluations taken by nine students, which have been analyzed using qualitative methods. The most difficult concepts for students to understand include concurrent computing and the mechanisms to change the program that the processor is running (interrupts, context switches, system calls, etc.). Six misconceptions regarding these concepts have been identified, helping to determine the specific problems that need to be resolved.

Categories and Subject Descriptors
D.4.0 [Operating Systems]: General; K.3.2 [Computers and Education]: Computers and Information Science Education—*Computer Science Education.*

General Terms
Human Factors

Keywords
Misconceptions, e-learning, online learning, Operating Systems.

1. INTRODUCTION

The literature, teachers and students agree that *Operating Systems* is a difficult subject. However, the difficulties and their causes are unknown. Traditional studies on teaching and learning operating systems only describe practical experiences without a theory that explains why certain tools or techniques facilitate learning better than others.

Our work seeks to advance a key issue for research in science education: identifying misconceptions that interfere with the learning process. Specifically, our objective is to study concept comprehension in an Operating Systems course and the causes for difficulties that prevent students from adequately learning this subject in an online learning environment.

This work is part of a broader qualitative study that includes analyzing documents in addition to those referenced herein.

Misconception is an important area of research for science education; its development initially began in physics [13]. Currently, references in diverse fields abound [28]. In computer science education research, concept comprehension in computer programming has been studied, including such concepts as assignment and sequence [36, 37], logical sentences [39], functions and parameter passing [19, 37], concurrent programming [18], semaphores [15], and object-oriented programming [37].

2. RELATED WORK

Fifty-nine studies on operating systems were considered that discuss such topics as teaching and learning methodologies in an Operating Systems course. To avoid over-extending this analysis, this section only describes studies published in the main conferences and journals for computer science education research from 1995 to the current year (2013). We systematically analyzed the following aspects for each study: the year of publication, course content on operating systems and proposed problems and solutions.

Analyzing the temporal dimension shows that such publications are continuous, and the interest in learning Operating Systems is valid.

By analyzing the operating systems course content and comparing it with the studies, we show that the course content herein is consistent with previous studies and the curriculum objective, as well as content recommended by the ACM/IEEE-CS Joint Task Force [1]. Therefore, the operating systems course can be used as a reference.

The primary operating systems course themes are as follows: management, scheduling, process synchronization, memory management, I/O management and file management. In addition, the studies analyzed afforded special importance to the following concepts: system calls, deadlock, semaphores, virtual memory and disk-scheduling algorithms.

Though the studies considered herein concur that the subject is difficult, they generally detail neither the specific difficulties nor possible causes.

The common objective for such studies was to facilitate teaching and/or learning of a subject through various proposals. Next, we describe the general solutions proposed .

- To develop code for the kernel in a real operating system, such as UNIX [27], Linux [16] and Android [3].
- To develop programs that do not entail modification of an operating system (using system calls and accessing the operating system data structure) to facilitate comprehension of the primary concepts discussed in the coursework [40].
- To use simulators that facilitate student exploration of operating system function without writing code [20] [24] [29–33].
- To motivate learning course materials via competing at cooperative online games [14].
- To analyze code in short programs with an unexpected result for the student to discern the causes of such behavior and delve into the course concepts [41].
- To perform experiments that measure operating system service performance and infer information about their implementation based on the results [10].
- To improve student academic performance using cooperative and problem-based learning [26].
- To improve concept comprehension and increase student interest using active learning techniques (observing and analyzing behavior in real operating systems and programming exercises) [17].

The preceding solutions are proposed without explaining why or how they aid in learning, and they lack a theoretical foundation. Research on operating system education is at an earlier phase compared with other areas in computer science education research, such as programming [12].

To move forward, our work focuses on a basic goal: to determine the causes of learning difficulties for operating systems by studying student comprehension. The research herein was performed in an exclusively online learning environment, which also distinguishes this study from previous studies in classroom environments.

3. THEORETICAL FRAMEWORK

The goal of the Operating Systems course is that the student understands the concepts taught and use such ideas to solve new problems, pose new questions or facilitate learning with new subject matter. This type of learning is meaningful learning, and it differs from rote learning, in which the student is limited to remembering the material in a manner similar to its presentation [21].

Bloom's revised taxonomy [2] is an adequate theoretical framework for meaningful learning because it describes cognitive processes and understanding levels, such that it aids in creating objectives and evaluation tests consistent with the course goals.

Bloom's original taxonomy [5] includes only one dimension, while the revised taxonomy has two dimensions: *cognitive process* and *knowledge*. The first dimension comprises nineteen cognitive processes organized in six categories: remember, understand, apply, analyze, evaluate and create. This dimension is critically important to our study because it allows us to differentiate the two aforementioned types of learning. *Rote learning* is related to remembering, and *meaningful learning* is related to the five remaining categories.

The second dimension includes four different types of knowledge: factual, conceptual, procedural and metacognitive. The current work is within the conceptual understanding dimension, which is related to bodies of extensive and organized content, such as concepts, principles, models and theories.

4. RESEARCH QUESTIONS

The purpose of this work was to study concept comprehension for Operating Systems with the ultimate goal of improving the learning with such subject matter. The following research questions guided the study.

- What *concepts* about operating systems are most difficult to comprehend?
- What *alternative conceptions* do students have about such concepts?

Next, we define the terms underlying the research questions:

- We utilize the term *concept* broadly, including ideas, objects or events that aid in understanding our environment [11].
- We define *alternative conceptions* as conceptions that are clearly incompatible with the accepted conceptions but are maintained persistently, even after instruction [1]. We prefer this general term over additional terms from the literature (e.g., misconceptions, misunderstandings and mistakes).

5. METHODOLOGY

This study was performed using a qualitative-research methodology. The qualitative data analyzed is from three tests with multiple-choice questions, wherein the students had to justify their answers.

The research herein is part of a qualitative case study, wherein additional data sources were analyzed.

5.1 Setup

The study was conducted using Operating Systems coursework for the second course of a computer engineering degree at an online university. The length of the course was 14 weeks from October 2012 to February 2013.

Next, the online learning course environment, the course manual and the activities performed by the students throughout the coursework are described.

5.1.1 Online learning environment

The online condition yields special characteristics to the study. On one hand, it is more challenging for the researcher to perceive difficulties because the student cannot be directly observed. On the other hand, the study is more aseptic with respect to the professor influence on the difficulties observed.

The conditions for the online course were as follows.

- The students had a virtual classroom through the learning management system Moodle, which included all of the course materials and activities.

- The virtual classroom provided a mechanism for interaction with the professor and other students enrolled in the course (forums).
- The students had a weekly schedule that included 8 hours in which they could contact the professor via telephone.
- The course was exclusively online, without in-person classes or video lectures in the virtual classroom.
- The activities the students used for learning were reading the manual and instructional activities. Both materials are described next.

5.1.2 Course manual

The course manual was expressly created for this subject with following characteristics.

- The manual contains the necessary information for students to perform the activities and tests in the course. However, this information does not include direct answers to the activities; the answers require that the student perform higher cognitive processes, such as interpreting, inferring, comparing and explaining. To pass the tests, the students must engage in meaningful learning, beyond the cognitive processes recognizing and remembering.
- Given that the objective of this study includes understanding whether the students comprehend the essential concepts discussed during the coursework and do not merely memorize such information, the manual could be used during each tests, including the final in-person evaluation test. Thus, if a student does not adequately answer a question, it is more likely that the problem is not a failure in recognition or memory.
- The manual was constructed in accordance with the curriculum recommended by the ACM/IEEE-CS Joint Task Force [1] and includes the following chapters.
    1. Introduction to operating systems
    2. Process management
    3. Process scheduling
    4. Communication and synchronization of processes
    5. Memory management
    6. I/O management
    7. File management
    8. Protection and security
    9. Case study 1: UNIX/Linux
    10. Case study 2: Windows 7

5.1.3 Tests

The tests given to the students in this course were as follows.

- A questionnaire on initial knowledge.
- Installing and using the Linux operating system.
- Three tests with ten multiple-choice questions.
- Two instructional activities with classic operating-system problems.
- An instructional activity using a simulator [24].

As mentioned above, this work only used tests with multiple-choice questions. For the full case study, the remaining course activities will be analyzed.

5.2 Data Collection

5.2.1 Sample

The **sample** comprises 9 students out of 13 enrolled in the course. The students selected include those who performed the instructional activities expected during development of the coursework (i.e., they did not withdraw from the course).

5.2.2 Evaluation Questionnaires

To answer the research questions, we designed three formative tests [6] with 10 multiple-choice questions. The tests correspond to the course manual content as follows.

- **Questionnaire I.** Introduction to operating systems and process management.
- **Questionnaire II.** Process scheduling. Process communication and synchronization.
- **Questionnaire III.** Memory management. I/O management. File management.

After answering each multiple-choice question, the student should justify his/her response by expounding on the basis for their selection. The students were asked to include a justification to explain their thought processes.

The questionnaires were created using the quiz tool existing in the moodle learning environment. The students had two weeks and two attempts at the questionnaire. When the student selected the incorrect answer, they received feedback.

Because the primary goal was to evaluate student comprehension of concepts from subject matter commonly included in "operating systems", the questionnaires were constructed based on the questions in the primary course textbook.

The process used to elaborate the questionnaires was as follows.

- The revised Bloom's taxonomy was used to classify all the multiple-choice questions included in the following textbooks [7] [8] [25] [9] [35] [38].
- The questions were selected using the following criteria.
    - The question content corresponds to the content established by the questionnaire.
    - The answers entail meaningful learning (i.e., the answers were not exclusively limited to the cognitive processes recognition and memory).
- The question terminology was adapted to the course manual used herein.

Tables 1, 2 and 3 show the following information for each questionnaire: the origin of each question, the cognitive process required to answer the question, the question's specific learning objective and, finally, the corresponding learning objective proposed by ACM/IEEE.

Table 1. Questionnaire 1

	Origin	Category	Cognitive process	Learning objective	Related ACM/IEEE objective
1	Exercise 1.4 Page 8 [25]	Understand	Inferring	Understand the operating system function	OS/OverviewOfOperatingSystems Learning Objective 1
2	Instructor Companion Site. Test bank Chapter 1 Question 1 [35]	Understand	Interpreting	Understand the operating system function	OS/OverviewOfOperatingSystems Learning Objective 1
3	Question 1.24. Page 52 [7]	Understand	Interpreting	Explain a system call	OS/Operating SystemPrinciples Learning Objective 4
4	Question 1.19. Page 51 [7]	Understand	Inferring	Infer the dual operation mode purpose	OS/OperatingSystemPrinciples Learning Objective 5
5	Question 1.33. Page 53 [7]	Understand	Comparing	Compare different I/O types	OS/DeviceManagement Learning Objective 5
6	Testbank. Chapter 1 Multiple choice questions Question 11 [38]	Understand	Interpreting	Understand the mode wherein multi-programming increases processor performance	OS/Concurrency Learning Objective 3
7	Testbank. Chapter 3 TRUE/FALSE Questions Question 3 [38]	Understand	Inferring	Infer the repercussions that a design change can cause in the structure or semantics of a process control block (PCB)	OS/Concurrency Learning Objective 4
8	Testbank. Chapter 1 TRUE/FALSE Questions Question 8 [38]	Understand	Inferring	Infer how interrupts aid in improving processor utilization	OS/Concurrency Learning Objective 6
9	Instructor resources TEST N1 Question 20 [8]	Understand	Inferring	Infer the situations wherein interrupts should be inhibited	OS/Concurrency Learning Objective 6
10	Instructor Companion Site. Test bank Chapter 3 Question 7 [35]	Understand	Interpreting	Understand the situations wherein a process is in the “Ready” state.	OS/SchedulingAndDispatch Learning Objective 1

Table 2. Questionnaire 2

	Origin	Category	Cognitive process	Learning objective	Related ACM/IEEE objective
1	Page 9. Question 1.2.4. [9]	Understand	Inferring	Infer circumstances and events that cause a process switch	OS/Concurrency Learning Objective 3
2	Instructor resources TEST N1 Question 21 [8]	Understand	Inferring	Infer the maximum number of processes that can remain blocked in a semaphore booted with a value of 2	OS/Concurrency Learning Objective 5
3	Page 75. Question 3.2.2. [9]	Understand	Interpreting	Interpret the signal operation code for a semaphore	OS/Concurrency Learning Objective 5
4	Page 75. Question 3.2.7. [9]	Understand	Inferring	Infer whether it is possible to execute two, consecutive wait operations on a semaphore	OS/Concurrency Learning Objective 5
5	Instructor Resources TEST N5 Question 21 [8]	Apply	Implementing	Calculate the number of resources necessary to avoid deadlock in a system with 3 processes such that each requires 3 units of a given resource type	OS/Concurrency Learning Objective 9
6	Instructor Resources TEST N3 Question 19 [8]	Understand	Inferring	Infer whether a given process-scheduling algorithm can cause resource starvation	OS/SchedulingAndDispatch Learning Objective 1
7	Instructor Resources TEST N4 Question 7 [8]	Understand	Inferring	Infer which transition is invalid between process states in a non-preemptive scheduling algorithm	OS/SchedulingAndDispatch Learning Objective 1
8	Testbank. Chapter 9 TRUE/FALSE Questions Question 8 [38]	Understand	Inferring	Infer whether the FCFS algorithm is better for short than long processes	OS/SchedulingAndDispatch Learning Objective 1
9	Page 8. Question 1.2.1. [9]	Understand	Inferring	Infer the time to affect process-scheduling algorithms (ready queue, queue for processes in execution or queue for blocked processes)	OS/SchedulingAndDispatch Learning Objective 1
10	Question 2.26. Page 87 [7]	Understand	Interpreting	Interpret the characteristics of the SJF algorithm	OS/SchedulingAndDispatch Learning Objective 1

Table 3. Questionnaire 3

	Origin	Category	Cognitive process	Learning objective	Related ACM/IEEE objective
1	Instructor Resources TEST N3 Question 32 [8]	Apply	Implementing	Select the disk scheduling algorithm most appropriate for a given situation	OS/SchedulingAndDispatch Learning Objective 7
2	Question 6.31. Page 267 [7]	Apply	Implementing	Select the disk scheduling algorithm most appropriate for a given situation	OS/SchedulingAndDispatch Learning Objective 7
3	Instructor Resources TEST N2 Question 15 [8]	Apply	Executing	Apply the elevator algorithm with variant C-SCAN to a given situation	OS/SchedulingAndDispatch Learning Objective 7
4	Instructor Resources TEST N1 Question 28 [8]	Apply	Executing	Given a system with virtual memory, calculate the page faults produced by the LRU algorithm in a given situation	OS/MemoryManagement Learning Objective 2
5	Instructor Resources TEST N1 Question 23 [8]	Understand	Inferring	Given a virtual memory system, infer the maximum move in the clock algorithm during the selection of a page	OS/MemoryManagement Learning Objective 2
6	Page 33. Question 2.2.3. [9]	Understand	Inferring	Infer what limits the size of programs in an operating system with virtual memory	OS/MemoryManagement Learning Objective 2
7	Instructor Resources TEST N3 Question 27 [8]	Understand	Inferring	Infer the main purpose of the introduction of virtual memory	OS/MemoryManagement Learning Objective 2
8	Page 14. Question 5.2.4. [9]	Understand	Inferring	Infer what I/O technique is more efficient	OS/DeviceManagement Learning Objective 5
9	Page 165. Question 6.2.1. [9]	Understand	Inferring	Inferring the implications of assigning non-continuous blocks to store a file	OS/FileSystems Learning Objective 2
10	Page 165. Question 6.2.2. [9]	Understand	Inferring	Inferring the implications of using linked allocation as a method of allocating space for files	OS/FileSystems Learning Objective 2

5.3 Analysis

The data were analyzed in three phases using the ATLAS.ti, qualitative analysis software [23].

In the first analysis phase, we developed a code [34] for each answer that is not entirely correct. Thus, it is important to remember that each multiple-choice question has two parts: answer selection and the reasoning used. Next, we describe the form for the aforementioned coding.

Answer selection:

- If the answer was correct, no code was created because difficulties were not detected.
- If the answer was incorrect, a code was created that represented the student's answer (e.g., "The change of process always originates with a clock interrupt").
- If the question was blank, a code was created that reported the question as unanswered (e.g., "does not answer which transition between states of a process cannot occur in a non-preemptive, scheduling algorithm").

Answer justification:

- If the justification was correct, no code was created because difficulties were not detected.
- If the justification was incorrect, a code was created, wherein the justification and answer were combined (e.g., "The same process cannot execute two wait operations on a semaphore because this operation is indivisible or atomic").
- If there was no justification, a code was created that reported answer as unjustified (e.g., "does not justify why it is possible to execute two consecutive wait operations on a semaphore").

The second analysis phase was used to codify patterns [22]. In this phase, we identified the possible causes for incorrect answers, and we grouped the codes with a common cause. This grouping facilitated the results for the second research question: alternative student conceptions.

Finally, in the third phase, the codes from the previous phase related to the same concept were grouped, and an answer to the first research question was generated, identifying the operating-system concepts that are more difficult to comprehend.

6. RESULTS

The operating-system concepts that were more difficult for the students to comprehend in our study are mechanisms to change the program that is running the processor, as well as concurrent computing. Six misconceptions regarding these concepts have been identified (Table 4).

Next, each concept and the alternative student conception are discussed. The alternative conceptions are treated systematically: first, the alternative student conception is presented, and later, we explain how it is incompatible with the accepted conception. Last, the students' answers that are related to the alternative conception are presented.

The questions shown in this section could be seen in Tables 5 and 6.

Table 4. Results

Concepts most difficult to comprehend	Alternative conceptions around these concepts
A. Mechanisms to change the program that the processor is running	A.1. Identifying the interrupt and process switch concepts A.2. Simplistic conception of the purpose for interrupts A.3. Simplistic conception of I/O operations
B. Concurrent computing	B.1. Concurrent computing of various processes is deterministic B.2. Relating the deadlock concept with the lock concept B.3. Conception of the semaphore influenced by common sense

6.1 Concept: mechanisms to change the program that is running the processor

The most important difficulty in this study is related to code alternation mechanisms (interrupts, context switches, system calls, etc.). For this concept, the students produced the following alternative conceptions.

1. Identifying the interrupt and process switch concepts

The students identified an interrupt with a process switch; they believe that an interrupt must correlate with a process switch.

However, interrupts and process switches are two different concepts, and one is not necessarily caused by the other. An interrupt does not always cause a process switch. For example, an I/O interrupt may yield one process ready for execution. However, the process will not be executed until it is selected by the scheduler. A process switch is also not necessarily caused by an interrupt. For example, when a higher-priority process arrives in a system with a preemptive scheduling algorithm, this process is immediately executed.

We found this alternative conception in four students and in answers to two different questions.

On one hand, three students answered question 1 on the second questionnaire similarly: "a process switch is always caused by a clock interrupt". The justifications for their answers are as follows.

"Because an interrupt serves to carry out a process switch, among other things."

"Every clock interrupt is correlated to a process switch".

The third student does not justify the answer.

On the other hand, in the second questionnaire, a student answered question 7 correctly but justified his/her response as follows: "when the algorithm is non-preemptive, it cannot yield a transition from execution to ready because the process does not wait on interrupts". In this assertion, the student utilized the term "interrupt" to refer to a process switch. In reality, the arrival of a higher-priority process would generate a process switch, but no interrupt would exist.

2. Simplistic conception of the purpose for interrupts

The students understand interrupts simplistically as a mechanism for changing the program that is executed at the moment and ignore its effects on system performance, specifically on I/O time use.

Interrupts are an efficient mechanism for communicating with the processor. Specifically, I/O interrupts allow the processor to execute other instructions while an I/O operation is in progress. An interrupt alerts the processor that the operation has ended or notifies it of an error. Not knowing such information significantly affects comprehension of operating system function, as discussed next.

We found this alternative conception in 3 students and in the answers to one question.

Two students answered question 8 in the first questionnaire similarly: "interrupts do not serve to improve processor utilization". Their justifications were as follows.

"Interrupts do not serve to improve processor utilization because they serve to change to the operating system privileged mode".

"Interrupts do not serve to improve processor utilization because an interrupt is a temporary program suspension, and if there are many interrupts, processes will be executed slower because the processor will be devoted to addressing such interrupts".

Another student answered correctly, but he did not have the arguments to justify the answer (i.e., the student did not explain why the interrupts serve to improve processor utilization).

3. Simplistic conception of I/O operations

The students do not know the sequence of events that occur during an I/O operation.

When a process initiates an I/O operation it stopped his execution and get blocked waiting for an interrupt arrival, which would indicate operation completion. During that wait time, the processor can continue executing additional processes to increase the system performance. When the I/O operation ends, the process is ready for execution and waits its turn on the processor.

We found this alternative conception in answers from 2 students and to three questions.

A student gave the following answer to question 7 from questionnaire two: "the transition 'blocked-executing' cannot occur in a non-preemptive scheduling algorithm because the process can only proceed from the 'executing' state to the 'finished' state".

In this answer, the student disregards I/O operation mechanisms; any process is blocked when an I/O operation begins independent of the scheduling system type.

Another student offers the following answers to questions 6 and 10 from questionnaire one, respectively.

Table 5. Questions related with the results section (Questionnaire 1)

	Origin	Question	Category	Cognitive process
6	Testbank. Chapter 1 Multiple choice questions Question 11 [38]	In a uniprocessor system, multiprogramming increases processor efficiency by: A. Taking advantage of time wasted by long wait interrupt handling B. Disabling all interrupts except those of highest priority C. Eliminating all idle processor cycles.	Understand	Interpreting
8	Testbank. Chapter 1 TRUE/FALSE Questions Question 8 [38]	Interrupts are provided primarily as a way to improve processor utilization. A. True. B. False.	Understand	Inferring
9	Instructor resources TEST N1 Question 20 [8]	What code do you think need to be run with interrupts inhibited? A. None, because the interrupts could be lost. B. All operating system code. C. Certain critical parts of the operating system code such as context switching.	Understand	Inferring
10	Instructor Companion Site. Test bank Chapter 3 Question 7 [35]	A process may transition to the Ready state by which of the following actions? A. Completion of an I/O event. B. Awaiting its turn on the CPU. C. Newly-admitted process. D. All of the above.	Understand	Interpreting

Table 6. Questions related with the results section (Questionnaire 2)

	Origin	Question	Category	Cognitive process
1	Page 9. Question 1.2.4. [9]	A process switch: A. is performed by the scheduler. B. modifies the entry in the process table of the process evicted. C. is always caused by a clock interruption. D. occurs whenever a process leaves the waiting process queue and enters in the ready process queue.	Understand	Inferring
4	Page 75. Question 3.2.7. [9]	Is it possible to execute two consecutive wait operations in a semaphore? A. Yes. B. Yes, if they are requested by two different processes. C. No.	Understand	Inferring
5	Instructor Resources TEST N5 Question 21 [8]	Consider a system with 3 processes so that each one of them needs 3 units of a particular type of resource. How many units of that resource must exist at least to avoid the deadlock? A. 6. B. 7. C. 8. D. 9.	Apply	Implementing
7	Instructor Resources TEST N4 Question 7 [8]	Which of the following transitions between states of a process can not occur in a system with a non-preemptive scheduling algorithm? A. Blocked to Ready. B. Running to Ready. C. Running to Blocked. D. Ready to Running.	Understand	Inferring

The student does not justify why multiprogramming in monoprocessor systems increases the processor performance through the I/O wait time.

Completion of an I/O operation cannot cause a process to proceed to the ready state.

To conclude, we show a difficulty that may be related to the three aforementioned alternate conceptions.

Four students did not explain nor justify why certain critical parts of the operating system, such as the process switch, should be executed with interrupts inhibited (question 9 from questionnaire one).

6.2 Concept: Concurrent computing

The second and last group of difficulties found in this study regards concurrent computing.

The following describes the students' misconceptions regarding this concept:

1. Concurrent computing of various processes is deterministic

Students consider concurrent computing to be deterministic; in other words, the different programs are always executed in the same order and consequently always produce the same results.

Nevertheless, generally in concurrent computing, the order in which different processes are executed cannot be determined. For example, the concurrent execution of various processes can sometimes cause deadlock, and other times it does not. Consequently, in order to detect possible deadlock situations, it is necessary to explore all the possibilities that may arise during execution. Following are two examples in which students determine if a deadlock could exist by analyzing a single situation (question 5 from questionnaire two).

"If there are 3 processes and each process needs 3 units of a type of resource, a minimum of 6 units is needed. Therefore, two processes use the 3 units of the resource they need and the third has to wait until the 3 units it needs are available."

"If there are 3 processes and each process needs 3 units of a type of resource, a minimum of 8 units is needed. Therefore, two processes use the 3 units of the resource they need and the third process has to wait until one of the units used by the other processes becomes available."

2. Relating the deadlock concept with the lock concept

Students treat the concepts of lock and deadlock as if they were synonymous.

Nevertheless, the concepts are different and the differences between them are about the duration of the lock state and its cause. For example, a lock is created when a process stops executing because it is waiting its turn to use a certain resource. The condition of a deadlock is more serious, as the duration of the lock is permanent and its cause is another process that is also locked, waiting for a resource. In other words, in a deadlock, there are at least two indefinitely locked processes. When one of these processes has a resource that the other needs, it cannot make it available because otherwise it could not continue with its execution. A lock is evidently a natural situation in concurrent computing, but a deadlock is a situation that must be avoided.

In the following definition of deadlock, students describe a situation that would cause a lock for one or two of the processes, but not a deadlock: "Because if they had fewer resources, two processes would try to access the same resource unit, and consequently a deadlock would be created".

Another definition that illustrates the confusion between lock and deadlock is as follows: "Deadlocks occur when a process needs a resource that is not available."

This misconception causes students to try to avoid deadlock situations by having a number of resources that is equal to the sum of all the resources needed during program execution (question 5 from questionnaire two).:

"Understanding that the three processes must have simultaneous access to 3 resource units, you would need 3 x 3 = 9 resource units to avoid a deadlock."

"If you want to execute the 3 processes at the same time, 9 units are needed."

3. Conception of the semaphore influenced by common sense

Difficulties have been encountered in understanding the classical synchronization mechanism – the semaphore. To understand how a semaphore works, students analyze similarities with real situations instead of analyzing the semaphore operations code.

Nevertheless, the semaphore is a creation, a mechanism that was invented to solve problems of synchronization between concurrent processes. Therefore, their characteristics are defined by their operations code and not by the similarities with real situations.

The following is a student's response, which explains that it is not possible to execute two consecutive wait operations in a semaphore (question 4 from questionnaire two). Reading the code would have allowed the student to respond affirmatively to the question. However, the student instead established a similarity with a real situation, which led him to the incorrect conclusion.

"In the example of a doorman at a nightclub, when he receives the P signal, the semaphore blocks access to more processes (people in the club). People begin to accumulate, forming a line until the other operation V, up, caused by a person leaving the club, allows the first person waiting in line to enter. I believe there can only be one wait operation because I think that one is enough and more would be redundant."

In the following answer, the student responded that it is possible to execute two consecutive wait operations on one semaphore only if they are requested by two different processes (question 4 from questionnaire two):

"In the case of a public establishment, for example, if customers want to use the locked bathroom, they take a key, decreasing the number of remaining available keys. Inversely, when they leave the bathroom, they return the key, increasing the number of available resources/bathrooms. Therefore, if there is more than one resource, more than one wait operation or key operation can occur at the same time, but only with different people-processes."

7. LIMITATIONS OF THE STUDY

This study is subject to the limitations inherent to a qualitative study [22]. Next, we discuss certain considerations for the analyzed data, the results and the study's objectivity.

The collected data are the answers and justifications from students to certain questions. The relationships between these data and students' conceptions are hypotheses which require more course documents to fully analyze the results.

The results depend on the questions, knowledge of each student and instructional activities the students performed during the course.

The following facts support the study's objectivity.

- The methods and procedures of the study are described explicitly and in detail.
- The data collection and processing sequence can be followed and support the conclusions.
- The results are explicitly related to the source data.

8. CONCLUSIONS AND FUTURE WORK

The study objectives have been met, providing answers to the proposed research questions. The operating systems concepts that are more difficult to understand in the studied online course include code alternation mechanisms (interrupts, context switches, system calls, etc.) and concurrent computing. Furthermore, six misconceptions regarding these concepts have been identified, which has helped to determine the specific difficulties associated with a course in operating systems. The misconceptions discovered regarding concurrent computing support the results of papers [15] and [4]. The other misconceptions identified in our study have not been studied in existing literature.

The contribution of this paper is to shed light on the learning process for the subject of Operating Systems, specifying the difficulties encountered. This type of exploratory study has never been conducted before in the area of Operating Systems.

This study is also an example of using qualitative research, which is rarely used in or associated with the area of Computer Science Education Research. This methodology is important because it allows us to achieve a better understanding of a specific learning scenario and explain how and why these phenomena occur. The purpose of qualitative methodology is discovery, and it is a valuable complement to quantitative research, which is more focused on confirming hypotheses.

This paper is part of a qualitative case study of an online course in the subject of Operating Systems over the course of a semester. The next step would be to conduct an analysis of the other course documents to supplement the results obtained from this study.

9. REFERENCES

[1] ACM/IEEE-CS Joint Interim Review Task Force 2008. Computer Science Curriculum 2008: An Interim Revision of CS 2001, Report from the Interim Review Task Force.

[2] Anderson, L.W., Krathwohl, D.R., Airasian, P.W., Cruikshank, K.A., Mayer, R.E., Pintrich, P.R., Raths, J. and Wittrock, M.C. 2000. A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Abridged Edition. Allyn & Bacon.

[3] Andrus, J. and Nieh, J. 2012. Teaching operating systems using android. Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12 (New York, New York, USA, Feb. 2012), 613.

[4] Ben-David Kolikant, Y. 2004. Learning concurrency: evolution of students' understanding of synchronization. International Journal of Human-Computer Studies. 60, 2 (Feb. 2004), 243–268.

[5] Bloom, B.S. and Krathwohl, D.R. 1956. Taxonomy of Educational Objectives: the Classification of Educational Goals, Handbook I: Cognitive Domain. Addisson-Wesley.

[6] Bloom, B.S., Madaus, G.F. and Hastings, J.T. 1981. Evaluation to improve learning. McGraw-Hill.

[7] Candela, S., García, C.M., Quesada, A., Santana, F.J. and Santos, J.M. 2007. Fundamentos de sistemas operativos: teoría y ejercicios resueltos. Editorial Paraninfo.

[8] Carretero, J., García, F., De Miguel, P. and Pérez, F. 2007. Sistemas Operativos. Una visión aplicada. McGraw-Hill.

[9] Casillas, A. and Iglesias, L. 2004. Sistemas operativos: problemas y ejercicios resueltos. Pearson.

[10] Downey, A.B. 1999. Teaching experimental design in an operating systems class. SIGCSE '99 The proceedings of the thirtieth SIGCSE technical symposium on Computer science education (New York, New York, USA, Mar. 1999), 316–320.

[11] Eggen, P.D. and Kauchak, D.P. 2012. Educational Psychology: Windows on Classrooms (9th Edition). Pearson.

[12] Fincher, S. and Petre, M. 2004. Computer Science Education Research. Taylor & Francis.

[13] Hestenes, D., Wells, M. and Swackhamer, G. 1992. Force concept inventory. The Physics Teacher. 30, 3 (Mar. 1992), 141.

[14] Jong, B.-S., Lai, C.-H., Hsia, Y.-T., Lin, T.-W. and Lu, C.-Y. 2013. Using Game-Based Cooperative Learning to Improve Learning Motivation: A Study of Online Game Use in an Operating Systems Course. IEEE Transactions on Education. 56, 2 (May 2013), 183–190.

[15] Kolikant, Y.B.-D., Ben-Ari, M. and Pollack, S. 2000. The anthropology of semaphores. Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education - ITiCSE '00 (New York, New York, USA, Jul. 2000), 21–24.

[16] Laadan, O., Nieh, J. and Viennot, N. 2011. Structured linux kernel projects for teaching operating systems concepts. Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11 (New York, New York, USA, Mar. 2011), 287.

[17] Lincke, S.J. 2005. Creating Interest in Operating Systems via Active Learning. Proceedings Frontiers in Education 35th Annual Conference (2005), S3C–7–S3C–10.

[18] Lönnberg, J. and Berglund, A. 2007. Students' understandings of concurrent programming. Koli Calling '07 Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88 (Nov. 2007), 77–86.

[19] Madison, S.K. 1995. A Study of College Students' Construct of Parameter Passing Implications for Instruction. PhD thesis, U. Of Wisconsin.

[20] Maia, L.P., Machado, F.B. and Pacheco, A.C. 2005. A constructivist framework for operating systems education. Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education - ITiCSE '05 (New York, New York, USA, Jun. 2005), 218.

[21] Mayer, R.E. 1998. The Promise of Educational Psychology: Learning in the Content Areas. Prentice Hall.

[22] Miles, M.B. and Huberman, A.M. 1994. Qualitative Data Analysis: An Expanded Sourcebook. SAGE Publications.

[23] Muhr, T. 2013. ATLAS.ti. (Computer software). Version 7.

[24] Mustafa, B. 2011. Visualizing the modern operating system: simulation experiments supporting enhanced learning. SIGITE '11 Proceedings of the 2011 conference on Information technology education (2011), 209–214.

[25] Pérez, F., Carretero, J. and García, F. 2003. Problemas de sistemas operativos. De la base al diseño. McGraw-Hill.

[26] Perez Martinez, J.E., Garcia, J., Muñoz Fernandez, I. and Sierra Alonso, A. 2010. Active Learning and Generic Competences in an Operating Systems Course. International Journal of Engineering Education. 26, 6 (Dec. 2010), 1484–1492.

[27] Pérez-Dávila, A. 1995. O.S. bridge between academia and reality. Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education - SIGCSE '95 (New York, New York, USA, Mar. 1995), 146–148.

[28] Pfundt, H. and Duit, R. 2009. Bibliography. Students' Alternative Frameworks and Science Education. Institute for Science Education at the University of Kiel.

[29] Robbins, S. 2004. A disk head scheduling simulator. SIGCSE '04 Proceedings of the 35th SIGCSE technical symposium on Computer science education (Mar. 2004), 325–329.

[30] Robbins, S. 2007. A Java execution simulator. SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education (Mar. 2007), 536–540.

[31] Robbins, S. 2005. An address translation simulator. SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 515–519.

[32] Robbins, S. 2002. Exploration of process interaction in operating systems. SIGCSE '02 Proceedings of the 33rd SIGCSE technical symposium on Computer science education (Mar. 2002), 351–355.

[33] Robbins, S. 2001. Starving philosophers. SIGCSE '01 Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education (New York, New York, USA, Feb. 2001), 317–321.

[34] Saldana, J. 2012. The Coding Manual for Qualitative Researchers. SAGE Publications.

[35] Silberschatz, Galvin and Gagne 2011. Operating System Concepts with JAVA. John Wiley & Sons.

[36] Simon, 2011. Assignment and sequence. Proceedings of the 11th Koli Calling International Conference on Computing Education Research - Koli Calling '11 (New York, New York, USA, Nov. 2011), 10.

[37] Sirkiä, T. and Sorva, J. 2012. Exploring programming misconceptions. Proceedings of the 12th Koli Calling International Conference on Computing Education Research - Koli Calling '12 (New York, New York, USA, Nov. 2012), 19–28.

[38] Stallings, W. 2011. Operating Systems: Internals and Design Principles (7th Edition). Prentice Hall.

[39] VanDeGrift, T., Bouvier, D., Chen, T.-Y., Lewandowski, G., McCartney, R. and Simon, B. 2010. Commonsense computing (episode 6). Proceedings of the 10th Koli Calling International Conference on Computing Education Research - Koli Calling '10 (New York, New York, USA, Oct. 2010), 76–85.

[40] Wagner, T.D. and Ressler, E.K. 1997. A practical approach to reinforcing concepts in introductory operating systems. Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education - SIGCSE '97 (New York, New York, USA, Mar. 1997), 44–47.

[41] Ziegler, U. 1999. Discovery learning in introductory operating system courses. SIGCSE 99 The proceedings of the thirtieth SIGCSE technical symposium on Computer science education (1999), 321–325.

Tracing Quiz Set to Identify Novices' Programming Misconceptions

Takayuki Sekiya
Information Technology Center,
the University of Tokyo
3-8-1 Komaba, Meguro,
Tokyo, Japan
sekiya@ecc.u-tokyo.ac.jp

Kazunori Yamaguchi
Graduate School of Arts and Sciences,
the University of Tokyo
3-8-1 Komaba, Meguro,
Tokyo, Japan
yamaguch@graco.c.u-tokyo.ac.jp

ABSTRACT

Novice programmers' understanding of conditional and loop constructs are often incomplete. They seem to understand a single conditional or single loop, but fail to understand the combination of them. We propose a method for finding misconceptions underlying this failure. We first developed a tracing quiz set to locate the exact points at which students will fail. Second, we identified some misconceptions from experiments on five courses. Third, to use and validate these misconceptions, we developed an interactive test system which showed the correct answers to the students and requested them to describe their explanations. The experiments showed that some misconceptions affected the overall performance of the students.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms

Experimentation

Keywords

novice programmers, CS1, tracing, misconception

1. INTRODUCTION

We often observe that some students who can read and write a code involving a single conditional or loop can not understand the combination of them. This is strange because the meaning of the codes of such a combination can be deduced from those of a single conditional and loop. Although some mistakes seem to be just due to each student's carelessness, but some students make same mistakes for similar questions. The reason of this failure may be that they understand conditional and loop constructs incompletely.

```
def a3(a)
  ans = 0
  for i in 1..a
    ans = ans + a
  end
  p ans
end
```

Figure 1: Code a3 used in our quiz

Some misconceptions of simple conditional and loop constructs do not appear for the usual problems of conditional and loop constructs. The misconceptions are carried on to the next stage, making recovery difficult. Those with such misconceptions can pass quizzes on simple conditional and loop constructs, but the deficiency appears when they are faced with a combination of them.

Recently, a lot of studies on novices' programming skills were conducted and among them the tracing skill [5, 6] is most related to the understanding of conditional and loop constructs. To detect misconceptions as early as possible, we developed a set of tracing quizzes.

Meaningful codes have been used in assessing the tracing skill. For example, "max" and "sum" are often used to explain the semantics of a loop construct. The semantics of "max" and "sum" obscure the repeating nature of the construct, and some students understand them just as simple patterns. Such patterns can not help the students to understand a bit more complex loop. Therefore, we use quizzes without practical meanings.

Figure 1 shows a sample quiz written in Ruby. The main body of the loop does not include the control variable `i` in the enclosing loop. This is unusual, but those who completely understand the semantics of the loop construct can answer that `9` is the output of `p ans` for `a3(3)` and `16` for `a4(4)`. Surprisingly, less than 30% of students could answer both correctly in one of our novice programming classes.

Those with some misconceptions may answer a quiz while failing to answer a slightly different quiz. Therefore, we generated a code as a combination of programming constructs, and we used a set of these slightly different codes to detect misconceptions. We call the set *tracing quiz*.

From students' answers to the quizzes, we mine common errors and infer what misconception may lead to the errors. The plausibility of the inferred misconception can be estimated by the number of errors the misconception can explain at this point. By using an interactive test system, we can promptly provide feedback on possible misconceptions a student may have, and also we can ask him why he makes a mistake.

We applied these methods to five courses. Seven possible misconceptions were identified that could explain half of the errors. The collected self-explanations from our interactive test system confirmed that the inferred misconceptions are mostly correct.

2. RELATED WORK

Many programming educators have been tackling misconceptions in novice programmers. For example, Pea identified parallelism, intentionality, and egocentrism as programming language-independent bugs [10]. Bonar et al. focused on novice natural language preprogramming knowledge and novice fragmentary programming knowledge [1]. Our research focuses on misconceptions related to tracing skills in some programming language in higher education; Language-dependent misconceptions are not excluded. Problem solving strategies are not so related. Students are novices in programming but matured in logical reasoning. Misconceptions identified in these previous works may be used in the "assume misconception" in our method explained in Section 3. In our limited experiment reported in Section 4, these misconceptions did not explain the identified common errors.

Recently, studies on novice programmers have been focused on programming skills [7, 17]. For example, Lopez analyzed an end-of-semester examination, and found hierarchical relationships among programming-related skills such as reading, tracing, and writing [6]. The tracing quizzes they used have practical meanings and are not suitable to our purpose.

Vainio found the following four factors in novice programmers' poor tracing: single value tracing, confusing function and structure, inability to use external representations, and inability to raise abstraction level [16]. They conducted a series of interviews with students who participated in an intensive course on introductory programming. About one hour long interviews were carried out while a student had been taking tracing exams. Their method may reveal important factors but it is time consuming while our method is easy to carry out. The four factors they identified did not manifest in our experiment.

Gobil identified the common errors made by students learning introductory programming [4]. For example, they found that most common errors in evaluating arithmetic expression are failing in evaluating integer division and type conversion. However, they did not investigate if such novice programmer's errors are crucial for their learning programming.

Intelligent Tutoring Systems (ITSs) have been extensively investigated. As directly related research to tracing skills, Chou developed ProTracer 2.0 [3], which requests a student to input each value of variables step by step during program execution. If the input is incorrect, ProTracer provides hints to the student. Moreno used "Jeliot3" as a tool to assist in teaching introductory programming course [8]. Jeliot3 is a program animation tool oriented towards novice programmers, however, its animations was hard for the novice students who joined Moreno's experiments. Sorva's "UUhistle" helps novice programmers to understand how given codes works [14]. For example, novice programmers can verify changes of variables by putting together programming blocks on UUhistle. Sirkiä and Sorva explored the logs collected from UUhistle and identified 26 mistakes [12]. Because a few novice programmers have good tracing skills without spending time on it, identifying novice programmers who have some problems in tracing skills by using our method is beneficial.

Simulation of a human thinking process has been also extensively investigated(e.g. [9]), particularly in computer programming, MARCEL [15] is a well known system that simulates a process of constructing a program. However, MARCEL is mainly focused on modeling, and is not a mechanism that can be applied in a real-world classroom situation. We do not deeply analyze a student's entire thinking process, but concentrate on the code tracing process. We assume that students understand most programming constructs, but they fail to trace codes due to misunderstanding certain constructs. Therefore, we can provide feedback as long as we successfully infer their misconceptions.

3. METHOD

Our method involves the following procedure.

write quiz First, we write codes of various combinations of conditional and loop constructs. Then we write a quiz asking for the printed or returned values by executing the codes for some parameters.

conduct test Students answer the tracing quiz through a paper test or an interactive test system. For the interactive test given after a class, it is better to make it clear to the students that obtaining answering, using computers or asking others is not rewarding but detrimental because the test is not graded but is for giving early feedback.

common error mining For each quiz or a set of quizzes using the same code, we find errors common to many students. For example, because 42% of students answered 3 for `a3(3)` and 4 for `a3(4)` for quizzes on a code in Figure 1, we use this as a common error for these quizzes of the same code (`a3`).

assume misconception We find out a misconception which causes the common errors. This is a discovery process. The already found misconceptions in [10, 1] may be used. In our experiment reported in Section 4, we found out the misconceptions mostly from our observations and some students' explicit comments on the mistakes they made.

simulate misconception To simulate the response of those who have misconceptions, we write a program to interpret a given code along the misconception.

assess explainability We apply the interpreter program to all the codes used in the quizzes and see how the interpreter's outputs match students' answers. How each student's errors coincide with the errors output by the interpreter is used to determine whether a student has a misconception.

automate feedback Once we collected misconceptions and implemented them in an interactive test system, we can show the student possible misconceptions he may

Table 1: Description of tracing quiz set

Quiz	Code	Description
1-2	a1	conditional with comparison `>`
3-4	a2	conditional with comparison `==`
5-6	a3	loop using no control variables (see Figures 1 and 3)
7-8	a4	loop using control variables
9-10	a5	conditional with comparison `>` using no control variables (see Figure 3)
11-12	a6	conditional with comparison `==` using no control variables
13-14	a7	conditional with comparison `>` using control variables (see Figure 3)
15-16	a8	conditional with comparison `==` using control variables
17-18	a9	loop using no control variables in conditional with comparison `>`
19-20	a10	loop using no control variables in conditional with comparison `==`

have immediately after he submits answers to the system. This function helps students to remedy their misconceptions as early as possible.

validate misconception By allowing students to enter their explanations for the presented misconception in the interactive system, we can collect information invalidating the inferred misconceptions suggesting modifications as well as information validating the misconceptions. Therefore, we can constantly improve the system and adapt it to any educational environment.

4. EXPERIMENT

4.1 Tracing Quiz Set

We created 10 Ruby codes by combining conditional and loop constructs in various ways[1]. A tracing quiz asks for the output value for a given input value. We created two tracing quizzes that vary the input values for each code, resulting in a total of 20 quizzes. We used them as the tracing quiz set. Table 1 summarizes these quizzes.

4.2 Data Sets

We administered the tracing quiz set introduced in Section 4.1 for five groups of students: **G2011S**, **T2011W**, **T2012S**, **G2012S**, and **T2012W**. The details of these groups are explained below.

G2011S This group consisted of about 80 students who took a course on basic programming in Ruby in 2011. Most were sophomores and majored in math education. They were divided into two classes according to their student IDs, but we used the answers as one group because the students' characteristics and the course materials were the same. They answered questions in the tracing quiz set after ten scheduled classes. The quiz scores were not used to grade them. We used the answers of 71 students who agreed to participate in the research.

T2011W This group consisted of about 60 students who took a course on introduction to information science in 2011. They learned basic concepts on information science through programming in Ruby. Most were freshmen and sophomores. Their majors were in various fields. The course did not require a student to have programming experience and most had no programming experience. The quiz scores were not used to grade them. We used the answers of 47 students who agreed to participate in the research.

T2012S This group consisted of about 50 students who took a course on fundamental data models such as graphs, lists, sets, and trees in 2012. The course required students to have basic programming skills. That is, the other groups were novices though those in this group were not. Most were sophomores and majored in various fields. The quiz scores were not used to grade them. We used the answers of 81 students[2] who agreed to participate in the research.

G2012S This group consisted of about 110 students who took the course on basic programming in Ruby in 2012. The students' characteristics and course materials were the same as those in G2011S. The difference was that 76 students in G2012S took the tracing test with an interactive testing system and 5 students took it on paper. We used the answers of 81 students who agreed to participate in the research.

T2012W This group consisted of about 70 students who took the course on introduction to information science in 2012. The students' characteristics and course materials were the same as those in T2011W. The students took the tracing test with the testing system used for G2012S. The tracing set used for them was different from the ones used for the other groups in that `while` was used instead of `for`. For example, the code in Figure 2 was used for the test instead of that in Figure 1. We used the answers of the tracing test of 63 students and the final exam scores of 54 students.

5. ANALYSIS

5.1 Misconception

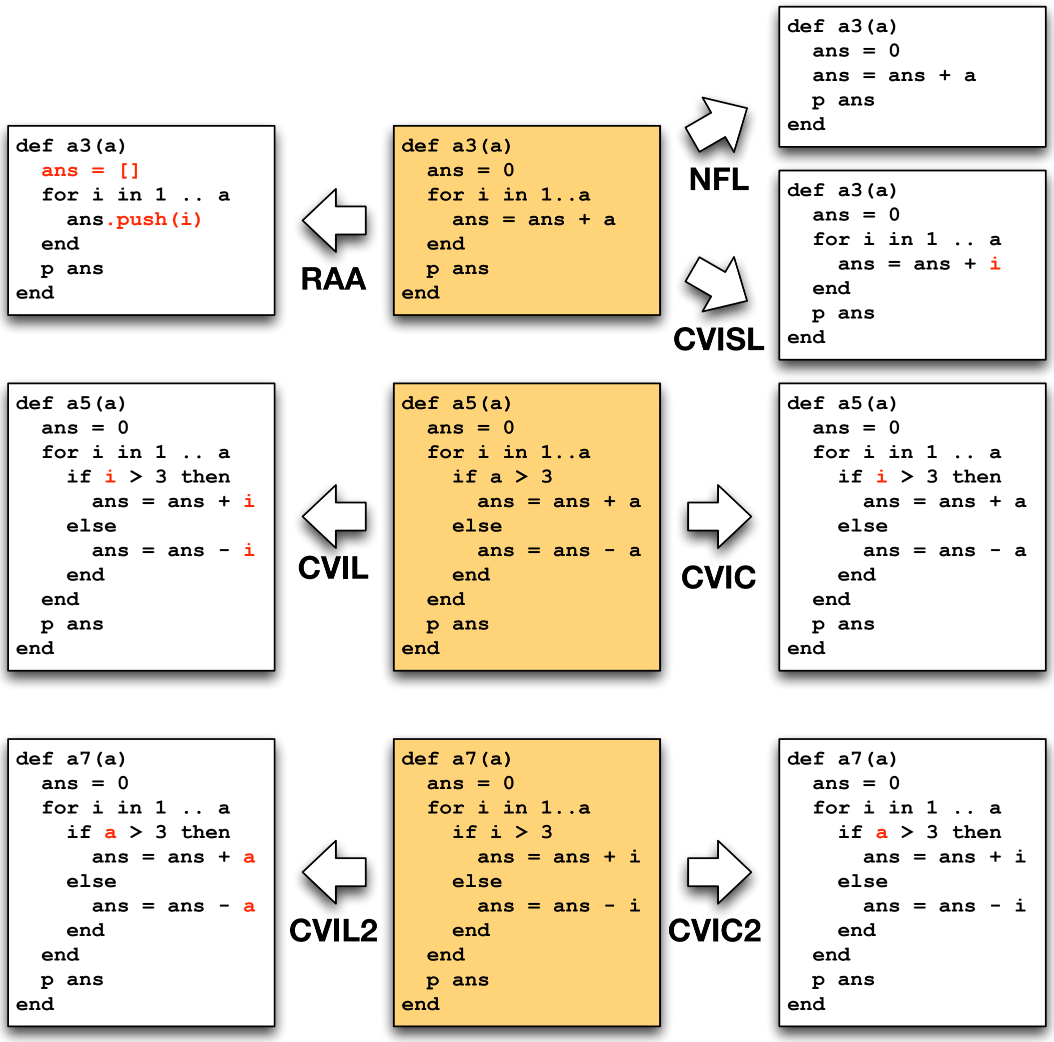
From the answers of the G2011S students, we mined common errors and inferred seven misconceptions that caused the errors from our experience. The seven misconceptions are listed in Table 2. Hereafter, we call an error that can be explained by a misconception NFL(Neglect For Loop) an

[1]All codes we used are available at `http://www.sekiya.ecc.u-tokyo.ac.jp/blog/research/codes_for_mtq.html`.

[2]They answered the tracing quiz set at the first class of this course. However, some of them gave up taking the course, therefore the number of participants of the quiz set was bigger than the enrollments for the class.

Table 2: Description of misconceptions and number of quizzes affected by them

	Name of Misconception	Description	Number of Quizzes
CVIC	Change Variable In Condition	Interpret variables in the conditional part of conditionals as control variables.	3
CVIC2	Change Variable In Condition 2	Interpret variables in the conditional part of conditionals as the end value of the enclosing loop.	3
CVIL:	Change Variable In Loop	Interpret variables in conditionals as control variables.	8
CVIL2	Change Variable In Loop 2	Interpret variables in conditionals as the end value of the enclosing loop.	6
CVISL	Change Variable In Simple Loop	Interpret variables in the body of conditionals as control variables.	4
NFL	Neglect(ignore) For Loop	Ignore loop when the body has no control variables.	8
RAA	Regard As Array	Interpret substitution to keep the previous values.	13



```
def a3(a)
  ans = 0
  ans = ans + a
  p ans
end
```

```
def a3(a)
  ans = []
  for i in 1 .. a
    ans.push(i)
  end
  p ans
end
```

```
def a3(a)
  ans = 0
  for i in 1..a
    ans = ans + a
  end
  p ans
end
```

```
def a3(a)
  ans = 0
  for i in 1 .. a
    ans = ans + i
  end
  p ans
end
```

```
def a5(a)
  ans = 0
  for i in 1 .. a
    if i > 3 then
      ans = ans + i
    else
      ans = ans - i
    end
  end
  p ans
end
```

```
def a5(a)
  ans = 0
  for i in 1..a
    if a > 3
      ans = ans + a
    else
      ans = ans - a
    end
  end
  p ans
end
```

```
def a5(a)
  ans = 0
  for i in 1 .. a
    if i > 3 then
      ans = ans + a
    else
      ans = ans - a
    end
  end
  p ans
end
```

```
def a7(a)
  ans = 0
  for i in 1 .. a
    if a > 3 then
      ans = ans + a
    else
      ans = ans - a
    end
  end
  p ans
end
```

```
def a7(a)
  ans = 0
  for i in 1..a
    if i > 3
      ans = ans + i
    else
      ans = ans - i
    end
  end
  p ans
end
```

```
def a7(a)
  ans = 0
  for i in 1 .. a
    if a > 3 then
      ans = ans + i
    else
      ans = ans - i
    end
  end
  p ans
end
```

Figure 3: Code conversions reflecting various misconceptions.

NFL error. Note that the same error can be an NFL error and a CVIL(Change Variable In Loop) error simultaneously because the same error can be explained by multiple misconceptions.

For simulating a misconception, we converted the original Ruby code into another Ruby code reflecting the misconception. For example, with the NFL misconception, the code in Figure 1 is interpreted as that of Figure 4. Other examples are shown in Figure 3.

A student with the NFL error can answer a quiz correctly if the quiz does not involve a loop construct. The "Number of Quizzes" column in Table 2 shows the number of quizzes affected by the misconception in the same row.

Table 3: Percentage of answers that match misconceptions. Percentages of total answers and incorrect answers for each group.

Misconceptions	G2011S		G2012S		T2011W		T2012W		T2012S	
CVIC	0.1	0.2	0.2	0.4	0.2	1.2	0.7	6.2	0.6	7.8
CVIC2	4.4	10.7	3.5	7.9	1.4	7.8	1.4	12.4	1.4	17.1
CVIL	6.9	17.0	6.8	15.5	3.9	22.3	0.6	5.5	2.2	27.1
CVIL2	0.8	2.1	0.1	0.3	0.1	0.6	0.7	6.2	0.0	0.0
CVISL	3.7	9.0	3.4	7.7	1.7	9.6	0.3	2.8	1.9	23.3
NFL	15.4	37.7	15.3	34.9	8.3	47.0	1.5	13.1	0.9	10.9
RAA	1.0	2.4	1.0	2.3	0.0	0.0	0.0	0.0	0.3	3.9
Matched Some	26.2	64.4	24.6	56.1	12.7	71.7	4.8	42.1	5.2	65.1
Matched None	14.5	35.6	19.3	43.9	5.0	28.3	6.7	57.9	2.8	34.9
Error	40.7	100.0	43.9	100.0	17.7	100.0	11.5	100.0	8.0	100.0
No Answer	3.4	—	1.7	—	0.6	—	9.4	—	4.3	—
Correct Answer	55.9	—	54.4	—	81.7	—	79.0	—	87.7	—

```
def a3(a)
  ans = 0
  i = 1
  while i <= a do
    ans = ans + a
    i = i + 1
  end
  p ans
end
```

Figure 2: Code a3 used in test for T2012W

5.2 Explainability

Table 3 shows how each misconception explains the result of the tracing test. For G2011S, 64.4% of the errors could be explained by the misconceptions listed in Table 2. For all groups, NFL errors dominated other errors. CVIL, CVIC2(Change Variable In Condition2), and CVISL(Change Variable In Simple Loop) incorrectly treat variables in conditionals enclosed by a loop. This group of misconceptions explains 39% of the errors.

Even though the number of errors was smaller for T2011W, 71.7% of the errors could be explained by the misconceptions in Table 2. NFL errors were 47% of the errors. The group of misconceptions in variables in conditionals enclosed by a loop explains 41.5% of the errors. For the all student groups except T2012W, more than half of errors can be explained by any of misconceptions listed in Table 2.

Next, we investigated how consistently students had misconceptions. Table 4 lists the number of students based the number of NFL errors. The tracing quiz set contains eight NFL errors, as shown in Table 2. The number of students is distributed with two peaks at each end and the minimum at its center, "4". For G2012S, the number of students is also distributed with two peaks at each end, but the minimum at "2". For both G2011S and G2012S, the differences among the number of students who answers "2"–"4" NFL errors are small (1–3). Therefore, we consider students with more than or equal to half of the NFL errors have the misconception and the other students have no misconception. A student considered to have an NFL misconception in this rule is called an *NFL student*, and one considered to have no NFL misconception is called a *non-NFL student*. For other misconceptions and other student groups, the distributions of number of students are different from each other. Therefore in this paper we adapt half of the maximum errors as the cut-off point as same as NFL.

Table 5 lists the number of students having each misconception. For G2011S, we found that 66.2% of students had misconceptions. Some of the T2012W students did not answer a part of quizzes (See the "No Answer" in Table 3), therefore the percentage in total students for T2012 was less than those for the other groups. Because the error rate of the T2012S students was only 12.3% (= 100 − 87.7. See Table 3), the remaining error is governed more by randomness, making it less consistently explained by misconceptions.

```
def a3(a)
  ans = 0
  ans = ans + a
  p ans
end
```

Figure 4: Code reflecting NFL misconception for code in Figure 1.

5.3 Stability and Sensitivity

Figure 5 shows the percentages of correct answers per quiz for all groups. The groups G2011S and G2012S were quite similar. These two groups had the same characteristics and were taught with the same course materials. This shows the stability of the tracing quiz set; similar students with similar course materials produced similar results.

Table 4: Number of students and percentage in total students according to number of NFL errors for G2011S and G2012S

	G2011S		G2012S	
Matched Errors	# of students	(%)	# of students	(%)
0	31	43.7	35	43.2
1	4	5.6	7	8.6
2	4	5.6	2	2.5
3	3	4.2	3	3.7
4	1	1.4	3	3.7
5	4	5.6	4	4.9
6	8	11.3	8	9.9
7	3	4.2	4	4.9
8	13	18.3	15	18.5

Table 5: Student misconceptions. Number of students and percentage in total students for each group.

Name of Misconceptions	G2011S		G2012S		T2011W		T2012W		T2012S	
CVIC	0	0.0	0	0.0	0	0.0	4	6.3	3	3.7
CVIC2	22	31.0	17	12.0	4	8.5	5	7.9	6	7.4
CVIL	7	9.9	9	11.1	2	4.3	0	0.0	0	0.0
CVIL2	1	1.4	0	0.0	0	0.0	1	1.6	0	0.0
CVISL	17	23.9	16	19.8	5	10.6	1	1.6	10	12.3
NFL	29	40.8	34	42.0	10	21.3	2	3.2	1	1.2
RAA	0	0.0	1	1.2	0	0.0	0	0.0	0	0.0
Matched Some	47	66.2	52	64.2	16	34.0	13	20.6	19	23.5
Total	71	—	81	—	47	—	63	—	81	—

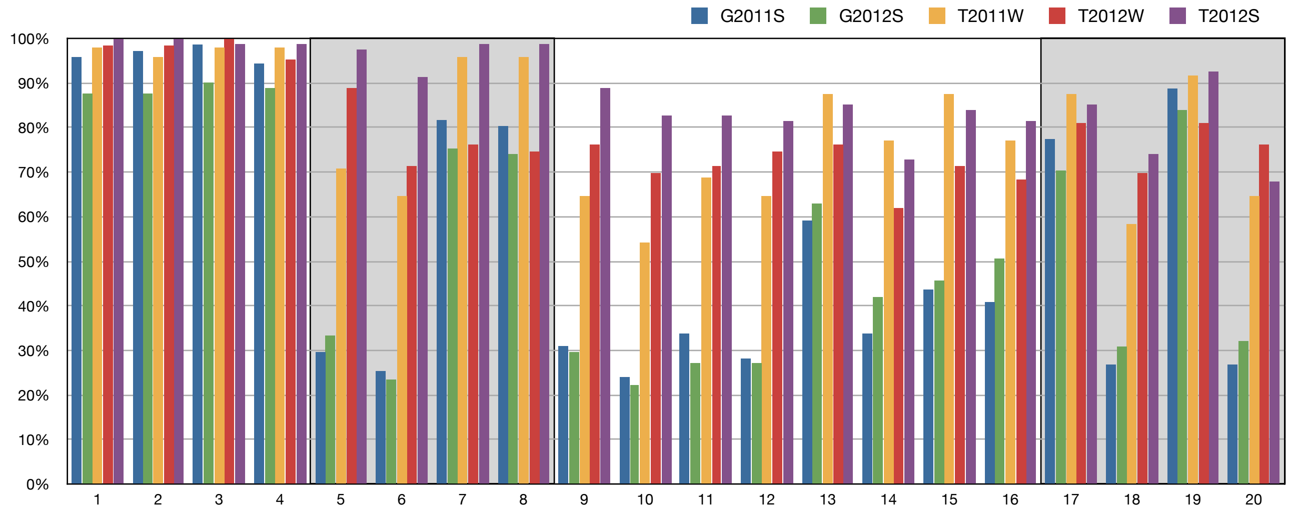


Figure 5: Percentages of correct answers per quiz. Quizzes 1–4 included only conditionals, quizzes 5–8 included only loops, quizzes 9–16 used conditionals in loops, and quizzes 17–20 used loops in conditionals.

The groups T2011W and T2012W were similar to G2011S and G2012S with respect to quizzes 1–4, 7–8, 17, and 19, but were better than G2011S and G2012S with respect to other quizzes. This is because T2011W and T2012W were different from G2011S and G2012S in student' characteristics and course materials.

The group T2012W performed better than T2011W in quizzes 5, 6, 9, and 10. This can be explained by the difference between `while` and `for` used in the quizzes for T2012W and T2011W, respectively. We found that some students ignore a loop when no control variables appear in the loop body like the code shown in Figure 1. In the `while` loop like quiz 5 for T2012W (See Figure 2), a control variable always appears in the loop body and this may prevent students from erroneously ignoring the loop. On the other hand, the group T2012W performed worse than T2011W in quiz 7 and 8. This may be explained by the fact that the code `a4` used for T2011W (See Figure 6) contains a control variable in the loop body and may prevent T2011W students from ignoring the loop. Besides, Table 3 shows that number of NFL errors for T2012W is smaller that the number for T2011W despite the fact that there is no big difference between the "Correct Answer"-s for T2011W and T2012W. The reasons of the differences between T2011W and T2012W stated here are preliminary ones and further research similar to Soloway's [13] is needed to validate them.

T2012S performed better than the other four because the students were not novices. This shows the sensitivity of the tracing quiz set; different students with different course materials produced different results.

5.4 Validity

To confirm that the inferred misconceptions are those the students had, we administered a paper-based questionnaire for T2012S. We returned marked sheets of their answers to students and asked them to describe how they answered the quizzes. For G2012S and T2012W, we used a web-based, interactive test system, as shown in Figure 7. After students submitted answers, the system showed the correct answers

```
def a4(a)
  ans = 0
  for i in 1..a
    ans = ans + i
  end
  p ans
end
```

```
def b4(a)
  ans = 0
  i = 1
  while i <= a do
    ans = ans + i
    i = i + 1
  end
  p ans
end
```

Figure 6: Code a4 for all groups except T2012W and Code b4 for T2012W used in quizzes 7–8

Table 6: Relationships between misconceptions and final exam scores for G2011S and G2012S. Number of students and percentage in total students for each group. The students of G2011S and G2012S are respectively divided into four groups, A(good score)–D(poor score), according to the quartiles of their final exam scores.

	G2011S								G2012S							
	A		B		C		D		A		B		C		D	
CVIC	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0
CVIC2	3	4.3	5	7.1	5	7.1	9	12.9	2	2.8	3	4.2	5	7.0	6	8.5
CVIL	2	2.9	2	2.9	1	1.4	2	2.9	2	2.8	3	4.2	1	1.4	2	2.8
CVIL2	0	0.0	1	1.4	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0
CVISL	4	5.7	4	5.7	3	4.3	5	7.1	3	4.2	4	5.6	4	5.6	3	4.2
NFL	1	1.4	7	10.0	11	15.7	10	14.3	2	2.8	6	8.5	11	15.5	9	12.7
RAA	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	1	1.4

Table 7: Explanations from T2012S, G2012S, and T2012W (original comments were written in Japanese)

Misconception	Comments
CVIC	"I mistook `a` for `i` at line 4." (T2012S)
CVIC2	"I mistook `i` in the line `if i > 3` for `a`, thought `ans=ans+i` would be executed when `a=4`." (T2012S)
CVIL/CVISL	"I mistook `a` in `ans=ans+a` for `i`, so calculated 1+2+3+4 = 10, but it was 4+4+4+4 = 16." (T2012S) "I thought it was $1 + 2 + 3$." (G2012S)
NFL	"I thought the answer was 3 because `ans+a` has no relation with `i`." (G2012S) "I forgot a symbol used for repeating." (T2012W)

and requested them to describe how they reached their answers. The system finally showed the inferred misconceptions if the students submitted the descriptions about their own answers. Although the system does not have any mechanism to request students to answer about the misconceptions, a few students gave us comments that the misconceptions might be helpful to understand the codes used in the tracing quiz.

Table 7 lists the comments from T2012S and G2012S students along with the misconception that explains the error. For example, a student in G2012S answered `3` to `a3(3)` for the code in Figure 1 and commented on the error that "I thought the answer was 3 because `ans+a` has no relation with `i`." This comment agrees with the description of NFL.

5.5 Effect

We analyzed the relationships between students' misconceptions and their end-of-semester examination scores to determine the effect of the misconceptions on the overall performance of the students. We divided each of G2011S and G2012S into four groups, A(good score)–D(poor score), according to the quartiles of their end-of-semester exam scores. The numbers of students in groups A, B, C, and D were 19, 17, 18, and 16 for G2011S, and 16, 19, 18, and 18 for G2012S respectively.

Table 6 lists the number of students according to misconceptions and groups (A–D). From this table, CVIC2 and NFL students had a tendency to obtain lower scores while CVIL and CVISL students had no such tendency for the both G2011S and G2012S.

To reexamine if CVIC2 and NFL students obtain poor scores, we divided the students into NFL/Non-NFL and CVIC2/Non-CVIC2 students following the rule explained in Section 5.2. For G2011S, the t-test on the differences between the averages of NFL/Non-NFL and CVIC2/Non-CVIC2 showed that the differences were significant with p-values (NFL/Non-NFL: $p = 0.001136 < 0.05$ and CVIC2/Non-CVIC2: $p = 0.04792 < 0.05$). For G2011S, the result of the same t-test did not show that the differences were significant (NFL/Non-NFL: $p = 0.05076 > 0.05$ and CVIC2/Non-CVIC2: $p = 0.209 > 0.05$), but NFL students had a tendency to obtain poorer scores than Non-NFL students. Table 8 lists the minimum, average, maximum, and standard deviation of the scores of these groups.

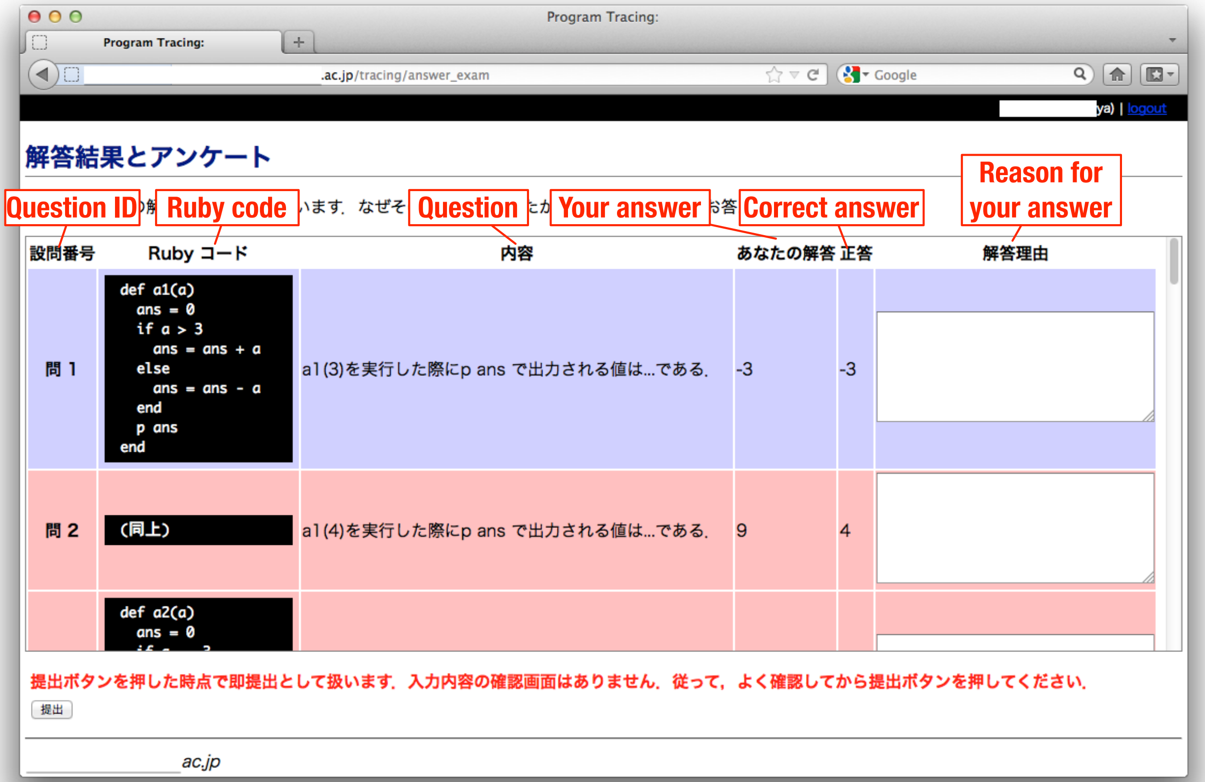
We also compared T2011W and T2012W students' misconceptions and their end-of-semester examination scores. Table 9 lists the numbers of students according to misconceptions and groups (A–D). In T2011W and T2012W, the numbers of students having misconceptions were much smaller than those in G2011S and G2012S; therefore, we could obtain statistically significant results. However NFL and CVIC2 students had a tendency to obtain similar poor scores to those in G2011S and G2012S.

These results show that NFL students' performance was significantly lower than others. In Ruby, a block that starts from `while`/`for` and ends at `end` is the basic control structure. NFL students do not completely understand even this basic control structure. Thie misunderstanding may effect not only tracing skill of programming but also other skills related to programming. Therefore such students may not be able to obtain good exam scores.

6. CONCLUSION

We proposed our tracing quiz set. From experiments with five groups of programming students, we found seven misconceptions that explain the significant portion of errors students make. In our research NFL was the most important misconception. Students with the NFL misconception performed significantly poorer; therefore early detection of the misconception may be helpful.

The tracing quiz set is easy to conduct and takes less than 30 min. It can be administered on paper or interactively. An interactive test system, like the one we have developed enables a student to get a prompt suggestion of his mis-



Program Tracing:

[redacted].ac.jp/tracing/answer_exam

[redacted]ya) | logout

解答結果とアンケート

設問番号	Ruby コード	内容	あなたの解答	正答	解答理由

問 1	`def a1(a)`
	`  ans = 0`
	`  if a > 3`
	`    ans = ans + a`
	`  else`
	`    ans = ans - a`
	`  end`
	`  p ans`
	`end`
		a1(3)を実行した際にp ans で出力される値は...である.	-3	-3	

問 2	(同上)	a1(4)を実行した際にp ans で出力される値は...である.	9	4	

	`def a2(a)`
	`  ans = 0`				

提出ボタンを押した時点で即提出として扱います．入力内容の確認画面はありません．従って，よく確認してから提出ボタンを押してください．

提出

[redacted]ac.jp

Figure 7: Snapshot of our web-based interactive test system

Table 8: Final exam score statistics of NFL, non-NFL, CVIC2, and non-CVIC2 students

	G2011S					G2012S				
	students	Min.	Ave.	Max.	Std.	students	Min.	Ave.	Max.	Std.
NFL	29	70.0	132.8	190.0	29.3	28	40.0	127.2	200.0	42.9
Non-NFL	41	50.0	160.5	200.0	38.9	43	10.0	149.1	200.0	48.6
CVIC2	22	70.0	135.9	190.0	35.2	16	40.0	127.7	200.0	44.4
Non-CVIC2	48	50.0	155.0	200.0	37.5	55	10.0	144.2	200.0	48.0

Table 9: Relationships between misconceptions and final exam scores for T2011W and T2012W. Number of students and percentage in total students for each score group. The students of T2011W and T2012W are respectively divided into four groups, A(good score)–D(poor score), according to the quartiles of their final exam scores.

	T2011W								T2012W							
	A		B		C		D		A		B		C		D	
CVIC	0	0.0	0	0.0	0	0.0	0	0.0	2	3.7	1	1.9	1	1.9	0	0.0
CVIC2	0	0.0	1	2.4	2	4.9	0	0.0	0	0.0	0	0.0	4	7.4	1	1.9
CVIL	0	0.0	1	2.4	0	0.0	1	2.4	0	0.0	0	0.0	0	0.0	0	0.0
CVIL2	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0
CVISL	0	0.0	2	4.9	1	2.4	2	4.9	1	1.9	0	0.0	0	0.0	0	0.0
NFL	0	0.0	0	0.0	5	12.2	3	7.3	0	0.0	0	0.0	0	0.0	1	1.9
RAA	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0	0	0.0

conceptions for his submission of answers. Even though the suggested misconceptions may not be his actual misconceptions, the comments he may write can be used to improve the collection of misconceptions.

Another advantage of the tracing quiz set is that its codes are defined as functions, and each quiz is just a specific instance of invocation. Therefore, students can not answer quizzes by just remembering the answers of the previous quizzes. This means that we can safely reuse and/or share codes that are the essential parts of the quizzes. For future research, we plan to analyze other type of programming concepts by developing new tracing quizzes, and to administer the same tracing quiz used in this paper. To examine the answers of students, in addition to our misconceptions, some misconceptions which are already reported by other studies (e.g [11, 2, 12]) seem available.

Acknowledgements

This work was supported by JSPS Grant-in-Aid for Challenging Exploratory Research Number 133800000012.

7. REFERENCES

[1] J. Bonar and E. Soloway. Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human-Computer Interaction*, 1(2):133–161, June 1985.

[2] B. D. Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73, 1986.

[3] C.-Y. Chou, B.-H. Huang, and C.-J. Lin. Complementary machine intelligence and human intelligence in virtual teaching assistant for tutoring program tracing. *Computers & Education*, 57(4):2303–2312, 2011.

[4] A. Gobil, Z. Shukor, and I. Mohtar. Novice difficulties in selection structure. In *Electrical Engineering and Informatics, 2009. ICEEI '09. International Conference on*, volume 02, pages 351 –356, Aug. 2009.

[5] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, ITiCSE-WGR '04, pages 119–150, New York, NY, USA, 2004. ACM.

[6] M. Lopez, J. Whalley, P. Robbins, and R. Lister. Relationships between reading, tracing and writing skills in introductory programming. In *Proceeding of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 101–112, New York, NY, USA, 2008. ACM.

[7] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, ITiCSE-WGR '01, pages 125–180, New York, NY, USA, 2001. ACM.

[8] A. Moreno and M. S. Joy. Jeliot 3 in a demanding educational setting. *Electron. Notes Theor. Comput. Sci.*, 178:51–59, July 2007.

[9] A. Newell and H. A. Simon. *Computer simulation of human thinking.* Rand Corporation, 1961.

[10] R. D. Pea. Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2(1):25–36, 1986.

[11] R. T. Putnam, D. Sleeman, J. A. Baxter, and L. K. Kuspa. A summary of misconceptions of high school basic programmers. *Journal of Educational Computing Research*, 2(4):459–472, 1986.

[12] T. Sirkiä and J. Sorva. Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, pages 19–28. ACM, 2012.

[13] E. Soloway, J. Bonar, and K. Ehrlich. Cognitive strategies and looping constructs: An empirical study. *Communications of the ACM*, 26(11):853–860, 1983.

[14] J. Sorva and T. Sirkiä. UUhistle: a software tool for visual program simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 49–54, New York, NY, USA, 2010. ACM.

[15] J. C. Spohrer and E. Soloway. Simulating student programmers. In *Proceedings of the 11th International Joint Conference on Artificial Intellegence*, volume 1 of *IJCAI'89*, pages 543–549. Morgan Kaufmann Publishers Inc., 1989.

[16] V. Vainio and J. Sajaniemi. Factors in novice programmers' poor tracing skills. *SIGCSE Bull.*, 39(3):236–240, June 2007.

[17] A. Venables, G. Tan, and R. Lister. A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the fifth international workshop on Computing education research workshop*, ICER '09, pages 117–128, New York, NY, USA, 2009. ACM.

An Easy Approach to Epistemology and Ontology in Computing Theses

Matti Tedre
Stockholm University
Department of Computer and Systems Sciences
Kista, Sweden
firstname.lastname@acm.org

John Pajunen
University of Jyväskylä
Department of Social Sciences and Philosophy
Jyväskylä, Finland
firstname.lastname@jyu.fi

ABSTRACT

In many research fields—notably social sciences but also in those fields where design, experiment-based science, and social sciences are mixed—researchers must often describe their epistemological and ontological commitments in research reports. The research literature describes those commitments in various ways, often grouped under research paradigms such as positivism, post-positivism, and constructivism, and described as "world views." This paper presents the bare bones of the ontological and epistemological questions in scientific practice. Ontologically speaking, subject matters can be mind-dependent or mind-independent. Epistemologically speaking, elements of research may be more or less open to interpretation. This paper introduces a simplified approach to standard research terminology for computing and engineering students by offering a rough-and-ready way for resolving ontological and epistemological questions.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms

Theory

Keywords

Thesis work, computing education, computer science education, philosophy, epistemology, ontology, research paradigms

1. INTRODUCTION

Computing combines a number of very different subjects under the same disciplinary umbrella (e.g., [4]). A look at the six research areas of computing [5] reveals a broad variety of subjects of study. In electrical and electronic engineering we study things like semiconductors and electromagnetic radiation. In computer engineering we study things like microprocessors and sensors. In computer science we study things like algorithms and computability theory. In software engineering, we study things like software development processes and formal methods. In information technology we study things like network design and lifecycle analysis. And in information systems we study things like organizational processes and decision support systems. Many computing fields, such as human-computer interaction and artificial intelligence, span across several computing fields and also intertwine with fields other than computing.

The variety of research topics in computing makes it nigh impossible to formulate an overarching prescription on how computing research should properly be done [24, 25]. Researchers in each computing subfield prefer a different set of methods. Analytical methods dominate in the theoretical fields, varieties of the scientific method are common in the quantitatively oriented empirical fields, methods from social sciences and the humanities dominate in the qualitatively oriented empirical fields, and engineering methods are often found in the design-oriented fields [23, 24]. It is uncommon to find pure engineering, pure science, or pure theory in computing disciplines, and some may consider it undesirable, too [12].

In computing literature, ontological and epistemological questions have received their most covering treatment in the field of information systems (e.g., [11, 16]). The lack of field-specific philosophical foundations is often not much of a concern, but as methodological literature often discusses research philosophy too, it is hard to completely ignore them. Most importantly, many students face, at some point of their thesis work, two grand words in the research literature: epistemology and ontology.

Research textbooks typically introduce the epistemological question as "What is the nature of the relationship between the knower ... and what can be known?" [8, p.108] and the ontological question as "What is the form and nature of reality and, therefore, what is there that can be known about it?" [8, p.108]. Methodology literature presents a variety of answers to these two questions, ranging from clear and simplified to abstract and vague. Guba and Lincoln [8, p.107] wrote that a research paradigm, which entails ontological, epistemological, and methodological assumptions, represents a "world view" that defines the researcher's view of the nature of the world, the individual's place in it, and the range of possible relationships to that world and its parts. On a more basic level, what the ontological and epistemological questions do in a research report is specify some basic facts about one's subject of study. Those questions, when properly formulated, can help students with

Koli Calling '13, November 14–17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526979

their framing of research, their method choice, and their research questions.

The classic definitions and treatments of research paradigms, as accurate and deep as they may be, are not always very useful for students in computing fields. Consider, for example, a student who does a quantitative study on users' perceptions on usability of a new tool. Should that student adopt the positivist viewpoint which entails naïve realist ontology and objectivist epistemology; or should that student adopt the constructivist viewpoint which entails relativist ontology and subjectivist epistemology [8]? Too many new and difficult concepts may unnecessarily complicate a discussion that is aimed at clarifying the thesis. The research paradigms are rooted in a number of fundamental ontological, epistemological, and methodological positions, and choosing between them requires quite a lot of knowledge of philosophy. It would be hard to argue that in-depth knowledge of such debates would or should play a very important role in computing students' training.

Yet, it is still important to recognize assumptions about one's research. Clarity about ontological, epistemological, and methodological stands in a research study is especially important in a field that has to maneuver between and across disciplines that deal with essentially different kinds of things. In computing, the same research study may involve mathematical and formal objects, artifacts made by people, and social or other kinds of conventions. This paper is primarily aimed at educators in the field of computing, and it proposes a practical, hopefully student-friendly approach to talking about ontology, epistemology, and methodology in computing disciplines. This article is intended to be short and to-the-point; there are only four crucial concepts: mind-dependence and independence, epistemological objectivity, and epistemological subjectivity. Those interested in deeper discussions on the topic will easily find plenty of good reading on the philosophy of science.

2. ONTOLOGY

Ontology is concerned with the most fundamental level of the world: what are the ultimate constituents of the world, "how" do those constituents exist, and what kinds of relations are there between those constituents. Some classical questions about ontology have occupied philosophers for ages. One set of such questions includes questions on substances: what are they like, and how many kinds of substances are there. Philosophers have proposed numerous answers to such questions, and some of those answers seem downright counterintuitive. A full appreciation of all possible positions would take us too far away from practical work, and we discuss only positions that we take to be relevant to the field, but we mention points of potential disagreement. We attempt to approach these difficult issues from an intuitive, common-sense point of view.

One of the major disputes in philosophy is concerned with existence of things, and their mode of existence. Existence of material things (such as processors and copper wires) seems unproblematic to common sense. Similar, for computer scientist the existence of things that we manipulate daily (such as electromagnetic radiation and electrons) often seems unproblematic. How mental things (such as wishes, emotions, thoughts, and feelings) exist might be slightly more problematic. And how social things (such as laws, conventions, first names, ideas, ideologies, and theoretical entities) exist

Table 1: Examples of mind-dependent and mind-independent things in computing disciplines

Mind-dependent	Mind-independent
Preferences, attitudes, programming languages, values, worth, standards, processes, procedures, models	Electromagnetic radiation, material properties, semiconductors, thermal noise

may be the least certain to common sense. Some philosophers argue that all the examples above exist only in people's minds, but that point of view, one kind of anti-realism, is rare. For the common sense, a realist position is often taken for granted when it comes to material things, but mental and social spheres can lead to problems.

The realist standpoint typically says that on the one hand, there are things and forces in the world that exist regardless of our thoughts and feelings—things such as silicon, electrons, copper, and electromagnetic radiation. We may not know everything about those things, but they nevertheless exist independently of us. Their existence is *mind-independent*: They would continue to exist even if there were no humans left on the planet. One must be careful to distinguish a term, say "copper" (which is a name we give to a thing), and the things that the term refers to (the actual thing, such as pieces of copper). Terms are mind-dependent but they may refer to mind-independent things.

On the other hand, there are things whose existence depends on people's states of mind, thoughts, or feelings about those things; take, for instance, first names, users' preferences, computer brands, organizational cultures, syntaxes, and agreements. The existence of those things is *mind-dependent*. They exist by virtue of people individually or collectively making them exist. Those things are created, maintained, and discarded by individual cognition or by collective agreements, and they cease to exist if there is no-one left to maintain them. Table 1 presents examples of mind-dependent and mind-independent objects. In Table 1 on the right hand side there are things whose existence is not at all relative to what people think about them, and on the left hand side there are things whose existence depends on us making them exist. It is along this dimension of ontology where people in a technical field usually face the ontological question. The reason is that many interdisciplinary fields, such as computing, deal with physical, mental, and social aspects—sometimes even within a single project.

In general there is a good understanding of the mind-independent mechanisms underlying physical systems (especially systems relevant to computing, such as electrical circuits), but the situation is not so good with minds and societies. In many sciences of the mind and societies, there is less understanding of how things work and why—and sometimes even no vision of how such consensus could even look like. Mind-dependent and -independent things are also studied differently: while mind-independent phenomena do not on the macro level care much about instruments used for studying them, mind-dependent phenomena may be irreversibly changed by the act of studying them. For instance, putting a person in usability lab probably affects that person's behavior. Similar, doing the same experiment on semiconductors in the 1970s and in the 2010s should yield the same results, but one does not even expect interview results to be the same between the 1970s and the 2010s.

When dealing with complex objects a situation may arise where one aspect of the object may belong to the sphere of mind-independent things and another to the sphere of mind-dependent things. In such a case we may say that the object of study is mind-involving. A relevant question then is: what kind of involvement is it? Artifacts are trivially mind-involving, because the very meaning of term artifact includes a mind (an artifact is made by humans and it has uses and functions that are mind-dependent). But artifacts have also physical properties that are mind-independent, and even though people have made them, their continuing existence does not depend on human minds. Artifacts have properties that are non-trivially mind-involving or not-mind-involving.

Ontology: Why?

Distinction between mind-independent things and mind-dependent things is important because there are remarkable differences between their properties. Consider, for instance, how persistent those properties are. Mind-dependent things, such as values, standards, opinions, and preferences may change radically over time, whereas many properties of mind-independent things are relatively stable: how semiconductors behave does not change over time. For example, a computer system for commercial purposes may have been effective in the 1960s economy and user base, but ceased to be so in the 2010s; even though the mind-independent parts (the computer system) have not changed, the mind-dependent parts (users and the economy) have changed.

Sciences, including many social sciences and humanities, are not directly concerned with ontological questions, yet every research study involves, implicitly or explicitly, ontological assumptions. In empirical research studies, ontology describes the most general background assumptions of the study: what sorts of things is the researcher dealing with. In natural sciences ontological issues are often not treated in university curricula or discussed in research papers, as there often is no controversy about those issues and many would consider them to be a waste of time [13]. Especially in natural sciences [13] it is a common view that there is no need to delve into philosophical discussions, as they would only slow down research. However, the situation is different in social sciences and humanities, where there is much less consensus about many fundamental philosophical issues.

Similar to many natural sciences, ontological issues are not explicitly treated in computing curricula and it is rare to see them discussed in research papers in computing. But only some parts of computing are similar to natural sciences. Much of computing research is directly concerned with mind-dependent things, which situates it closer to social sciences than natural sciences. In some clearly human-oriented branches of computing, especially information systems, ontological descriptions are more common than in branches that are associated more closely with formal or natural sciences. Yet, in all research ontological commitments have more or less direct ramifications on the epistemological and methodological aspects of the study. Especially in cross-disciplinary collaboration, computing researchers should be able to state their general ontological position in clear terms.

Ontology: How?

A crude but effective way of thinking about ontology in computing research is to ask whether the existence of the subject of study has anything to do with people's mental states—thoughts, feelings, anxieties, and so forth. Examples of subjects of study that have nothing to do with people's mental states are plenty in electrical engineering, computer engineering, and information technology. If a study is about things like wireless network signals or semiconductors, the study is concerned with things that are mind-independent. Neither the existence nor the properties of those things depend on what anyone thinks about those things.

Alternatively, if a research subject exists by virtue of people's feelings, agreements, thoughts, or other mental states, the study is concerned with things that are mind-dependent. Examples of such topics can easily be found in the fields of information systems, computer science, and software engineering. If a study is about, say, IEEE standards, OSI layers, programming languages, or users' preferences, it studies things that are mind-dependent. Those things exist because humans individually or collectively make them exist, they may be changed in the course of time, and they cease to exist if there is no-one left to hold them. Often research projects do not, however, concern objects as much as features or properties of objects—and those too can be mind-dependent or -independent.

It is important to note that mind-independent subjects of study, take a computer mouse, for example, can have both mind-dependent and -independent properties [19]. Firstly, the existence of a specific mouse does not depend on our feelings about it. Secondly, there are several features of that mouse that are mind-independent: for instance, that it has a certain mass, form, location, and chemical and physical composition. Thirdly, many other features of the mouse are mind-dependent: for instance, that we call it a mouse, that it is meant for controlling a computer, that this is Mary's mouse, and that Mary's mouse is a particularly good mouse.

Resolving the Ontological Status of a Subject of Study

Figure 1 presents two simple questions for resolving the ontological status of one's research subject. The first question resolves whether the subject of study is mind-dependent or -independent. In the case of mind-independent subjects, many studies still focus on their mind-dependent features, functions, or properties. Hence, the second question finds out whether the features studied are observer relative (mind-dependent) or intrinsic (mind-independent). That distinction is—much of the time—enough to frame ontology for a study.

Ontological status of research subject says nothing about how difficult is the science involved; it only denotes the subject's mode of existence. Research on both kinds of subjects can reveal interesting and important facts about the world, but those facts are of different types. Those facts that concern mind-independent aspects of the world have been called *brute facts*, and those facts that concern mind-dependent aspects of the world have been called *institutional facts* [19]. For example, it is a brute fact that transistors switching in processors cause heat. They cause heat no matter what we think about semiconductors, processors, current, and resistance. There again, it is an institutional fact that 8 bits

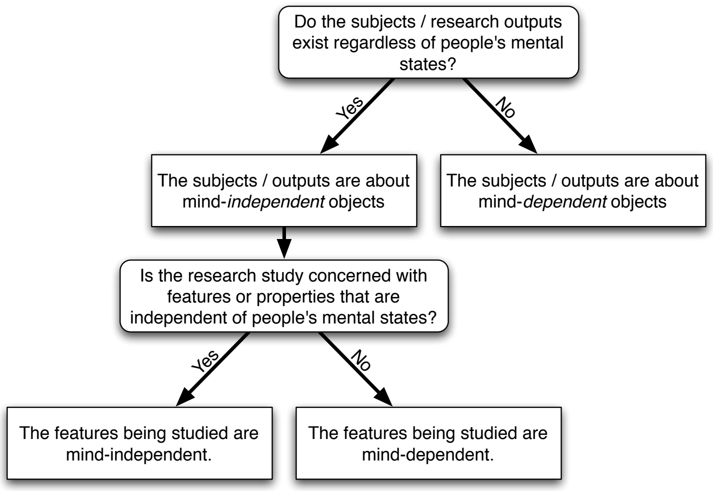

Do the subjects / research outputs exist regardless of people's mental states?
Yes
No
The subjects / outputs are about mind-independent objects
The subjects / outputs are about mind-dependent objects
Is the research study concerned with features or properties that are independent of people's mental states?
Yes
No
The features being studied are mind-independent.
The features being studied are mind-dependent.

Figure 1: Resolving the ontological status of elements of study.

make a byte: That fact is created and collectively maintained by people, and it is a fact due to people's collective agreement.

3. EPISTEMOLOGY

Epistemology refers to the theory of knowledge and is concerned with, for example, what knowledge is and how one may gain knowledge. Historically speaking, two major schools of epistemology have been empiricism, which emphasizes perception (such as research based on empirical data collection), and rationalism, which emphasizes reason (such as any logical or mathematical reasoning). The old philosophical dispute between empiricism and rationalism is often not a concern for computing students, although ignoring the fundamental differences between the two modes of knowledge creation risks some critical misunderstandings.

All scientific disciplines have some traditions and conventions concerning credible ways of acquiring and justifying knowledge claims. But computing is a diverse discipline, and spans from highly abstract topics, such as computability theory, to quite practical topics, such as information technology. Hence, understanding epistemological issues plays an important role in clarifying research studies in computing, too.

Epistemology: Why?

There are various kinds of reasoning in different branches of science. Different kinds of reasoning and different kinds of information all have their own limitations and pitfalls. For instance, mathematical and formal models are precise and unambiguous, yet they are confined to the world of abstractions and they fail to capture the unbounded richness of the physical and social world. Narratives and ethnographies are rich in dimensions and sensitive to detail, yet they are equivocal and context-dependent. Scientific experiments enable accurate prediction and statistical description, but cannot capture things like meaning and significance. Narratives have little predictive power, and formal proofs have little explanatory power regarding things like usability preferences and much of the human experience in general. There again, the predictive power of mathematical and computational formulations is uncanny: Computational models have a miraculous, "unreasonably effective" capability of accurately predicting things in seemingly unrelated domains [9]. Confusing the limits of different intellectual traditions undermines the credibility of a research study, and credibility in part involves the recognition of known pitfalls and problems related to each source of knowledge.

Although scientific experiments and empirical research are able to produce new knowledge about the world, they both entail various epistemological issues. For example, perception is limited as a source of knowledge, and it is not always trustworthy: dip a straight stick in water, and it looks like it is bent. Measurement tools may offer a very limited picture of the phenomenon they measure. Existing theories, experiences, memory, and presuppositions affect how people interpret what they perceive. However, not only empirical work, but also theoretical reasoning entails problematic issues ranging from affirming the consequent, to hasty generalizations, to equivocation, and beyond. Even introspection (examining one's own mental states) is fallible—memories fade, change, and some 'memories' never happened. In any research study that involves things like observational, reflective, or reflexive practices, one needs to be cognizant of the epistemological issues involved, so that one can take into account the possible errors and try to work out the research design to accommodate biases, misrepresentations, misunderstanding, and other kinds of problems.

On the most basic level, many research studies follow a question–answer structure, where the answer makes a claim or a statement about how things are. One of the central meanings of epistemology refers to the *objectivity or subjectivity of judgments or statements* [19]. Whereas in the ontology above things are either mind-dependent or -independent, in epistemology objectivity or subjectivity is a *matter of degree*: judgments can be between subjective and objective [19]. In everyday language, *objective* judgments are those that are independent of single people's attitudes towards them; often those shared judgments are called intersubjective. Respectively, *subjective* judgments are those

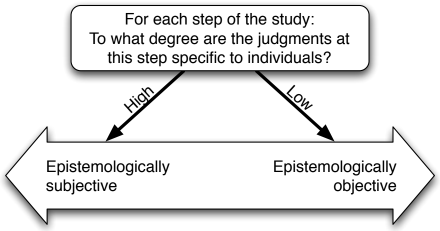

For each step of the study:
To what degree are the judgments at this step specific to individuals?
High
Low
Epistemologically subjective
Epistemologically objective

Figure 2: Evaluating the epistemological status of elements of study.

that depend on people's attitudes or feelings. Objective judgments are often called facts or knowledge, whereas subjective judgments are often called opinions, preferences, or beliefs.

Research studies on both mind-dependent and mind-independent subjects face similar epistemological questions, often discussed in different terms. There is the question of how closely does the data represent reality. There is the question of how strongly do the data support conclusions. There is the question of how certain the results are. There is the question of how well does the tool measure what it is supposed to measure. There are questions about the generalizability of results, and many more.

Epistemology: How?

One approach for reflecting on epistemological dimensions of a research study is to try to map in the same picture the question, a tentative (imagined) answer, and all the steps in between. Different steps may involve different kinds of epistemological stances: To what degree are the data epistemologically subjective or objective? Studies can be about things like people's preferences (more subjective), as well as about things like people's religious nominations (more objective, ascertainable from the population register). To what degree does the data collection itself involve interpretation versus recording of data? To what degree does the analysis involve interpretation versus mechanical processing? For instance, considering results, one can ask, "how much do these results depend on the researcher's and the reader's interpretation, attitudes, or preferences?" The more objective the results, the less they depend on any kind of interpretation.

The formulation of the research problem, aims, and research questions rely on existing research studies and researchers' interpretations of valuable and worthwhile questions, and the question could be about previously unknown things or conflicting views. Data is not collected randomly, and in the data collection process, only a minor part of infinite available data is collected, based on what is already known about the domain. The data is collected using formal or informal data structures that are created to record or model some aspects of the phenomenon as well as possible. Because data structures are built according to what one already knows about the domain, data cannot be independent of one's previous knowledge. Treating such data as if they were 'independent' empirical discoveries or theoretical results has been called 'inscription error' [22, p.52]. All results are also interpreted, often on multiple levels. Raw data are often interpreted for reporting results, and results are interpreted for discussion of findings—and in the end the critical reader and the scientific community are the final interpreters of the research report. There is no research without interpretation somewhere.

Epistemology: From Subjective to Objective

Many judgments depend on individual preferences. For example, the judgment that OS X is a good operating system is a matter of viewpoint, and that judgment varies by people's feelings about aesthetics, user experience, proprietary products, and various other things. So, the judgment that OS X is a good operating system is *epistemologically subjective* to some degree, and many may disagree with the judgment. There are many types of research studies that study epistemologically subjective matters (such as preferences and attitudes). Some interpretive research looks at things that rely heavily on individuals' personal states of mind—take, for instance, aesthetics or users' emotions regarding interface design. Many studies that try to understand individuals' meanings, interpretations, or feelings study epistemologically subjective matters.

There again, there are some judgments that are considered to be more than mere opinions. Many people may hold the same judgments (sometimes called intersubjectivity), or those judgments may form a part of a coherent structure of statements. For example, the statement that there are 7 OSI layers is an objectively ascertainable fact. That judgment is *epistemologically objective* [19] or *intersubjective* to a high degree. Note that although the OSI model—as a standard created by networking professionals—exists mind-dependently, statements about the OSI model can be objective in a sense that those statements cohere with commonly accepted standards or definitions; they are not only subjective opinions.

Intersubjective matters can also be studied in many different ways. Many mind-dependent subjects can be studied in a way that does not leave much room for idiosyncratic interpretation—take, for instance, studies on IEEE standards or commonly agreed models. Other research studies try to understand and interpret the subjective meanings and mechanisms between intersubjective things—take, for instance, studies on groupware or computer-supported collaborative learning. Many such subjects are an outcome of negotiation, common making of meaning, and social processes. Indeed, there is a continuum that ranges from intersubjective agreements that are fixed to a high degree (statements like "eight bits make a byte") to subjective preferences (statements like "pair-programming works well for me"). At the highly objective end of the spectrum, with very strong intersubjective agreement, are things like axioms in mathematics and transformation rules in logic.

Objective out of Subjective

If one's research data lies at the subjective end of the spectrum, that does not mean that results could not be at the opposite, objective, end of the spectrum. For instance, a presidential poll is essentially a survey on subjective preferences ("X would make a better president than Y would"). The results are highly objective facts, such as the fact that Barack Obama won 52.92% of votes in the American 2008 presidential elections. Surely such survey has various limitations, such as the fact that the respondents are not a representative sample of the population, but those limitations do not change the vote count, which is highly objective: the

Table 2: Example statements with various ontological and epistemological aspects of research.

	Epistemologically Subjective	Epistemologically Objective
Mind-dependent	1. *It's more fun to learn Python than it is to learn Java*	2. *The Java language has fewer reserved words than the Cobol language has*
Mind-independent	3. *Thermal noise in conductors is an undesirable phenomenon*	4. *The speed of electromagnetic radiation in vacuum is about* $3 \times 10^8 m/s$

election results are not an interpretation but as long as the rules of counting remain the same, any new count of the votes would come up with roughly the same result.

Epistemological standpoints affect, in many ways, what a research study can achieve. Research studies that deal with epistemologically objective matters (such as standards, theoretical constructions, or procedures) often yield results that can be right or wrong. Studies on epistemologically objective as well as subjective matters can yield statistics or empirically testable models. Studies on epistemologically subjective matters (such as opinions, preferences, or attitudes) can have a lot of variety in their results. Some of those studies come up with objective facts whereas others come up with interpretive, subjective results, such as case studies, ethnographies, or other types of rich description. The latter type of studies are often judged by their credibility, transferability, coherence, or confirmability.

The strength of many kinds of research is in the rich description they can provide of subjective things—of people's reasoning, motivations, fears, hopes, anxieties, expectations and so forth. In some other kinds of research, subjectivity is considered to be negative. All in all, many subjective aspects in research are unavoidable—from interpretation and communication of results to the choice of axioms—yet those often do not undermine the study's credibility much. Other subjective aspects can be avoided, and avoiding them may improve the credibility of results. For example, in research with human subjects, the individual researcher can make a sharp distinction between the beliefs and knowledge of researchers and beliefs and knowledge of the research subjects. Various techniques (multiple reviewers, coding books, etc.) can be employed to measure the level of intersubjectivity of results. Statistics provides a multiplicity of tools for establishing confidence on results.

It is important that researchers are aware of the epistemological aspects of their research; false assumptions may lead to decreased credibility of a research study. A researcher should have an understanding of the human condition and epistemic condition of the research subject. It is also important to separate between subjects, data, and results of the research—those are very different things.

4. DISCUSSION

All combinations of the ontological and epistemological aspects of research can be found in computing research. Table 2 presents examples of each combination. Sentences 1 and 2 on the first row concern mind-dependent things (programming languages), while sentences 3 and 4 on the second row concern mind-independent things (thermal noise and speed of light). Epistemologically speaking, statements in the first column are more subjective whereas statements in the second column are more objective.

In Table 2, the first statement concerns learning programming languages. Both learning as well as programming languages exist by virtue of our thinking, and are hence mind-dependent. As judgment 1 is highly contextual and dependent on one's opinion, it is epistemologically subjective. The second statement is also concerned with programming languages, which are mind-dependent. But judgment 2 is not a matter of anyone's opinion. Statement 2 is an epistemologically objective judgment about a mind-dependent thing, so statement 2 is an institutional fact.

In Table 2, the third statement concerns thermal noise, which is an intrinsic feature of conductors. Both conductors and thermal noise are mind-independent, for both exist independent of any observers or mental states. Statement 3 is, however, epistemologically subjective, as it is contingent on observers' preferences. Physicists, for one, do not consider thermal noise as desirable or undesirable—"it just is"—whereas for many engineers thermal noise is an unwanted phenomenon. In the fourth statement of Table 2, speed is an intrinsic feature of electromagnetic radiation, which does not know or care about any observers or anyone's feelings towards it. The statement 4 is not, however, anyone's subjective opinion, but an objectively ascertainable fact, and hence epistemologically objective. Being an epistemologically objective statement about an ontologically objective thing, statement 4 is a brute fact.

Some Difficulties and Relationships to Other Research

There are topics in computing that are difficult to classify in the categories presented above. Take theoretical topics, for instance. Ontology of mathematics is a notoriously tricky issue. For instance, Gödel [7, 8,§1] argued that mathematical objects exist in the very same manner as tables and screwdrivers do. In mathematics, ontological realists argue that there are mathematical objects that exist independently of people; ontological idealists argue that mathematical objects exist by virtue of our minds; and nominalists argue that mathematical objects do not quite "exist" [21, 25,226–227]. The image of mind-independence of mathematics was questioned by, for example, Lakatos [14] and Bloor [1]. In computing fields, De Millo et al. [3] discussed the social processes of knowledge creation in the discipline.

In computing fields, one bone of contention is the ontological and epistemological status of algorithms [6, 22]. Some argue that we find computations (and apparently algorithms) in the nature—that is, they are mind-independent [20, 15, 26]. Others argue that algorithms as well as properties of algorithms are mind-dependent (cf. [18]). However, in theoretical branches of computing epistemological and ontological positions in research are rarely debated, and epistemological and ontological positions do not affect method choice or research frameworks in theoretical fields. Cognitive science, with which computer science is often linked, poses another set of epistemological and ontological difficulties [19].

This article partly coheres with some research framework descriptions in computing fields. For example, Hirschheim & Klein [10] and Roode [17] adapted Burrell & Morgan's [2] four quadrants of sociological research to the field of information systems: those quadrants are the functionalist paradigm, the interpretive paradigm, the radical humanist paradigm, and the radical structuralist paradigm. The axes that cut through the four paradigms are, however, only partly the same as the axes in this study: As the study of Burrell and Morgan [2] was fully focused on sociological subjects, it did not touch on mind-independent matters at all.

Hence, the "subjective–objective" dimension presented by Hirschheim & Klein [10] and Roode [17] is the same as the epistemological dimension in this article, but the "order–conflict" dimension in those studies is specific to sociological research. Hirschheim & Klein [10] and Roode [17] divided ontology into realism—where "an empirical organizational reality that is independent of its perceiver or observer is believed to exist" [10]—and nominalism, where "reality is not given, immutable "out there," but is socially constructed" (*ibid.*). The problem with the type of "ontological realism" presented by those authors is that it argues for an independent social reality that would continue to exist even if all observers would cease to exist, and hence faces difficulties with metaphysics.

Conclusions

Computing's interdisciplinary linkages and computing researchers' increasing collaboration with people from other fields has led to a point where computing researchers too need to be able to understand and use some standard research terminology. In many research projects it is important that the authors can describe the mode of existence of their research subject and some epistemological aspects of their research data and results. Those questions—the ontological and the epistemological question—are in the research literature explained in various ways.

This paper proposes a simple way of framing the ontological and epistemological questions. The paper proposes that a computing research study's ontological and epistemological linkages can be determined through a few simple questions. Firstly, in many cases the ontological status of a study's subjects can be resolved by asking whether the subjects, their properties, or their features exist regardless of people's mental states. That question creates a division into mind-dependent and mind-independent subjects, which lead to very different sets of methods applicable. Secondly, the epistemological status of data, research results, and other judgments can be evaluated by considering the degree to which they depend on particular individuals' judgments. The ontological distinction is a clear "either/or" juxtaposition, whereas the epistemological dimension is a continuum; a "more-or-less" type of a question. These two dimensions offer a rough-and-ready starting point for discussing ontology and epistemology in computing theses.

5. REFERENCES

[1] D. Bloor. *Knowledge and Social Imagery.* Routledge & Kegan Paul, London, UK, 1976.

[2] G. Burrell and G. Morgan. *Sociological Paradigms and Organisational Analysis: Elements of the Sociology of Corporate Life.* Heinemann, London, UK, 1979.

[3] R. A. DeMillo, R. J. Lipton, and A. J. Perlis. Social processes and proofs of theorems and programs. *Communications of the ACM*, 22(5):271–280, 1979.

[4] P. J. Denning. Great principles of computing. *Communications of the ACM*, 46(11):15–20, 2003.

[5] J. J. Ekstrom, S. Gorka, R. Kamali, E. Lawson, B. Lunt, J. Miller, and H. Reichgelt. *Computing Curricula*, volume Information Technology. ACM, 2005.

[6] J. H. Fetzer. Program verification: the very idea. *Communications of the ACM*, 31(9):1048–1063, 1988.

[7] K. Gödel. Russell's mathematical logic. In S. Feferman, J. Dawson, S. Kleene, G. Moore, R. Solovay, and J. van Heijenoort, editors, *Kurt Gödel Collected Works (1990)*, volume II: Publications 1938–1974, pages 119–141. Oxford University Press, Oxford, UK, 1944.

[8] E. G. Guba and Y. S. Lincoln. Competing paradigms in qualitative research. In N. K. Denzin and Y. S. Lincoln, editors, *Handbook of Qualitative Research*, pages 105–117. SAGE, London, UK, 1994.

[9] R. W. Hamming. The unreasonable effectiveness of mathematics. *The American Mathematical Monthly*, 87(2):81–90, 1980.

[10] R. Hirschheim and H. K. Klein. Four paradigms of information system development. *Communications of the ACM*, 32(10):1199–1216, 1989.

[11] H. K. Klein and M. D. Myers. A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly*, 23(1):67–94, 1999.

[12] D. E. Knuth. Theory and practice. *Theoretical Computer Science*, 90(1991):1–15, 1991.

[13] T. S. Kuhn. *The Structure of Scientific Revolutions.* The University of Chicago Press, Chicago, USA, 3rd edition, 1996.

[14] I. Lakatos. *Proofs and Refutations: The Logic of Mathematical Discovery.* Cambridge University Press, Cambridge, UK, 1976.

[15] S. Lloyd. *Programming the Universe: A Quantum Computer Scientist Takes on the Cosmos.* Vintage Books, London, UK, 2007.

[16] M. D. Myers. Qualitative research in information systems. *MIS Quarterly*, 21(2):241–242, 1997.

[17] J. D. Roode. Implications for teaching of a process-based research framework for information systems. In *Proceedings of the International Academy for Information Management Conference*, Orlando, Florida, USA, 1993.

[18] J. R. Searle. Minds, brains, and programs. *The Behavioral And Brain Sciences*, 1980(3):417–457, 1980.

[19] J. R. Searle. *The Construction of Social Reality.* Penguin Press, England, 1996.

[20] C. Seife. *Decoding the Universe. How the New Science of Information is Explaining Everything in the*

Cosmos, from Our Brains to Black Holes. Penguin Books, London, UK, 2006.

[21] S. Shapiro. *Thinking About Mathematics: The Philosophy of Mathematics*. Oxford University Press, New York, NY, USA, 2000.

[22] B. C. Smith. *On the Origin of Objects*. MIT Press, Cambridge, Mass., USA, MIT paperback edition, 1998.

[23] M. Tedre. Computing as engineering. *Journal of Universal Computer Science*, 15(8):1642–1658, 2009.

[24] M. Tedre. Computing as a science: A survey of competing viewpoints. *Minds & Machines*, 21(3):361–387, 2011.

[25] M. Tedre and E. Sutinen. Three traditions of computing: What educators should know. *Computer Science Education*, 18(3):153–170, 2008.

[26] S. Wolfram. *A New Kind of Science*. Wolfram Media, Champaign, IL, 2002.

Analysing computer science students’ teamwork role adoption in an online self-organised teamwork activity

Rebecca Vivian Katrina Falkner Nickolas Falkner
The School of Computer Science
The University of Adelaide
Adelaide, South Australia, 5005
firstname.lastname@adelaide.edu.au

ABSTRACT

Computer Science (CS) professionals are regularly required to work in teams to solve complex problems. Agile software development processes are increasingly popular in organisations as a method for teamwork but the self-organising nature of the method and lack of strictly allocated roles means that graduates need to know how to adopt and transition between roles effectively. While online teamwork makes team processes and behaviours transparent, educators are often confronted by the amount of data and difficulty in how to judge roles and behaviours to provide meaningful feedback to students. Furthermore, assessment of teamwork does not necessarily ignite a need to identify roles and behaviours as feedback is usually based on the product, rather than processes and behaviours.

Using Dickinson and McIntyre’s teamwork roles, we extend the framework to include explicit behaviours to analyse one class of students’ self-organised team interactions in an online discussion space as solved open-ended problems. The collaborative activity did encourage role adoption, however not all students moved fluidly through the roles. Despite the lack of defined roles, one or two students adopted leadership roles, but attempts at leadership were not always successful. We discovered other less-obvious roles were equally important for maintaining and progressing team discussions. In this paper, we discuss the roles that emerged and suggest strategies for encouraging and assessing online teamwork. Our framework may prove to be a guide for others seeking to analyse students’ teamwork and provides a guide for what behaviours a teacher might look for in online environments. Our findings support the need to develop tools that provide real-time visual feedback to students and teachers about student behaviour and roles when undertaking teamwork in online spaces.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Collaborative Learning

General Terms

Human Factors.

Keywords

Teamwork; collaboration; human-to-human; students, team roles.

1. INTRODUCTION

Computer Science (CS) professionals are often required to work in teams on complex software development projects that require them to solve problems and accomplish complex tasks [1]. The significance of teamwork to the CS profession is acknowledged by CS employers, who identify that the ability to engage in teamwork is critical to CS professional practice and therefore they claim to seek graduates who demonstrate team skills [2, 3]. However, university students reportedly struggle with teamwork, in terms of producing software designs [4] and in applying collaborative skills [5]. Moreover, many CS graduates continue to struggle with teamwork in their new workplaces [6-8] and struggle to voice when they do not understand or know how to achieve a teamwork task [5-7], which may cause significant problems for their team and goal-attainment. Consequently, CS employers identify teamwork as an area in need of significant improvement [9].

Agile software development processes [10] are increasingly popular in organisations as a method for teamwork but the self-organising nature of the method and lack of strictly allocated roles means that graduates need to know how to transition and adopt new roles as their team requires. This places further pressure on graduates to have a sound knowledge of effective teamwork practice and roles; something that they currently lack. As a result authors have called for more *proactive* measures and teaching of teamwork in CS university programs [5].

To provide students with opportunities to practice teamwork and role adoption at university, students are often required to work in teams with peers on collaborative activities or group projects. However, educators are faced with issues in assessing teamwork as the processes are often “invisible” and when teamwork is captured online, discussions may be large and complex in nature. While some students may emerge as “leaders” and others “inactive”, the complex nature of roles that students adopt in their teams and how they move between roles may not be so obvious or easily identifiable. Such issues are hampered by time constraints that teachers often face. As a result, assessment of teamwork usually focuses on the product, rather than on the processes that led to the creation of the product [11]. Where assessment of “teamwork” does occur, it is typically through self- or peer-assessment and is usually based on post-reflection of self- and peer-contributions to the product. Furthermore evaluations typically focus on contributions in terms of workload, rather than on performance as a team member. Despite feedback being an important part of students’ learning development, such factors make it difficult for educators to provide meaningful feedback on students’ teamwork behaviour. Feedback is a crucial part of the learning process as it provides learners with information about their performance or understanding toward goal attainment [12]. Students may use feedback to alter their behaviour to meet their goal, or by teachers who can alter learning activities or guide students toward their goal. Since employers identify teamwork as critical to the CS profession, assessment should not only include feedback about technical knowledge but explicit and accurate feedback about teamwork performance.

Koli Calling '13, November 14 - 17 2013, Koli, Finland
Copyright 2013 ACM 978-1-4503-2482-3/13/11…$15.00.
http://dx.doi.org/10.1145/2526968.2526980

As a result of the lack of assessment and feedback of teamwork in CS, it is argued that CS education treats teamwork as something that students will develop through 'experience' [5] by partaking in a CS degree, rather than explicitly taught and assessed. Little attention is focused toward providing students with feedback about how to improve their teamwork behaviour or which roles they adopt and could develop in future collaborative activities, in preparation for the CS profession. Furthermore, there is a lack of research that seeks to investigate assessment of CS teamwork roles.

This paper reports on our endeavour to develop a framework for identifying explicit behaviours that align with Dickinson and McIntyre's previous model of teamwork. Using the framework as a guide, we code students' utterances at particular behaviour "nodes" to 1) determine if content analysis of behaviours can provide information about the roles students are adopting in their teamwork activity, and 2) investigate the extent students adopt particular teamwork roles in the online collaborative activity. We use the results to inform 1) what roles students are adopting in online self-organising teams to solve complex open-ended CS questions, 2) to determine if the identification of roles can inform us about how to adapt the course to encourage role adoption and development and 3) to determine if such a process of analysis would prove to be useful for devising automatic feedback tool to students and teachers in the future.

This paper is situated in the context of reflecting on a re-design of a number of key courses across the Computer Science curriculum to incorporate activities that involve collaboration and content creation, with one aim being to encourage students to actively participate in teamwork. We explore mechanisms for evaluating the agile teamwork roles that may provide indication of performance and in doing so observe CS students in an online Computer-Supported Collaborative Learning (CSCL) environment as they solve open-ended problems in the domain of networking. We construct a coding framework devised on the work of Dickinson and McIntryre's teamwork model [13]. This work helps us understand the roles that students adopt during teamwork and further facilitates academics in providing guidance to students regarding role occupancy, efficacy and management, and how students can develop effective online teamwork processes.

2. RELATED WORK

Our discussion of related work to this study begins with a discussion of the importance of teamwork to CS and teamwork roles identified in the literature. Our discussion concludes with a discussion about how teamwork is currently assessed in higher education and our motivation for the study.

2.1 Teamwork in Computer Science

Online teams or *virtual* teams are 'geographically, organizationally and/or time dispersed workers brought together by [ICTs]' to accomplish a task [14]. Although "teamwork" refers to team performance, effectiveness, or the individual team members' contributions to the team [11], team*work* does not necessarily mean team *success*. Success may be influenced by factors such as individual contributions, the context, and the history of team members. Successful teams are identified as agile, self-organised, have defined objectives, monitor and evaluate progress, and have equal member participation [5]. In virtual teams, those who reflect on group processes and employ strategies to improve team performance are more successful [15]. Computer scientists are increasingly required to collaborate across distributed networks and use ICTs for teamwork purposes as they are affordable (or free), allow for collaboration and sharing, regardless of time or place, and are part of a computer scientists' professional's immersive environment.

The agile manifesto [16] is one popular software development method that requires individuals to operate in self-organising teams and is a method which favours individuals and interactions, working software, customer collaboration and response to change [17, 18]. Reflection is identified as a key behaviour, where teams determine more effective ways of operating and adjust their behaviour accordingly. In preparation for self-organised teamwork approaches like this, we need to ensure that our students are prepared to proactively participate in a group and that they know how to adopt certain roles as their group requires.

2.2 Teamwork Roles

In teamwork, members may be assigned specific roles or allowed the freedom to self-organise. In self-organising teams, members often display spontaneous role behaviours and, despite the omission of a designated leader, planning behaviours still emerge [19]. However, it has been found that a lack of coordination behaviours exist in comparison to teams with allocated roles [20]. Furthermore, leadership is not usually dispersed evenly among members of self-organised teams and leadership behaviour may alternate between individuals, depending on who has the specific abilities and skills required for the team at a point in time [20].

To encourage students to practice various teamwork roles and build confidence in their CS program, one method used might be to scaffold students in various team roles using the pedagogical approaches similar to Contributing Student Pedagogy (CSP) [32]. The educator may demonstrate particular roles or allocate and rotate roles to students in collaborative tasks. In these tasks, students are implicitly or explicitly encouraged through team roles and are made aware that one can transition between roles. To teach the roles, educators might adopt published models of teamwork as a guide. There have been a number of teamwork models that have emerged in the literature and we will proceed to discuss some of the models and their relevance to CS.

Teamwork role identification is a topic of interest in management science and psychology [13, 21-23]. In response to the recognition that research on teamwork was fragmented, Salas, Sims [22] undertook an extensive review of the literature to classify common teamwork components and as a result identified five core components of teamwork: *team leadership*, *mutual performance monitoring*, *backup behaviour*, *adaptability*, and *team orientation*. While an advantage of their model is the solid grounding in literature, a critique of their work is that their model is not focused toward self-managing teams [18].

One teamwork model that focuses on practical use and self-organising teams [18] is the Dickinson and McIntryre model [13]. In a review of the literature on teamwork, Dickinson and McIntyre [13] identified a need to focus on components of teamwork to develop assessment measures. In their work they classified seven core components of teamwork: *Team Leadership, Team Orientation, Monitoring, Coordination, Communication, Feedback* and *Backup Behaviour* (see Table 1) and a model to guide the construction of measures. The authors recommend three formats for constructing teamwork measures: a behavioural observation scale, behavioural summary scale, and behavioural event. While their model is focused on practical use and self-organising teams [18], the measurement relies on how teams respond to critical events according to scales, rather than the explicit identification of behaviours. Furthermore, because the measures involve post-critical incident evaluation they may not necessarily be useful for real-time evaluation of online behaviour. Moe et al. [17] used the teamwork model in a holistic investigation of essential teamwork components found in agile software teams in organisations. They discovered barriers to team effectiveness existed in problems with team orientation, team leadership, coordination and division of work. They claim that within organisations there are significant challenges with moving from individual work to self-organised teamwork [17] and that some leaders focused on their own processes, leaving others to ponder what they are doing [24].

Hoda et al. [25] also investigated agile software teams in organisations and identified certain roles, similar to those previously discussed, important for agile teams. These six roles were categorised as mentor, co-ordinator, translator, champion, promoter, and terminator. The findings are important for understanding roles that professionals adopt to develop software, however we are interested in developing students' role adoption and this is better enabled by exploring teamwork using existing roles so that the identification of underused roles can be made. Furthermore, these examples are situated in organisations, where team members are often working on long-term projects with colleagues whom they are familiar with, however, there is a great deal that can be learnt and improved by studying the way that CS students come together and behave for small team projects.

2.3 Teamwork Assessment & Feedback

In higher education, we hope that students in self-organised teams adopt and transition between various teamwork roles with equal participation. In reality, many students express a dissatisfaction toward teamwork because of negative experiences with the inequality of member participation [26], the presence of perceived "slackers" [27] and pressure to complete a majority of the workload when peers are not contributing [26]. Assessment at university does not necessarily favour students in these situations as teamwork assessment often focuses on the product, rather than individual contributions and team processes. To address this concern and make students more accountable to the team project, efforts are made to assess students based on their involvement, such as by peer review, self-reporting, and group assessment reports. However, a review of the literature on these methods reveals that the components measured largely focus on student contributions. Some examples are the use of the 'contribution matrix', percentage-sharing scheme, and self-and peer-ranking methods [28]. Such an assessment focus may result in team fragmentation and a concentration less on *teamwork processes* and more on *individual achievement* and *knowledge contribution*.

There are arguments that if teamwork is not assessed in course activities, students generally will not take it seriously or do it [28]. While teamwork assessment that focuses on contribution may encourage student contributions to the task, if we want students to begin thinking about the way they engage in teamwork by their behaviour, roles and types of contributions, we need to find ways to assess these aspects.

2.4 Motivation

There is considerable unanimity over workplace skills required in Australia and overseas and while the non-technical skills employers demand have not changed much over the last 50 years, the expectations and priority given to certain skills has changed [29]. A recent report reveals a growing trend toward a need for higher-order skills, which includes 'the skills and knowledge to work effectively as a member of a team' [29]. Simply measuring by 'contribution' or 'product', often captured by assessment methods, does not capture the dynamic and complex skills and knowledge required to operate effectively as a team member; for example, the ability to fluidly move between 'roles' as required by a team. In the context of this study, students are responsible for developing their own graduate attributes, with teamwork being one [30]. More specifically, it is envisioned that the CS will produce graduates who operate effectively as members of multi-disciplinary and multi-cultural teams, who have the ability to lead or manage as well as behave as an effective team members [31].

Recently, our Computer Science programs have undergone a review and update of its curriculum, using Contributing Student Pedagogy (CSP) as a guide, to re-design key courses across the curriculum and instil a focus on student content-creation and collaboration. For an in-depth discussion about the approaches and design undertaken these are reported in a paper that detail the mixed-methods participatory action-research used to inform the re-design [32]. One focus of the re-design was informed by the use of CSP in conjunction with Dickinson and McIntyre's teamwork model. An aim of the re-design was to incorporate the value of student perspective and establish student-to-student networks in which students are active and willing participants.

Teachers will not be available to guide teamwork practices when graduates enter the workforce and this leads us to ponder whether the CSP and teamwork activities we implement prepare graduates to engage in fluid role adoption and participate effectively in self-organising teams. Our experience tells us that, at a surface glance, some students are more active than others are and little is known about what characterises team involvement and what those behaviours mean for teamwork. We often assume that students develop teamwork skills through participating in teamwork activities [5], however, how can we ensure our students have effective teamwork skills and how can we assist students to develop skills they are lacking?

3. RESEARCH METHODOLOGY

This research project sought to answer the following research questions:

1. Can a framework with the identification of teamwork role behaviours assist us in identifying teamwork roles students adopt in an online environment?
2. What teamwork roles do students adopt in self-organised teams to CS solve problems?

An instrumental case study approach allowed us to use a particular case to understand students' self-directed teamwork roles and collaboration in a CSCL environment [33, 34]. Additionally, it allowed us to refine and test our coding framework. A small case size, although not generalisable, allows for an in-depth examination of the interactions and behaviours between individuals [35]. In this study, we examined a final year undergraduate CS course because students in this year level have had practice in teamwork activities throughout their degree and we hope students are able to demonstrate teamwork skills and adopt roles as required.

3.1 Case Context

This case study is conducted with one CS course at the University of Adelaide. The course chosen as the case study for this research structured around the discussion of complex problems inherent with computing operations carried out across a network, over more than one computer. We deliberately introduce the students to practical problems and thought experiments that are open-ended and we expect students in this final year undergraduate course to apply their knowledge and to synthesise solutions. The summative assessment includes three individual assignments and an exam. In addition, the course involves three collaborative tasks that contribute a modest 2% of the student's final grade. The role of the collaborative sessions is to expose the students to difficult problems that require them to consider all of the possible issues that could occur during tasks carried out in this domain. The first collaborative task was conducted as a face-to-face activity, and the final two collaborative tasks, which were observed for this study, were conducted online. The first task asked students how they would implement an Adaptive Timer System based on a previous assignment that used the Java RMI system. Students were required to write a report to provide architecture for implementing the adaptive time on both the client and server side and consider the design issues. The second task invited students to devise a network design and discuss issues associated with the creation of a Massive Multi-User Online Roleplaying Game experience.

The instructor randomly allocated the 26 students to one of eight groups for the collaborative tasks. Each group was required to construct a joint wiki response in the collaborative software site, Piazza [36], by editing the contents of a page, to address the problems posed. The Piazza framing for this concept includes the allocation of two development spaces: one for students and one for the instructor. Instead of the traditional comment-response model found in traditional online environments, a Piazza solution exists as a whole. This allows students to review the current "best" solution easily without having to compose a set of existing responses in a potentially complicated semantic alignment. It is unlikely that one student holds all of the expertise in an area, which we hope encourages fluid transition of roles and the sharing of information between those who hold particular expertise at a point in time. Students were instructed to hold their team conversations using the Piazza discussion feature, positioned underneath the wiki. This paper reports on analysis of the discussion component of the activity.

3.2 Coding Framework and Content Analysis

We chose to focus on the identification of roles based on those proposed by Dickinson and McIntyre [13] because the roles encapsulate those we hope CS students would adopt and they are commonly taught to our students in CSP activities [32]. Dickinson and McIntyre [13] provide a summary description and allude to what each role's behaviour might involve, however, they lack explicit behaviours to look for, particularly in online interactions.

In their framework, Dickinson and McIntyre adapted the critical incidents methodology [37] to construct teamwork measures, using anti-warfare teams as a case to guide their measures. Although, the measures are valuable, they are critical incident reports and are for post-hoc measurement of self-organised teamwork. When studying the roles that our CS students adopt, we would like to measure their activity through the identification of particular behaviours associated with roles, with the intention to make this automated in the future. While scale measures of competency may be useful, we wished to make these as accurate as possible using frequencies from the utterances of student activity. Therefore, we needed to develop a measure to appropriately identify behaviours one would expect to see in students problem-solving in CSCL environments. To achieve this we examined online CSCL behaviours identified in frameworks from argumentative knowledge construction [38], collaborative knowledge construction [39], and learning processes in CSCL environments [40] and selected the identified behaviours that matched role behaviours and sorted these accordingly, as we saw fit, to each of Dickinson and McIntyre's seven components of teamwork (see table 1). While the CSCL studies concentrate on understanding behaviours and processes exhibited in CSCL environments, rather than role adoption among team members, we feel that the behaviours they identify are useful as indicators for coding behaviours that can then be sorted within team roles.

Table 1: Teamwork Roles (Dickinson & McIntyre) with the coding behaviours

Components of Teamwork & Description ([13], p. 25)	**Role behaviours**
Team Leadership: Involves providing direction, structure, and support for other team members. It does not necessarily refer to a single individual with formal authority over others; several members can show team leadership.	Restate problem; identify problem space; identify new problems/components of the problem to address; initiate group planning; elicit group responses/involvement.
Team Orientation: refers to the attitudes that members have toward one another and the team task. It reflects acceptance of team norms, level of group cohesiveness, and importance of team membership.	Social presence aspects and group cohesion: use of inclusive pronouns; communication that is social or a function of social communication; expression of emotions/humour
Monitoring: refers to observing the activities and performance of other team members, It implies that team members are individually competent and that they may subsequently provide feedback and backup behaviour.	Monitor group processes, monitor group progress, monitor group activities.
Coordination: refers to team members executing their activities in a timely and integrated manner. It implies the performance of some team members influences the performance of others.	Report learning activities and processes; initiate goal setting.
Communication: involves the exchange of information between two or more team members in the prescribed manner and by using proper terminology. Often the purpose of communication is to clarify or acknowledge the receipt of information.	Monitors/expresses own cognition or monitors cognition of peers; identifies conceptual problem space (knowledge required); elicit peer elaboration of information; rephrase previous claims (own or others). Responds to others' initiated goal-setting or planning.
Feedback: involves the giving, seeking, and receiving of information among team members. Sub-components: giving, seeking, and receiving feedback.	
Giving feedback: refers to providing information regarding other members' performance information among members	Evaluate or assess peer contributions (express agreement/disagreement, counterarguments, and critical evaluation).
Seeking feedback: refers to requesting input or guidance regarding performance.	Seeks critical evaluation or feedback to contributions.
Receiving feedback: refers to accepting positive and negative information regarding performance.	Accepts feedback about performance.
Backup Behaviour: involves assisting the performance of other team members. This implies that members have an understanding of other members' tasks. Sub-components: seeking and supporting feedback activities.	
Seeker: Members are willing and able to seek assistance as required.	Seek information from group; seek social assistance about group task/processes from group.
Supporter: Members are able and willing to provide assistance as required.	Elaborate on previous contributions, integrate ideas; revise previous contributions.

We also included regulation behaviours from the works of Zimmerman [41] to capture behaviours that involve students reflecting on their own and their team's processes and included any of those relevant behaviours within the teamwork roles. We also saw the value in dividing Dickinson and McIntyre's roles of Feedback and Backup Behaviour into their sub-categories for analysis as "seeking" and "providing" behaviours are different in their purpose and we wanted to know when students were providing support or seeking support.

We employed directed content analysis [42] to code students' online teamwork discussions, using the framework behaviours in Table 1. For actual coding purposes we developed a fully comprehensive guide that included each behaviour within the particular role. Each behaviour included a description of what the behaviour might involve and a quote as an example, taken from an initial pilot exploration of the students' online activity. For an example, we include in table 2, the behaviour of 'identifying the problem space' taken from the 'team leadership' role. Under each teamwork role, more indicators would follow which are those listed in Table 1 under the column 'role behaviours'. The developed coding guide was presented by the researcher to two CS academics at the university, experienced in CS education research and teaching CS to university students. The two academics confirmed that the behaviours aligned with the roles and that the descriptions and examples of behaviour in the guide were appropriate.

Table 2: Example of the coding guide

Behaviour	Description	Example
Team Leadership		
Identifying the problem space	Retelling or rephrasing of the original problem or questions.	"I think there are a number of points we need to consider…" [student lists points]

An important aspect of content analysis is to clarify the unit of analysis to be coded prior to coding. Previous studies that have involved coding student CSCL collaborations [19, 39, 40] and teamwork roles [43] have coded *discussion posts* by students in their entirety as the unit of analysis. However, the basic unit of analysis in this project were *coding units* [44], which included sections of text responses or "utterances", of any size. Sections of text, such as a sentence, word or phrase, were coded as long as the selection represented a single category in the framework [42]. Focusing on the micro-level of text in postings is important because broad-scale coding may lose fine-grained behaviours and multiple role adoption that might be evident in a single post.

The use of numbers in qualitative content analysis provides precision to the frequency, observations, and patterns of a particular phenomenon being investigated [45]. Our desire was to extract quantitative frequencies of the content analysis and so we coded data as being within a mutually exclusive category [42]. Frequencies of student activity in NVivo were exported to a Microsoft Excel spread sheet, for further examination. The content analysis frequencies, combined with qualitative examples of student activity allow for a rich picture of student teamwork practices.

Discussion data from Piazza were coded using the qualitative software NVivo 10. One researcher, experienced in using NVivo for content analysis, methodically worked through the discussion data for each group, coding the conversation data to the relevant behaviour category, which was organised under the teamwork roles. To ensure reliability, the coding process was iterative by reviewing coded content and refining codes. The researcher began coding the activity, starting with two student teams. Before going further the researcher 'explored' the data by re-reading the coded text and checking the coded text against the coding guide. These processes were undertaken to ensure coding was consistent and reliable. The researcher continued to complete the coding for all teams. The researcher clarified coding results with the two CS academics in stages throughout analysis. Our future work will include conducting inter-rater reliability testing on more samples of data.

4. RESULTS

The following sections present the findings obtained from the analysis. We begin with an examination of the overall frequencies of roles across all tasks and groups and present how students were adopting the role with examples of students' quotes and summarised behaviours. The second section examines role distribution across the groups and we select 2 groups that are contrasts of one another to provide a more detailed analysis of how the identification of teamwork roles can provide insight to how teams are performing and interacting.

4.1 Frequency of Teamwork Role Behaviours

Table 3 presents the coding frequencies for the teamwork roles across both collaborative tasks, calculated according to the total of teamwork behaviours coded. The findings suggest that students were engaged primarily in leadership, team orientation and response behaviours; however, less frequently displayed behaviours that involved monitoring and regulation of group processes.

Table 3: Frequencies of teamwork behaviours across all groups and activities

Teamwork categories	Freq. (%)	Mean	Std. Dev.
Orientation	47.2	27.5	23.4
Leadership	15.5	9.1	8.7
Feedback (provider)	15.1	8.8	6.2
Coordination	3.1	1.8	2.9
Monitoring	2.2	1.3	2.9
Communication	7.8	4.5	3.1
Backup (supporter)	4.7	2.7	2.4
Feedback (seeker)	2.3	0.9	1.5
Backup (seeker)	1.6	2.7	2.4

The following section provides qualitative examples of how students were engaging in the various active and passive teamwork roles within this online collaborative context.

4.2 Students' Teamwork Role Behaviours

4.2.1 Team leadership

A number of the team leadership behaviours were initiated (6.8%) posts, and therefore, it was not surprising to find students exerting a leadership role in the first post. For example, one student prompts:

Hey guys, let's get it started…. Let's identify the problem first so that we can derive a solution (student 3).

Leadership behaviours involved instances where students would direct the group toward an aspect of the collaborative problem. For example, students would specifically re-state or rephrase the question or component of the problem. Students typically used this as a strategy to draw attention to what aspect of the problem they were referring to, or to encourage members to respond to an aspect of the problem. For example, one student proposed to his peers:

I'm expecting some kind of further discussion on ALL those areas (student 12).

Leadership also involved instances where students delegated tasks to specific individuals. For example, by asking: 'Could [student 25] do the honor to do the full revised answers for question2?' (student 11). Similarly, a number of students sought group consensus on particular aspects of the task so that the group could progress. This was particularly noticeable when the conversation became stagnant or divided. For example, one student (5) makes a stand and requested of their group: 'ALL AGREE???'. By someone taking charge, fruitful conversations would follow. Leadership behaviours complemented Team Coordination behaviours.

4.2.2 Coordination

Coordination involved goal-setting and reporting activities. Across all groups, only one student initiated setting a goal for the completion of the assessment task prior to the deadline. He did this by requesting: 'Shall we give ourselves a cutoff timing to finalize our question by Thursday evening?' (student 11). The other students eventually agreed to this proposal; however, it took the student three comments in each collaborative task for the group to respond. Another aspect of coordination involved students stating their task performance to the group. For example, one student reported to their group: 'Have added my answer for question one' (student 4). Often such an action would be one of the first initiated posts, to which discussions about the contribution, or further development of the solution would follow. Statements of performance appeared to be a way to manage group task completion and notify team members of progress.

4.2.3 Monitoring

Monitoring involved awareness of group processes and performance and acting on that knowledge. This included judgments about discussion progress toward the task, such as, '... I think the discussion for this particular follow up is at least getting somewhere' (student 12). Students were also displaying monitoring behaviours about the topic being discussed, and would comment when they perceived conversations digressed off-topic, bringing the focus back to the task. This would involve a student making a statement that signals to the group the conversation is moving away from their desired outcomes. For example, one student notices that, although the team conversation is active, the discussion is moving away from what they are being asked to achieve. As a result he suggests: '[s]ince this is rather off-topic, we may continue discussion offline' (student 7). In addition, students engaged in assessing and monitoring group processes to provide group direction toward certain tasks, for example: '[t]onight is the cutoff time, it would be great to start revising the answer' (student 11). Alternatively, students made statements when they perceived the group had reached reasonable completion of a task. Such task-navigation behaviours were important for team processes.

4.2.4 Team orientation

Team orientation characterizes the level of social presence one brings to the group, rather than a specific role one would adopt in a team. Such characteristics included the expression of emotions, expressed through emoticons and the use of humour. Socialization was another aspect of this component, which included the use of communication that was social in its nature, such as 'hi', 'hey' and 'thanks' and the use of inclusive pronouns such as 'we', 'our' and 'us' when discussing the task. Students would often refer to others' messages, or address specific team members by following up with the use of member names. Although this aspect of teamwork is not necessarily an active role, in the sense of leadership, it appeared that students who engaged in such practices were actively involved and connected to their group in a social and task manner.

4.2.5 Communication

Communication essentially involved the clarification and reception of information. Students displayed communication behaviours by identifying the conceptual space, such as, '[w]e should probably first identify the two types of locks' (student 6). Communication also involved students monitoring their own cognition. For example, students would share their understanding, or lack of to the group by stating something such as, '..not quite sure what this questions means… '(student 23). To seek clarification and reassure their understanding, students would elicit elaboration from group members, such as by requesting: 'Can [you] explain what retries are you referring to?' (student 25). In response, students would rephrase contributions to clarify understanding. In addition to monitoring their own cognition, students displayed an awareness of their peers' understanding by making an announcement and/or rephrasing a previous contribution. This was important to correct misunderstandings, particularly in terms of course content or when a member appeared to misunderstand a proposed solution. One example is when a student stated: 'maybe i didn't phrase it correctly' (student 19). Additionally, students would also follow-up, or check group cognition by eliciting a response that sought to clarify understanding. Communication behaviours were important for information sharing and knowledge construction and also allowed for students to peer-tutor one another and share expertise and knowledge.

4.2.6 Feedback seeking

Feedback seeking entailed occasions where students would seek feedback about contributions they posted. This would typically involve requests from students to review their wiki contributions, such as, 'Feel free to contribute any suggestions or improvements' (student 4). Alternatively, students would explicitly seek peer evaluation, such as 'do you think it will be better this way?'(student 11). Feedback seeking behaviours generally resulted in students critiquing and evaluating peer contributions.

4.2.7 Feedback provider

Feedback provider included behaviours where students provided evaluations to their own or other's contributions. In addition, students were engaged in critical feedback to contributions through statements of agreement, disagreement, or judgments. We observed some critical feedback such as the following:

'I think that if we send out request and probe together we can reduce the delay of the first probe. But if we send out probe first to test if server is alive it sounds weird…' (Student 12)

Feedback also included instances where students would compliment one another, such as: 'I believe (student) and (student) have indicate some great points for the RPC timer'. Such evaluations were supported with reasoning. At times, this feedback was requested by feedback seeking behaviour. Feedback was an essential component to the knowledge construction process and a way for students to derive a successful solution to the problem.

4.2.8 *Backup seeking*

Backup seeking comprised of instances where students would seek help from their group in relation to task processes, such as, where to discuss the content, where to post responses, or the assessment details. One example is when a student asked:

are we base on the future rmi or multi thread server system that we discuss in the previous collaborative? (student 20).

In addition, another backup seeking behaviour involved students seeking information from their group about course content or task content.

4.2.9 *Backup behaviour*

Backup behaviours were supportive and involved those that involved elaborating on existing contributions, integrating ideas (building a consensus between members) and revising work. These were often in response to group member requests, or self-initiated. The behaviours appeared to 'keep the peace' and assist in the construction and application of knowledge toward the problem. For example one student attempts to integrate both student contributions by proposing: 'I think we should incorporate a bit of both solutions' (student 15). Such behaviours were more so about building on existing ideas, than initiating new contributions.

In this context we did not observe students providing feedback about performance although we did observe them providing feedback about other aspects, such as content or proposed solutions. Rather than explicitly acknowledging the feedback provided, the feedback would ignite other responses or assist in the continuation of discussion. As a result, we did not code any behaviours for 'feedback receiver' but this is not to say that it would not arise in other CS teamwork contexts.

These roles and behaviours have provided insight into the overall processes. The following section will explore the adoption of roles by individuals within each group.

4.3 Role Distribution

Figure 1 reveals that all groups had higher levels of Orientation and Leadership in comparison to their other roles, except group 3 who had low orientation and some evidence of leadership. Orientation represents cohesiveness of a team, which indicates that the team displayed low cohesiveness, but overall the group did have low interaction altogether. Other role adoption is generally low across all groups, with a number of roles having '0' frequency or just above.

We selected two groups to highlight and discuss role adoption according to the frequency of their behaviours for each role, for the purpose of this paper. We selected, group 6, with both high activity overall (Figure 2) and, group 3, with low activity (Figure 3). The graphs illustrate the dispersion of individual's behaviour frequencies for each role within groups.

In group 6, all members were actively involved in editing the wiki response and contributed in some way to the discussion by integrating and responding to ideas. Two of the four students displayed frequent leadership behaviours, with another providing some leadership support. Student 22, however, although not obvious when looking at the discussion board, demonstrated less self-initiated role adoption. Overall, this group, across both collaborative activities demonstrated role adoption primarily in leadership and orientation, but also coordination, providing support and feedback and somewhat evidence of monitoring group processes and seeking feedback. This team produced a wiki response that was complex and detailed.

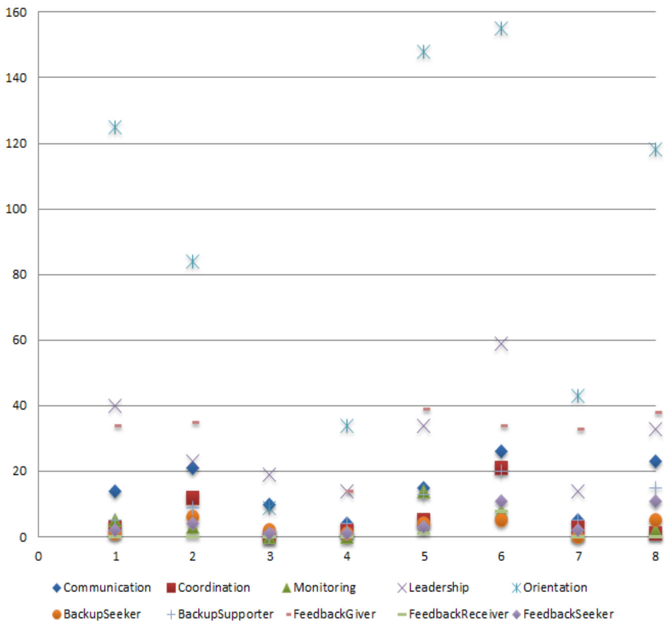


Figure 1: Scatterplot of observed role behaviour frequencies present in each group

In comparison, role adoption activity in group 3 was minimal. The discussion board comprised of two threads, where one student attempted to initiate discussion among members. However, despite attempts to lead the group, none of the other members responded. This resulted in the "leading" student (19) to create the wiki response primarily on her own. In the second task, the "leading" student edited the wiki, without attempting to initiate a discussion. The previous task saw the course instructor provide lower marks to the two non-participating team members, which may have encouraged one student (8) to participate and contribute to the wiki. Both wiki products from this team were basic and contained dot points, with little evidence of discussion.

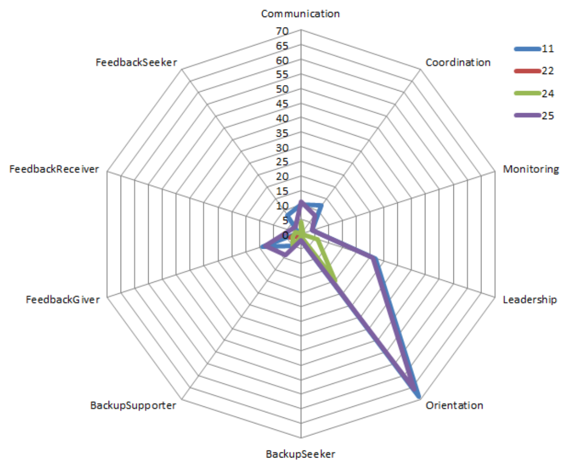


Figure 2: Radar graph of students' observed role behaviour in group 6

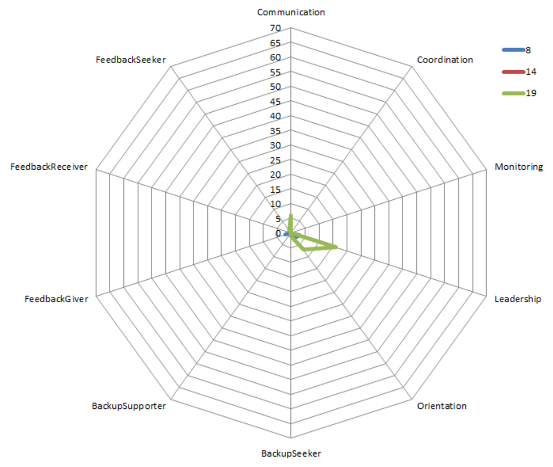


Figure 3: Radar graph of students' observed role behaviour in team 3

The two student groups presented indicate that students vary in the frequency of role behaviours. It also highlights that the existence of a leader to initiate discussions does not always result in high activity. The radar graphs illustrate that we can encourage these students to adopt a variety of other roles, in particular those that relate to group processes: coordination and monitoring. Although not conclusive, the results suggest that the frequency of role adoption evident in the discussion may contribute to the quality of students' wikis. The following sections will summarize the findings in light of the literature.

5. DISCUSSION

Dickinson and McIntyre's measurement of teamwork roles is usually measured post-incident and along a scale measurement. Our research has demonstrated that their teamwork model can also be a useful framework for guiding more detailed measures of online activity. We demonstrate this can be achieved by explicitly listing behaviours associated with their teamwork roles so that content analysis can be applied to students' discussions by coding the students' utterances to explicit behaviours. Furthermore, by identifying behaviours that represent each role, we alleviate any vagueness of the roles, making it much clearer as to what each role should involve. Behavioural data collected can provide results about the roles students adopt by calculating the frequencies of total behaviours for each role. The frequency data collected by our analysis of teamwork behaviours can be presented visually as graphs to illuminate how students are performing in particular teamwork roles, particularly by comparing students within teams according to their role adoption. In doing so, we can visually see where students are reasonably inactive and potentially 'at-risk' or putting pressure on other group members to take on a majority of the workload.

The results confirm that students in self-organised teams are self-initiating the adoption of some teamwork roles defined by Dickinson and McIntyre [13]. However, the frequency of behaviours within the roles varied: students were more commonly displaying behaviours associated with orientation, leadership and proving feedback. The ability to negotiate group goals, monitor and regulate group processes is identified as critical for self-organising group success, particularly for agile teams, as reflection on team performance is what members rely on to adapt behaviours and improve performance [16], but, we observed that team goal-setting and regulation of processes were scarce in the student discussions. In online agile teams, it is suggested that teams intermittently perform this behaviour [16] and it could be argued the ability to communicate reflections about team processes and performance is vital and should be part of a team's communication. Although one strategy is for educators to model roles and role transition in the classroom [32], it begs us to question what roles are being modelled and explicitly taught and whether there might be a tendency to focus on positive "orientation" behaviours, leadership and feedback, and less on seeking support, team coordination and monitoring. Furthermore, in university teamwork, educators may request students to engage in collaboration and provide a "participation mark" in an online space; but this may cause students to assume a focus on knowledge-construction and frequency of "contribution", rather than reflection on collaborative processes.

It becomes apparent that leaving students to develop teamwork skills through experience is not enough [5]. To what extent are we facilitating the process of teamwork skill development within courses and across our CS programs and how does assessment influence students' behaviour to engage and/or develop confidence and skills in teamwork? How are we measuring change and providing feedback to students about teamwork? Such questions raised instil a need to be critical of our role in students' teamwork development so that we are preparing graduates who not only know content, but are able to behave effectively as team members. Activities and assessment can be designed to better support teamwork development. Teachers could present students with teamwork models and discuss or design activities that emphasise particular roles and expected behaviours. Alternatively teachers could use or develop a model like ours (table 1) or ask the class to identify behaviours they think would represent each role, which could engage and involve students in creating and adhering to their own teamwork expectations. Teachers could request "reflection of team processes" as one component for students' participation grades to encourage meaningful participation as a team member and encourage reflection on team processes. Alternatively, assessment tasks could request that students reflect on their group's performance and processes and ask students to post their reflections in discussion spaces or write reflective essays discussing the extent they adopted particular roles and what they intend to do next time to improve. Future research could determine strategies, like these suggested, that support teamwork development and team processes and determine how the design of activities and assessment influence students' self-initiated role adoption and development.

Similar to studies where leaders emerge in software organisation teams [19], one or two leaders usually emerged in the teams observed. These leadership and coordination roles could be described as the "glue" or structure to the team collaboration, as they require students to initiate, lead, and monitor the group process. However, responsive and supportive roles are equally important for the construction of ideas and solutions in a collaborative task as they involve having a sense of group awareness, reflection of processes and learning of self- and others, and the ability to seek help when required. Seeking behaviours are also important, as they require students to express lack of comprehension; a skill that new graduates often struggle with [5-7]. Essentially, both spectrums of behaviours are necessary for collaboration and it appears to be important that team members strike a balance between the two types.

One or two students were noticeably disengaged from the task and these students were at times unobvious in discussion activity. Such students may be those who go unnoticed, particularly in online collaborative tasks, and are at-risk of failing. This lack of engagement from particular individuals appeared to place pressure on those trying to enact leadership or coordination behaviours and resulted in some students having to take on a majority of the workload or "leading" students to try to motivate members through the task. Workload allocation was also an issue in software organisations [17] and a number of studies have found students dislike group work because of unfair participation. We saw examples where attempts to lead the team were not always successful and despite continued attempts to motivate peers and initiate discussion, sometimes leaders received no response. These findings support why students often dislike and feel anxious about teamwork [26, 27] and emphasise the value of illuminating student participation by having students collaborate in an online space. Visibility of collaboration provides educators with an opportunity to monitor participation and roles and determine if an intervention is required. However, in the workplace, there will not be a "teacher" available to intervene and students need to be equipped with skills to manage poor peer team-participation in constructive and positive ways. This problem might be alleviated by encouraging students to be aware of their own active and passive behaviours and by encouraging students to transition between behaviours as the need arises to alleviate workload pressure on particular members.

5.1 Limitations and Future Research

This is a case study of one 3rd year CS class and a "snapshot" of teamwork for students in one semester, offering potential for comparisons with other cases. Students' behaviour in teamwork activities may change across their degrees and such findings may provide guidance about how to structure teamwork activities across the program to develop skills. We focused on role adoption across groups and within groups; our future work will compare the findings in this study to a robust evaluation of students' wiki products to determine if role adoption affects the *quality* of work produced.

Our framework that guided our analysis included a number of behaviours we associated with each of the teamwork roles identified by Dickinson and McIntyre. However, we realise that in some cases particular behaviours could be appropriately suited to more than one role. This offers opportunities for other researchers or educators seeking to analyse student teamwork to identify the behaviours they expect to be evident of particular roles, or alternatively roles they would like students to enact. Furthermore, researchers could take an exploratory approach to analysis and code behaviours as they arise, creating their own behaviours and connecting them with roles. Alternatively, in a practical sense, educators could have students identify explicit behaviours they feel should represent each role and follow-up with an assessment of their teamwork based on the class-constructed behaviours. In involving students in the assessment process it may assist them in becoming familiar with teamwork roles and expectations and therefore assist post-reflection of theirs and others' of processes. While we encourage other approaches to investigating teamwork behaviours, our work has contributed to the field by identifying particular behaviours and in encouraging an active approach to assess student teamwork, beyond outcomes or contributions.

Our findings demonstrate the value in visualising such data and the need to continue to investigate ways to make this data available to students and teachers in "real time" so that they can reflect on their behaviour and adapt accordingly. We will continue to investigate how to include teamwork processes and skills into assessment so that students can be provided with meaningful feedback that will assist them in improving their skills for subsequent teamwork tasks. We hope that others working in the field of learning analytics will be seeking to provide meaningful, real-time data about how students behave.

6. CONCLUSIONS

Our research has demonstrated that using Dickinson and McIntyre's model of teamwork, with the extension of explicit behaviours has proven valuable in providing us with information about how students are performing in online teamwork. We found that students were self-initiating leadership and coordination roles, however these were greatly dependent on how their team mates responded. Encouraging students to practice leadership roles is important, but we stress the need to practice responsive and supportive roles as well. In our analysis we did identify a lack of team regulation and goal-setting behaviours, suggesting that educators may need to scaffold students to engage in behaviours that are vital for monitoring and maintaining agile team success. Possibly future research could investigate how CSCL tools can be incorporated into online environments to facilitate regulation of team processes and goal-attainment.

Research has identified that new graduates are often lacking teamwork and communication skills [6-8], and perhaps we too often assume students naturally employ these skills through experience [5]. It is unknown why students in this class lacked the adoption of some roles, particularly coordination, seeking and reflective behaviours, however, understanding why students adopt certain roles and not others warrants further exploration. If team success is linked to teams who reflect on group processes and employ strategies to improve team performance [15], then assessment of teamwork activities needs to recognise these behaviours as important and provide feedback to students about improving these behaviours.

This research serves as the basis to continue the development of evaluation mechanisms for assessing teamwork roles and skills in collaborative activities. Teamwork is essential to the field of CS and CS education can lead investigations in effective teamwork practices, proactive teaching methods and ways of assessing teamwork so that students may be provided with valuable feedback that can improve their performance for teamwork situations they will most likely encounter as a computer scientist.

7. REFERENCES

[1] Bender, L., et al. 2012. *Social sensitivity correlations with the effectiveness of team process performance: an empirical study*, in *International Computing Education Research*, ACM: Auckland, New Zealand. p. 39-46.

[2] Robles, M. 2012. *Executive perceptions of the top 10 soft skills needed in today's workplace.* Business Communication Quarterly, 2012.

[3] Archer, W. and Davison, J. 2008. *Graduate employability: what do employers think and want?*, in *Graduate Employability: The Views of Employers*, R. Brown and K. Herrmann, Editors. 2008, The Council for Industry and Higher Education: London, United Kingdom. p. 1- 18.

[4] Loftus, C., Thomas, L. and Zander, C. 2011. *Can graduating students design: revisited*. in *Technical Symposium on Computer Science Education*. Dallas, Texas: ACM.

[5] Lingard, R. and Barkataki, S. 2011. *Teaching teamwork in engineering and computer science*. in *Frontiers in Education Conference*. Rapid City, South Dakota: IEEE.

[6] Begel, A. and Simon, B. 2008. *Struggles of new college graduates in their first software development job*, in *SIGCSE*

Technical Symposium on Computer Science Education. ACM: Portland, USA. p. 226-230.

[7] Begel, A. and Simon, B. 2008. *Novice software developers, all over again*, in *Workshop on Computing Education Research*, ACM: Sydney, Australia. p. 3-14.

[8] Radermacher, A. and Walia, G. 2013. *Gaps between industry expectations and the abilities of graduates*. in *Technical Symposium on Computer Science Education*. Denver, Colorado: ACM.

[9] Ruff, S. and Carter, M. 2009. *Communication learning outcomes from software engineering professionals: a basis for teaching communication in the engineering curriculum*. in *Frontiers in Education Conference*. San Antonio: IEEE.

[10] El-Abbassy, A., Muawad, R. and Gaber, A. 2010. *Evaluating agile principles in CS Education.* International Journal of Computer Science and Network Security, **10**(10): p. 19- 28.

[11] Hughes, R. and Jones, S. 2011. *Developing and assessing college student teamwork skills.* New Directions for Institutional Research, 2011. (149): p. 53-64.

[12] Hattie, J. and Timperley, H. 2007. *The power of feedback.* Review of Educational Research, **77**(1): p. 81- 112.

[13] Dickinson, T. and McIntyre, R. 2009. *A conceptual framework for teamwork measurement*, in *Team Performance Assessment and Measurement: Theory, Methods, and Applications*, M. Bannick, E. Salas, and C. Prince, (Eds). Lawrence Erlbaum Associates: Mawah, New Jersey.

[14] Powell, A., Piccoli, G. and Ives, B. 2004. *Virtual teams: a review of current literature and directions for future research.* SIGMIS Database, **35**(1): p. 6-36.

[15] ChanLin, L.-J. and K.-C. Chan, K.-C. 2010. *Group learning strategies for online course.* Procedia - Social and Behavioral Sciences, **2**(2): p. 397-401.

[16] Beck, K., et al. 2001. *Manifesto for agile software development*. Available from: http://agilemanifesto.org/.

[17] Moe, N., Dingsøyr, T. and Dybå, T. 2010. *A teamwork model for understanding an agile team: a case study of a Scrum project.* Information and Software Technology, **52**(5): p. 480-491.

[18] Dingsoyr, T. and Dyba, T. 2012. *Team effectiveness in software development: human and cooperative aspects in team effectiveness models and priorities for future studies*. in *International Workshop on Cooperative and Human Aspects of Software Engineering*. Zurich, Switzerland: IEEE.

[19] Strijbos, J.-W., et al. 2005. *Functional versus spontaneous roles during CSCL*, in *Computer Support for Collaborative Learning*, International Society of the Learning Sciences: Taipei, Taiwan. p. 647-656.

[20] Weinberg, G. 1971. *The psychology of computer programming*. Computer Science Series, New York, United States: Van Nostrand Reinhold Company.

[21] Belbin, M. 2010. *Team roles at work*. 2nd ed, Oxford, United Kingdom: Elsevier Ltd.

[22] Salas, E., Sims, D. and Burke, C. 2005. *Is there a "big five" in teamwork?* Small Group Research, **36**(5): p. 555- 599.

[23] O'Neill, T., Goffin, R. and Gellatly, I. 2012. *The knowledge, skill, and ability requirements for teamwork: revisiting the teamwork-KSA test's validity.* International Journal of Selection and Assessment, **20**(1): p. 36-52.

[24] Moe, N., Dingsoyr, T and Dyba, T. 2009. *Overcoming barriers to self-management in software teams.* Software, IEEE, **26**(6): p. 20-26.

[25] Hoda, R., Noble, J and Marshall, S. 2010. *Organizing self-organizing teams*. in *International Conference on Software Engineering*. Cape Towan, South Africa.

[26] Falkner, K., Falkner, N. and Vivian, R. 2013. *Collaborative learning and anxiety: a phenomenographic study of collaborative learning activities*. in *Technical Symposium on Computer Science Education*. Denver, Colarado: ACM.

[27] Oakley, B., et al. 2007. *Best practices involving teamwork in the classroom: results from a survey of 6435 Engineering student respondents.* IEEE Transactions on Education, **50**(3): p. 266- 272.

[28] Race, P. 2001. *A briefing on self, peer and group assessment*, Learning and Teaching Support Network Generic Centre: York.

[29] DWEER. 2012. *Employability skills framework: final report*, Department of Education, Employment and Workplace Relations: Canberra, Australia.

[30] The University of Adelaide. 2012. *University of Adelaide Graduate Attributes*, accessed 13 June 2013, Available from: http://www.adelaide.edu.au/learning/strategy/gradattributes/.

[31] School of Computer Science. 2013. *Graduate attributes.* accessed 4 June 2013, Available from: http://cs.adelaide.edu.au/programs/compsci/attributes/.

[32] Falkner, K. and Falkner, N. 2012. *Supporting and structuring "Contributing Student Pedagogy" in computer science curricula.* Computer Science Education, **22**(4): p. 413- 443.

[33] Creswell, J. 2013. *Qualitative inquiry and research design: choosing among five approaches*. 3rd ed, Thousand Oaks, California: SAGE Publications.

[34] Crowe, S., et al. 2011. *The case study approach.* BMC Medical Research Methodology, **11**(1): p. 100.

[35] Zanial, Z. 2007. *Case study as a research method.* Jurnal Kemanusiaan, **9**: p. 1- 6.

[36] *Piazza*. 2012. accessed 15th July 2012, Available from: http://piazza.com/profs.

[37] Flanagan, J. 1954. *The critical incident technique*. Vol. 51, American Psychological Association.

[38] Weinberger, A. and Fischer, F. 2006. *A framework to analyze argumentative knowledge construction in computer-supported collaborative learning.* Computers & Education, **46**(1): p. 71-95.

[39] Hmelo-Silver, C. 2003. *Analyzing collaborative knowledge construction: multiple methods for integrated understanding.* Computers & Education, **41**(4): p. 397-420.

[40] Hmelo-Silver, C., Chernobilsky, E. and Jordan, R. 2008. *Understanding collaborative learning processes in new learning environments.* Instructional Science, **36**(5): p.409-430.

[41] Zimmerman, B. 2008. *Investigating self-regulation and motivation: historical background, methodological developments, and future prospects.* American Educational Research Journal, **45**(1): p. 166-183.

[42] Zhang, Y. and Wildermuth, B. 2009. *Qualitative analysis of content*, in *Applications of Social Research Methods to Questions in Information and Library Science*, B. Wildermuth (Ed), Libraries Unlimited: Westport. p. 308-319.

[43] Williams, K., Morgan, K. and Cameron, B. 2011. *How do students define their roles and responsibilities in online learning group projects?* Distance Education, **32**(1): p.49-62.

[44] Krippendorff, K. 2004. *Content analysis: an introduction to its methodology*. 2nd ed, Thousand Oaks, California: SAGE Publications, Inc.

[45] Maxwell, J. 2010. *Using numbers in qualitative research.* Qualitative Inquiry, **16**(6): p. 475- 482.

A Case Study of the Development of CS Teaching Assistants and Their Experiences with Team Teaching

Elizabeth Patitsas
University of Toronto Department of Computer Science
Toronto, Ontario, Canada
patitsas@cs.toronto.edu

ABSTRACT

Teaching assistants play a vital role in lab-based teaching at large institutions, with a large impact on students' success in CS1. How do TAs develop as teachers? We extended existing models of teacher development for our context of teaching CS labs in pairs. We found practice, teaching multiple courses, mentoring, effective staff meetings, team teaching, and feedback all contributed to TAs' development. Team teaching was a positive experience for our TAs, and allowed them to learn from each other. While teaching labs, TAs learnt mostly from partners who had more course-specific experience, rather than general teaching experience.

Categories and Subject Descriptors

K.3.2 [**Computers and Information Science Education**]: Pedagogy, education research

General Terms

Human factors

Keywords

Computer science education, teaching assistants, labs

1. INTRODUCTION

Consider the scene: a lab session has just ended. The TA hangs around, to debrief the lab with another TA. While mundane, for TAs this is a valuable source of professional development – yet this is ill-studied. How do TAs hone their teaching skills? Who and where do they learn from?

At the University of British Columbia (UBC), CS is taught by 55 faculty and about 200 TAs. The TAs are responsible for over half of the contact hours in first year CS, and, with the lower student-to-teacher ratio, are positioned to have a large impact on their students.

In this paper, we provide a case study of the experience and instructional development of the TAs in CS at UBC. We are interested in how to improve the teaching of our TAs – how can we most effectively and efficiently support our TAs?

Note that this is a case study; while TAs are important to how CS is taught at our institution, our heavy use of TAs is not universal. We leave it to the reader to determine the relevance of our context to their own.

1.1 The Importance of Teaching Assistants

We know from the small literature on lab TAs that their teaching has an effect on student retention [1], and final exam scores [2]. Indeed, students' performance in TA-taught labs have been found to be a predictor of success on final exams in CS1 [3]. Studies in the general education literature on teaching assistants tend to document issues of (lacking) TA quality (e.g. [4]), often with little empirical guidance for how to improve the matter.

Our use of TAs is typical of research-intensive North American institutions, where first-year students spend 30-50% of classroom hours with TAs [5]. The use of undergraduate TAs in teaching introductory programming labs is also common to our type of institution [6, 7, 8], and has been found to be effective in providing a positive learning environment [6], particularly for women and other minorities. Effective use of TAs has the potential to improve the retention of women and minorities in CS, and reduce failure rates.

Despite TAs being highly used, they are not highly trained, and *"few faculty members set as a career goal the supervision of graduate teaching assistants"* [9]. In our experience, many faculty see TAs as poor at teaching; yet the evidence is that TAs – like all other teachers – develop with experience, feedback, support, and positive socialization [9, 10, 11]. Existing literature on TAs comes from the humanities and other fields without labs; as such, we must draw on those studies, and general teaching development, to inform our understanding of TA training and development.

1.2 Teacher Development

How do teachers develop and change professionally? Guskey created a model in 1986 of teacher change; in 2002 he republished his model with almost twenty years of empirical evidence for the model [12]. Guskey found that the naïve model that "*knowledge* $\rightarrow$ *change in attitudes/beliefs* $\rightarrow$ *change in behaviour*" is false [13]: change in attitudes/beliefs is unlikely to happen from solely informing people about new teaching techniques. Instead, change in attitudes and beliefs comes *from* a change in behaviour. Guskey's simplified model looks as such [12]: *professional development* $\rightarrow$ *change in teachers' classroom practices* $\rightarrow$ *change in student learning outcomes* $\rightarrow$ *change in teachers' beliefs/attitudes*

As we see here, the change in beliefs comes from a change in behaviour (the classroom practices). So what prompts teachers to change their classroom practices? Teachers are highly motivated to improve their practice – but are wary that changing their approach could potentially result in *less* student learning [12]. Teachers who have had success changing their practices in the past are more confident about further changing their practices [12]. For effective professional development, teachers need to receive regular feedback, and receive *continued* support that is understanding of the fact that change is gradual and difficult for teachers [12]. It is hence not surprising that coaching approaches to teaching development are staggeringly more successful than isolated, one-time tutorials on pedagogy [14].

Kugel, in his paper on how professors develop as teachers [15], observed that the teaching abilities of professors develops in stages. He presented three stages of development:

1. *Focus on self* and their own role in the classroom
2. *Focus on subject material*
3. *Focus on student* and their ability to absorb and use what they have been taught

In his paper, he breaks the third stage into three: a focus on students' ability to absorb knowledge, a focus on students' ability to use what they have been taught, and a focus on students' ability to learn independently [15]. These three sub-stages are empirically difficult to separate and for the purpose of our paper, we will be treating them as one stage.

Kugel also presents how professors transition between these three stages. The first transition happens once the professors develop confidence that they are not talking too quickly or quietly, or covering too much/little material, etc – and from there, begin to worry that they are not doing a good enough job of teaching the material. In the second stage, professors increase the quantity of what they teach as they develop enthusiasm for the material – and then begin to wonder why students fail to understand all the new material, blaming this on the students. In the second transition, professors begin to focus on the students themselves.

Kugel's transitions are consistent with the research that behaviour change leads to attitudinal change – but his described transitions are simplistic and focus only on the professor. What external forces act and help the teacher to develop? We know that knowledge transfer among teachers is "pull transfer" – teachers pull on their colleagues for aid when they see a problem – rather than "push transfer" [16].

Sprague [9] and Staton [10] describe two external forces that affect TA development: supervision, and socialization. Sprague has a three-stage model of TA development that parallels Kugel's model. She focuses on how TAs should be differentially supervised depending on their stage of development; it should be noted her model comes from an arts background, with no labs.

1. *Senior learner* – a new TA, who is making the transition from being a student to a teacher (focus on self)
2. *Colleague in training* – "as TAs settle into the new role, they become more concerned about their lack of teaching skills" [9]; it is at this stage that they begin to develop a teaching style and focus on delivering the content (focus on subject)
3. *Junior colleague* – "their primary concerns involve discovering ways to help students learn ... [they] are able to transcend, combine, and create systems of instruction ... they are just the people we would all like to hire as assistant professors" [9] (focus on self)

Sprague argues for "progressive delegation" of tasks to TAs – giving the junior colleagues more responsibilities (such as running tutorials), and the senior learners fewer ones (grading papers). Importantly, she notes that the supervision of these TAs should also change: senior learners need a manger; colleagues in training need a role model; and junior colleagues need a mentor [9]. TAs at all stages can benefit highly from feedback and effective supervision – and from relationships with other TAs. Senior learners need a support system of fellow TAs/co-learners; colleagues in training need the fellow TAs as resources; the junior colleagues act as mentors and role models to the other TAs [9].

Per Staton, *"new friendships are a vital component of the TA socialization experience. In fact, having a group of people with whom one can share concerns, fears, triumphs, and challenges ... can make a considerable difference in later success as a faculty member."* [10] For new TAs, it is vital to their development to have friendships with new TAs and senior TAs as role models. TAs are also affected by the culture of their department: micro-messages from fellow students/faculty about the importance of teaching make a large difference [10].

Insufficient social support for TAs is a common issue [5, 4]: TAs tend to feel "overworked, underpaid, and unappreciated" [5], and faculty are unmotivated to focus on TA supervision [9]. A positive social environment for TAs makes a large difference in their motivation as teachers, and their quality of teaching as a result [5].

1.3 Social learning and knowledge transfer

Social learning, also known as observational learning, occurs through observing, retaining and replicating new behaviours seen by others. It is one form in which *knowledge transfer* (distribution of experiential knowledge) may occur [17]. Szulanski identifies three factors in knowledge transfer: the ability of the recipient to identify, value, and apply new knowledge; the depth of knowledge of the source and their usefulness as a role model; and the ease of communication and intimacy of the relationship [17]. As such, we would expect more experienced TAs and course instructors to be more useful sources of knowledge transfer – and more transfer to happen when TAs have more social support.

2. CONTEXT

At the time of this study, UBC CS TAs teach labs in pairs. Lab sections typically contain 20-30 students, and last 2-3 hours depending on the course.

There are 60 undergraduate TAships every year, and 150 graduate TAships. Graduate students are mostly first-year MSc students hired as part of their guaranteed funding package; more experienced graduate students can apply for additional TAships. These experienced graduate students, and the undergraduates, are hired selectively.

In this study we are focusing on TAs who teach first and second year CS. These courses are large; they have hundreds of students, 1-4 instructors, and dozens of TAs. Typically these courses have weekly staff meetings; first-year courses will often have TAs perform the labs in advance during these staff meetings. The organization of the staff meetings varies depending on the lead instructor for the course.

New TAs are strongly encouraged to attend an initial training session. While TAs are told it is obligatory, in practice, TAs frequently skip the session without recourse. No further formal TA training is available to TAs.

The convention in TA assignment is to place TAs who have been hired previously onto the course they had last taught, in the hope of maximizing the experience on a given course. As a result, the vast majority of two-term TAs have only taught one course. More experienced, sought-after TAs have more leverage to request shifting to new courses.

3. METHODS

We held hour-long, semi-structured interviews with nine TAs, using the data to refine Kugel and Spragues' models of development to a model of TA development in our context.

TAs were sampled to yield a maximal spread of experience. We interviewed two first-time TAs, four second-time TAs, two fourth-time TAs, and +6-time TAs. Given the high turnover of TAs in our department, finding TAs with more than three terms of experience is difficult; we had hoped to have a second +6-time TA but were unsuccessful to find one that satisfied our constraint of only interviewing TAs that the author had not worked with[1]. We list our participants in Table 1; names have been changed.

[1]This proved to be a substantial constraint: the author had been a prominent TA for nine terms.

Participant	Terms as a TA	Courses TAed
Alice	1	1
Arthur	1	1
Bob	2	1
Ben	2	1
Bill	2	1
Charlie	2*	1*
David	4	3
Daniel	4	3
Evan	6	3

Table 1: Summary of participants by experience; Charlie has significant additional non-CS teaching experience as a sports coach

3.1 Interviews

The interviews began with a *grand-tour question*[2], an open ended question which allows the interviewee to set the direction of the interview [18]. We would ask the TAs to list their experience, including all the different duties they had had over the years. From there, example questions we asked were (in typical order):

- Why did you become a TA?
- What do you see your role as being as a TA?
- What is your favourite part of being a TA?
- What is your least favourite part of being a TA?
- Who has influenced you as a TA?
- How were you trained for your job?
- What do you do to prepare for the labs?
- Have you sought advice from others about TAing?
- How many lab sections have you taught? Could you describe each one?
- Who did you teach those lab sections with? What was your experience of working with them? What was your first impression of them?
- What do you think of your own teaching ability?
- Has your teaching style or ability changed since you began? How so?
- Overall, how would you characterize your experience as a TA?

3.2 Qualitative analysis

The interview analysis happened in multiple stages, using an Affinity Diagram [19]. Our goal at this point was to examine these research questions:

RQ1. How does the TA experience change with development? (section 4)

RQ2. What influenced our participants to transition to new stages, and promote their development? (section 5)

First, interviews were transcribed. Then, each interview was coded: each new theme, idea, or issue was summarized on a post-it note. Post-it notes were colour-coded by the participant and the amount of experience they have. Once all the initial codes were put on post-it notes (approximately 300), the post-it notes were iteratively grouped by theme, until there were approximately 25 groups.

Then, for each thematic group, we took the post-it notes in the group and sorted them by how experienced the TA was: Alice/Arthur, then Bob/Ben/Bill, then Charlie, then David/Daniel, then Evan. We decided that since Charlie has more teaching experience than Bob, Ben and Bill that we would analyze him as a separate category.

We split thematic groups into two categories: those where a TA's experience of that theme changed with how experienced they were (e.g. asserting authority was hard for Alice/Arthur but easy for Evan), and those that did not (all TAs looked up to course instructors).

Using the themes which changed with experience, we looked at when the changes happened, to identify when stages of development begin/end; and fitting to Kugel's and Sprague's stages in a data-driven approach. Once we had fit our participants to those three-stage models, we looked at what factors influenced our participants' transitions.

3.3 Additional analysis

As one of the factors which emerged as influential was team teaching in the lab, and there was no existing literature on how TAs teach in teams, we focused more on this. We then added these questions:

RQ3. How do TAs work together in pairs? (section 6)

RQ4. How does knowledge transfer flow? (section 7)

We performed 8 hours of observational study of TAs in their labs, observing how pairs worked together (described in [20]). After these observations, we went back to the interview scripts, and re-coded participants' answers about how they work with their partners and their experiences with their partners. For these two research questions, the unit of analysis was the pair. From interviews, we had descriptions of 23 pairs; we also had observed 4 pairs in the lab.

4. WHAT CHANGES WITH EXPERIENCE

4.1 Confidence

More experienced TAs described their own teaching ability more confidently, particularly in terms of asserting their authority and forcing the students to work. As the most experienced participant noted, *"I think I've gotten more stern [over the years]... now I'll enforce a sort of 'put in the effort' to the students. I have a policy of never giving students 'The Answer', and many students don't like that; I can tell students aren't happy about it... at this stage, it gets frustrating, not holding students' hands as much as I used to. I'm not as popular, but always respected." (Evan)*

The TAs with four or more terms of experience were comfortable asking students to, as one put it, *"eat their spinach"*. As Evan describes, *"I may not be popular, but I always feel respected"*. The newer TAs were less comfortable with this. Alice, a first-time TA, described her teaching ability as "hit or miss"; and a second-time TA rated his ability in the classroom as *"better than having no TA there." (Bob)*

The aspect of increasing self-confidence manifests itself elsewhere, such as in interacting with students. Overall, experienced TAs considered themselves to do a better job of teaching, particularly compared to when they began.

4.1.1 Regular Preparation

We saw three stages of TA preparation based on experience: diligent but potentially ineffective preparation, overconfidence, and then effective preparation.

Alice, Arthur, Bill, Ben and Bob all described diligently preparing – looking over labs, but not necessarily doing the labs themselves. They would identify where they expected students to have difficulty. Not having much experience teaching these labs, this was based mostly on their own experiences as students.

For Charlie, David and Daniel, the TAs who had taught more than one course, the days of worrying about lab preparation were behind them. Instead, they would describe times

[2]Typically, "What has your experience been like as a TA?" Alternate wordings were used.

where they neglected to prepare for their labs, assuming they could "wing it" based on previous experience. With their increased confidence, their jobs became easy: *"it was so simple it [preparation] didn't really matter at all" (Charlie)* but later noted that *"I had a problem with preparing for [the labs], out of hubris for having done the labs... one time where I was doing something completely wrong and [my partner] caught me... the preparedness thing was something I could have worked on... [In time] I tried to detach myself from my ego." (Charlie)*

Teaching a course multiple times also would make it harder to motivate onesself to prepare: *"when you've taught the material a few times, and you remember that you've taught it, you have to bring yourself back to the point where you didn't know it, and you have to reset it." (David)*

Evan described himself as reliably preparing. His preparation was less than the first-time TAs, but more targeted; he could identify when he needed preparation, and when he did not. Noticeably, only Evan, David and Daniel mentioned talking to fellow TAs outside of lab as a source of preparation – for the more junior TAs, preparation was generally a solo experience.

4.2 Technique

4.2.1 Approach and Focus

When first-time TAs assessed their own teaching ability, they would describe their approaches to answering questions. One described her approach as: *"I try to simplify it, try to break it into steps. Sometimes I'm leading the person to the answer and sometimes I think I'm dragging them to the answer ... If they don't get it, I try to take them back to where they last understood, and take them from there. It's hit and miss." (Alice)*

The second-time TAs would also focus on answering questions, but described their ability to do so as successful, and focused on the content of the questions – whether they could answer a question on HTML, or how to use the debugger in Eclipse. For both the first and second-time TAs, their evaluation of themselves was largely based on how well-received they were by the students. The first-time TAs were clearly at Kugel's "focus on self" stage, but Kugel's model doesn't quite fit here for the second stage. The second-time TAs were comfortable with their ability to answer questions, and focused on content (focus on subject) – but evaluated themselves based on how students perceived them (focus on self). Here is where Kugel's model doesn't quite apply to TAs – we think it is because they do not determine the subject material, and have less responsibility for it, and so it is harder for them to focus on only the material.

Charlie, David, Daniel and Evan also described using multiple ways of reaching their students. They would lecture the class, or target students and "guide them along" rather than waiting for those students to ask questions. These four TAs described multiple heuristics in teaching students, shifting between them as appropriate. They were the only ones describing a Socratic approach – *"I don't give them answers, I just get them to find answers." (Daniel)* They described their approaches as focusing on equipping students to learn independently, and would evaluate themselves based on their impression of student learning. By Kugel's model, these four TAs are at the "focus on student" stage.

4.2.2 Communication skills

The more junior TAs (Alice, Arthur, Bill, Bob, Ben) also tended to discuss their communication ability when describing their teaching ability. For example: *"one person wrote [in my evaluations that] I should take some public speaking lessons, and maybe I should, and it's a bit hard for me in front of the class but it's easier one on one. There were also a few cases where I might have misled someone." (Bob)* – And: *"[My teaching ability has] room for improvement... I have to make a mental effort to slow down when speaking to students" (Arthur).*

In contrast, the more senior TAs (Charlie, David, Daniel, Evan) did not mention their communication ability when assessing themselves as teachers. They did not note difficulty in communicating with students, although note was made of improving over time: *"[Teaching is] a great experience... it's an experience for growth. You have to know things to quite a high level unlike [teaching sports], and you're developing it at the same time as you're developing the soft skills." (Charlie)*

4.3 Interactions with Students

4.3.1 Relationship to Students

All participants noted that their favourite part of their work was helping their students, and guiding them to so-called "Eureka moments".

For junior TAs, interacting with students in a friendly manner was important. They were also eager to have more interactions: *"what is important is that I get more interaction with students." (Ben)* And as another put it, *"[My favourite part about being a TA is] I get to interact with the same group of students, so you develop a friendship sort of thing. It's fun knowing they can turn to me when they need help in lab." (Alice)*

For the more senior TAs, interacting with students in a mentorship style was more important. One wanted *"to convey that CS is pretty cool, and when students get it, that's a pretty good feeling" (Evan).* Another said, *"I don't chat with the students or [my partner] socially [while teaching]" (Daniel)* and that during lulls in the lab, he would instead focus on the struggling students: *"[it] takes out a lot of my time to try and help them." (Daniel)*

The difference between the mentorship approach and the friend-making approach could sometimes cause tension between TAs of different stages. For example, in describing his less experienced partner, Charlie described that *"[The partner] would spend more time talking with the kids, talking about random stuff ... there was a couple [of students] that he really liked to chat with. I would also chat with the students, but not as much as he did. As a TA you want to be friendly and nice, but you don't want to have a 20 minute discussion about your favourite video game." (Charlie)*

4.3.2 Authority

Asserting authority was a salient problem for the first-time TAs, particularly given their young age: *"As a first year [myself], it's weird interacting with students in first year who are in classes with me, and with older students ... My position as an authority is a bit [pauses] I have to be a bit more careful in what I do." (Arthur)*

Arthur and Alice's descriptions of their unclear authority in the lab was very much consistent with Sprague's description of the "senior learner": *"they tend to identify more with the students in their classes than with the instructors they are assisting ... this is a troubling and confusing transition[:] Can I really do this? Do I look like a teacher?"* [9]

Bob, Bill and Ben were more comfortable in their roles; they could assert authority when they had to, and were generally unworried about whether they were seen as authorities to their students. And for Charlie, David, Daniel and Evan, this was not an issue at all.

Related to this is how TAs would respond to questions where they did not know the answer. More senior TAs generally responded by calling over their partner to see if they knew the answer – regardless of how experienced their partner may be. As one put it, *"If there's something I'm not confident on, I'll refer to [my partner] – we want to get the* *best answer possible [for our students]." (Daniel)* Junior TAs were less likely to defer to their partner. They were less likely to admit that they did not know something, worrying that they would look incompetent. For example, Ben noted that if a student asked him too advanced a problem, he would brush it off rather than ask his partner.

4.4 Support

4.4.1 Teamwork

First-time TAs tended to ignore their partners, "too busy" while teaching to check in with their partners or observe their work. Second-time TAs, however, tended to interact directly with their partners, such as in socializing with them or intentionally observing how they answered questions. Junior TAs were immediately trusting of their partners; as one noted, *"It was kind of implicit – we never thought of it. It was a given. We were both in the same section, we were both TAs. What was there not to trust about?" (Ben)*

For more senior participants, the trust was to be earned. Some would report experiences working with unprepared TAs where they had to perform "damage control"; these experiences tended to stop TAs from automatically trusting their partners. The senior TAs would take more of a supervisory role when paired with an inexperienced TA, taking them under their wing. Evan describes a partner: *"[She] was frequently unprepared as a TA [last term]; this impression [of her] has not changed. [This term] She is more comfortable with the material now, can think on her feet more. ... I trust her now more since she's more familiar with the material... not so much when first TAing with her, wasn't sure she'd always give good advice to students." (Evan)*

The senior participants would take the initiative to communicate with their partners about the lab, such as in discussing the lab beforehand, or debriefing together afterwards. For one, the process was: *"[We] would huddle up and talk the lab over... same thing with [a partner in another section], huddle up at the beginning to talk about what the lab is about and who does the marking" (Daniel).* Another TA used the lulls for this: *"when there's a slow period, and nobody asks a question, then we'll talk until somebody asks a question. It was actually pretty neat to see how he [the partner] was doing the labs." (David)*

4.4.2 Getting Advice and Encouragement

Beyond their partners, all participants sought advice and support for their work, and found mentorship important in their growth. Participants of all stages looked to their course instructors for mentorship.

Novices also went to external sources: friends, family, and past TAs were noted. The senior TAs noted going instead to more experienced coworkers and their research advisors.

When asked about their favourite part about teaching, the more experienced TAs noted collaborating with the other staff as a favourite part about teaching. One, for instance, noted that staff meetings were one of his favourite things, and that *"I really, really enjoy working with [two course instructors]. Our staff meetings are awesome." (Charlie)* This was not noted by the junior TAs.

Outside the labs, TAs of all stages would also seek advice from other, experienced TAs. *"At one point I asked another TA [Daniel] about another lab. I was wondering how they handled people who couldn't keep up. [And how to handle a difficult student.] And I did ask some other TAs to ask what their experiences were like, and I could use that to generate a strategy to work with him." (David)*

Similarly, Arthur went to his TA from when he was a student (Charlie) for advice about students not finishing labs on time, and was reassured that *"it's not your fault they didn't all finish on time"*.

4.5 Perception of the Job

4.5.1 Least Favourite Parts about Teaching

While no TA enjoyed seeing their students fail, nor fail to complete on time, nor having students who didn't put in the effort or keep up with the material, the extent to which these things distressed the TAs differed between the junior TAs and senior TAs.

Alice, Arthur, Bill, Ben and Bob found these issues highly distressing, listing them as their least favourite parts about teaching. By contrast, the senior TAs described these matters with a large degree of acceptance. Indeed, on the matter of students not finishing on time, Alex noted a "too bad, so sad" approach; he would cut labs off precisely on time to be fair to all the students.

Senior TAs listed a number of different things in response to "what is your least favourite part about teaching?". These were: 8AM staff meetings, bad answer keys, issues with recording grades, and managing grades with Blackboard; in other words, logistical issues that differ from term to term. Senior TAs tended to focus on complaining about matters they felt could be changed — such as rescheduling staff meetings, or changing the course management software.

4.5.2 Triage

Senior TAs demonstrated more incisiveness in how they allotted their time and effort as TAs. As Evan noted, *"in my first term, I would not have thought twice about spending 40 minutes with a student that hasn't put in the effort..."*

Junior TAs did not note this discrimination: *"I'd often stay up to an hour and half [overtime]... The labs were tiring since they were 3 hours, and most students took half that time. Some 'exceptional cases' took longer, and I'd wind up working overtime. I'd spend a lot of time working on false problems: ambiguous instructions, lab machine issues, so on – I'd be wasting a lot of time on these." (Ben)*

When it came to the 'exceptional cases' of students who were very far behind, the senior TAs described a form of triage in rationing their time, and learning to move on for students that "can't be helped". None of the junior TAs mentioned passing over these students, instead devoting as much time as they could to them.

Junior TAs struggled with overexerting themselves, like that TA who would be spending extra hours in the lab helping students. In contrast, one senior TA would *"not stick around after the lab; it reduces cross-pollination between sections and is more fair to the class [as a whole]... I get told I come off as unfriendly, but I'm working on it... when students come in late I won't repeat earlier explanations, to enforce timeliness." (Daniel)*

4.5.3 Motivation and Role as a TA

When asked why they became TAs, the chance to help others always came up. Junior TAs, however, tended to note benefits to themselves: the pay, getting job experience, practice at communicating, and consolidating their knowledge of the material. Senior TAs tended to focus more on philosophical reasons — to "pay it forward", to make up for *"the bar being set so low"* in terms of TA quality, and to replicate the effect that an influential TA had on them as a young student. It is plausible that the TAs who teach for these reasons are more apt to gain more experience.

When questioned about what they see the role of a TA as being, senior TAs tended to describe the role first as that of a teacher and role model, and secondly as that of an assistant to the course as a whole, while junior TAs tended to describe the role primarily as an assistant to the instructor, reinforcing their work in lecture.

As one first-time TA put it, *"We all want to get these kids through the lab and get them through as best as we can. I* *see my job as clarifying what students are having difficulty with, reinforcing what they're learning in lecture. By doing things, they learn it better; helping them see why they'd do something." (Arthur)*

Being a role model was a theme in the senior TAs' answers; *"We are on the front lines, we are the ones the students see; they make their impressions based on us. And it's our task to make sure they learn the stuff, and I'm willing to spend extra time to make sure that's possible. And on the flip side, we also have to make sure things run smoothly, so the professors don't have stuff to worry about." (David)*

5. FACTORS PROMOTING GROWTH

5.1 Practice

For the Sr. Learners, teaching gets less "intimidating" as time proceeds and they gain more practice at it: *"[Teaching] was really scary at first... it is about people skills... but not as hard as I initially thought it would be ... I started off being very math-heavy; now I'm trying to draw from a wider range of examples and different ways of approaching the problem." (Arthur)* and *"over the course of the term, the labs became less intimidating" (Bill)*

One Jr. Colleague described that *"this term I'm much more comfortable" (Daniel)*, referring to his fourth term. In contrast: *"[For my first experience] I was nervous... I expected it to be harder, and eased into it after a few lab sections, and y'know, developed confidence about it, like 'sure, I know this stuff'. It wasn't too bad. [The term after,] I switched to [a second course] to get some variety and to work for [a particular instructor]." (Daniel)*

In addition to the day-by-day practice, TAs also used their term-by-term experience. For example: *"[I] made a point of learning students' names in the first week this term ... it felt awful handing students the marksheet last term [when I did not know their names], especially when most of them knew my name." (Bill)*

5.2 Teaching a Different Course

While Sr. Learners described their first terms as TAs as "intimidating", the more senior TAs described switching to a second course as their most challenging experience, and that this was harder than than beginning as a TA.

Indeed, the senior TAs all noted the experience of moving to a different course as pivotal in their development; their first and third courses were less discussed in this regard. Indeed, having taught multiple courses appears to be the distinguishing factor between the Colleagues in Training and the Jr. Colleagues.

As one Jr. Colleague notes: *"[My first experience as a TA] was fun. Felt prepared, since I'd done well at [the material] at both the grad and undergrad level ... the next one I TAed was CSXXX; It was a different experience. [...] since the course material was new to me. I had to learn the stuff in advance to be able to teach it right back." (David)*

For another: *"[When I first TAed the second course] I was nervous, especially around [the course instructor] ... It took me a term to get used to [that course], there is a special way of doing things [compared to his first course].." (Evan).*

The process of having to adapt to a new way of teaching, and new material, made the Jr. Colleagues reflect on their teaching and generalize their skills to the new courses. It also counteracted the "boredom effect" of teaching the same material repeatedly, and encouraged David and Daniel to prepare for the new material (cf. subsubsection 4.1.1).

For the Colleagues in Training there was a desire to try a second course. *"I was getting tired of the course [by the time I applied for a second TAship]... I plan on continuing to TA for as long as possible ... Hopefully not [the course I'm on]. I don't want to be in the same course for too long; I want breadth of experience. Actually, I listed in my preferences everything but [the course I'm on] for this term." (Ben)*

The Sr. Learners, in contrast, reported not feeling "ready" to try another course: *"I don't think I'd be qualified to teach anything else." (Alice)*

5.3 Mentoring

Our participants reported mentoring as being helpful in their growth – and for Evan, being a mentor helped him. Mentoring could come from course instructors, research supervisors, or more experienced TAs.

Indeed, numerous study participants reported going to Evan for advice, or listening to his remarks in staff meetings. He was also aware of being a role model: *"there is a 'social strata' in [this] course, once you've done it once before you're in [the 'old folks'] crowd. Among that crowd, I think I'm the only one whose done the course for more than one term... Now, TAs will ask me questions and expect a definite answer. I'll try to be hands off with the 'new folks', I don't want to give them the impression that I think they can't do it." (Evan)*

For the graduate student participants, research supervisors were noted as role models, among others: *"I have been influenced by many teachers: [my research supervisor] has been especially influential; I also had another mentor... she has passed away... she helped me deal with many things that come to you at once, and make all the students feel acknowledged and not feeling ignored. And I'm learning a lot from [the two course instructors] and I learnt a lot from [one of the course instructors] by watching him and how he handles questions." (David)*

5.4 Working with other TAs

5.4.1 Staff meetings

Staff meetings form Sr. Learners' primary resource for advice on labs, and were noted as very valuable to them. These TAs tended not to actively contribute to the meetings, but listened carefully to the discussions between the other TAs and the instructors.

Arthur described their training for the job as: *"There were pointers on how to run the lab [at the staff meetings, which] came from the other TAs as a general discussion... At the meeting, the main contributors to the warnings [about pitfalls] include Daniel and David, but everybody tends to pitch in. [The course instructor] talks about the labs [also.]"*

The other Novice adds: *"I don't surround myself with other TAs [off the job] ... so I don't really get influenced by them... At the meetings, [the course instructors] will give tips and I'll take notes and try to use them ... One time I brought up that my students were working together in lab and one student was just copying. I brought it up in one of our TA meetings and they gave me some helpful pointers... mostly when we have TA meetings I listen to other issues that are brought up, so it's okay in that sense. " (Alice)*

Hearing their coworkers talk about teaching appears to support Sr. Learners in reflecting about their teaching. Alice noted: *"[Since the start of term] I've become more self-aware. At the beginning, I was more telling them the answer, now it's more I'm telling things to get them towards to the answer. I ask them questions to get them engaged... I've become more aware about how I approach things." (Alice)*

5.4.2 Team teaching

For Colleagues in Training and Jr. Colleagues, team teaching was noted as useful for gaining tricks, examples, and advice on how to teach the labs. For Ben, talking to their partner was enjoyable and motivated preparing for the labs: *"[My partner and I] would let the students go crazy with the* *labs and we would sit at the front."* They'd try out things on their computers. *"I would do little projects, and we'd share. It was fun, talking to [my partner] in the lab. We'd be talking about things outside the students' league, like assembly programming. But we would also be doing stuff related to the lab. And then we would try to do the lab, too." (Ben)*

Bob found that working with a much more experienced partner in one of his sections helped in learning the labs: *"I thought that [one of my partners last term was] pretty good. I would put myself at the same level [of teaching ability] as him had I not the lab with [a more experienced partner that term] and access to what she was doing. I could see how she would explain things; I could correct myself more often. Helpful in filling gaps in my knowledge."*

Evan, in reflecting on an earlier term as a TA, recalled *"It was nice to have [the other TA] around to use as a gauge of whether [our course instructor] was being stern or upset; I was nervous the first time I was on [that course], especially around [that instructor]." (Evan)*

As we noted previously, Sr. Learners differed from the other participants in how little they reported interacting with their partners in the lab. Indeed, Arthur, who hadn't reported talking to his partners much, noted during the interview's debrief that *"in doing the interview, I'm reflecting on what I'm doing and not doing in the lab... I think I should start talking to the other TAs more." (Arthur)*

5.5 Feedback

Sr. Learners reported being hungry for feedback, finding it useful for their growth. We have already noted mentoring – one source of feedback. But for TAs at our institution the only formal performance feedback TAs receive is from end-of-term student evaluations.

A number of participants noted the qualitative feedback they had received from students. For example, *"I would have my laptop out and not be disturbed for an hour, hour and half ... [My TA] evaluations described me as ignoring [the students], that they didn't feel comfortable interrupting [me], so I changed this in later terms." (Daniel)*

Rarely, instructors would give performance feedback in staff meetings, such as "you are all doing a great job". TAs described this as useful, particularly for being motivated midway through the term. One-on-one feedback from instructors was described as particularly powerful.

As students were the main (if only) source of performance evaluations, Sr. Learners tended to focus on pleasing their students. They worried that poor evaluations would lead to them not being rehired, and were intimidated to take a firm hand with their students as a result. Jr. Colleagues did not note this intimidation; after being rehired numerous times they had confidence in their job security.

6. HOW TAS WORK TOGETHER

As team teaching influenced TAs' development, and is unstudied in the literature – how do TAs work together?

The TAs responded unanimously that working with another TA in the labs was a positive experience; as one put it, *"the two TA thing was perfect" (Charlie).* None of the TAs would have preferred to work solo, and reception to adding a third TA to a lab section was generally lukewarm: "*adding a third TA would make it harder to coordinate ... there might be more conflict.*"

Only Daniel thought that adding a third TA may be better: *"I've always wondered how it would go with three TAs instead of two; it would reduce the wait time for students, which is their biggest complaint. A third TA would give them time to sit down and help them out."* But as Bob put it: *"That size lab felt pretty hectic... we were just trying to keep track of everyone, so I think two is kind of ideal."*

The student-to-TA ratio (25 students to 2 TAs) was consistently described as manageable, except in CS1 – TAs noted the relative neediness of students in this course compared to other classes meant that the student-to-TA ratio was slightly too high. The CS1 labs also require students to show intermediate work to TAs at specified "checkpoints" and to wait until they have TA feedback before proceeding. More experienced TAs on the course would ignore the instructor's instructions about the checkpoints and let students work ahead after finishing checkpoints, so that there would not be bottlenecks in questions.

6.1 Advantages of Team Teaching

Our participants noted four benefits of team teaching:

Division of labour: *"makes the lab more efficient"*

Security: *"It's nice to have somebody covering your back"*

Teamwork: having another TA to socialize with during lulls, or *"bounce ideas off of"*

Diversity: *"sometimes you just can't see something and you need another view"*; *"we could combine our knowledge"*

For less experienced TAs, the last point was particularly salient: if they did not know how to help a student, their partner would be there to help them out – when their partner "has their back" it is making up for their own lack of knowledge or experience – and would give them a chance to fill those gaps, as we see in subsubsection 7.1.2.

For experienced TAs, a partner who "has their back" often meant they had more freedom in how they spent their time in lab – *"More TAs mean you can get to a student faster, or you can spend more time with a student and somebody else takes up the slack."*

6.2 Conflict in the Lab

Overwhelmingly, TAs had positive things to say about their partners and the experience of teaching in teams. However, conflict could arise between partners.

The most frequent negative comment about their partners was that they had been blunt or insensitive to their students, and all came from the undergraduate TAs:

- *"I think he would have been great for fourth year, but first years are a bit fragile and he should smile more." (Charlie)*
- *"he could be insensitive... very straightforward about what he tells students about their mistakes. Won't sugar coat it. Students have told me they found him insensitive..." (Ben)*

Experienced TAs were most concerned about their partners' preparation and professionalism.

As for direct conflict between partners in the lab, the only issues that our participants noted as contentious have been (regardless of experience level):

Part marks: whether students were being marked too leniently or harshly – for many CS TAs, having thrived in a culture of yes-or-no marking, any subjectivity in marking schemes is uncomfortable, and TAs get little guidance on the matter.

Punctuality: when their partner was coming consistently late, or very late

6.3 Approaches to Teamwork

TAs spent most of their time in the lab working independently. Generally, TAs would brief with their partner at the beginning of the section and debrief at the end, only checking in with each other if issues arose. In labs with lulls in student questions, TAs would also talk to their partners during these breaks.

About half of the TA pairs went further than this, by either actively observing their partner, talking strategies in the labs, or socializing. As some examples:

- *"I would see how [my partner] would explain things ... we spoke more about the issues that were popping up and how we could resolve them."*
- *"[My partner] would sit at the back [and I at the front] and whenever one of us would see a question, the closest would go over and sort it out... at the beginning of lab we would huddle up and talk the lab over... what the lab is about and who does the marking."*
- *"It was cool to get to know him a bit more... [working together] is good. He's really enthusiastic about his stuff. He knows how to help most stuff, and when he doesn't, I've been able to help him out."*
- *"There are these two girls who sit in the front row who don't talk to [my partner and I]... we talk about this and what we can do about it."*
- *"[My partner and I] would do the labs together [as preparation so] that we wouldn't look so clumsy in front of the students."*
- *"Our styles complement each other; I explain things theoretically, and she explains things concretely."*

Generally, TAs would not get to spend much time talking to their partners – for the majority of sections the TAs were "going from one question to another" for at least the first half, and last quarter of the lab – but the brief interactions were described as very useful. We categorized the types of reported in-lab discussions between TAs as such:

Lab issues: problems and bugs in the lab
Solutions/prep: lab content and solutions
Strategies/stragglers: identifying students in need of special attention and how to help them
Social/'chitchat': socializing
Logistics: who marks what, "could I have the marksheet", having to leave early, who enters grades, etc.
Do not talk - language barrier
Do not talk - no chance: the TAs were too busy in the lab to talk at all

The chitchat, while off-topic, was useful for beginning TAs to form friendships with their colleagues – and made it more likely for TAs to talk about lab issues and strategies. Undergraduate TAs, in particular, noted spending time with fellow undergraduate TAs outside of class and forming a social support network with their colleagues.

7. KNOWLEDGE TRANSFER

In our interviews, we saw that most of the knowledge transfer received by TAs came from fellow TAs – through informal mentoring, staff meetings, and working together in the lab. For the first two types of interaction, the knowledge transfer generally happened from an experienced TA to a less experienced TA – the "grandmasters" were the sources of knowledge that other TAs would look to. Noticeably, TAs did not pay attention to whether a grandmaster was a graduate student or an undergraduate.

More junior TAs would also emulate other teachers as they developed a teaching style of their own. Often they would look to more experienced TAs in this regard, as well as the course instructors. Often, for the undergraduate TAs, the TA who had taught them in the course they were now teaching would still be accessible (or even still teaching the course). Such veteran TAs were more likely to be sources of advice, and were most likely to be emulated – *"Charlie was once described at a staff meeting as 'walking on water'... very likable, very approachable, helps you figure out what you were doing, easy to understand... I try to be like him, and try to show the same enthusiasm" (Arthur)*

7.1 Knowledge transfer in the lab

In our field observations, we observed that one TA would be dominant in running the lab. The students and the other TA would defer to this *alpha TA* as an authority; in observing behaviours, we saw this TA would give most or all of the announcements to the class, that the *beta TA* would ask more questions of the alpha TA than vice versa, and the alpha TA would spend more time speaking than the beta TA – both to students and to each other.

We noticed that a TA did not always have the same position in every lab section – a given TA may be alpha in one section, and beta in another. We did, however, notice that positions were stable within a pair – a TA who was alpha on the first day of lab would continue to be alpha. Furthermore, we noticed that novice TAs were in many cases alpha TAs, particularly when two novices were paired together.

When interviewing TAs about each of their partners, we asked the questions "would you ask your partner more questions, or would they ask you more questions?" and "would your partner give more class announcements, or would you?". We then quantified their response as such:

2 pt: "they ask me questions and I don't ask them questions" / "I give all the announcements"
1 pt: "they ask me more questions than I ask them" / "I give most of the announcements"
0 pt: "we ask each other questions equally" / "we split the announcements equally"
-1 pt: "I ask them more questions than they ask me" / "they give most of the announcements"
-2 pt: "I ask them questions and they don't ask me questions" / "they give all the announcements"

We then add the question score and the announcement score for each dyad: if the TA we interviewed was the alpha in the dyad, the score for that dyad would typically be about 3 pts; for a beta, -3 pts. No dyads had a score of 0.

For each dyad, we determined the TA's previous TAing experience to that section, with breakdowns by how many labs they had taught previously, whether those had been for that course, whether they had taken the course, and whether they had taught the course. We also determined the number of labs earlier in the week the TA had taught. We then used the `lm` package in `GNU R` to model the alpha-ness as a function of those different types of experience.

We found only two factors were statistically significant ($p < 0.05$): whether the TA had taught sections earlier in the week, and whether the TA had taught these labs before in previous terms. Whether the TA had taught the labs in previous terms was a stronger factor.

This fits with the cases we observed in which an experienced TA took on a beta role to a less experienced TA – their partner had more experience with those labs. Hence, we see two types of knowledge that TAs draw from:

General teaching knowledge: ability to teach; related to the Sr. Learner to Jr. Colleague axis
Course-specific knowledge: knowledge of given labs, relevant subject material, and how a given course works.

In the lab, TAs look to the partner with more apparent course-specific knowledge. In contrast, when TAs described who they sought advice from outside the lab, they explicitly selected for friends, colleagues and role models with more perceived general teaching knowledge.

7.1.1 First impressions

One question we asked TAs was about their first impression of each of their partners. We observed that the words used to describe this impression varied by whether they were alpha or beta, indicating that these roles are determined very early in the term. Charlie, an experienced TA, described a TA that he took a beta role to as, *"First time I got to talk to him, he was lecturing [to the class] already."*

Alpha TAs were typically described as "experienced", "organized" and "intimidating"; beta TAs were "quiet" and "uninterested" – but were usually seen more positively over time, particularly as inexperienced TAs "learnt the ropes".

7.1.2 Learning from the Alpha

For junior TAs, the beta role appears to be extremely valuable. TAs new to a course would learn from alpha partners in an informal and often unrecognized apprenticeship. A sampling of comments from beta TAs about working with alpha TAs reveals a transfer of knowledge:

- *"[During the lab] she would call me over when there were problems arising, so I could see them"*
- *"It was pretty neat to see how he was doing the labs. "*
- *"I like it better when [my partner] is around. Just because he knows what he's doing, because he's done the lab before. So if there's minor details I don't know, I can ask him. And if there's something I can't explain, then maybe [he] knows how to do it. And it's harder for me to do the challenge problems."*

A number of TAs that had beta roles earlier in the lab week would hold alpha roles later in the week. Bob was in such a role, and said: *"I would put myself at the same level [as my beta partner] had I not the lab with [my alpha partner] and access to what she was doing."*

7.2 Other Influences

After their fellow TAs, course instructors were mentioned as the most influential people on our participants' development. (Other sources of influence that came up were mentors, research supervisors, and friends.) The influence of the course instructor was not consistently positive. One TA noted one of the reasons he tried hard to be a good TA was because he was afraid of the course instructor's ire – *"[this instructor] doesn't suffer fools gladly." (Evan)*

Another TA described a course instructor as a negative influence that contributed to his job dissatisfaction – *"[this instructor] was really lax about standards... [this instructor] takes these breaks... there aren't clear instructions on what to do when [they] leave." (Ben)* Instructors who did a poor job of management, or were disinterested in supervising TAs, were identified as negative parts of their teaching experience.

For the TAs who noted course instructors as an influence on their work, staff meetings would come up as important times for them. One TA noted that their course instructor would praise the TAs during staff meetings for their hard work; three TAs noted they had received good advice during the meetings. David, who sat in on lectures, noted watching the instructor and how he handled questions to be inspiring.

It should be highlighted that two of the nine TAs noted that they had at one point chosen a TA assignment solely on a desire to work with a particular course instructor. Five of the nine TAs noted course instructors to be a factor when listing preferences for TA assignments.

8. DISCUSSION

As a case study, this work provides a rich view of the TA experience at our institution, useful for improving TA support. Some threats to validity that should be considered are the recall bias of the participants, and the filtering effect of which TAs are rehired. Our analysis used descriptions of experienced TAs recalling earlier times – which would not be as reliable as having interviewing them years ago. Also, at our institution, only generally dedicated TAs apply for TAships past their 2nd appointment: TAs more motivated to develop as teachers are more likely to gain experience.

Due to ethics reasons, the study author was the only person who coded the interview data – some bias in coding will be inevitable as a result. Were the study to be repeated from the start, we would have added more researchers into the coding process. Lastly, we should note the identity of the researcher likely had a (positive) effect in the interviews: by being a peer to the participants, we feel we had their trust; we feel TAs were as a result more open and honest during interviews. We speculate that had a faculty member run the study, we would not have heard anecdotes about being uninterrupted for "an hour, an hour and a half" or being "intimidated" by their course instructors.

8.1 Implications for Practitioners

8.1.1 Implications for TA Training

Based our findings, we highly recommend formally mentoring TAs – but realize that such mechanisms are time-intensive and difficult to maintain. For most CS departments, we expect they would get better results by instead improving their TA training. We suggest:

- Offer *two* TA training courses. One for Senior Learners (those who have not taught before), and one for Colleagues in Training (those that have).
- Training for Sr. Learners should focus on communication skills, asserting authority, and triaging student questions. This is also a place to teach TAs how to use the department's chosen technologies for grading and handin, navigating the computing resources for undergraduates, etc.
- Training for Colleagues in Training should focus on pedagogy, effective teamwork, and preparation.
- Both groups benefit from teaching observations, though the focus on where to improve will differ.

8.1.2 Implications for Instructors

We recommend that instructors pair TAs for teaching labs. The social support provided by team teaching benefits all TAs. We also recommend running weekly staff meetings where TAs can discuss past and upcoming labs.

Positions of leadership – such as a Head TA position, or curriculum development – should be given to Jr. Colleagues. Courses that offer tutorials in addition to labs should prioritise assigning these TAs to tutorials. Office hours and grading should be given mostly to Sr. Learners. These TAs benefit from building confidence with the course material, and should be assigned a minimal number of lab sections.

In assigning TA pairs to labs, ensure each lab section has a TA that has either taught the lab before, or teaches another section earlier in the week. Course-specific experience is more important here than general teaching experience.

In running staff meetings, we recommend taking the time to give TAs feedback on their work. Solicit TA feedback on labs, and debrief together. We recommend viewing the staff meetings as a learning opportunity for TAs.

Large courses should also offer weekly staff meetings where the TAs work through the labs as a group, to help Sr. Learners with the material – and to ensure more experienced TAs prepare at all. These staff meetings should be run by a head TA – a Jr. Colleague – to encourage inter-TA collaboration (and to lighten the load for the instructor!)

Optimally, TAs would benefit from having one-on-one feedback from instructors as well as teaching observations from either fellow TAs, instructors, or external staff.

Finally – and perhaps most importantly – is that culture is important for raising TAs. Graduate students should be encouraged to do well at TAing by their research advisors. Course instructors should treat TA supervision as a TA mentoring opportunity. TAs will do better when they are encouraged from all sides to take the role more seriously.

9. CONCLUSIONS

We found that our participants' experience of development could be broken into three stages that followed Sprague and Kugel's models – although Kugel's second stage of "focus on subject" was less applicable as TAs do not determine subject material in our courses. We saw developmental differences in TAs along lines not predicted by either model:

1. Sr. Learners and Colleagues in Training were diligent at preparing for labs, but Jr. Colleagues could fall prey to underpreparing.
2. Sr. Learners were generally too overwhelmed in the lab to coordinate with their partners, and were too insecure to defer student questions where they did not know the answer.
3. Colleagues in Training, like Sr. Learners, were immediately trusting of their partners. Jr. Colleagues needed trust to be earned. Both Colleagues in Training and Jr. Colleagues coordinated with their partners, with Jr. Colleagues doing so in a more systematic fashion.

We found that practice, teaching a different course, mentoring, effective staff meetings, team teaching, and feedback all promoted TA development. For Sr. Learners, the staff meetings and practice were the most important factors; for Colleagues in Training it was team teaching, mentoring, and feedback. And for Jr. Colleagues, teaching a new course was a pivotal experience.

Team teaching was an important, positive experience for the TAs. For Sr. Learners it gave them security in the lab; for more experienced TAs it allowed for a division of labour, teamwork, and diversity in approaches. While conflict was occasionally present over ambiguity in marking, and professionalism, the experience of learning from another TA was clearly valuable for our participants.

Outside the lab, TAs sought advice from "grandmaster" (Jr. Colleague) TAs and course instructors, and saw them as role models. Inside the lab, however, knowledge transfer in a pair happened differently. In a given pair, knowledge transfer flows almost entirely from what we have termed the "alpha TA" to the "beta TA", and these roles would be fixed over the course of a term. A given TA may be an alpha in one pair and a beta in another pair.

Interestingly, which role a TA assumes is not related to their total TAing experience – but how much experience they have teaching a specific set of lab activities. Having taught the course previously, or even teaching a lab section earlier in the week, factors into which TA is the alpha. For example, we saw Jr. Colleague TAs taking a beta role to Colleagues in Training who had more course-specific experience.

It appears that TAs draw on two types of experience: general teaching experience, and course-specific experience – and both should be considered when assigning TA pairs. Instructors have the power to improve social support, feedback and mentorship for TAs – and should support TAs differentially based on their development.

10. ACKNOWLEDGMENTS

We would like to thank our study participants for their involvement, as well as: Meghan Allen, Patrice Belleville, Michelle Craig, Steve Easterbrook, Jon Pipitone, Kimberly Voll, Steve Wolfman, and anonymous reviewers.

11. REFERENCES

[1] Christopher O'Neal, Mary Wright, Constance Cook, Tom Perorazio, and Joel Purkiss. The impact of teaching assistants on student retention in the sciences: Lessons for TA training. *Journal of College Science Teaching*, 36(5):24–29, 2007.

[2] Cassandra Paul, Emily West, David Webb, Brenda Weiss, and Wendell Potter. Important types of instructor-student interactions in reformed classrooms, 2010. American Association of Physics Teachers Summer Meeting.

[3] Jens Bennedsen and Michael E. Caspersen. An investigation of potential success factors for an introductory model-driven programming course. In *Proceedings of ICER '05*, ICER '05, pages 155–163, New York, NY, USA, 2005. ACM.

[4] Valbona Muzaka. The niche of graduate teaching assistants (GTAs): perceptions and reflections. *Teaching in Higher Education*, 14(1):1–12, 2009.

[5] S. S. Bomotti. Teaching assistant attitudes toward college teaching. *Review of Higher Education*, 17(4):371–393, 1994.

[6] Eric Roberts, John Lilly, and Bryan Rollins. Using undergraduates as teaching assistants in introductory programming courses: an update on the Stanford experience. *SIGCSE Bull.*, 27(1):48–52, March 1995.

[7] Stuart Reges. Using undergraduates as teaching assistants at a state university. In *Proceedings of the 34th SIGCSE*, SIGCSE '03, pages 103–107, New York, NY, USA, 2003. ACM.

[8] David G. Kay. Large introductory computer science classes: strategies for effective course management. *SIGCSE Bull.*, 30(1):131–134, March 1998.

[9] Jo Sprague and Jody D Nyquist. TA supervision. *New directions for teaching and learning*, 1989(39):37–53, 1989.

[10] Ann Q Staton and Ann L Darling. Socialization of teaching assistants. *New directions for teaching and learning*, 1989(39):15–22, 1989.

[11] Robert D Abbott, Donald H Wulff, and C Kati Szego. Review of research on TA training. *New Directions for Teaching and Learning*, 1989(39):111–124, 1989.

[12] Thomas R Guskey. Professional development and teacher change. *Teachers and Teaching: theory and practice*, 8(3):381–391, 2002.

[13] Doug McKenzie-Mohr. Fostering sustainable behavior through community-based social marketing. *American Psychologist*, 55(5):531, 2000.

[14] John A Ross. Teacher efficacy and the effects of coaching on student achievement. *Canadian Journal of Education*, pages 51–65, 1992.

[15] Peter Kugel. How professors develop as teachers. *Studies in higher education*, 18(3):315–328, 1993.

[16] Sally Fincher and Josh Tenenberg. Warren's question. In *Proceedings of ICER '07*, ICER '07, pages 51–60, New York, NY, USA, 2007. ACM.

[17] Gabriel Szulanski. Exploring internal stickiness: Impediments to the transfer of best practice within the firm. *Strategic Management Journal*, 17(Winter Special Issue):27–43, 1996.

[18] Del Siegle. Qualitative research, 2002.

[19] Karen Holtzblatt, Jessamyn Wendell, and Shelley Wood. *Rapid Contextual Design*. Elsevier, 2005.

[20] Elizabeth Patitsas. A case study of environmental factors influencing teaching assistant job satisfaction. In *Proceedings of ICER '12*, ICER '12, pages 11–16, New York, NY, USA, 2012. ACM.

Computer science students' causal attributions for successful and unsuccessful outcomes in programming assignments

Rebecca Vivian Katrina Falkner Nickolas Falkner
The School of Computer Science
The University of Adelaide
Adelaide, South Australia 5005
firstname.lastname@adelaide.edu.au

ABSTRACT

While some students excel in introductory programming courses, others find the course to be significantly challenging and demanding. The way that students reason about the factors that contribute to success or failure may affect their self-efficacy, motivation, future success and whether or not they persist in Computer Science (CS). What factors do students' perceive to cause successful or unsuccessful learning outcomes in first-year programming assignments? Such findings can assist us in identifying causal reasoning that may be detrimental to future success and persistence. We use Attribution Theory (AT) as a framework to explore the "causal attributions" that students apply to explain their causes for success or failure in introductory programming assignments, alluded to in their reflective essays about performance in a course. Our research demonstrates that reflective essays, integrated into learning tasks, can be one effective and efficient way to extract students' casual attributions. Our results indicate that the students raised a number of causal attributions in their essays that were specific to the CS-context and were attributed to both internal and external causes. We highlight problematic areas of casual reasoning and a need to correct misleading reasoning to ensure CS students understand their control over the success of their future programming assignments. This research offers opportunities for future research to develop activities that may encourage students to correctly identify causes of performance outcomes in programming assignments and to determine if such interventions can prevent students from leaving CS.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education, self-assessment.*

General Terms

Computer Science Education and self-assessment.

Keywords

Attribution theory; university students; programming assignments; attributes; success; failure; self-reflection.

1. INTRODUCTION

There are a number of challenges to educating students in first-year computer science (CS) courses, as many enter CS programs with little or no prior discipline experience and are required to master programming skills and knowledge as well as skills and knowledge of how to be an effective learner. Learning to program is a unique experience for each student [18] and while some excel in introductory programming courses, others struggle with mastering programming skills and development processes [14; 15] and find the course to be significantly challenging and demanding [11; 12]. First-year CS students not only drop-out because of 'critical' reasons, but also as a result of multiple minor reasons that cumulate and eventually result in withdrawal [11], such as having a lack of motivation and or time for study [12].

While enrolment and retention in CS are often explored by examining students' reasons to withdraw, it is not well understood what students perceive to be the contributing factors to their success or trouble in learning to program, particularly at the assignment-level. Poor performance in first-year programming assignments may have a major impact on an individual's emotional state and desire to persist in CS. A great deal of students' self-efficacy in the domain of CS is influenced by prior programming experience and studies have found that as students progress through CS degrees their self-efficacy in programming increases [18]. However, what makes some students persist through a course long enough to gain experience and increase self-efficacy? Unfortunately, many students may decide to withdraw early if they encounter early experiences of poor performance or difficulty in programming assignments.

One possible way of examining students' perceptions of poor performance is by Attributional Theory (AT) [8]. This theory is based on the premise that individuals will assign causal attributions that explain reasons for success and failure [27]. These reasons might be due to internal (self) or external (environmental) and stable (fixed) or unstable (changeable) factors. What we hope that CS students perceive their performance to be due to internal reasons, which are changeable. For example, positive outcomes can be acquired through increased effort and/or the use of a software design, rather than external reasons such as harsh marking, which can instil feelings of hopelessness and no control of their learning performance.

In other discipline areas, students have been trained to change their causal attributions for performance which has been found to improve performance and retention [8; 17]. However, in CS we have factors that are unique to our field, such as the presence or lack of previous programming experience and certain software design processes and strategies that can contribute to successful software programming [2]. Identifying problematic reasoning for causal factors that students attribute to successful and unsuccessful outcomes in an early programming course, may assist our understanding of how to guide or 'retrain' first-year thinking about programming processes and learning processes, in their authentic environments. Such understandings may assist us to prevent students from attributing to external and unchangeable events that ultimately may cause feelings of 'helplessness', 'shame' or 'humiliation' [27] that may result in them dropping out from the degree in the long-term.

Koli Calling '13, November 14 - 17 2013, Koli, Finland
Copyright 2013 ACM 978-1-4503-2482-3/13/11…$15.00.
http://dx.doi.org/10.1145/2526968.2526982

We apply content analysis, using the AT framework, to students' essays in which they reflect on their learning experience in a first-year programming course. We use the framework to identify causal attributions students raise and to determine if students' certain causes to lead to successful or unsuccessful outcomes. Furthermore, we wanted to determine if students mention causal attributions particular to the CS-discipline. We begin this paper with a discussion of literature relating to the use and importance of reflection in learning, followed by a discussion of AT in the field of *academic achievement* and factors currently identified in the literature that explain students' reasons for success and failure, with a particular focus on the CS context.

2. BACKGROUND AND RELATED WORK

2.1 Reflection in learning

The transition from novice to expert programmer is assisted by reflection on prior successes and failures [19], followed by analysis of potential areas for improvement. Reflective thought is an act in which individuals carefully consider beliefs or knowledge that lead to conclusions [4]. When reflecting, individuals rely on evidence of some form that leads to belief. In the context of success and failure, individuals will weigh up evidence that they believe has led to particular outcomes. When individuals engage in reflective thinking about their learning processes, these thoughts can inform individuals about how they may adapt their behaviour to perform better in the future. However, misguided reasoning may be detrimental to performance or self-efficacy.

A number of studies have sought to understand students' learning processes in software design. When students have been asked to reflect on software processes, students focus on non-discipline specific strategies [6; 22], in particular those associated with design-oriented stages [16]. Furthermore, when students are asked to provide advice to other first-year programmers studying, they are highly focused on non-discipline specific learning strategies, such as time management and planning [6]. When it comes to improving their practice in programming tasks, students have been found to struggle with developing appropriate plans to improve learning performance [22] and only in the final stages of programming processes have students began to connect practices with potential improvement strategies [30]. An inability to identify how to improve practice is problematic because a critical variable of practice and competence has been found to be, not time spent engaged in an activity, but the intent involved in figuring out how to improve practice [30]. The results bring us to question if students acknowledge stages of the software development process that we 'preach' as leading to effective practice and contributing to their successful outcomes?

Reflective tasks are often incorporated into the CS curriculum to encourage students to consider how to improve their software development processes. Authors have argued that reflective writing can provide a critical role in promoting reflective learning and can encourage students to make connections between learning tasks [21]. In their experience of open-ended reflective essays, students did not reflect on their learning processes and it was only when students had explicit prompts that they began to reflect on learning processes [7]. The authors suggest that if wanting to extract such information through open-ended questions, it appears to be important to include explicit prompts to guide students toward such aspects of their learning experience.

One aspect of reflection that maybe be elicited through reflection may involve an individual making causal attributions: reasons that explain success and failure [1]. The following section will discuss this theory in light of the relevant literature.

2.2 Attribution Theory (AT)

Attributions, in combination with motivation, has been found to predict almost 50% of the variance in students' university Grade Point Average (GPA) scores [28], suggesting that causal reasoning plays an important part in the learning experience. AT rests on the presumption that the result of an action is felt to depend on one of two conditions: factors being within the person or factors external to the person [9; 23]. Heider [9] originated AT, however, Weiner and colleagues have developed the previous work to include constant and variable internal causes and understandings about the role of emotions and motivation [23-26].

Figure 1 depicts AT as a 2x2 representation model [27]. The model includes four determinants of behavioural outcomes (ability, task difficulty, effort and luck) and two causal dimensions (locus and stability). Stability is measured by how much control one has over the situation and if a cause is subject to change in future events. For example an attribute such as effort or luck has potential to change, making it unstable, whereas someone's perception of his or her ability or personality at a point in time is fixed. Theory suggests that when people attribute their successes to unstable causes (luck or effort) and their failures to stable causes (ability or task difficulty), the probability of persistence is low [28]. Another dimension of the model is measured by internal or external causes. Internal causes are about the self, such as one's ability or effort and external causes are due to factors in the environment or external to the person, such as task difficulty or a perception of harsh marking.

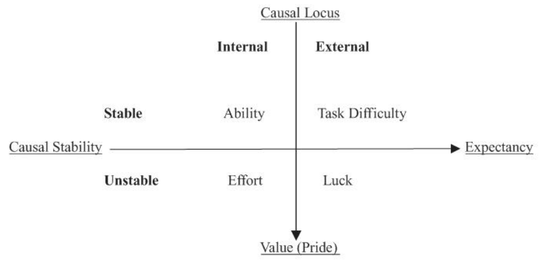


Figure 1: Weiner's (2010, p. 32) representation of the four main causes of behaviour

There are a number of attribution-emotion linkages that have been made between aspects of the model. When people fail, it is believed attributions that fall within the internal/stable dimension can lead to shame or humiliation because these are causes attributed to the self and as something that cannot be changed. Feelings generated by this dimension can influence one's feelings of self-efficacy in a domain, which are beliefs about competency to perform a goal [1]. Research has shown that a primary influence on CS students' self-efficacy is influenced by prior programming experience [18]. A poor experience in programming, attributed to internal/stable causes, such as ability, may create negative feelings toward future programming tasks and can potentially lead students to believe they are not capable of being a computer scientist because programming is an innate ability [20]. In the internal/unstable dimension feelings of guilt may arise because individuals recognise they had personal control over how they prepared for academic tasks and consequently brought about their own outcomes. Although we don't want students to feel guilt, we hope that students realise that they have control over their learning processes because this leads students to perceive future success can be achieved by adapting learning processes and behaviour. Typically success attributed to internal reasons generates more pride than by external reasoning [27]. When individuals perceive their failure is due to external/stable reasons, such as task difficulty, they may experience feelings of 'helplessness', a feeling that any outcome in the situation was beyond their control [1], which is something that we want students to avoid. Feelings of hope can be generated when individuals attribute failure to external and changeable factors, such as luck, because they hold the belief that things could be different the next time.

In a study of nursing students, authors analysed two open-ended survey questions to determine what they attribute to unfavourable and successful learning outcomes in a course [5]. Using content analysis the researchers identified whether students were making internal or external causal attributions and found that the majority of respondents (84%) attributed their academic successes in the course in part to internal causes, and a number of respondents (68%) attributed their academic failures, in part, to external causes. Overall, most students primarily attributed successes to controllable, unstable causes such as 'effort'.

In the CS context, 19 narrative interview extracts about students' lived experience in a foundational computer-programming course were analysed. Researchers identified ten causal attributions that emerged in reaction to the question: 'what do you think have caused the course outcome [of your grades]?' [7]. Responses included: learning strategy (40%); lack of study (31.1%); lack of practice (22.2%); appropriate teaching method (13.3%); subject difficulty (4.5%); and (all with 2.2%) a lack of effort, exam anxiety, cheating, lack of time and unfair treatment. The authors claim that, of the ten causal attributions, only two (lack of effort and subject difficulty) were amongst the four causes that Weinberg (1958) identified, as ability and luck were not mentioned. The authors reason that 'ability' may not have arisen in the interviews because the course was not based on mathematics, which is where AT is commonly applied and that luck did not surface because their exams were in lab environments where students use an existing development tool to build a solution. This is somewhat surprising because 'ability' was one contributing factor acknowledged as why students choose to major in CS [13]. Furthermore, the literature presented so far suggests that causal attributions of ability are not only restricted to the field of mathematics, as although small in size, 3 of 75 students in the discipline of nursing attributed their course success or failure to 'ability' [5].

A survey of 45 first-year CS students [10] revealed those with an optimistic attribution style performed better than those who had less optimistic styles. Students who explained positive events as internal and stable had better grades than students with pessimistic attributions. The authors reason this could be due to optimistic attribution styles as having a predicative effect on programming performance or that success is often attributed to personal causes and unsuccessful outcomes to external factors. Ability accounted for the largest proportion of the variance in students' course performance and the causal dimension of stability was significantly related to performance; suggesting that students perceive the causes of their performance are relatively stable over time. The authors suggest strategies such as providing students with achievable programming tasks early on to experience success and to encourage students to interact with teachers or peers about reactions to performance so that those who experience failure may be encouraged to view it as something that is temporary.

Lewis, Yasuhara and Anderson [13] undertook semi-structured interviews with 31 students enrolled in introductory programming courses at two colleges. Using grounded theory, they analysed interview transcripts and identified five factors that influenced students' decisions to major in CS, which were: ability, enjoyment, fit, utility, and opportunity. In their paper, they explore 'ability' and how students measured their ability in terms of speed, grades and previous experience. They discovered that these experiences came about because they were influenced by experiences in their environment and through their perception of ability as being fixed or malleable. These factors identified by the researchers are very similar to the facets identified in AT. Although the authors explored students' perceptions using their own grounded theory approach, examination of their paper reveals that many of the students' quotes could be understood using the AT framework, in particular those beliefs of ability being fixed (internal/stable) or malleable (internal/unstable) and by experiences in the environment (external stable/unstable). For example, one student states that their success was due to being a 'fast learner', which could fall within the internal/stable trajectory. Or another example is that a student believed that their success was not accurately reflected because the grades they receive never depict their actual level of understanding, which could be an external and stable perception. What we realise is that, AT offers a framework for analysing students' attributions in the field of CS and, besides gathering attributions via surveys, the framework can also be applied on qualitative reflections.

In recognising that failing first-year courses can be an overwhelmingly negative event for students, researchers sought to determine if an Attributional Retraining (AR) treatment intervention could have any influence on performance through a series of reflection activities. AR involves restructuring students' causal explanations of poor performance by encouraging students' to re-think how they attribute causes for failure. AR encourages students to consider controllable attributions, such as effort and learning strategies instead of unchangeable attributions such as ability or intelligence [8]. The results found that the AR treatment was found to be successful in a Canadian first-year psychology course [8; 17] and that the AR intervention reduced the likelihood of failure for those students who undertook the program. Additionally, the program's success was found to surpass several other common factors that influence academic outcomes, such as gender, age, past performance and learning environment variables, such as orientation involvement. However, it is stated that CS involves unique aspects that can influence learning [2], such as the presence or lack of previous programming experience and design-oriented strategies, making it necessary to explore causal attributions that students use in the CS context, to identify problematic causal reasoning that we may be able to correct through CS education.

2.3 Motivation for the study

Many of the studies discussed are based primarily on 'snap shot' survey instruments, where students are forced to respond to certain causal attributions about a whole course or their CS program experience. We would like to know how students describe their own personal attributes for 'success' and 'failure' in terms of their assignment submission and software development process; both of which students are able to plan for; implying they have time to be able to prepare, research, test programs and monitor their process over the duration of a course. We wanted to see if we could apply the AT framework to code students' essay reflections about their design process and programming assignments. In doing so we sought to determine if students make causal attributions to 'task difficulty', 'ability', 'luck' and 'effort' and if there were other CS attributions that students made.

The research tells us that external attributions are detrimental to learning processes and we seek to identify those external attributions specific to the CS degree as well as identify internal attributions and areas where students feel these contribute to success or failure. In doing so we may be able to identify and correct misconceptions and celebrate positive attributions. Such information could assist us in making adaptions to better scaffold students to take ownership for the success of their learning outcomes in the CS-discipline.

3. RESEARCH METHODOLOGY

We adopted an instrumental case study design, whereby data were collected and subject to directed content analysis [29]: an approach commonly used to validate or extend a previous conceptual framework or theory. Our approach extends upon existing frameworks of AT; applying AT within the context of software development processes. An instrumental case study approach uses a particular case as an illustration to understand a phenomenon, which in this case is students' attributions toward successful or unsuccessful outcomes of their submission of programming assignments [3; 29]. The small sample size of case studies allow researchers to capture the complexities of a phenomenon and explore in qualitative detail students' experiences. Using this research approach we investigate the following research questions:

1. Are there CS-specific causal attributions that we can identify in students self-reflections using the AT framework?

2. What causal attributions do CS students believe lead to successful or unsuccessful outcomes in programming assignments and what causal attributions are potentially problematic?

As this course is about algorithm design and data structures, we anticipate and hope that students identify that the use of a design leads to successful outcomes and that a lack of design can impact negatively on their performance in a programming assignment.

3.1 Case Context

The case considered in this research project is a final course in our introductory programming sequence at The University of Adelaide (UoA) in 2012. Students within this course have completed 1-2 prior programming courses, providing them with competencies in the application and tracing of fundamental programming constructs, and design skills within small scale problem solving. The learning objectives for this course include awareness and application of simple data structures, related algorithms and algorithm complexity, and initial experiences in medium-scale problem solving and software engineering. This course contains, for many students, their first experiences with nontrivial complexity software design and development. The course includes a substantial, supervised practical component, of between 2-4 hours per week, in addition to small-group collaborative work within lectures and tutorials. Students are assessed on the functional outcome of their programming assignments, and their process, via design documents and descriptions of testing strategies. In addition, the course includes two structured reflective exercises that require students to describe their current software development processes, how they have changed and a description of how they intend to change them in the future. These exercises provide a small contribution, less than 2% in total, to their final grade. We have analysed the second reflective essay for analysis as this enables us to capture students reflecting on their progress and processes for the semester. We randomly selected 85 (of 98) students' reflective essays for analysis. Students were asked to write a reflective essay about their learning experiences in the course in relation to the production of their programming assignments. The essay was to be 2-4 pages in length and was marked out of 10 (although only contributing to 1% toward their course grade). Because the essay was based on opinion, marks were allocated primarily toward the structural components: grammar and spelling (4), length (2), quality of essay response (2) and advice for students taking the course in the future (2). Students have been found to struggle with reflecting on learning processes [7], therefore, to encourage students to reflect we included explicit "prompt-like" questions in the essay task description that asked students to consider aspects such as, if they submitted their work on time and why or why not.

3.2 Coding Framework and Content Analysis

We began by importing the 85 self-reflection essays into NVivo 10 for analysis. We created four nodes (causal attributes) identified by Weiner of 'ability', 'task difficulty', 'luck' and 'effort'. We coded the students' utterances in the essays that fell into any of Weiner's categories. If the student made a causal attribution and it was not one of the existing attributes or was CS-specific, we created a new node for that attribution. In the end, we had a list of 16 causal attributes that the 85 students had identified. These attributes will be discussed shortly. Each student could identify multiple attributes in their essays and could identify factors that contributed to success and difficulties. When an utterance was coded, we also coded the utterance as being an 'unsuccessful outcome' or 'successful outcome'. For example, if a student believed the marking criteria was too complicated to understand and because of it they received poor grades, we would code this as an attribute for 'marking criteria' and for 'unsuccessful outcome'. Using this method, we are able to determine which attributes are associated with certain positive or negative outcomes. In section 4 we elaborate on our methods and use of AT as we present the results. For anonymity each student is provided with a random number when quotes are used.

One researcher, with a background in education research and previous experience in content analysis, whom was not involved in teaching at the university, conducted the coding. Upon completion of the data analysis, the researcher presented examples of coding to two CS academics at UoA with expertise in CS education research to confirm coding decisions. We found no disputes in reasoning, however, we recognise that the results are based on three researchers' judgements with westernised views about where particular attributions fit in the AT model and which attributions are perceived as successful or not, however we endeavour to recognise and explain our judgements where possible. Future work will involve an external researcher to conduct coding comparisons for accuracy of content analysis with more cases. We acknowledge that student essays were perceptions about what *students think* caused outcomes in their learning and how the researcher interpreted their essay. In future work we will involve students to discuss what we coded as their causal attributions to determine the accuracy of our interpretations.

4. RESULTS

The pie chart in Figure 2 illustrates the types of causal attributes students raised in their reflections. We can see that time management and design strategy are two primary areas, although this is not surprising as the task asked students to reflect on these components. Effort, ability and the pre-assessment of task difficulty were other primary factors. Using the attributions above, we extracted whether they were coded as an 'unsuccessful' or ‘successful’ outcome using an NVivo coding matrix and were able to report the percentage of the total of 85 students who mentioned the attributes as leading to positive or negative outcomes. We present the causal attributes associated with unsuccessful outcomes and successful outcomes in Table 1.

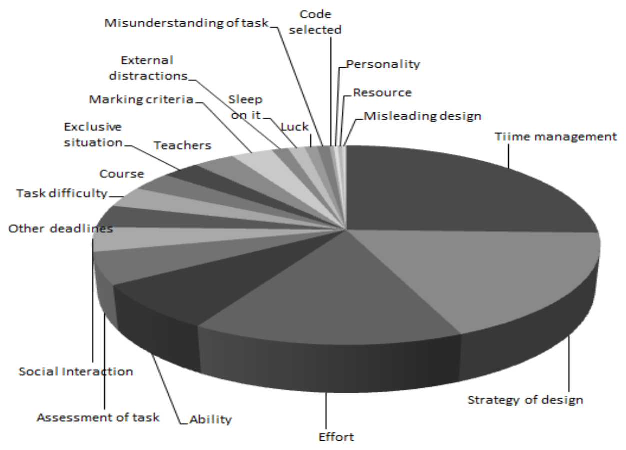


Figure 2: Pie chart that illustrates the number of students raining particular attributions in their reflective essays

As mentioned previously, each student may have discussed multiple attributes that contributed to their success or difficulties. The results reveal that when reflecting on difficulties in programming assignments a number of students attributed causes to time management strategies, effort, lack of design and programming ability. Although not as common, students also perceived factors such as their personality and teachers to play a role in poor performance. Overall, what we see is that across all student essays, a number are attributing unsuccessful outcomes to internal and external factors. We will explore these in relation to the AT framework in the following section. Table 1 also reveals that a number of students perceived time management, presence of a design and effort to influence the success of their programming assignments. Additionally, social interaction with peers and teacher guidance played a role in successful outcomes.

Table 1: Causal attributions identified by students that resulted in unsuccessful and successful outcomes (n=85)

Unsuccessful		*Successful*	
Time management	43.5%	Time management	40.0%
Effort	32.9%	Design strategy	31.8%
Design strategy	25.9%	Effort	10.6%
Ability	21.2%	Social interaction	8.2%
Pre-assessment of task	14.1%	Teachers	8.2%
Other deadlines	11.8%	Pre-assessment of task	4.7%
Task difficulty	11.8%	‘Sleep on it’	4.7%
Course	9.4%	Ability	3.5%
Social interaction	7.1%	Luck	3.5%
Exclusive situation	5.9%	Marking criteria	2.4%
Marking criteria	5.9%	Exclusive situation	1.2%
External distractions	4.7%		
Misunderstanding task	3.5%		
Teachers	3.5%		
Personality	1.2%		

Using the AT model with the stable/unstable and internal/external matrix, we “mapped” each causal attribute (previously coded nodes in NVivo in Table 1) by sorting them individually to an appropriate dimension of Weiner’s causal attribution model (Figure 3). To sort the attributes, the researcher applied Weiner’s measurement of *causal stability*; which involved judging whether students perceived the causal attribute to be stable (fixed /uncontrollable) or unstable (possibility for a different outcome in the future and/or changeable), and also according to the *causal locus*, so whether it was a result of the self (internal) or due to external factors. For example, the presence of a design strategy was mapped as being internal and unstable because, like effort, students would use one or not but the choice was up to them to create one for their programming task. In Figure 3, we colour coded each of the attributes to represent them as leading to a successful or unsuccessful outcome. In the previous tables we presented results that indicated a lack or absence of a particular attribute could lead to an unsuccessful outcome (e.g. effort), here we colour code attributes based on whether the students believed the presence of a factor would lead to successful outcomes, even if they had done the opposite. We wanted to see what factors students recognise as leading to successful or unsuccessful outcomes, even if they had not ‘followed their own advice’ or only realised this on reflection. For example, a student may have said they were lazy in the course, which led to poor grades but had they put in more effort they could have done better. A statement indicates the student knows higher effort equals success. Factors that are perceived to lead to successful outcomes are coloured green and attributes that are perceived to contribute to poor performance are coloured red. Blue attributes indicate students perceived the attribute to lead to both successful and unsuccessful outcomes. For example, in regard to ability, some students felt their previous programming experience allowed them to undertake the task easier than others, whereas others reported that their lack of programming experience or skill caused them to struggle in assignments.

By mapping students’ attributions we are able to identify the attributes that students incorrectly perceive to lead to certain outcomes in programming assignments and factors they feel they have no control over. In doing so, we are able to identify misconceptions that need to be corrected. We proceed to explore each dimension and the attributes identified so far in detail.

4.1 Examining attributions

We explored students’ attributions within the four dimensions of Weiner’s model and whether students perceived the attributions as causing successful or unsuccessful outcomes. Most students demonstrated responsibility over aspects of their behaviour in programming assignments, however a number of causal attributions mentioned demonstrated a lack of ownership over particular outcomes. Table 2 provides examples for how different students with different reasoning in each of the dimensions might approach and consider a complex programming task that a teacher sets. Those with a disposition to typically attribute outcomes to external causes may approach a task differently to those with a tendency to reason toward internal factors.

4.1.1 The internal & stable dimension

The internal and stable dimension involves students attributing toward personal factors that are ‘unchangeable’; those perceived to be ‘the way things are’. In this dimension we identified students attributing to ability, personality and assessment of the programming task.

Weiner [27] identifies ‘ability’ as being an attribute in this area and similar to previous studies [10; 13] we also found students discussed ability as a contributing factor. Ability in the CS context related to ability in programming languages, programming skill and ability to understand the programming task. Students identified a high-level of ability and confidence in programming as leading to successful outcomes.

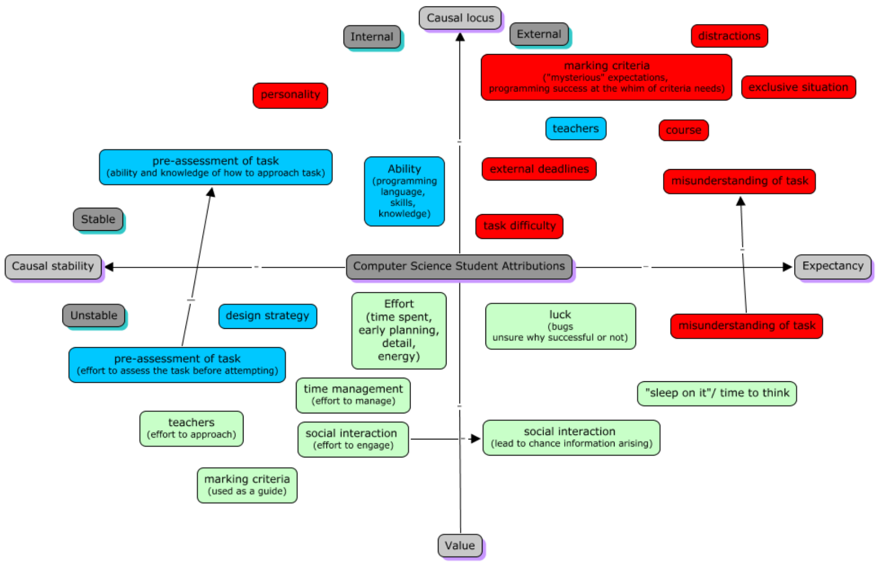

Causal locus
Internal
External
distractions
marking criteria ("mysterious" expectations, programming success at the whim of criteria needs)
exclusive situation
personality
teachers
course
pre-assessment of task (ability and knowledge of how to approach task)
Ability (programming language, skills, knowledge)
external deadlines
misunderstanding of task
Stable
task difficulty
Causal stability
Computer Science Student Attributions
Expectancy
Unstable
design strategy
Effort (time spent, early planning, detail, energy)
luck (bugs unsure why successful or not)
misunderstanding of task
pre-assessment of task (effort to assess the task before attempting)
time management (effort to manage)
"sleep on it"/ time to think
teachers (effort to approach)
social interaction (effort to engage)
social interaction (lead to chance information arising)
marking criteria (used as a guide)
Value

Figure 3: Students' causal attributions reflecting on a programming course using Weiner's framework

Table 2: Example of how a student might approach and perceive the outcome of a challenging situation

Internal/stable They did not have the required knowledge or ability to solve the programming problem. Tried but they are not good at programming yet.	**External/stable** The programming task was too difficult and the marking criteria were vague. The teacher also marked harshly.
Internal/unstable Able to complete the task because they identified key concepts they did not understand, searched Google and taught themself.	**External/unstable** Successful outcome viewed as 'lucky' because they had no idea what they were doing and hacked away at their code until it worked.

One student was '*able to complete the work quicker as I learnt more about programming as well as having had more practice'* (Student 1). On the other hand, students identified a lack of ability as a disadvantage. For example one student believed that '*having started uni when Java was taught, I didn't have the same depth of knowledge and experience with C++*[as others]' *(Student 2).*

Students identified the step of assessing the difficulty of programming assignment tasks as one important stage in their assignment process. We placed this as both internal/stable and internal/unstable because at times, students would reason that their assessment of the task was a result of 'effort' or because of their 'ability' to assess the difficulty. In this dimension, examples include those where students incorrectly assessed the difficulty of the task: *I thought I should be able to finish the practice in 4-5 hours. However, when I begin to work on the practice I realize that it is more complex than I thought. I wrongly assess the difficulty of the assignment (Student 3).* Students expressed that a level of ability was required to accurately assess the necessary knowledge for the assignment, the difficulty and/or time needed to complete. Many students believed this was developed through experience and a number of students reported getting better at this skill over the semester.

Another internal/stable attribute was due to 'personality' factors. One student believed starting programming assignments late on a regular basis was due to their 'personality', that it was in their nature to work on tasks that appeal to them at the time rather than what needs to be done; hindering their performance.

4.1.2 The internal & unstable dimension

Internal and unstable attributes are factors that students perceive they have control over, that influence their learning outcomes. Many of these attributes are familiar to academic contexts, such as effort invested into assignments or time management; however, we also found other self-initiated factors, such as the use of a design or pre-assessment of tasks, also played a role in students' successful completion of programming assignments.

This course was focused on teaching students about designs for software engineering and it was anticipated that, through the activities implemented and the course content, students would see the value in using designs and that use of designs can contribute toward efficiently created and successful programs. While some students, like the following, saw the value in designs and followed through with using detailed designs for their assignments, others despite being encouraged, did not implement a design.

Using a full design always resulted in a better outcome, a better final product completed in less time. Knowing both what algorithms were going to be used and how they functioned meant that it became much easier to locate and fix errors encountered during compile or run time. (Student 12)

Most students realised that a lack of design resulted in lost time and therefore, lost marks. Two practicals, class 4 and 5, were designed to be significantly more complex than previous programming tasks and as a result the practical complexity resulted in students realising that a lack of design produced a poor assignment. As this one student states: *'I also made this mistake in practical 4 and 5, where I didn't write complete design of how I'm going to solve the problem, which cost me a lot of time and marks' (Student 13).* While some changed their behaviour after practical 4 and 5 to include a detailed design to increase performance, a number continued to go without designs, knowing it would potentially hinder their performance. Such behaviour is closely linked with effort because students would put the effort in to create a design and their effort also determined how much detail they invested into their designs.

Effort is an attribute that Weiner identified and, in this context, effort involved the extent to which students invested effort to find additional resources, read lectures or assignments '*thoroughly'*, or to investigate concepts that they did not recognise or know very well in the programming task. Although some cases of 'effort' could also be coded as 'time management', students articulated some aspects of design, programming or learning processes as being based on effort. For example: '*In order to complete the work on time, all assignments were initially read through as thoroughly as possible. This gave maximum time to think about the problem at hand and find an optimal solution' (Student 4)*. In contrast, students who identified themselves as being 'lazy' and approaching the task carelessly or with a lack of effort acknowledged that such behaviour resulted in work that was of poor quality and with additional effort they could have achieved a higher grade. For example: '*I lost marks I shouldn't have as I was too lazy or forgot to add in comments or format my code*' *(Student 32)*. In some cases, students completed designs as an afterthought because it was something that was required for marks and many students lost time by going back to do the design after because of difficulty in remembering processes. As we hoped students attributed success to having a detailed, proactive design, but unfortunately they did not always follow their own advice.

In the essays students mentioned assessing the difficulty of their programming assignment as a step in their software design process. As mentioned previously students perceived the success of their outcomes at times relied on their ability to pre-assess a task, however, they also discussed this step in terms of effort. This is because students did or did not put the effort in when receiving their assignment to assess the difficulty. If they did, they would use this information to inform their design and how much time they would allocate to the task.

Assessing the difficulty of an assignment before beginning to code was an important step. Often this was helped along by a good design that reflected how the practical was going to be constructed. (Student 14)

For students who did not assess the difficulty of the task, or underestimated the difficulty, it resulted in poor time management and often students submitting incomplete or poorly written code. For example, a student claims: '*I rarely took the difficulty of an assignment into consideration before beginning work on it…* [therefore, I] *needed to rush certain aspects of the coding in an attempt to complete it on time' (Student 12).*

Having good time management was often mentioned as contributing toward successful programming assignment outcomes and timely submissions. Time management is crucial to programming assignments, as students need to account for unforeseen issues and time for testing, problem solving, and de-bugging. Students who were able to cater for these factors reported having good time management.

I can attribute most of my assignment submission success from simply allocating plenty of time in case things went wrong. (Student 6)

However, a number of students struggled with time management, identifying that their lack of time management skills resulted in a build up of assignments and rushed submissions. As one student said, '*I found myself regularly starting assignments quite late… rushing to get my work done… I had to compromise marks' (Student 9)*. Although not uncommon in academic assignments, what makes programming assignments unique is often the allocated time required for particular design-stages and testing of code that, like editing an essay in another discipline area, allows students to refine and edit their code to produce higher quality assignments. It appears that this is a stage students often omit.

While we encourage students to use their programming assignment marking criteria to guide the development of their design and planning of time, we are unsure whether students actually use this process and how they perceive the use of this strategy. We found that some students used the marking criteria as a guide and felt it assisted their pre-assessment and completion of programming tasks: *I always followed the structures given because they were logical and very flexible approaches to the problems… This worked well in preventing me from procrastinating because there was pressure to complete tasks by a certain time (Student 11)*. In these instances students perceived the use of the marking criteria led to favourable outcomes, whereas others had a negative perception of the marking criteria and perceived the marking criteria from an external/stable position, which we will discuss shortly.

Despite CS being perceived as a solo experience, a number of students mentioned social interaction as influencing their success in programming assignments. This included approaching peers or teachers for help, assistance or ideas, or by casual conversations with peers or engaging with the discussion forum.

Nine times out of ten if you find a problem and can't work it out there will be someone else that has a different view and can give you some advice in another direction to take. (Student 16)

While this could be perceived as an 'external' influence we placed it within the internal/unstable dimension because students discussed this aspect in terms of needing to put the effort into interacting with others. For those students who did not draw on social support, they felt it led to more work on their behalf. For example two students discussed how they had to 'Google' how to do certain aspects of the task or research lecture slides in advance, knowing that their peers were likely discussing solutions to such aspects in the forum. Although it did not directly result in a poor outcome, this was time spent on activities that could have been spent elsewhere, such as testing, de-bugging or refining code. As one student states: *I did not have enough communication with tutor. When I met problem, I did not turn to anyone. As the result, I had to contribute more time to my work (Student 17).*

Similarly students perceived interaction with teachers as contributing to their success and this was because the student had made an effort to approach the teacher in the practical sessions.

One of the practical demonstrators gave me some great advice which I started using straight away… and to my amazement, my coding life so to speak was a whole lot easier! (Student 18)

The structure of our practical classes encourages such situations, as students are able to check their practical work and ask questions in class. However, this requires students to take control and approach the tutors in-class, as required.

Internal and changeable attributes are favourable and we have discussed a number of examples where students have perceived their outcomes to be caused by attributes that fall within this dimension. We hope students attribute to causes within this dimension because it means they are able to reflect on *their role* and how *their behaviour* influenced an outcome. When students fail and attribute their cause to a certain factor within this dimension, it is important they are also able to consider appropriate strategies that can improve performance. Without effective strategies or the ability to improve behaviour, the student may still encounter unsuccessful learning outcomes.

4.1.3 The external & unstable dimension

The external and unstable attributes are factors that students reason are caused by chance. For example, in a positive outcome, a student may attribute the outcome to being lucky, whereas in a negative outcome they may consider the outcome to be unfortunate or unlucky. Two aspects that students' considered that led to successful outcomes that fell within the external and unstable dimension were luck and the act of 'sleeping on it'. Students perceived that their successful submission and code was due to being 'lucky', with no understanding of how their learning or software development processes or other internal attributional factors contributed to this success.

it was completed on time… mostly due to sheer luck, as the program was not working rather close to the deadline and only with some aimless tinkering did I actually find out the cause (Student 21)

Students also thought that giving themselves time to process information led to better outcomes or an epiphany. One student claimed that they '*found helpful… to sleep on it. Giving my brain time to process the information gave more clarity and cohesiveness to the code' (Student 22)*. While 'sleeping on it' could also be considered as an internal/unstable attribution because students choose to behave a certain way (by taking a break), we have made a judgement to place it within the external/unstable dimension as we viewed this as students believing clarity will be achieved through 'chance' or luck. While students who receive this clarity may have a fortunate outcome, for others who do not experience clarity and receive a negative outcome, they may believe this to be the result of factors that are internal and unstable, such as poor time management or external and stable factors, such as task difficulty.

Within this dimension we found students expressed 'misunderstanding the task' as an unfortunate and unlucky event that lead to unsuccessful outcome. One example of this behaviour is when a student stated that '*the library practical … is what I found snuck up on me and would have liked some warning myself' (Student 24)*. Such situations may have been prevented with the use of learning strategies that other students used, such as pre-assessing the task or managing plans to account for unforeseen circumstances and de-bugging.

4.1.4 The external & stable dimension

The external and stable dimension involved attributions that students perceived to be causing outcomes that were beyond their control and due to reasons that were the result of other individuals, the environment or because of the situation. These included course factors, marking criteria, teachers, other deadlines, and distractions.

Students perceived course factors, such as length of time to complete tasks and the structure of courses had a negative impact on the time they had to complete tasks. Furthermore, some students believed that the limited resources they received were hindering their ability to learn course concepts, as one student states: '*[t]here were some resources provided but they didn't provide the depth that I got by reading the textbook almost cover-to-cover later in the semester' (Student 2)*. Although this student went on to use the textbook, these students could benefit from knowing that others who also did not understand course concepts had researched how to approach the task or asked the tutor or their peers for assistance.

Although 'marking criteria' surfaced as an internal/unstable attribute, where students incorporated the criteria into their programming design plan and time allocation, other students perceived the ambiguity or difficult wording on the marking criteria to fault them. For example this student claims '*at no point was any clear definition given regarding what form these designs should take…. it apparently didn't sufficiently fit into the 'mystery box' of expectations that the marking criteria demanded' (Student 26)*. While there were a number of other strategies the students could have used to prevent this situation, such as drawing on peers or teachers, or researching components of the problem; these students felt 'stuck' or 'tricked' and as though they could not change the situation.

While teacher assistance was perceived to contribute to success for those who initiated help seeking (in the internal/unstable dimension), for those who experienced negative outcomes, some believed their teacher was the cause, in terms of marking or teacher effort. The student above who attributed an unsuccessful outcome to the marking criteria also believed their assignment was *'deemed unsuitable for assessment, because, I assume, the demonstrator didn't have time to read it.* Similarly another student felt they could not receive help in practicals because '*the practical sessions are always very busy and we have to wait for long time to get mark. There is little chance to ask questions or let tutors help me to fix my problems' (Student 27)*. Both examples demonstrate the sense of 'helplessness' that students felt and depict the perceived lack of control about their situation.

Interestingly, one student took a positive perspective on their situation. The student believed the situation had caused them to approach their assignment differently, leading to better quality work by reasoning that '*not having access to a programming environment at home forced me to think about my code without the temptation to begin coding. Through this, I developed good habits and discovered the usefulness of a detailed design' (Student 28)*. However, with a negative outcome, others perceived their situation put them at a disadvantage, for example by missing a practical or tutorial for reasons such as going on holiday. Similarly, and although not uncommon in university contexts, students externalised their failure to reasons such as other course deadlines. Students displayed a sense of 'helplessness' and inability to adapt their behaviour to take into account deadlines.

I was unable to start practicals 6 or 9, as one of my other subjects demanded an entire week's worth of work on those weeks. I don't think this is something that could have been improved upon with better time management; this was an unfortunate clash of priorities. (Student 31)

Rather than taking ownership over their time management and planning, these students had exhibited helplessness in their situation. Even when discussing distractions students described being distracted with an attitude of helplessness over their learning environment that contributed to poor use of study time.

5. DISCUSSION

While Hawi [7] was able to demonstrate that narrative interviews were a successful means to uncover causal attributions, our work reveals that students were able to reflect on casual explanations for successful or unsuccessful outcomes in their programming assignments through a narrative reflection task. This approach, in comparison to interviews, reduces the inconvenience and time required of students. A number of students were able to reflect on causal attributions using reflective prompts, provided in the essay description, making it a useful method to ignite students to reflect on their learning processes and the causes of success and failure. Although it is not certain what impact the use of self-reflective essays had on subsequent performance and some students were attributing to problematic causes, the reflective essays encourage most students to realise aspects of their performance they can improve. Such findings suggest that the use of reflective essays are valuable and could be applied in other CS courses where students are required to develop their programming or learning processes.

Students identified a number of multiple causal reasons for success or troubles encountered in their programming assignments. Like previous studies [6; 22] the students did focus on non-discipline specific strategies as we identified a number of causal attributions that could be present other discipline areas, such as: effort, other deadlines, time management and the perceived impact of assessment criteria and task difficulty. However, many of these in the CS-context have a specific focus, which assists our understanding of how students' specifically think about these attributions within their programming experience. For example, while students' in other discipline areas might describe 'effort' as time spent on an assignment or level of exertion into perfecting their assignment, in the CS-context this also included level of detail on their code annotations, researching and teaching themselves unfamiliar programming concepts to complete assignments, as well as care in formatting code. Additionally they also referred to deign-oriented strategies [16], which is what we anticipated because of the nature of the course. Unlike a previous study of first-year programming students' narratives about performance [7], our students mentioned 'ability' and considered this aspect in the CS-context to include skill and knowledge of programming languages, ability to accurately pre-assess programming assignments and level of understanding of CS concepts. Another example is how, in the CS-context, 'time management' involves students needing to account for time to test, de-bug and refine their code. Such findings highlight the value in identifying discipline-specific strategies and causal attributes so that particular processes can be examined and improved. When teaching students how to adjust their causal attributions, being discipline-specific about the strategy and reasons for thinking particular ways may be more beneficial than telling students to 'manage their time' or generally 'put more effort into tasks'. Students can also benefit by having a more in-depth assessment of their learning processes and this may assist students who struggle to reason about causes for outcomes.

When we examined attributes that students perceived as leading to successful or unsuccessful outcomes and mapped them according to being internal/external or stable/unstable on Weiner's AT model, we were able to identify students' reasoning that is somewhat problematic. As we would expect and hope, students identified a number of causal attributions that they have control over and are attributed to personal causes, such as creating a design based on the marking criteria or interacting with peers on the discussion forum. However, a number of students also believed good outcomes were the result of uncontrollable factors, such as luck. While these are troublesome, more problematic, were causal attributes students perceived to lead to unsuccessful outcomes in the external dimension, such as other external deadlines, marking criteria and task difficulty. While literature typically suggests that students will attribute failure to external outcomes and positive outcomes to the self, such reasoning is an issue because when students reason failure in programming to external factors they are not realising their role and responsibility in the learning process. Furthermore attributions identified in the stable dimensions are troubling, as this causal reasoning has been found to significantly decrease persistence in tasks [28]; a prime concern for CS education.

In our exploration of the causal dimensions, we were able to identify 'successful' strategies adopted by some students, which they believed assisted them in overcoming obstacles that others perceived to be beyond their control. This suggests there could be great value in connecting problematic causal reasoning with learning and CS-specific strategies to identify ways that students can obstacles in programming assignments. This emphasises the importance to also to teach students appropriate reasoning about learning outcomes and the use of learning processes and strategies. We map some examples in Table 3.

Table 3: Aspects of causal reasoning and possible strategies

Perception/ Behaviour	Possible Intervention Strategies
Unknown causes (external/unstable)	
Does not realise what factors contribute to a particular outcome. Difficulty in reflecting on learning processes and identifying causal attributions.	Encourage reflection, focusing on specific learning processes. Discuss possible outcomes in programming assignments and causal attributions. Use case studies and students identify causal attributions of outcomes.
Unchangeable causes (internal/stable and external/stable)	
Does not recognise personal role in a particular aspects of the learning process. May cause negative feelings toward learning or feelings of inability to succeed. Behaviour may not change if unable to recognise particular behaviour influence outcomes.	Guide reasoning from external causations to internal. Case studies where students identify what might be perceived to be external attributions and then replace with internal/unstable reasons. Brainstorm discipline and non-discipline strategies to overcome issues. Encourage responsibility.
Changeable causes (internal/ unstable)	
Realises certain behaviours and strategies influences learning processes and identifies aspects of behaviour or strategies that can be adapted to improve performance.	Focus on detailed discipline-specific strategies and encourage detail about non-discipline and CS-specific strategies. Identify multiple possible options to improve performance in an aspect and continue to self-evaluate.

5.1 Limitations and Future Research

Our research confirms that CS is a unique experience for each student [18], not only in terms of performance, but how students perceive their learning experiences and the causations they apply to explain their outcomes, which influence emotions and determine how they approach future tasks. In their essays, some students expressed how their outcomes made them feel, however, this was not something that was consistent across essays and we could not measure this aspect. Nonetheless, this offers opportunities to explore the link between CS students' attributions and emotions, particularly self-efficacy and motivation. Explorations could track students across their CS experience and determine if there are changes in causal attributions and how emotions influence persistence and motivation in CS degrees.

Our research has raised an issue in regard to the way that we consider the use of reflective essays in CS. While reflective essays are implemented to encourage students to reflect on their programming processes and learning strategies, to what extent are we providing useful feedback to students about the effectiveness of their strategies or the way they are reasoning? Currently, it appears, an assumption is that the use of reflective essays is valuable enough in itself to promote reflective thinking. However, what we find is that, through analysing students' causal reasoning using the AT framework, a number of students had problematic reasoning that could affect future performance. Students who focused on external or stable causes of failure may not change their behaviour to induce favourable learning outcomes because they do not recognise their responsibility in the learning process. Using our map of attributes, we are more easily able to identify when students are using problematic reasoning in their essays and offer guidance to correct this, either directly or via activities. Such activities could involve having students share their experiences, successes and problems encountered with one another so that students who are attributing to external causes might hear experiences of students who, despite encountering difficulties, were able to control their outcomes through using discipline-specific and learning strategies. This offers opportunities for CS education research to explore if students' behaviours do change over time with feedback or the correction of misguided reasoning and determine if intervention methods can have a positive impact on programming performance and learning, as it has been found to in other disciplines. Whilst we adopted Weiner's earlier model, this offers opportunities for research to develop understandings in the light of more recent theories. Researchers in other disciplines may apply similar content analysis to students' reflections to determine similarities or differences across disciplines or courses.

With the intention to encourage CS students to be self-directed learners, we have examined causal attributions by focusing on the changes students can make in their thinking and learning processes. However, external causal attributions could also provide guidance for re-designing courses to enhance learning, for example, in terms of program assignment clarity, expectations and making resources more visible.

6. CONCLUSION

This research has contributed to identifying non-discipline and CS-specific causal attributions students make about their outcomes of CS programming assignments. Our analysis reveals a number of casual attributions that may be problematic to students' future performance and their persistence in CS. Given that lack of time, difficulty and motivation are some of the main reasons students leave CS, our findings suggest there may be ways to improve students' CS experiences by exploring how to retrain problematic reasoning of programming assignment outcomes by encouraging students to control learning situations, by the use of internal/controllable reasoning and learning and discipline-specific strategies. This research highlights the value in continuing to investigate how we can improve CS students' experiences by exploring the way they perceive success and failure in CS programming assignments.

7. REFERENCES

[1] Alderman, M. 2013. *Motivation for achievement: possibilities for teaching and learning*. Taylor & Francis: London.

[2] Barker, L., McDowell, C., & Kalahar, K. 2009. Exploring factors that influence computer science introductory course students to persist in the major. *SIGCSE Bulletin, 41*(1), 153- 157.

[3] Crowe, S., Cresswell, K., Robertson, A., Huby, G., Avery, A., & Sheikh, A. 2011. The case study approach. *BMC Medical Research Methodology, 11*(1) 100.

[4] Dewey, J. 1910. *How we think*. D.C. Heath & Co.: Lexington.

[5] Dunn, K., Osborne, C., & Rakes, G. 2013. It's not my fault: understanding nursing students' causal attributions in pathophysiology. *Nurse Education Today, 33*(8), 828- 833.

[6] Hanks, B., Murphy, L., Simon, B., McCauley, R., & Zander, C. 2009. Cs1 students speak: advice for students by students. In *SIGCSE,* Chattanooga, TN, 19- 23.

[7] Hawi, N. 2010. Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers & Education, 54*(4) 1127- 1136.

[8] Haynes Stewart, T., Clifton, R., Daniels, L., Perry, R., Chipperfield, J., & Ruthig, J. 2011. Attributional retraining: reducing the likelihood of failure. *Social Psychology Education, 14*, 75- 92.

[9] Heider, F. 1958. *The psychology of interpersonal relations*. Psychology Press: New Jersey.

[10] Henry, J., Martinko, M., & Pierce, M. 1993. Attributional style as a predictor of success in a first computer science course. *Computers in Human Behavior, 9*(4), 341- 352.

[11] Kinnunen, P. 2009. Challenges of teaching and studying programming at a university of technology - viewpoints of students, teachers and the university. *Department of Computer Science and Engineering,* Helsinki University of Technology, Helsinki, Finland, 259.

[12] Kinnunen, P., & Malmi, L. 2006. Why students drop out CS1 course? In *ICER* Canterbury, UK, 97- 108.

[13] Lewis, C., Yasuhara, K., & Anderson, R. 2011. Deciding to major in computer science: a grounded theory of students' self-assessment of ability. In *ICER*, ACM, 3-10.

[14] Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., et al. 2004. A multi-national study of reading and tracing skillls in novice programmers,". *SIGCSE, 36*(4), 119- 150.

[15] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., et al. 2001. A multi-national, multi-institutional study of assessment of programming skills of firstyear cs students. *SIGCSE Bulletin, 33*(4), 125- 140.

[16] Parham, J., Gugerty, L., & Stevenson, D. 2010. Empirical evidence for the existence and uses of metacognition in computer science problem solving. In *SIGCSE*, Wisconsin, ACM, 416-420.

[17] Perry, R., Hechter, F., Menec, V., & Weinberg, L. 1993. Enhancing achievement motivation and performance in college students: an attributional retraining perspective. *Research in Higher Education 34*(6), 687- 723.

[18] Ramalingam, V., LaBelle, D., & Wiedenbeck, S. 2004. Self-efficacy and mental models in learning to program. In *ITICSE,* Leeds, UK, 171- 175.

[19] Salajan, F. D., Schönwetter, D. J., & Cleghorn, B. M. 2010. Student and faculty inter-generational digital divide: Fact or fiction? *Computers & Education, 55*(3), 1393-1403.

[20] Scott, M., & Ghinea, G. 2013. Educating programmers: a reflection on barriers to deliberate practice. In *Learning and Teaching in STEM Disciplines* The Higher Education Academy, Birmingham, UK.

[21] Upchurch, R. L., & Sims-Knight, J. E. 1999. Reflective essays in software engineering. In *FiE,* IEEE, vol. 13.

[22] VanDeGrift, T., Caruso, T., Hill, N., & Simon, S. 2011. Experience report: getting novice programmers to THINK about improving their software development process. In *SIGCSE*, Dallas, TX, ACM, 493-498.

[23] Weiner, B. 1985. Attribution Theory. In *Human Motivation* Springer New York, 275- 326.

[24] Weiner, B. 1985. An attributional theory of achievement motivation and emotion. *Psychological review, 92*(4), 548- 573.

[25] Weiner, B. 1985. "Spontaneous" causal thinking. *Psychological bulletin, 97*(1), 74.

[26] Weiner, B. 1986. Attribution, emotion, and action. *Handbook of motivation and cognition: Foundations of social behavior, 1*, 281- 312.

[27] Weiner, B. 2010. The development of an attribution-based theory of motivation: a history of ideas. *Educational Psychologist 45*(1), 28- 36.

[28] Wilson, B. 2002. A study of factors promoting success in computer science including gender differences. *Computer Science Education, 12*(1-2), 141- 164.

[29] Zanial, Z. 2007. Case study as a research method. *Jurnal Kemanusiaan, 9*, 1- 6.

[30] Zimmerman, B. 1989. A social cognitive view of self-regulated academic learning. *Journal of Educational Pyschology 81*(3), 329- 339.

Social Media in Everyday Learning

How Facebook Supports Informal Learning Among Young Adults in South Africa

Tina Klomsri
Stockholm University
DSV
Kista, Sweden
tinaklomsri@gmail.com

Linn Grebäck
Stockholm University
DSV
Kista, Sweden
linngreback@gmail.com

Matti Tedre
Stockholm University
DSV
Kista, Sweden
first.last@acm.org

ABSTRACT

Social media has in the recent years become a part of people's daily lives, and with it has come a new way to communicate and interact. The functions of social media in formal and non-formal learning are well studied, but much less attention has been paid to their role in informal learning. In South Africa the dominant social networking service is Facebook, which crosses many digital divides in the country. This study explored, through semi-structured interviews, the use patterns of Facebook among young, low-educated South Africans adults, and analyzed, using grounded theory, the potential of those use patterns for informal learning. The results show that the social interactions on Facebook support informal learning and personal development. Facebook puts the individual and the social interactions in focus without the constraints of time, objectives, or curricula. Young adults are daily exposed to various kind of information on Facebook, while maintaining control over their own Facebook activities. Self-directed learning and intrinsic motivation promote continuous discovery of new knowledge, yet sometimes the educational benefits are undermined by a lack of critical attitude towards information in social networks.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education; K.3.1 [**Computers and Education**]: Computer Uses in Education—*Collaborative learning, distance learning*

General Terms

Human Factors

Keywords

Social media, Facebook, informal learning, social interactions, South Africa, ICT4D, self-directed learning, intrinsic motivation

Koli Calling '13, November 14–17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526983

Acknowledgments

This paper is based on the B.Sc. thesis "*Social Media in Everyday Learning: A Qualitative Study about What Use Patterns of Facebook Support Informal Learning among Young Adults in South Africa*," Stockholm University, Department of Computer and Systems Sciences (DSV), by Tina Klomsri and Linn Grebäck [15]. The authors would like to thank prof. Henrik Hansson, the thesis co-supervisor.

1. INTRODUCTION

The role of theoretical and formal knowledge has steadily strengthened throughout the human history. For the past few hundred years, knowledge gained by formal and empirical methods has become the pinnacle of knowledge construction (cf. [25, p.19]). In terms of learning, formal learning is highly valued in society, while informal learning is often seen as less valuable [8]. But the whole picture of learning is much broader. The diffusion of modern information and communication technology (ICT) has changed the way people live, learn, and communicate, and the effects of those changes to learning are not yet fully understood [8]. Along with the Web 2.0, the increased use of social networking sites, such as Facebook, has made it possible for large amounts of information to be exchanged and spread globally at no time. That new media society has created new conditions for informal learning [8] and for the lifelong learning process.

While the changes brought by modern ICT are most widespread in wealthy countries, the changes are dramatically seen in developing and emerging economies. In South Africa, for instance there are many ongoing initiatives made by the South African government to increase access to education for the entire population. One of the initiatives is to reduce unequal access to infrastructure, facilities, and learning resources within the country [32]. The South African government hopes that information and communication technology will "play an essential role in addressing societal challenges - for instance, universal education, training, health, inclusion, security and environmental management" [27].

The modes in which ICT already addresses the South African government's aspirations are many, and one of the promising strands of research has focused on the explicit use of social media for learning. Much research on learning through social media has focused on the organized, formalized, and directed aspects of learning. This study focused on the less studied, informal aspects of learning, and it asked, "*Which use patterns (if any) of Facebook among low-educated, South African young adults are potentially beneficial for informal learning?*".

1.1 Research Context

This study was situated in Johannesburg in the Gauteng province of South Africa. The Gauteng province has the highest population among South Africa's provinces (12.2 million people, or 23.7% of total population) even though it is the smallest province (16 963 km^2) [29]. When it comes to education in South Africa, statistics show that 8.6% of people aged 20 and above have never attended any kind of formal education [28], which is reflected in the literacy rate of 89% [32]. Of people aged 20 years and above, 33.9% have attended some secondary education and 28.9% have finished the 12th grade. Only 11.8% of people 20 years and above have attended tertiary education [28].

The most common piece of modern ICT equipment used among South Africans is the mobile phone. Among all the mobile phone owners, young people have embraced technology the fastest: Already in 2007 a survey study found that some 72% of the young people in South Africa, aged 15–24 years, owned a cell phone [1]. Of those who had access to Internet, 16.3% accessed it via cellphone. Concerning the uses of Internet, a 2009 market research study showed that social networking was one of the main internet activities among South Africans (74% of users) [31]. When asked which social network sites they used, the study showed that Facebook was the leading social network site (82% of social network users). Social network sites are usually free of cost and anyone with an e-mail account can become a member. The same research study [31] showed that main activities on social network sites were as follows: 75% sending messages to people, 62% updating status, and 61% uploading photos or videos. A lot of information is constantly distributed via different social network platforms.

The report also showed that social network use among a representative sample of 401 South Africans aged 16 years and older was 74% [31]. But as Kreutzer [16, p.6] pointed out, there is still little research on how South Africans "choose to use mobile phones to access information or entertainment media or to create and distribute their own media." There is a need for a qualitative study, which investigates how young, low educated South Africans use social network sites to achieve personal goals and interests. This is of value in terms of learning insofar as it sheds light on those processes of learning that happen outside formal education settings. The availability of social networking can work as a cheap and efficient way to learn and exchange information by "virtually meeting people from other age-groups and socio-cultural backgrounds, linking to experts, researchers or practitioners in a certain field of study and thus opening up alternative channels for gaining knowledge and enhancing skills" [24].

1.2 Three Levels of Formalization

One common conception of learning today is that which happens in institutions like schools and universities [6, p.217]. But that is but one view of learning. One can take a broader look at learning, and discuss learning in three different terms, based on their level of formalization: formal learning, non-formal learning, and informal learning [34].

Formal learning

Formal learning corresponds to an institutionalized kind of learning that is established and accepted in society, and that typically takes place in schools, universities, and other educational institutions [34]. It is rule-bound and bureaucratic, follows a rigid curriculum and often confers a degree or diploma. The knowledge gained from formal learning is often focused on propositional and decontextualized knowledge, gained through one-way communication from teacher to student (although modern pedagogical views often challenge that process), and organized and taught by professional teachers or tutors [22, p.19].

Non-formal learning

Non-formal learning is learning that occurs in situations with a lower level of formality, but with a plan and objectives for learning. Such situations include, for example, study circles and homework, and exist outside of any formal institution [22, p.20]. Such study environments for non-formal learning are typically minimally guided (which is not an optimal choice for novice learners [14]) and offer little affective support [2].

Informal learning

The learning that takes place in everyday life is called informal learning. Informal learning is personal and spontaneous, without structure, and can be either conscious or unconscious [22, 34]. In this type of learning, the individual is not always aware of the learning nature of the situation. For example, if a person visits a museum on his/her own initiative, and does not have to meet any explicit learning objectives, then the situation counts as informal education [34]. Informal learning is often contextualized and takes place where an individual can make immediate use of the knowledge obtained (e.g., [17]).

Today a lot of information is digital and available through modern ICT equipment, and hence the context of informal learning can also be virtual, physical, or a blend between the two. For example, a person can learn something by watching a video online and make use of the knowledge immediately in the real world context. As another example, a farm fair visitor may acquire more information about the products through mobile Internet. Contextualized learning contributes to lifelong learning, as understanding the context in which the learning takes place provides scaffolding that helps one to remember what was learnt and to pass on the knowledge [23, pp.30–31]. Since informal learning is controlled by individuals themselves, it has the great advantage of supporting intrinsic motivation (see [22, p.31], [17, 5]). Formal, non-formal, and informal learning are neither exclusive categories nor hierarchical levels of 'better' and 'worse.' The different kinds of learning can supplement each other so that they meet the needs of individuals and the society [34].

2. PREVIOUS RESEARCH

Learning through social networks is a well-researched topic. A lot of the literature is about introducing social networks and interactive media in institutional establishments. Examples include, for instance, Bull et al.'s [8] study on connecting informal and formal learning experiences in social media, and Redecker et al.'s [24] survey on how social networking can be, and is used to gain knowledge. Many research studies are about the different ways in which the interactive and social aspects in digital media can promote learning, and should therefore be introduced in formal education institutions.

Why social networks are popular

Human is by nature a social being, and socializing with others and communicating with the world around us is vital for our learning process [11, pp.33–34]. Social networks answer to some basic human needs, such as wanting to be included in a group and a context, as well as to understand one's place in the world and to reflect on how others see us [23, p.23]. The social nature of humans explains the ways in which people interact and behave in social networks. From a socio-cultural perspective, the sense of belonging in a social context is a driving force for learning, and relates to personal growth. This perspective suggests that interaction in social contexts offers different ways of development of personal identity [23, p.28].

Social networking in formal education

Social networks offer a new way to communicate, access and exchange information, and realize personal goals [24]. Physical distance, time and even language barriers no longer limit people from communicating and exchanging ideas. Social networking allows users to communicate, interact, and create their own content and share it with whomever they choose [4, p.128]. Social networks and informal learning are linked together, because social networks are not constrained by specific learning objectives, curricula, or deadlines [8]. In Europe, one study showed that social networking offers new opportunities to enhance innovation and creativity by producing one's own digital materials, involve learners more actively in their own learning process, "actively support lifelong learning by offering accessible, flexible and dynamic learning environments that can complement and supplement initial training" [24], reduce inequalities by increasing access to information, and reinforce active citizenship.

Previous research studies have shown that some of the most common problems in formal education settings are lack of motivation, failure to meet the demands derived from formal education, and shallow learning process (such as memorizing) [17, 22]. Bull et al. [8] and Åkerlund [23] suggested that introducing non-formal systems, such as social networks, could provide new opportunities for innovation and modernizing education, which could solve some of the problems above. Bull et al. pointed out that by linking students' attention with academic content exclusive of academic constraints such as deadlines and curricula, students can learn at their own pace and choose the most appealing topics [8]. That can be done with the use of social networks. Such linking of media, social interaction, and technology is well explored in education, and through social networks such conception can be more multifaceted and relevant and therefore more interesting for the students [11].

There are also various barriers for introducing social networks into formal education settings. Unlike individual teachers, formal education systems are notoriously slow to adopt changes [4, p.131]. Yet, also individual teachers would be required to update their ICT skills regularly and rethink their role to "act as guides and mentors, enabling and facilitating self-regulated learning processes" [24] (see also [26, 21]). It has been argued that in formal education settings there is a lack of clear definitions of terms such as formal, non-formal, and informal learning, which makes it difficult to establish basic non-formal or informal features within schools and universities [34]. There again, it is exactly the lack of any institutional characteristics that define informal learning.

Critical and reflective media consumption

There are different forms of knowledge and information, but for it to be valued, it has to be considered in critical ways [8]. Especially adolescents sometimes lack the attitude to critically evaluate the quality, veritableness, and authenticity of content in digital media [24]. In a many-to-many media, such as social networks, misinformation and disinformation abound—some of it misleading, some downright dangerous. To overcome this, teachers need to supply students with necessary skills to critically use social networks [24].

Social networks and innovation

Redecker et al. [24] listed 'four C:s' of the innovation-supporting character of social networks. Firstly, *content* "supports learning and professional development in a lifelong learning continuum, contributes to equity and inclusion and puts pressure on education and training institutions to improve the quality and availability of their learning material" [24, p.8]. Second, by *creating* and publishing digital content, learners can gain new knowledge along the way. This contributes to a vast amount of user-generated content, which other learners and teachers can take advantage of. Third, *connecting* with other learners and exchanging knowledge in different fields of interest supports innovation. Fourth, *collaboration* means the possibility for teachers and learners to work together, "pooling resources and gathering the expertise and potential of a group of people committed to a common objective" [24, p.8].

Social networks and inequality

There are many potential benefits arising with the introduction of social networks, but it is important to keep in mind that various faces of the digital divide still remain. It has been argued that the Internet and ICT may have increased, not decreased, inequality between the rich and the poor, by providing the best technological facilities only to those already in a privileged position in society [16, 20]. In South Africa, access to the web has only been available to the wealthier part of the population. Only recently, the Internet and ICT in general have become more available to the rest of the population [16].

3. METHODOLOGY

Due to this study's exploratory stance and the paucity of earlier research on the topic in this context, this study adopted the grounded theory strategy with interviews as the data collection method [9, p.63]. The research question of this study sought for use patterns that are potentially beneficial for informal learning, insofar as there exist any. To find these, semi-structured interviews were arranged to gather qualitative data about young adults' Facebook habits, thoughts, opinions, and use patterns. Interviews are well suited for gathering personal opinions, feelings, and views of a certain matter, as they are not constrained in the way questionnaires are [33, p.154].

The main reason for choosing semi-structured interviews instead of other types of interviews was that they are flexible and generate personal answers from respondents [7, p.45]. Respondents expressed their answers in their own ways (there were no predetermined answers to choose from) while the interviewer could ask follow-up questions on interesting topics. During the interviews, the interviewers had a guide to support them by providing a list of the most important themes to discuss [7]. This ensured that each important question was discussed and that the interviews did not get sidetracked. The same guidelines were used for every interview, and the actual interviews were preceded by three pilot interviews followed by refining of the guidelines.

Selection of Informants

In qualitative research it is very common among researchers to pursue purposive sampling [9, p.127]. Sampling is done in accordance with the research questions and the overall purpose of the research. When the aim of this study is to explore, not to generalize, the researcher can focus the sample on the entities (such as individuals) of interest. This study set the sampling frame using a type of purposive sampling called *theoretical sampling* [9, p.127], which is closely related to grounded theory methods of analysis. Theoretical sampling considers what is relevant and meaningful for the theory and selects each subsequent participant according to the information from the previous coding processes, so that the theoretical ideas can be validated [7, pp.394–395].

Considering the research question and focus of this study, the informants were young adults between 18 and 29 years old, as they belong to the group of active Facebook users. The informants were low-educated (no tertiary education) young adults from a less affluent suburb of Johannesburg. The interviewees that fulfilled the above mentioned criteria were arbitrarily selected in the suburb, based on the sampling frame, and asked if they want to participate in the study. This is called convenience sampling and facilitates the collection of data. Convenience sampling makes it impossible to generalize the research result since there is no way to know what the sample is representative of. When a suitable person was interested in participating in the interview, a location and time for the interview were agreed on. The collection of data ended when new data no longer provided new information and no new categories were identified in the coding process. This is called data saturation [18], and the saturation point was reached after twelve interviews, which is relatively common (typically 5–25 in case of phenomenology [9, p.128] and case studies [19]).

Data Collection Procedure

The interviews were conducted face-to-face. The reason not to interview via phone or online communication tools (such as Skype or social media) was that respondents have proven to be more likely to cancel an interview that is not carried out face to face, and there is always the risk of technical problems [7, p.433]. Also gestures and body language of the respondent will go missing, which would be a pity since it can provide a better understanding of answers [7].

Before every interview, the interviewers introduced themselves to the interviewees, informed them about the purpose of the study, and requested their informed consent. By the end of every interview, the interviewers thanked the respondents for their participation. The interviews were conducted in an undisturbed environment, with only the interviewers and interviewee present. This way the interviewee could speak freely and without worrying for anyone to hear or getting interrupted. Interviewers strove for objectivity and neutrality when interviewing by, for instance, not asking leading questions [7, p.420] [12, p.54].

Three pilot interviews were the basis for the formulation of the final interview questions. The interviewers used the same guideline for every respondent, and made sure all important questions and topics were discussed. Everything of importance that was said during the interview was recorded with a digital recorder on a mobile phone. After an interview was done, it was transcribed and analyzed as soon as possible, before conducting the next interview. This gave the opportunity to discuss every interview thoroughly while the memory was still fresh and to reflect on new categories and codes identified [33, pp.160–161].

The two authors interviewed one respondent at a time. One of the authors had the role of the interviewer, while the other author took notes. The average time of the interviews was around 15 minutes. All of the interviews were recorded and later transcribed. All respondents participated voluntarily in the study and without being compensated. Before every interview, the authors socialized with the respondents to make the respondents feel comfortable before beginning the interview.

The respondents were also asked for permission to be recorded. The interviewer followed an interview guide during the interview but since it the interviews were semi-structured, the interview guide was not followed strictly. The first questions in the interview were about the respondents' background, such as age, education, occupation, and why they joined Facebook. This was to get to know the respondents better and to be able to put their answers in a proper context. Next questions were about the social aspects of Facebook. Those questions asked how the respondents interact with other Facebook users. The respondents' answers on these questions described how they used Facebook to socialize and connect with other members of Facebook, as learning is a part of a social process. The last theme of the interview was about informal learning and contained questions that were direct and therefore minimized the chance of misinterpretation in the coding of data.

Research Ethics

Standard research ethics were followed as described in Bryman's four guidelines [7, pp.131–132], and the study was not concerned with very sensitive matters. Firstly, interviewees were provided with information about the research aims. It was made clear that participation is voluntary and that the data collected will only be used for research purposes. Secondly, informed consent was recorded, and since the study did not involve minors, the individuals had the right to decide whether or not they wanted to participate in the study. Thirdly, the respondents were guaranteed anonymity, and they were informed that the interview recording will be protected from unauthorized access. Fourthly, the material the respondents provided will not be used for any other purposes than this study.

Data Analysis Procedure

Methodologists recommend grounded theory strategy for qualitative studies where the intent is to go beyond description and develop an abstract schema for future action [9, p.63],[30]. Hence, grounded theory was chosen as data analysis method. One of the main tasks in grounded theory is to code the analyzed data by identifying the parts that could be of theoretical or practical significance for the research. The data works as indicators of behaviors and events. By finding common and deviant indicators and organizing them, the researcher is able to separate, label and compile the data as a first step to generate a theory [7, p.514]. This process is performed continuously.

Strauss and Corbin [30] described three stages of coding data. The first step is open coding, which generates indicators by separating, analyzing, comparing, and categorizing data. Identified indicators are grouped and divided into categories. The second step is axial coding, which is described as "a set of procedures whereby data are put back together in new ways after open coding, by making connections between categories" [30]. During this step the codes are put into context, giving them a meaning. The final step is called selective coding. By relating the categories to each other, the researcher identifies a core variable, which functions as a framework where all categories are integrated. In this study, those principles worked as a base and guideline during the analysis process of the empirical data. To assist the memory, notes and memos were used as a reminder of what different indicators, groups, and categorizations meant. A case-based memo was written down directly after each interview consisting of the researchers' impressions, thoughts, and interpretations of what was said during the interview.

4. RESULTS AND DISCUSSION

The open coding process generated 19 indicators that were analyzed and grouped into four categories: *Personal development, Social interactions, Simplifying life*, and *Privacy issues.* The first row of Table 1 presents each of the four categories, and after the first row, each column presents the indicators grouped under that particular category. Privacy issues are their own box to save space because they were smallest, only one-item category.

Table 1: Nineteen Indicators and Four Categories

Personal Development	**Social Interactions**	**Simplifying Life**
Channel for expression	Asking	Easy way to communicate
Create	Direct or personal information	Keeping updated
Critical view on information	Encouragement and support	Saving time
Identity	Exchange of information	Sharing information
Independent information seeking	Group communication	
Personal goals		
Personal interests		
Playing games		**Privacy Issues**
Using knowledge		Facebook privacy

The indicators belonging to the category *Personal development* were focused on the respondents' Facebook activities that could lead to new knowledge or developed skills. *Social interactions* arose from the data as a category because many of the indicators directly referred to some type of social interaction, such as *asking* or *exchange of information.* The indicators included in the category *Simplifying life* were grouped together as they are all about aspects of Facebook that simplify the respondents' lives in some way, such as by *saving time*, by helping the respondents to *keep updated* with different people and things, and by providing an *easy way to communicate. Privacy issues* was defined as a category even though it consists of only one indicator, *Facebook privacy.* This group is about privacy and identity theft problems that the respondents experienced with Facebook. The indicator *Facebook privacy* did not fit into any of the other three categories but was still important enough to be left in the analysis.

The core category identified in the final part of the coding process was *social interactions*, since all of the indicators are products of social interactions on Facebook. Even the indicators that are not directly about social interactions are still related to it, as all content and activities on Facebook are social and connect everything on the website.

Personal development

"*I want to know about people's views especially when you put up some comment. When you put something on Facebook you have to view some people's comment and what they think of stuff.*"
(Male, 28 years old)

Just like face-to-face interactions, social networks like Facebook are places where people express themselves and form an identity by interacting with others. The respondents are aware that the information they post on Facebook will be seen and evaluated by others. Their Facebook activities can therefore receive both positive and negative feedback, as well as start discussions where the respondents might have to explain themselves or consider other perspectives on a matter. By exchanging views and communicating on Facebook, the respondents learn new things about their world and gained informal knowledge. Those respondents who ask for advice on Facebook, gain even more informal knowledge, since it is easier to gain a deeper understanding when the knowledge is contextualized [23]. As one respondent explained how he learned surfing techniques by watching videos posted by his Facebook friends: "*They are pro surfers so they are better than myself so I look it up to just see how they are doing everything, you know. What's new? What's old? Keep up with the surfing world*".

Personal goals and personal interests are important aspects of informal learning, and many examples of this can be found in the respondents' Facebook use. The interviews revealed that the knowledge the respondents gain is linked with personal interests and personal goals, since the respondents pay attention to those posts they find interesting on their newsfeed and groups, and manage their Facebook so they get updates on topics that interest them. Things that the respondents mentioned as interesting was either a personal hobby, or things like: *"a cool quote or a nice picture", "something useful", "funny stuff"* and *"what's going on in the world, what's new".* When one respondent had to answer what he found interesting on newsfeed, he answered: "*nice photos, nice clothes, what's happening, who bought a car, who did this and this, that catches my attention. This other group, people post their photos and by the end of the day they chose who has the best photo. Then people will comment on it. Apart from that I check news*". When learning occurs as a result of personal interests or to reach personal goals, learners are more likely to remember what they have learnt [23]. According to the respondents, Facebook is a favorable platform for sharing digital content (both self-created and the content found on the Internet) since it generates useful and/or encouraging feedback from their Facebook friends. Positive feedback from Facebook friends works as an incentive to the respondents to share more information, and sometimes even to do more activities in real life. Feedback also provides a channel for increased critical attitude towards media by one's peers.

Playing games is something the respondents do on their own initiative and is connected with personal interests. Depending on the game, gaming activities can be very social when Facebook friends play with or against each other. Some games are connected to real-life events and keep the respondents updated with what is going on in real time. Since playing games is something the respondents find a lot of fun and interesting, they are attentive and motivated while doing it and the chance of learning is high. Sometimes certain knowledge or skills are required in order to beat the game, which motivates the respondents to achieve what is required. This means that the respondents, while playing games, can increase their motor skills. One respondent explained that by playing football games she learns strategy skills, but since she is having so much fun while doing so she does not realize how much she actually learns.

Some of the information that the respondents get exposed to on Facebook leads to a critical view on information and independent information seeking—two things that in turn contribute to personal development. When the respondents take a critical view on information they read or watch, they tend to seek further information on the search engine Google: "*if I find something that seems you know a bit unlikely I would ask if it's true or Google it*". Often the respondents go back to comment on the topic after finding out more about it, and by doing so they join a social process of exchanging and sharing knowledge.

The respondents engage in a broad variety of cognitive activities and knowledge-seeking in Facebook. By using Facebook, the respondents can gain knowledge either spontaneously, by for example scrolling through the newsfeed, or consciously by searching for certain information. In both cases, the knowledge gained can be of use for the respondents, either in the virtual or the real world. This is especially true in the latter case, as many of the respondents use Facebook when they need advice or information that can help them with something practical. One respondent explained: "*sometimes I ask for advice about studying, like when I'm in my room and I just can't sit down and study. Then I asked my friends on Facebook, "what should I do, how can I study?". They told me to not sit in the couch, sit on something hard, like on a bench. Don't be comfortable, sit straight, then you can get things done*". Another respondent said: "*if I have a problem, if I'm in trouble, I mismanage my finances, I always got a lot of older friends who I can turn to and ask them in their inbox*". This is a good example of how informal learning works as it takes place in an everyday situation and derives from the respondents' own personal interests and needs. Even though informal learning occurs by, for instance, reading a book or visiting a museum, it is often a result of social interactions, which today are occurring more and more within virtual spaces. Social interactions of all kinds are the core category that brings together all aspects of Facebook.

Social interactions

"*We have this group that we have, where we have goals that are under the group. So basically it encourages us, young ladies, to grow up. In a way like with manners and respect and things like that. It helps us to devote to something and you know learn that. It can encourage them, they can share with their friends also*"
(Female, 26 years old)

This category is a core category and it resonates with the theory that learning is a social process. By using communication as the main learning and teaching tool, knowledge exchange can occur [11, 13, 25, 17]. Facebook is a social network that the respondents use to communicate with their friends and family. This is the main reason for the respondents to create a Facebook account and to continue using it for many years. During the interviews other important aspects of socializing were discussed, like the exchange of information between Facebook users as a way to gain knowledge. By reading other Facebook friends' posts and comments, the respondents can learn new things about their world, whether the learning is conscious or not. The indicators 'asking', 'exchange of information' and 'group communication' are all about how the respondents gain knowledge by socializing with others, in a context where they are active at their own terms. This shows how Facebook is a platform for informal learning, which is personal and spontaneous and puts the individual in focus [34].

The indicators 'encouragement and support' and 'direct/personal information' can also be related to the informal learning process, as they affect how the respondents interact and learn on Facebook. Arroyo et al. [3, p.2] wrote that "feedback plays a fundamental role in the educational process" and the results showed that feedback, such as comments and 'likes,' are encouraging for the respondents and sometimes make them more active users. Encouragement can be given directly from one person to another, or within a group of people, or it can be given as a motivational status update that is not necessarily directed towards anyone special. Two respondents mentioned that they are members and administrators of groups where people who are facing difficulties help and encourage each other. One of the respondents explained the interaction within the group: "*everyday we get to share like two blogs, so I'll share two blogs and they'll share two blogs. Who ever read the blogs can comment on it. Or if they find it interesting or if they find that it will help them with whatever they are facing, like we have a topic that we post on so*". Another respondent continued on the same topic: "*people they want interesting stories, so some of the stories they don't get much comments. Funeral stories, they got a lot of comments, and a lot of condolences. Funny stories, they got like 100+ comments, people are commenting and commenting.*" Group communication is a social interaction towards and between many people, and it supports informal learning by providing people with information that they personally find interesting.

Furthermore, the respondents are more likely to engage with information on Facebook that is directed personally to them, which happens in social interactions and not by just checking the newsfeed. By using the search engine Google and Facebook's built-in translation function, the respondents take part in information exchange posted in a foreign language. This confirms the suggestion that social networking is no longer strictly limited by language barriers, time, or physical distance [4]. However, there were differences as to what kind of information the respondents chose to expose themselves to and what the society in which the respondents live in consider being valuable.

Simplifying life

> "*It (Facebook) helps you to get in contact with a lot of people outside the country, inside the country, make friends and a lot other stuff. A friend of mine, he went to the UK and with the communication and stuff, I think Facebook is the easiest way to communicate with him. It's very easy, very cheap, instead of making calls and other stuff. You can also see him physical with pictures and other stuff*"
> (Male, 29 years old)

The indicators belonging to this group are about how the respondents use Facebook as a means to simplify their lives. 'Easy way to communicate' is about accessibility and availability of Facebook. Many respondents expressed that more and more people they know are joining Facebook and since it is free, it has become a generic and preferred way to communicate: "*... it's the best way to communicate with family members, you know, socialize, connect with other people. Since I live in South Africa and my family is in Zimbabwe. And I have friends, old schoolmates, we need to connect so, that's why I joined Facebook*". In order to join Facebook one must only be over the age of 13, have an Internet connection, and have a valid email address. Internet access is still unequally distributed among South Africans and for many respondents, daily Facebook use is not always possible. The respondents who have a high-speed Internet access tend to check their Facebook page more often, which means that they can take part of more information exchange and communicate more often.

The use of Facebook is an optional and spontaneous activity, which the respondents mainly use to keep updated with friends and/or family, groups, games, events, and so on. Even though a few respondents did not have high-speed Internet access, they would find a way to log on to Facebook: "*when I quit work, or on my day off, I go to an Internet cafe and go on Facebook for maybe four hours. And then, in those four hours, I can learn something new, I can check if I missed something, what is new. So in that way it simplifies my life*". One respondent expressed that she knew a lot of people who could not operate a computer but knew how to use Facebook because "*it is so much fun.*" When an everyday activity is fun and of personal interest, it is more likely that the learner will gain a deeper understanding of the subject [10]. The informal knowledge gained from different Facebook activities can later on be consciously or unconsciously applied to other situations—including a sort of informal introduction to "21st Century skills" or ICT literacy. Facebook offers the respondents a non-traditional and immediate way for communicating, sharing, creating, and keeping updated with people and things while having fun. The activities on Facebook serve as a way to fulfill the human needs of belonging to a group, situating oneself within a context, and identifying oneself [23].

Since humans are social beings, sharing information—be it face-to-face or virtual—is a part of the socializing process. As the amount of available information increases in the Internet, the respondents can obtain more information and share it with many people. This does not necessarily mean that the information the respondents share or understand is considered valuable according to the norms of society, or highly appreciated within formal education settings, since informal learning is devoid of objectives and curricula [34]. The interviews revealed a major problem, too: the information posted on Facebook is most of the time not viewed critically. This could contribute to disinformation, misinformation, rumors, and it might conduce inequality.

The respondents use Facebook mainly as a way to keep updated with things like different people, organizations, and celebrities, and by doing so Facebook saves the respondents' time and financial resources since Facebook helps gather all information updates in one place: "*I follow Arsenal, that's my team, so there is this Arsenal fan page. But I'm not really into it, I just simply get it on my newsfeed, because I like the page. Like if Arsenal is playing, they make updates. When I'm busy I can just quickly check, "Arsenal is leading" or "Arsenal is losing", then I know. The other thing is what's happening in the world, keeping updated. I did not know, but I saw it on Facebook, they were launching this new Blackberry phone*". The amount of time the respondents spend on Facebook varies, as does the amount of information perceived by the respondents everyday. What the respondents do with the information varies too, but consciously or not, they are participating in a wealth of information-sharing through Facebook. By socially engaging in information flows, no matter if it is about sharing knowledge through digital text, photos, videos, or by reading other people's status updates, the respondents are participating in a spontaneous way to learn—namely informal learning [22, 34].

Privacy issues

> "*It (Facebook) simplifies my life a lot. It can also be complicated with the privacy issues, when you have certain friends on Facebook that you don't want to see some things, or your family. So it can be restricting, so, if they made like separate functions... I know you get separate lists but it's so difficult to add people and then you want them to see some things and you want them to not see some things. So it's a bit complicated, but really, it does simplify my life, it actually makes socializing so much easier.*"
> (Female, 27 years old)

Many respondents experienced that sharing information can be restrictive as they have to be conscious about what information their Facebook friends are allowed to see. This leads to information awareness among the respondents since they do not want to offend any of their Facebook friends with their posts, or share too much information with certain friends or family members. A consequence of this is that informal learning process can be affected negatively since the young adults may feel inhibited by other Facebook members.

4.1 The Connection Between Categories

The interviews made clear that the information and activities on Facebook are not limited to Facebook only. There is a constant flow of information between Facebook and other websites as well as the outside world of the respondents. Facebook forms a network that connects people all over the world by linking their accounts together, and expands as people interact in more than one network of friends. By sharing each other's posts, joining and inviting each other to groups, events and other social phenomena, the social interactions are very much unlimited and connect all content on Facebook in a vast spider web. Every day millions of activities take place on Facebook; meetings between old and new friends to keep each other updated on what is going on in their lives. Information updates from different pages are shared with millions of people, who in turn interact with the posts. Facebook is a network where all aspects of social interactions exist and come together. It works as a news channel, communication tool, meeting place, photo album, forum for discussions, billboard for advertising, spare time activity, a channel to express personal opinions, spreading awareness, and so on. The common denominator for all activities on Facebook is that it is social, and that was also the crucial element tying together the four categories identified in this study.

5. CONCLUSIONS

The social interactions on Facebook happen through various kinds of information exchange. Facebook is a social networking site that puts their individuals and their social interactions in focus, which is an ideal setting for informal learning. The interviewed young adults in Johannesburg reported using Facebook spontaneously and often without being conscious of the knowledge gained (such as increased ICT literacy and reading skills) when using Facebook, mainly because it is a fun activity. As active Facebook users, the interviewees are exposed to a large amount of diverse information, but at the same time they are allowed to be in control over their own activities on Facebook, which intrinsically motivates intake of information and knowledge.

Since Facebook is all about sharing, the interviewed young adults are exposed to a rich flow of information everyday, and on every post there is a possibility to socially interact by giving feedback. These activities allow the interviewees to show support, encourage others, and express their own thoughts on other members' posts. This helps them to develop as persons, as they get other people's views and opinions on different matters through their own and other people's posts. By watching, listening, and reading through posts and comments, the interviewees participate in an informal learning process, and gain new perspectives of the world by communicating with their Facebook friends with different backgrounds, cultures, and languages.

Interviewees reported that Facebook gives them a chance to express their personal opinions and question different matters, which empowers them as active producers and critics of information, not only passive consumers. There are no demands or objectives when using Facebook—it is a social networking site used to socialize and enjoy. This is why the interviewees found Facebook so appealing. Posts on Facebook are more likely to catch the interviewees' attention simply because the information is directly relevant to their interests, and they can engage in discussions without feeling pressure like in formal education settings.

The results indicate that Facebook supports informal learning in the following ways: firstly, the multiple ways to share different kind of digital content to one and/or many recipients encourage young adults to explore the world through the digital posts made by themselves and/or their peers. Facebook allows the young adults to be in control over their own Facebook usage (e.g. by customizing) which makes it an efficient tool to reach personal goals and to meet personal interests, without having to fulfill any objective or follow a strict curricula. Hence, every single interaction and discovery of new knowledge on Facebook come from the young adults' intrinsic motivation and these are the characteristics that support informal learning.

Secondly, the interviews showed that active Facebook usage (unlike passively viewing information without giving or receiving feedback) is essential to the learning process as feedback provides information, encouragement, support, and sometimes forces the respondents to consider alternative perspectives on different matters. Thirdly, the social interactions on Facebook are not limited by language, time, or physical distance. The members of Facebook are from different age-groups, socio-, cultural-, and economical background, which makes Facebook a unique place where people can virtually break barriers, meet, exchange information and learn. A problem connected to the large amount of information in Facebook is the lack of critical information analysis. Since Facebook is a spontaneous spare time activity some young adults neglect the importance of questioning information posted on Facebook.

The results of the study were not surprising, but rather a confirmation of what was already suspected. The results make it easier to situate social networking in the broader educational context, as education moves towards a world with a greater focus on social and interactive online learning, both in formal and informal settings.

Theoretical Implications

The results showed that social interactions are the essential factors that support informal learning in Facebook among the interviewees. This is well aligned with the perspective of learning as a socio-cultural process, which suggests that interacting in a social context motivates people to learn and develop as persons [23]. Social interactions allow the interviewees to get feedback on their own thoughts and actions. This is essential for personal development, since it gives people a deeper and more multifaceted understanding of the world and how others see them [23]. The results suggested that feedback and encouragement increase the interviewees' Facebook activities that support informal learning. This corresponds to previous research findings that social media facilitates and increases active and interactive Internet use [24, p.5]. The results also supported previous research findings [24] that suggested that the possibility of sharing digital content plays an important role in the knowledge exchange that exists on social networks.

Previous research on social networking and learning is often concerned with evaluating the benefits of social networks within formal education setting, and with how to achieve those benefits in schools, universities, and other educational institutions. That focus derives from the common problems with extrinsic motivation [22]. There was no lack of intrinsic motivation in this study: all respondents have had a Facebook account for many years and are still using it, mainly because it is fun and because it is an easy way to keep updated with people and events. It is, however, uncertain how the respondents' Facebook activities would change if Facebook was a part of learning environment in a formal education setting. The results may also be culturally bound. This study also revealed a dire need for educating the young adults about critical attitude towards information on Facebook.

Previous studies [24] have discussed how social networking can lead to innovation, personal development, and lifelong learning, which were also shown in this study. Similar benefits of sharing digital content were found in this study, as all respondents reported that they share some kind of digital content. When the interviewees create and publish digital content on Facebook, a dynamic and flexible learning environment results from the social interactions. Knowledge can be gained and exchanged in the interviewees' fields of interest, which makes it possible for young adults with a common objective to work together and learn from each other. By sharing, viewing, listening, and reading information, knowledge is passed from one user to another and new knowledge is gained. It was discovered in this study that the respondents are more likely to participate in information exchange if the information is directed to them personally, which subsequently facilitates informal learning.

Limitations of The Study

Due to the nature of this study, this study did not aim at high transferability, but at a rich description of the informants' use patterns of Facebook. A quantitative study would be needed to establish generalizability of these results. Concerning credibility, although saturation was reached, there was no possibility for informant checks or triangulation. Interpretive research is always influenced by earlier experiences and theories, but by formulating the research questions to be as neutral and open as possible, the interview protocol aimed at giving room to the interviewees' own voices. The study did not look for disconfirming instances to increase confirmability, as the research aim and question did not aim at pinpointing them. Some respondents might have felt the need to answer the interview questions 'correctly' as an attempt to please the interviewers with what they thought were the right answers. The interview bias is always a risk as it is impossible to reassure the respondents to express their minds fully, even if it was assured that the interview material would be treated confidentially [12].

Possible Impacts of The Study

The results of this study suggest that the various social interaction patterns on Facebook do support informal learning. Facebook is not only a potential tool for formal education—it is already a tool for learning of the informal kind. Facebook should be considered as a useful way for gaining, sharing, and distributing knowledge and information outside formal education settings—especially as a learning tool that helps to maintain the young adults' motivation level. Various studies have established the essential influence of motivation on learning outcomes [5], [3, p.1]. It is a matter of further research to find out how well the intrinsic motivation shown in informal learning correlates with formal learning using Facebook as an educational tool.

Further Research

The utility of Facebook in informal learning can be improved in various ways. Since Facebook is designed for social interactions, it is an ideal environment for exchanging knowledge, and by inserting functions that promote learning Facebook might increase its use as an interesting and fun tool for informal learning. Added functions could consist of *related search results*: when a user searches for something, a result consisting of similar and related topics would appear in order to give the user a possibility to explore those topics as well. Another function is to connect Facebook to *digital encyclopedias*. This way Facebook users can rapidly search for information on things they find interesting on Facebook, and learn something new at the same time. Additionally, a function that could encourage learning through information exchange on Facebook is to *change the way information is organized on newsfeed*. By categorizing information in different tabs and giving them various headlines, such as friends, close friends, news, sports etc. Facebook users would have a clear overview of all posts retrieved from several places. This way, the users are able to structure information flow and obtain more information easily. The proposed functions should, however, be paired with development of tools for supporting critical media literacy. It is essential that young adults realize the importance of questioning information.

Furthermore, the study hopes to encourage educational authorities to see a possibility in Facebook and the way it supports informal learning. Facebook is a social networking site where personal interests play a major role in what information its users participate in, and this is shown to motivate at least the young adult users. If educational authorities take advantage of this motivation and the possibility to use Facebook as a platform for education and diffusion of information, young adults can gain knowledge while at the same time increasing their digital skills. This would have the potential for increased equity, transparency, and information literacy, all of which resonate well with the view that ICT plays an essential part of addressing societal challenges in many societies.

It is important to emphasize that all interviewees in this study had the digital skills required to use computers and knew how to use Facebook. Since all interviewees were middle class and only a few of them had poor Internet access, it would have been interesting to conduct a study about how informal learning does occur on Facebook among people with low digital literacy.

Another suggestion for further research is to compare the Facebook use of young adults with high education with young adults with low education in the context of South Africa. While most of the interviewees of this study had secondary education, some of the interviewees had not. The interviews hinted that the uses of Facebook might vary between participants with different educational backgrounds. The generalizability of these results should also be established with a broader, quantitative survey.

There are many research studies about how social networks can be introduced in formal education settings, however majority of those studies are carried out in developed countries. Since education differs around the world, it would be interesting to investigate what effects the introduction of social networks in formal education settings would have in various educational contexts in different developing countries. If benefits were found, one should study their impact on cost of education, access to education, equity, and the benefit for the poorest and most marginalized members of society. As shown in this study, Facebook motivates young, low-educated adults in South Africa to gain new knowledge, and this should be put to good use both inside and outside the classroom.

6. REFERENCES

[1] Young South Africans, broadcast media, and HIV/AIDS awareness: Results of a national survey. Survey, The Henry J. Kaiser Family Foundation, 2007.

[2] M. Apiola and M. Tedre. New perspectives on the pedagogy of programming in a developing country context. *Computer Science Education*, 22(3):285–313, 2012.

[3] I. Arroyo, K. Muldner, W. Burleson, B. Woolf, and D. Cooper. Designing affective support to foster learning, motivation and attribution. In *Proceedings of the 14th International Conference on Artificial Intelligence in Education Workshops*, Brighton, UK, 2009. IOS Press.

[4] O. Bälter and J. Thorbiörnson. Sociala medier som stöd för lärande. In S. Hrastinski, editor, *Mer om nätbaserad utbildning - fördjupning och exempel.* Studentlitteratur, Lund, Sweden, 2011.

[5] J. Biggs and C. Tang. *Teaching for Quality Learning at University: What the Student Does.* Open University Press, New York, NY, USA, 4th edition, 2011.

[6] T. Blom. Formell kunskap och praktiskt hantverk i samverkan inom akademien - en (o)möjlig ekvation? In A.-K. Högman and M. Stolare, editors, *I lärandets gränsland - formella, icke-formella och informella studier igår och idag*, pages 217–228. Gidlunds Förlag, Vilnius, Lithuania, 2009.

[7] A. Bryman. *Samhällsvetenskapliga metoder.* Oxford University Press, Oxford, UK, 2nd edition, 2008.

[8] G. Bull, A. Thompson, M. Searson, J. Garofalo, J. Park, C. Young, and J. Lee. Connecting informal and formal learning experiences in the age of participatory media. *Contemporary Issues in Technology and Teacher Education*, 8(2):100–107, 2008.

[9] J. W. Creswell. *Qualitative Inquiry and Research Design: Choosing Among Five Approaches.* Sage Publications, Thousand Oaks, CA, USA, 3rd edition, 2007.

[10] L.-O. Dahlgren. Qualitative differences in learning as a function of content oriented guidance. In R. Säljö, editor, *Som vi uppfattar det - Elva bidrag om inlärning och omvärldsuppfattning.* Studentlitteratur, Lund, Sweden, 1989.

[11] P. Diaz. *Webben i undervisningen - Digitala verktyg och sociala medier för lärande.* Studentlitteratur, Lund, Sweden, 2012.

[12] R. Ejvegård. *Vetenskaplig metod.* Studentlitteratur, Lund, Sweden, 2009.

[13] A.-K. Högman. Alternativa vägar till lärande inom räckhåll. In A.-K. Högman and M. Stolare, editors, *I lärandets gränsland - formella, icke-formella och informella studier igår och idag*, pages 130–153. Gidlunds Förlag, Vilnius, Lithuania, 2009.

[14] P. A. Kirschner, J. Sweller, and R. E. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2):75–86, 2006.

[15] T. Klomsri, and L. Grebäck. Social Media in Everyday Learning: A Qualitative Study about What Use Patterns of Facebook Support Informal Learning among Young Adults in South Africa, B.Sc. thesis for Stockholm University, DSV, 2013.

[16] T. Kreutzer. Generation mobile: Online and digital media usage on mobile phones among low-income urban youth in South Africa, 2009.

[17] F. Marton and R. Säljö. On qualitative differences in learning – 2: Outcome as a function of the learner's conception of the task. *British Journal of Educational Psychology*, 46(2):115–127, 1976.

[18] M. Mason. Sample size and saturation in PhD studies using qualitative interviews. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, 11(3), 2010.

[19] J. M. Morse. Designing funded qualitative research. In N. K. Denzin and Y. S. Lincoln, editors, *Handbook of Qualitative Research*, pages 220–235. Sage Publications, Thousand Oaks, CA, USA, 2nd edition, 1994.

[20] H. Mpogole, H. Usanga, and M. Tedre. Mobile phones and poverty alleviation: A survey study in rural Tanzania. In J. S. Pettersson, editor, *Proceedings of the 1st International Conference on Mobile Communication Technology for Development*, pages 62–72, Karlstad, Sweden, December 11–12 2008.

[21] D. J. Nicol and D. Macfarlane-Dick. Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in Higher Education*, 31(2):199–218, 2006.

[22] S. Paldanius. Studieointresserades syn på formellt och informellt lärande. In A.-K. Högman and M. Stolare, editors, *I lärandets gränsland - formella, icke-formella och informella studier igår och idag*, pages 16–31. Gidlunds Förlag, Vilnius, Lithuania, 2009.

[23] D. Åkerlund. Ungas lärande i sociala medier. In M. Alexandersson and T. Hansson, editors, *Unga nätmiljöer - nya villkor för samarbete och lärande*, pages 23–39. Studentlitteratur, Lund, Sweden, 2011.

[24] C. Redecker, K. Ala-Mutka, and Y. Punie. Learning 2.0 - the impact of social media on learning in Europe. Policy Brief JRC56958, European Commission, Joint Research Centre, Institute for Prospective Technological Studies, Luxembourg, 2010.

[25] B. Rognhaug. *Kunskap och lärande i IT-samhället.* Runa Förlag, Hässelby, Sweden, 1:2 edition, 1998.

[26] R. M. Ryan and E. L. Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist*, 55(1):68–78, 2000.

[27] STATSSA. Final draft: Information and communication technology. research & development and innovation strategy. Strategy paper 71204.4A, Statistics South Africa, Pretoria, South Africa, 2007.

[28] STATSSA. Census 2011. Statistical Release P0301.4, Statistics South Africa, Pretoria, South Africa, 2011.

[29] STATSSA. Mid-year population estimates 2011. Statistical Release P0302, Statistics South Africa, Pretoria, South Africa, 2011.

[30] A. L. Strauss and J. M. Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* Sage Publications, Thousand Oaks, CA, USA, 2nd edition, 1998.

[31] TNS. Friendship 2.0. Survey, TNS Research Surveys, 2009.

[32] UNDP. Millennium development goals - goal 2: Achieve universal primary education. Country Report South Africa, United Nations Development Programme, 2010.

[33] P. Weaver. *Success in Your Project: A Guide to Student System Development Projects.* Prentice-Hall, New Jersey, USA, 2004.

[34] C. Zaki Dib. Formal, non-formal and informal education: Concepts/applicability. In *Cooperative Networks in Physics Education - Conference Proceedings 173*, pages 300–315, New York, NY, USA, 1988. American Institute of Physics.

Exploiting Sentiment Analysis to Track Emotions in Students' Learning Diaries

Myriam Munezero[*]
School of Computing
University of Eastern Finland
mmunez@cs.joensuu.fi

Calkin Suero Montero[*]
School of Computing
University of Eastern Finland
calkins@uef.fi

Maxim Mozgovoy
The University of Aizu
Aizu-Wakamatsu, Fukushima
mozgovoy@u-aizu.ac.jp

Erkki Sutinen
School of Computing
University of Eastern Finland
sutinen@cs.joensuu.fi

ABSTRACT
Learning diaries are instruments through which students can reflect on their learning experience. Students' sentiments, emotions, opinions and attitudes are embedded in their learning diaries as part of the process of understanding their progress during the course and the self-awareness of their goals. Learning diaries are also a very informative feedback source for instructors regarding the students' emotional well-being. However the number of diaries created during a course can become a daunting task to be manually analyzed with care, particularly when the class is large. To tackle this problem, in this paper we present a functional system for analyzing and visualizing student emotions expressed in learning diaries. The system allows instructors to automatically extract emotions and the changes in these emotions throughout students' learning experience as expressed in their diaries. The emotions extracted by the system are based on Plutchik's eight emotion categories, and they are shown over the time period that the diaries were written. The potential impact and usefulness of our system are highlighted during our experiments with promising results for improving the communication between instructors and students and enhancing the learning experience.

Categories and Subject Descriptors
I.2.7 **[Artificial Intelligence]**: Natural Language Processing – *Text Analysis*

General Terms
Design, Experimentation, Human Factors

Keywords
Emotion detection, sentiment analysis, learning diaries, visualization

[*] These authors contributed equally.

1. INTRODUCTION
Instructors are constantly looking for ways to understand and address the challenges that their students face during the learning process. Emotional obstacles, in particular, are known to hinder a learner's progress: students learn and perform better when they feel joy, satisfaction and contentment about a particular subject [13]. One approach that instructors use to be aware of their students' emotional welfare is the incorporation of learning diaries which are reflective of a student's journey throughout the duration of a course [16]. Understanding students' personal diaries to uncover their feelings toward the learning experience as a whole (i.e., the instructor, the learning material and topics, and themselves and their performance) can lead to improvements in the quality of the instructor-student relationship and the manner of teaching [2].

Over the past two years, a large number of sentiment analysis (SA) programs have been developed to discover the sentiment content of texts in various genres including news headlines for polarity and emotions [28], movie reviews for polarity [20] and Twitter posts for emotions [11]. However, not much work has been done in applying SA in educational settings, particularly in the analysis of students' learning diaries. Applying SA to the educational field holds many possibilities for improving the communication pathways and learning opportunities between instructors and their students. That is, it is well documented that students' emotions toward the learning experience have an important influence on learning outcomes [9] and that happy learners are generally more motivated to accomplish their set goals throughout the course [13]. Hence, the prompt detection of students' emotional problems or of students that need particular attention is of vital importance. Through the automatic analysis of the sentiments and emotions expressed in students' learning diaries it is possible to promptly identify students that are in need of immediate and personalized feedback. Additionally, using SA on students' text is less invasive than, for instance, personal interviews, which is desirable for instructors as they obtain information on a student's emotions without disturbing his or her learning [25].

SA is framed within the area of natural language processing (NLP) and is broadly defined by Pang and Lee [20] as the computational treatment of opinions, feelings, emotions, and subjectivity in texts. Current work in SA focuses on classifying sentiments based on the polarity/valence (positive, negative, neutral) of text [20]. In this paper, for a richer understanding of the students' texts, we go beyond polarity classification and identify expressions of emotions such as joy, sadness, anxiety, and frustration and how these emotions evolve over the period of time when the diaries are written. Our work is relevant to the education community as an application of SA in the educational context. As Goleman [7] points out in his book, Emotional Intelligence, expert teachers are able to recognize a student's emotional state, allowing them to respond in an appropriate manner that has a positive impact on the learning process. Goleman [7] also adds that students should also be able to recognize and accurately label their emotions and how they may guide their actions.

Hence, in order to provide instructors with a fast method to identify students' emotional states, this paper presents a functional system that takes students' learning diaries as input and gives a graphical visualization of the changes in emotions in the analyzed learning diaries as output. In addition, our system illustrates the polarity scores and the various topics covered in the diaries.

Our system's main contribution relies in providing the instructor with new perspectives for the detection, analysis and prompt addressing of the emotions that the students express. Our system also facilitates swift interventions and the creation of personalized feedback to students who so require, hence improving student motivation and performance [25]. It is not the aim of the system to predict a student's learning process or progress; it simply provides a window into the emotional well-being of students during their time of writing the learning diaries. The instructor is allowed to monitor and respond accordingly when emotions such frustration and anxiety are observed to be extreme or lasting for a too-long period of time. Not all of the emotions might need to be addressed. In addition, students themselves can use the system to assess their learning and motivational progress according to their own needs.

2. BACKGROUND

2.1 Learning Diaries

Learning diaries are containers for writing that are usually recorded over a period of time [16]. They are included in educational settings as a means of facilitating or assessing learning. They may provide valuable insights into what students think and feel during lectures and any problems that they might be having. They are vehicles for reflection for the student, without which might not be possible to do in the classrooms. The diaries usually accompany a program of learning or a research project. Moreover, the diaries can come in many different forms and be used to fulfill different purposes [16]. Thus the nature of learning diaries makes them largely subjective. As Altrabsheh *et al*. [1] explain; subjectivity represents facts and also emotions, feelings, views, and beliefs.

2.2 Sentiment Analysis on Students' Texts

Sentiment Analysis (SA) is a field that works on making sense out of textual material [1], and using it to analyze students' learning diaries can help instructors understand the learning behaviors of students. SA, however, has not been widely applied to the educational sector. A majority of the SA research has been built around user reviews corpora (e.g., movie reviews, product reviews, etc.) [20, 19]. This is because these reviews, similar to learning diaries used in the paper, are subjective and contain information about the user experience with the product or movie [10].

Works on SA with student texts have been applied to various forms of texts, especially those retrieved from e-learning platforms. Santos *et al*. [26], for instance, used SA to analyze emotional reports written by students while they were conducting an activity. Our work differs from theirs in that we go beyond the analysis of emotional valence (how pleasant or unpleasant an emotion is) into the identification of the categories of emotions present in text. In another relevant work, Rodrigues *et al*. [25] extracted emotions from essay texts produced within the classroom and also within an adaptive learning environment that supported dynamic task recommendation. They made use of emotion dictionaries and word-spotting techniques in order to classify the texts into four emotion categories: joy, anger, sadness and fear. Our study differs from the work by Rodrigues *et al*. [25] in that we use texts from more than one student and then analyze and visualize the emotions and their changes over a period of time.

2.3 Feedback in Education and Learning

SA has also been investigated to improve the feedback given to students [1]. It is important to notice that good feedback, among other things, will encourage motivation and self-esteem, which are directly related to the student's emotional state. As Hattie and Timperley [8] explain, it is beneficial to give positive feedback, even if what has to be communicated is negative (e.g., when a student has solved a problem wrongly). Analyzing students' learning diaries can also help in understanding the different issues that students go through, including their lack of understanding of a subject. These learning diaries can in fact become an important source of feedback to the instructor. Through this type of feedback, a student can convey his or her feelings in short expressions or words [1]. Analyzing online students' feedback, Feng *et al*. [5] created patterns to find which words are associated more with emotions, and they also created sentiment adjustment strategies to help students in certain situations like being frustrated after being criticized by a teacher.

Hence, it is reasonable to say that the information on emotions in the learning diaries have the potential to prompt the teachers to tailor their teaching styles in a manner that better matches the learner's requirements.

2.4 Categorizing Emotions

It is beneficial to investigate the kind of emotions students express and experience during the learning process, and how these emotions evolve over a period of time [25]. Lists of primary or "basic" emotions have been put forward prominently in the psychological field by Frijda [6], Ekman [4], and Plutchik [22], among others. The basic emotion categories used in these lists include anger, sadness, joy, love, surprise, happiness, fear and disgust (see [18, 27], for a detailed compilation of primary emotion lists). It is difficult, however, to settle on a category of emotion labels given the gradations and subtleties of the way emotions are expressed in language [29]. Furthermore, in literature, there is no consensus on which basic emotions to use. Thus we decided to concentrate on Plutchik's eight emotions: joy, sadness, fear, anger, anticipation, surprise, disgust and trust, as these adequately fit our purpose of identifying several basic emotions and in addition, they can be used to derive two other emotions that have been found relevant in the learning context: frustration and anxiety [12]. We include frustration and anxiety in particular, as they can impede the progress of the students toward their learning goals [14]. Plutchik's categorization of emotions further provides us with the conceptualization of blending the eight primary emotions to obtain secondary and tertiary emotions, such as frustration and anxiety [22].

3. SYSTEM DESCRIPTION

For the purposes of emotional analysis of learning diaries, we have designed and implemented an automated system that performs several actions. The system accepts students' learning diaries as input and then fragments the diaries by the date of each diary entry. It then extracts the emotions present in the diary entry, their negative and positive attributes, and the topics present. Finally, using the extracted diary entry time elements, the system produces a fine-grained visualization of the emotional information flow in the entire diary (i.e., all the analyzed entries). Figure 1 illustrates the overview of our proposed system.

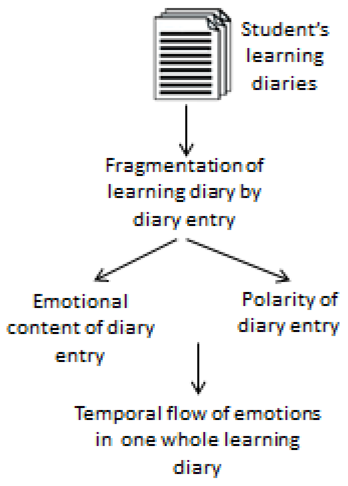
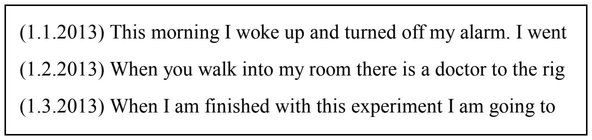

Student's learning diaries
Fragmentation of learning diary by diary entry
Emotional content of diary entry
Polarity of diary entry
Temporal flow of emotions in one whole learning diary

Figure 1. System flow diagram.

3.1 Uploading Students' Learning Diaries

Our system allows a user (e.g., the instructor) to enter a student's name and upload the learning diary belonging to that student for analysis. Upon uploading the diary, the system fragments the diary time-wise. Usually when a student updates a learning diary, the date of the new diary entry is recorded which makes it possible for the system to fragment the whole learning diary into individual diary entries. Figure 2 illustrates an example where one student's learning diary had three diary entries. Figure 2 shows the time stamp of the diary entry along with the textual content belonging to that time stamp. From these diary entries, we are able to create several visualizations (see Section 3.3).



(1.1.2013) This morning I woke up and turned off my alarm. I went

(1.2.2013) When you walk into my room there is a doctor to the rig

(1.3.2013) When I am finished with this experiment I am going to

Figure 2. Student's learning diary, fragmented by entry date.

3.2 Extractions of Emotions

In order for a richer exploration of the emotions expressed that goes beyond the emotion polarity of learning diaries, we extract emotions from the learning diaries by comparing each sentence in a diary against the NRC word-emotion association lexicon [15]. The lexicon has been manually annotated into eight emotion categories according to Plutchik's [22] eight basic emotions: joy, sadness, fear, anger, anticipation, surprise, disgust and trust. The annotations also include scores for whether a word is positive or negative. Each score in the lexicon is simply a Boolean marker, denoting whether the given word belongs to a given emotion category. In our calculations, when a word in a learning diary matches a word in the lexicon, we mark that word with a score of 1 within the matched emotion category, and when the word does not match any word in the lexicon, we mark it with a score of 0. Currently, the lexicon includes emotional annotations for 6,468 unique words. Further description of the lexicon can be found in an article by Mohammad and Turney [15].

In our work, an emotional score *(eScore)* is calculated for each one of Plutchik's eight categories represented in each diary entry as follows:

$$eScore_{(category)} = \frac{eWords_{(category)}}{eWords_{(all)}}$$

where:

- $eWords_{(category)}$ is the number of words in the uploaded learning diary that have nonzero emotional score for the category according to the NRC lexicon.
- $eWords_{(all)}$ is the number of words in the uploaded learning diary that have nonzero emotional score for any category according to the NRC lexicon.

Using the eight categories of emotions, we were also able to calculate frustration and anxiety as follows [23]:

$$Frustration = Average\left(eScore_{(Anger, Surprise, Sadness)}\right)$$
$$Anxiety = Average\left(eScore_{(Anticipation, Fear)}\right)$$

where *Frustration* is given by the average eScore of anger, surprise and sadness; and *Anxiety* by the average eScore of anticipation and fear.

3.3 Data Visualization

Our system visualizes obtained scores with a variety of graphical forms. The visualization shows the following:

3.3.1 *Emotion Distribution*

Emotional scores, calculated over a sequence of diary entries, are visualized on a radar chart that directly resembles Plutchik's wheel of emotions.

This radar chart (Figure 3) has eight independent axes, corresponding to the individual primary emotions. For each axis, we calculate a point of average emotional score over the given diary entries (i.e., for each diary entry, we calculated the *eScore* and then summed up all the *eScores* and divided the resulting value by the number of entries). Then these points become vertices of a filled polygon, thus providing a convenient visualization for the eight primary emotional scores. As frustration and anxiety are mixed emotions [22], they are not included in the radar chart. Those two emotions are visualized separately in a time chart (see Section 3.4).

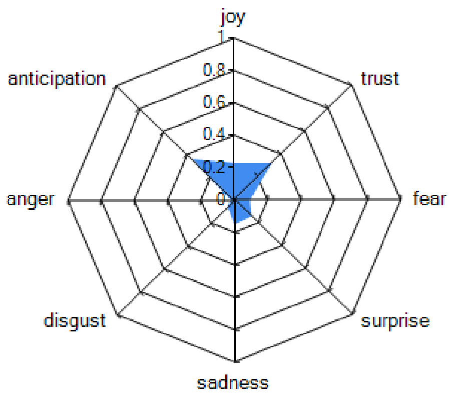


Figure 3. Distribution of emotions within a diary entry.

3.3.2 *Emotion Polarity*

In addition, polarity attribute scores are visualized on a bar graph (see Figure 4). Since "positive" and "negative" annotations are directly present in the NRC lexicon, in Figure 4, we make use of this information. The visualization shows the positive/negative average values for the given range of diary entries.

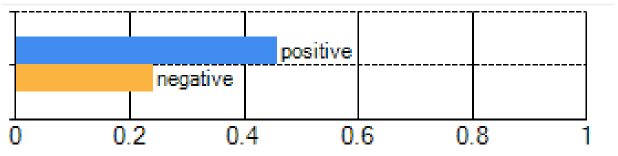


Figure 4. A diary entry's positive and negative sentiment averages.

3.3.3 *Topics*

Most topics found frequently in the uploaded student's diary are displayed as a word cloud, which is convenient for identifying frequent topics conveyed by one particular student (see Figure 5). Before building a word cloud, we apply a stop-word removal procedure and Porter's stemming algorithm [24]. These steps help to focus on the linguistically significant components of text by removing common words that are not topics, such as "without", "soon", "sometime", etc.

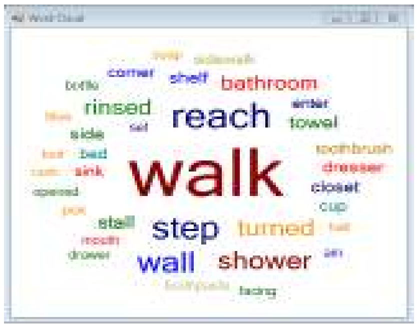


Figure 5. Word cloud view of topics present in the uploaded learning diary.

3.3.4 *Temporal Flow of Emotions*

Our system also allows for the visualization of the temporal flow of emotions in the learning diaries. The temporal flow-charts are divided into two views: The first view (Figure 6a and Figure 6b) displays the flow of the eight primary emotions. The second view (Figure 7) displays the mixed emotions, frustration and anxiety. In each of the views, a user has the choice of visualizing all the emotions or selecting a combination of emotions. Figure 6a shows visualization when all the emotions are selected, and Figure 6b shows an example where one emotion (i.e., fear) is selected.

For the temporal flow-chart, we build average emotional scores for each time entry and thus obtain a calendar-like view of emotional changes in the writings. In the given examples, the entries are given for three different dates; hence, the graphs are built for three time points. The Y-axis values for the flow chart are calculated in the same manner as for the *eScore*, (i.e., number of emotional words for the given category divided by the number of emotional words).

4. PRELIMINARY EVALUATION

4.1 Dataset

We performed a preliminary evaluation of our visualization system with samples of diaries from the Newman *et al.* [17] corpus[1]. The diaries used were written by first year college students, where they described their experience of going to college. There were a total of 18 female participants and 17 male participants, and each participant wrote in their diary three entries in three different occasions in sequential order, for a total of 54 entries written by females and 51 entries written by males. Table 1 shows a description of the entries.

Table 1. Description of diary entries by gender with average word count (Avg) and standard deviation (STD).

Female				***Male***			
Diaries	Entries	Avg	STD	Diaries	Entries	Avg	STD
18	54	459	119	17	51	384	105

[1] We obtained the corpus through personal communication with James W. Pennebaker, a professor of psychology.

Since the diary entries did not have a date time stamp on them, for our preliminary evaluation, each diary entry was assigned a date in the month-year form. Hence, each diary entry had the format entryNumber_monthYear.txt, transforming the sequential order of the diary entries into three-month time stamps. This format made it easier to place the diaries in a time sequence that allowed us to generate the flow of emotion charts, as shown in Figures 7, 8, and 9. We tested all 35 diaries, of which 18 diaries were written by females and the 17 diaries were written by males.

4.2 Results

With our system, it is possible to visualize the variation over time in all the emotions, including frustration and anxiety, as detected in the students' diaries. This allows the identification of students whose anxiety and/or frustration levels are increasing. It is also possible to determine the proportionality between anxiety and frustration that the students are experiencing.

In our preliminary evaluation, eight participants (five female and three male) showed frustration and anxiety levels that were proportional to each other over the three diary entries (i.e., both anxiety and frustration increased and decreased proportionally at the same time). Figure 8 gives an example where such a relationship was observed. In the other 27 participants (13 female and 14 male), anxiety increased while frustration decreased or vice versa. Moreover, within the dataset, we observed that with 30 participants, their three diary entries contained more anxiety than frustration, as indicated by the Y-axis values in Figure 7. No participant's diary showed more frustration than anxiety over the three diary entries, and five participants' diaries intertwined (i.e., they showed more anxiety than frustration in one diary entry and then more anxiety than frustration in another). Table 2 shows a summary of the results.

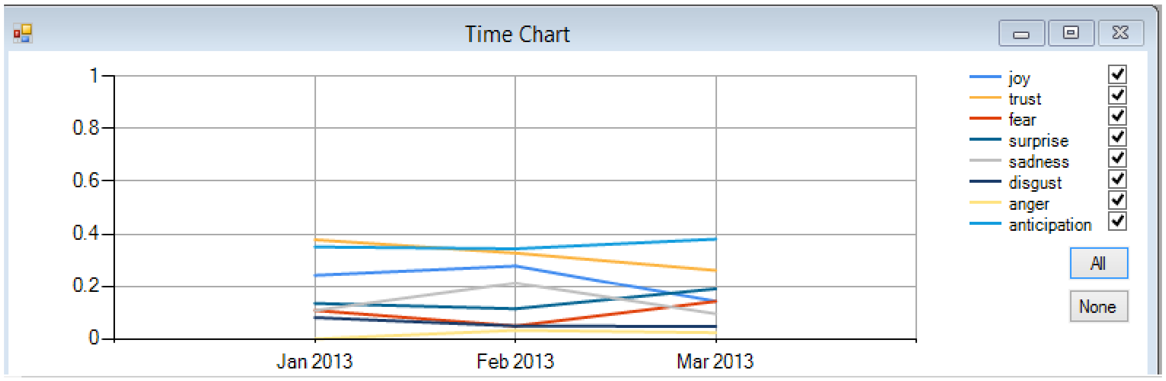


Figure 6a. View of the flow of all the eight Plutchik [22] emotions within a learning diary.

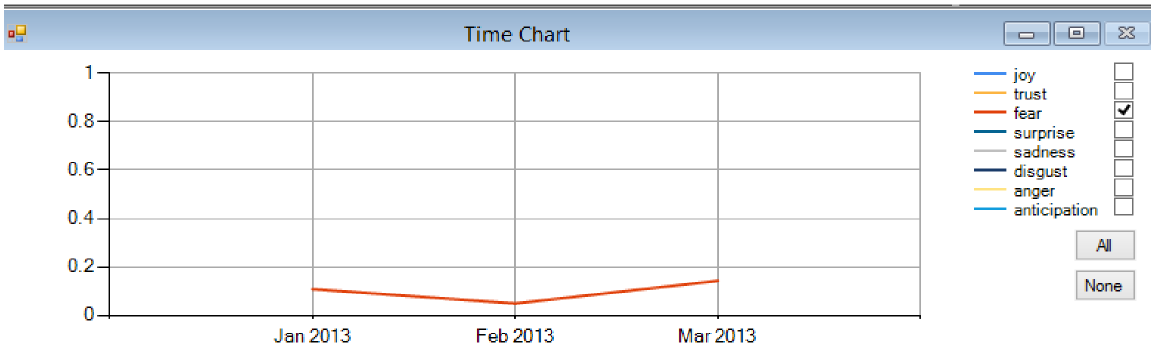


Figure 6b. Fear flow within the learning diary.

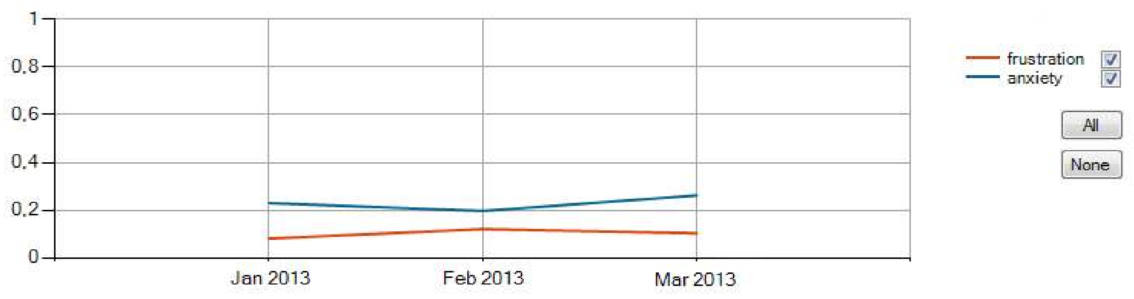


Figure 7. Frustration and anxiety emotional flow within a learning diary.

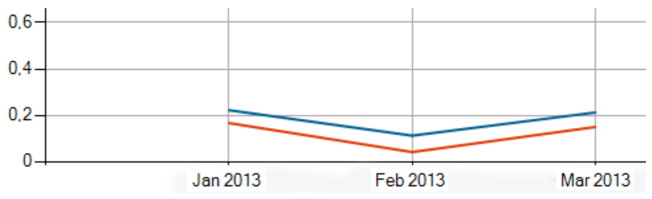


Figure 8. Example where frustration and anxiety have a parallel relationship.

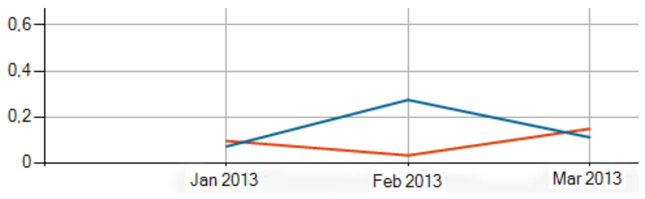


Figure 9. Example where Frustration and Anxiety intertwine.

Table 2. Distribution of anxiety and frustration within the students' diaries (all entries included).

	Female	*Male*	*Total*
More anxiety in each of the three diary entries	17	13	**30**
More frustration in each of the three diary entries	0	0	**0**
Intertwined	1	4	**5**
Total	**18**	**17**	**35**

Figure 9 shows an example where the frustration and anxiety levels are intertwined. Interestingly, only one participant registered anxiety levels of 0.4. Otherwise, all the participants had frustration and anxiety levels less than 0.4, with frustration in particular, registering levels less than 0.2 within 34 participants.

Thus, within the dataset, we see that the majority of participants expressed relatively low frustration whereas most of the participants expressed higher anxiety levels. This could be interpreted from the context within which the diaries were written: the students were expressing their anxiety toward their new college life experience. An interesting direction to explore is the selection of an empirical threshold, whereby if frustration and anxiety levels go beyond it, a signal is given to the instructor.

4.3 Implications

Our visualization system can provide important insights into the students' pedagogical well-being, which is a vital part of the learning experience since according to the "Explaining Student Performance" report (2005) by the European Commission, data from PISA (Programme for International Student Assessment) suggests that students who have higher levels of performance in their scores are less anxious about the learning process. Also, the report showed that there is a positive correlation between interest and enjoyment of a subject and the students' PISA achievements[2].

By observing the flow of emotions within a diary, an instructor is given the opportunity to timely address any issues or concerns that might be causing any of the negative emotions such as frustration. It is important to notice that an instructor's feedback, among other things, will encourage motivation and self-esteem, which are directly related to the student's emotional state.

By taking the information on the student's emotional state into account, instructors can have a holistic picture of the students' emotional progress. Our system can also serve as a self-evaluation platform in which the students can assess their emotions and motivational progress. Hence, our visualization system can positively contribute to enhancing the traditional educational setting by providing a means of surveying the student's well-being and, at the same time, helping instructors to personalize their feedback. This will result in an overall improved learning experience.

From the students' perspective, we are aware that allowing for student self-evaluation might prompt them to manipulate the content of their diaries; however, with the system, there are no wrong or right emotions. The emotional flow of a student is not part of that student's performance assessment. With that emphasized, we believe that the students will use the system as a reflection tool and will not try to manipulate the content of their diaries as to improve their grades.

5. CONCLUSION AND FUTURE WORK

In this paper we have explored the automatic analysis and tracking of emotions within student' learning diaries. The developed system presented here aimed to function as an aiding system for improving instructors' teaching methods and feedback and serving as a reflection medium for students.

[2] http://ec.europa.eu/education/more-information/doc/basic_en.pdf

The preliminary evaluation showed that the system successfully presented information in an easy-to-understand manner and that the emotional flow of the students during the learning experience, as expressed in their learning diaries, can be meaningfully extracted.

Future work involves studying student diaries that have longer time stamps and analyzing their long-term implications. We also plan to continue the validation of our emotion detection system by comparing the results and performance with other systems on the same learning diaries dataset.

Future versions of the system can be incorporated in e-learning environments or learning management systems (LMSs), where text-based documents produced by the students can be automatically analyzed and the results combined with student profile information. The combination can be used to customize teaching material for students. Also, we plan to conduct a deeper linguistic analysis to better understand the expressed emotions. As the current version of the system makes use of a keyword based approach for detecting emotions, we plan to extend the capabilities of the system by incorporating approaches such as phrase- and sentence-level emotion analysis and common-sense analysis for broader emotion detection. Advantageously, using the keyword approach allows the system to handle the use of slang words and misspellings and even handle keyword sets of different languages.

Furthermore we plan to include event analysis within the next version of the system so as to better inform an instructor of any events such as sitting for an exam, receiving bad news or passing a course, that might have led to the observed emotions.

6. REFERENCES

[1] Altrabsheh, N., Gaber, M. M., and Cocea, M. 2013. SA-E: sentiment analysis for education. In *5th KES International Conference on Intelligent Decision Technologies*, (Sesimbra, Portugal, June 26-28 2013).

[2] Bergström, P. 2010. Process-based assessment for professional learning in higher education: Perspectives on the student-teacher relationship. *International Review of Research in Open and Distance Learning.* 11, 2.

[3] D'Mello, S. K., Craig, S. D., Witherspoon, A., McDaniel, B., and Graesser, A. 2008. Automatic detection of learner's affect from conversation clues. *User Modeling and User-Adapted Interaction.* 18, 1-2, 45-80.

[4] Ekman, P. 1992. An argument for basic emotions. *Cognition and Emotion,* 6, 3, 169-200.

[5] Feng, T., Zheng, Q., Zhao, R., Chen, T., and Jia, X. 2009. Can e-Learner's emotion be recognized from interactive Chinese texts?. *IEEE 13th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2009,* 546-551.

[6] Frijda, N. H. 1986. Emotional behavior. Chapter 2, In *The Emotions*. Studies in Emotion and Social Interaction, Cambridge University Press.

[7] Goleman, D. 1995. *Emotional Intelligence* (The 10th anniversary edition), Bantam Dell, NY.

[8] Hattie, J., and Timperley, H. 2007. The power of feedback. *Review of Educational Research*, 77, 1, 81–112.

[9] Järvenoja , H., and Järvelä, S. 2005. How students describe the sources of their emotional and motivational experiences during the learning process: A qualitative approach. *Learning and Instruction*, 15, 5, 465-480.

[10] Kim, S. M., and Calvo, R. A. 2010. Sentiment analysis in student experiences of learning. In *Proceedings of the 3rd International Conference on Educational Data Mining,* R. S. J. D. Baker, A. Merceron, P. I. Pavlik Jr. Eds. 110-120.

[11] Kim, S., Bak, J., and Oh, A. 2012. Do you feel what I feel? Social aspects of emotions in twitter conversations. In *Proceedings of the AAAI International Conference on Weblogs and Social Media.*

[12] Kort , B., Reilly, R. and Picard, R. 2001. An affective model of interplay between emotions and learning: Reengineering educational pedagogy-building a learning companion. In *Proceedings of the IEEE International Conference on Advanced Learning Technology: Issues, Achievements and Challenges, Madison, Wisconsin: IEEE Computer Society,* 43-48.

[13] Lawson, C. 2005. The connections between emotions and learning. *Center for Development and Learning.* Available at http://www. cdl. org/resourcelibrary/articles/connect_emotions. php. [Retrieved on 3 June 2013].

[14] McQuiggan, S. W., Lee, S., and Lester, J. C. 2007. Early prediction of student frustration. *Affective Computing and Intelligent Interaction*. Springer Berlin Heidelberg, 698-709.

[15] Mohammad, S. M., and Turney, P. D. 2012. Crowdsourcing a word-emotion association lexicon. *Computation Intelligence*, DOI: 10.1111/j.1467-8640.2012.00460.x

[16] Moon, J. 2003. Learning journals and logs, reflective diaries. *Centre for Teaching and Learning Good Practice in Teaching and Learning*. University of Exeter.

[17] Newman, M. L., Groom, C. J., Handelman, L. D., and Pennebaker, J. W. 2008. Gender differences in language use: An analysis of 14,000 text samples. *Discourse Processes*, 45, 211-236.

[18] Ortony, A., Clore, G. L., and Collins, A. 1994. The structure of the theory. Chapter 2, in *The Cognitive Structure of Emotions*, Cambridge University Press, 15-33.

[19] Pang, B., and Lee, L. 2004. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics* (ACL '04). Association for Computational Linguistics, Stroudsburg, PA.

[20] Pang, B., and Lee, L. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval,* 2, 1-2, 1-135.

[21] Pérez-Marín, D., Alfonseca, E., and Rodríguez, P. 2006. On the dynamic adaptation of computer assisted assessment of free-text answers. I*n Proceedings of the Adaptive Hypermedia Conference, LNCS,* 4018, Springer-Verlag.

[22] Plutchik, R. 1980. A general psychoevolutionary theory of emotion. *Emotion: Theory, Research, and Experience*, 1, 3, 3-33.

[23] Plutchik, R. 1991. *The Emotions* (revised ed.), University Press of America Inc, Lanham, MD.

[24] Porter, M. F. 1980. An algorithm for suffix stripping. *Program: Electronic Library and Information Systems*, 14, 3, 130-137.

[25] Rodriguez, P., Ortigosa, A., and Carro, R. M. 2012. Extracting emotions from texts in e-learning environments. *Sixth International Conference on Complex, Intelligent, and Software Intensive Systems,* 887-892.

[26] Santos, O. C., Salmeron-Majadas, S., and Boticario, J. G. 2013. Emotions detection from math exercises by combining several data sources. *Artificial Intelligence in Education Lecture Notes in Computer Science,* 7926, 742-745.

[27] Shaver, P., Schwartz, J., Kirson, D., and O'Connor, C. 2001. Emotion knowledge: Further exploration of a prototype approach. *Emotions in Social Psychology: Key Readings*, G. W. Parrott Ed., Taylor & Francis, USA, 25-56.

[28] Strapparava, C., and Mihalcea, R. 2007. SemEval-2007 task 14: affective text. In *Proceedings of SemEval-2007,* Prague, 70-74.

[29] Wyner, S., Shaw, E., Kim, T., Li, J., and Kim, J. 2008. Sentiment analysis of a student Q&A board for computer science, *The 9th KOCSEA Technical Symposium,* Vienna, VA.

Building Collaborative Quizzes

Bruno Sampaio
Dept. Informática, FCT,
Universidade Nova de Lisboa
Quinta da Torre, Caparica,
Portugal
b.sampaio@campus.fct.unl.pt

Carmen Morgado
CITI/Dept. Informática, FCT,
Universidade Nova de Lisboa
Quinta da Torre, Caparica,
Portugal
cpm@fct.unl.pt

Fernanda Barbosa
CITI/Dept. Informática, FCT,
Universidade Nova de Lisboa
Quinta da Torre, Caparica,
Portugal
fb@fct.unl.pt

ABSTRACT

Building appealing online quizzes is not a simple task, although there are many tools to build quiz tests, usually they are not a very motivating activity to students. Typically students look to these quizzes as an obligation (assessment) and not as a tool that can help them in their learning process.

Epik (Edutainment by Playing and Interacting with Knowledge) is an online application, which allows the easy development of quizzes games, through a graphical environment. Epik quiz games may be distributed through existing LMS (Learning Management Systems) or directly on the Epik framework. Epik quizzes may be individual or collaborative, they consist of a sequence (collection) of interactive scenarios. Within each scenario there is a set of questions of type: multiple choices, true or false or matching. Didactic contents such as: texts, slides, images or videos, can also be placed on the scenarios. Because of its integration with an LMS, learning materials already created on the LMS can be easily imported and reused.

In order to increase the student knowledge acquisition, each question can have a set of "helps" (didactic contents; hints; and incorrect answers removal (50/50)). In order to promote the cooperation between students, Epik has the collaborative quizzes. These type of quizzes has mechanisms to encourage collaboration among team players. Through the use of "helps" collaboration between team members is encouraged. Team players who have already correctly answered a given question, can help others. The game developer can easily configure the scores bonus and penalties values of each question, as well as the question "helps".

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education—*collaborative learning*

General Terms

Human Factors

Keywords

quizzes; educational games; collaborative learning environments

1. INTRODUCTION

One of the oldest and most used learning activities are the quizzes. They can be used in class or as homework in almost any educational area, namely in computer science courses. Covering almost any education area and level, being used in teaching first year students, as well as all levels of school education. Online quizzes can give students an immediate feedback about their learning progress. Typically, they are individual activities, although when used as a group activity tends to increase participation as well as the ability to work as a team [20]. Group quizzes can provide a rich learning experience, where students learn by doing and through communication and interaction with others.

Collaborative learning, where a group of students works together to solve a problem, is usually widely used in undergraduate and graduate computer science courses. This type of learning activity increases the development of some of student skills such as communication, cooperation and coordination, all of them relevant in the learning process and individuals formation [22].

Nowadays there are many frameworks [2] [10] for the development of online quizzes, some of them with the ability to integrate some competition mechanisms, but almost none of them have mechanisms that promote collaboration among the quiz participants. Another problem regarding the existing frameworks is that, although many of them are able to integrate multimedia contents on the quizzes, they do not have integrated tools to easily manage and reuse these didactic materials.

Most of existing LMSs (Learning Management System), as Moodle [4], have functionalities to create and manage quizzes. But quizzes created by these systems are not very interactive and are not fully integrated into the learning environment. Most of them seem a little with the tests used in student evaluations.

On the other hand, the possibility given by the games of joining competition with team work (collaboration) offers an entertaining and stimulating way of introducing the collaboration in the learning process [8] [17]. The intrinsic characteristics provided by games (competition and collaboration) can make of them a powerful tool on the learning process. Particularly the use of games on the learning process can have two key aspects: to motivate, the desire to explore the motivational power of games in order to "make learn fun"; for practice, the belief that "learning through doing", with mechanisms (like for example the simulations), offers a simple and powerful tool. In the last decade, many studies were conducted to investigate the effectiveness of the use of educational computer games in teaching a variety of different topics, such as for example mathematic [14], software engineering [12], computer science [18]. Previous studies have reported that educational computer games can enhance students' interest and motivation on learning [11] [13]. They argue that educational computer games have a great potential on helping students to improve their learning performance as well as their motivation [15]. Although there are many educational games, most of them are specific for a given theme/topic.

However, the games are still not widely used as a teaching activity. Some difficulties in a wider use of games in the learning process are related with their development and also with the production and management of didactic contents to use in the games. Another problem is related to the difficulty of inserting games as a learning activity in a working environment in the classroom or as a homework. So the possibility to integrate games as a learning activity on existing LMSs is a requisite. The possibility of having games as a collaborative activity, in order to incorporate the notion of working groups (a typical organizational structure that exists in class), is also a requisite. Nowadays there are not many generic platforms [21] for developing educational games for any given educational area. Most of the existing game development platforms require a prior knowledge of programming for the construction of games, apart from which, generally, these platforms are not specialized and targeted for education/learning.

Epik (Edutainment by Playing and Interacting with Knowledge) [3] is a framework that allows the easy development of different educational games types, namely individual and collaborative quiz games. The development of an Epik game, is done easily resorting to using a graphical environment and without requiring any prior programming knowledge. Quizzes developed with Epik can easily be distributed to students as an external learning activity through the LMS used at classes. The teachers, when building the game, can also reuse the didactic contents already existing in the LMS Epik also provides some collaborative mechanisms that can be easily incorporated into a game in a fun and interesting way.

In this paper, we will describe the Epik framework and how to build a quiz game with it. We also present a discussion of different platforms to develop online quizzes, and a brief analysis of some of the Epik key features that distinguish it of others.

2. ONLINE FRAMEWORKS FOR BUILDING QUIZZES

Nowadays there are several online frameworks to build quizzes, just as there are many online quizzes covering many educational areas and topics ranging from the teaching of languages to mathematics or physics. We analyze some online frameworks we think are the most used and relevant to build and distribute quizzes:

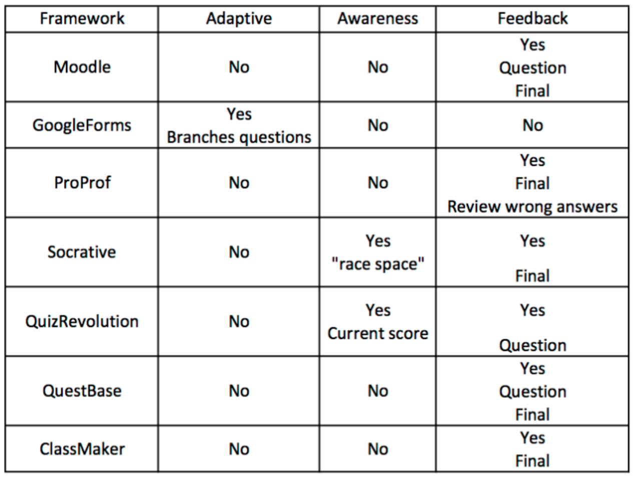
- Google Forms: can be used to develop short multiple choice quizzes, that may be post, for instance on a classroom blog. The students answers are present in an easy-to-grade spreadsheet.
- ProProf [5]: offers an easy way to make a quiz for education or fun. The quizzes questions may be true or false, fill in the blank, short and long answers, essay, check-boxes, yes/no, multiple choice questions. It is possible to add videos, pictures, articles, PowerPoint and documents (word or pdf) to the questions, and record student' names and their score, answers and time spent, this information can be used to students assessment. The quizzes may be distributed on a website, blog or a class management system (developed by the same software company).
- Socrative [9]: is a smart student response system that empowers teachers to engage their classrooms through a series of educational exercises and games via smartphones, laptops, and tablets. Questions can be multiple choice, short answer, or a combination of both. It is possible to create teams of students ("rooms"). In a given room, students can compete among them using a very simple rocket race game. In this game each student controls a rocket, where its speed is defined by the students' answers.
- Quiz Revolution [7]: offers an easy way to make a quiz about anything in minutes. Questions may be multiple choice, write-in answers and timed questions. It is possible to add images, videos and different scores associated to each question responses.
- QuestBase [6]: is a cross-platform application that provides a set of functionalities to create and manage assessments, tests, quizzes and exams, both online and printed. Quizzes may be building without any technical skills. It can manage several question types, and it is possible to randomize questions and answers, assign different scores and feedback messages, and add custom instructions and hints to questions. Moreover, the questions may be imported from files (e.g. Access, Excel)
- ClassMaker [1]: offers an easy way to make quizzes, with multiple choice, short answer and essay questions. It is also possible to randomize questions and answers, and assign time limit, images and documents to questions.

These frameworks are online web applications, which have forms to edit question. Several of them (ProProf, QuizRevolution, QuestBase, ClassMaker) allow the addition of resources to questions, in order to reuse didactic contents. Although, given the way how the questions are constructed, the quizzes are not very appealing to be used in learning environments.

The quizzes built with these frameworks may be distributed to students in websites or class blogs, and can also be used to students assessment, but they are not totally integrable into existent LMSs.

Usually these quizzes allow an immediate feedback, but are not adaptable to the student's knowledge and almost none of them have a notion of awareness (Table 1), features that are very important to motivate student' participation. Note that these are two typical features of educational games, that are fundamental to the learning process

Table 1: Environment features of quizzes



Framework	Adaptive	Awareness	Feedback
Moodle	No	No	Yes Question Final
GoogleForms	Yes Branches questions	No	No
ProProf	No	No	Yes Final Review wrong answers
Socrative	No	Yes "race space"	Yes Final
QuizRevolution	No	Yes Current score	Yes Question
QuestBase	No	No	Yes Question Final
ClassMaker	No	No	Yes Final

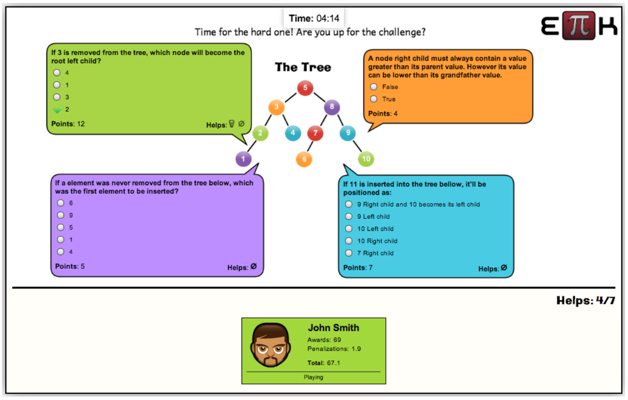
Table 2: Collaboration features of quizzes

Framework	Collaboration	Competition
Moodle	No	Yes Question score
GoogleForms	No	No
ProProf	No	Yes Question score; Final score
Socrative	No	Yes ”race space”
QuizRevolution	No	Yes Question score
QuestBase	No	Yes Question score
ClassMaker	No	Yes Question score

[16]. The idea that the activity fits the student’s knowledge, makes the activity more appealing and more dynamic allowing multiple execution paths. The student perception of the environment on activity motivates him to participate, thus increasing his/her knowledge acquisition.

According to the collaborative features of these online quizzes, neither of them have mechanisms to promote collaboration, although most of them have some competition mechanisms, basically based only on questions and final scores, as illustrated in Table 2. Only Socrative framework allows a team of students to perform together an activity, that is coordinated by a user with teacher’ role.

In summary, quizzes created by these frameworks are very similar to traditional exam quizzes, however they have the advantage of online distribution, immediate feedback, and some may incorporate educational resources. Only QuestBase quizzes allow the association of helps to questions which make quizzes more a learning activity and less only an evaluation activity. But although all of these frameworks are very useful to build quizzes quickly and easily, manage online quizzes and student outcomes in quizzes, they all lack in some important features to enable the full use of quizzes as a learning activity.



Time: 04:14
Time for the hard one! Are you up for the challenge?

If 3 is removed from the tree, which node will become the root left child?
- 4
- 1
- 3
- ✓ 2

Points: 12 Helps:

A node right child must always contain a value greater than its parent value. However its value can be lower than its grandfather value.
- False
- True

Points: 4

If a element was never removed from the tree below, which was the first element to be inserted?
- 6
- 9
- 5
- 1
- 4

Points: 5 Helps: Ø

If 11 is inserted into the tree bellow, it'll be positioned as:
- 9 Right child and 10 becomes its left child
- 9 Left child
- 10 Left child
- 10 Right child
- 7 Right child

Points: 7 Helps: Ø

Helps: 4/7

John Smith
Awards: 69
Penalizations: 1.9
Total: 67.1
Playing

Figure 1: An example of a layout of an Epik quiz scenario.

3. EPIK FRAMEWORK

Epik is a web-based framework dedicated to management of didactic contents and development of educational games based on quizzes [19]. In order to progress and complete the game players must correctly answer a set of questions sequentially presented to them. At the end a score will be assigned to the players or teams based on their performance.

3.1 Key features

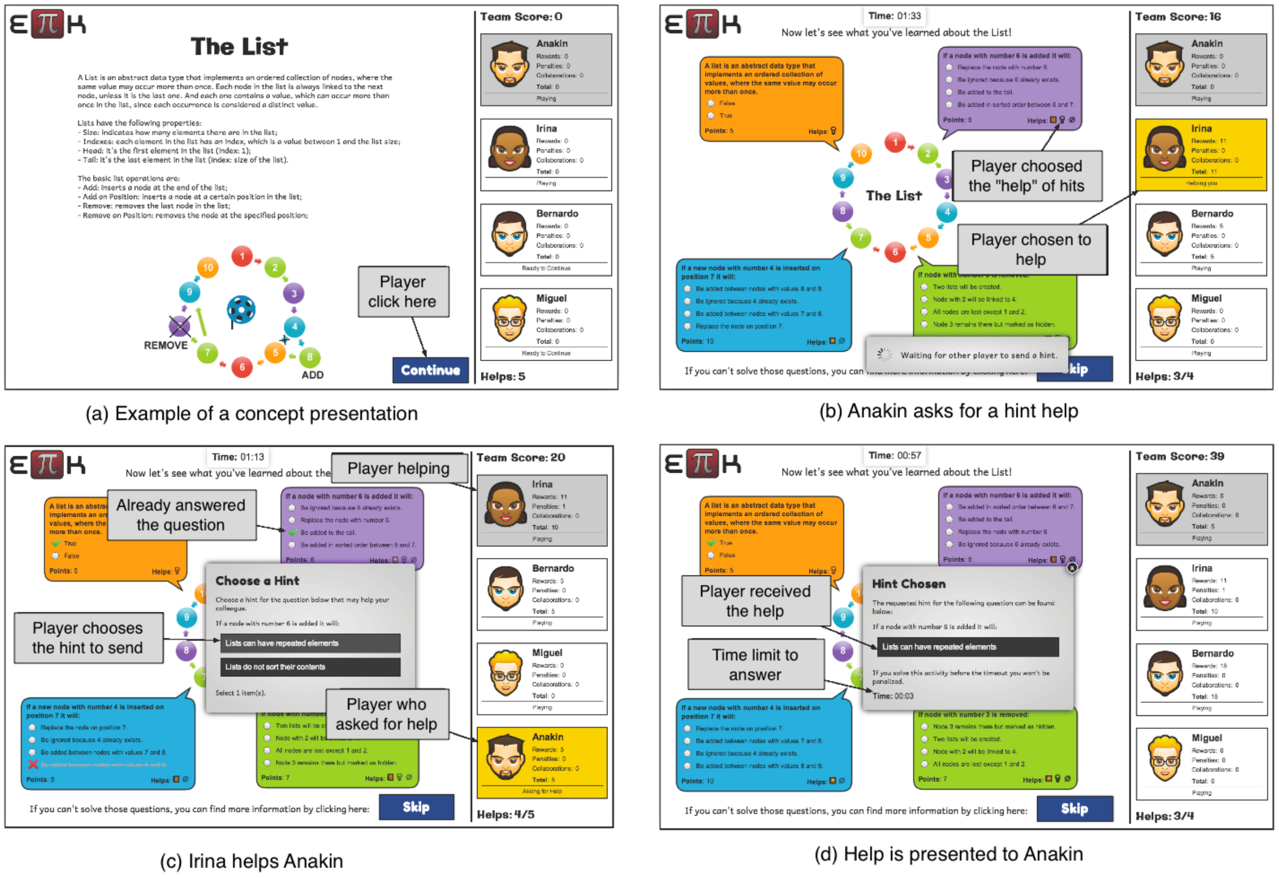
The main Epik’s key features are related to the creation of games based on sets of quizzes, and integration of collaboration mechanisms between team players.

Epik support multiple question types such as multiple choice, true or false and matching. To each question can be given a score and can be associated to set of ”helps”. The ”help” can be a didactic content (text, video our image) or a hint given by other team member (when working with collaborative quizzes), however each time a ”help” is consulted by the player a penalty is given to the score answer. Although this penalty is less than when the wrong answer is given. The ”helps” (contents and hints) and scores associated to each question are defined by the teacher at creation time.

Epik quiz is structured as a sequence of scenarios at each scenario we can have multimedia contents and several questions. To be able to proceed to the next scenario the user (player) must complete the scenario activities. This, associated with the fact of having an immediate feedback of the evolution of each participant (given by a score), a time limit to complete the quiz and a ranking of the best scores, gives a sense of game to the quizzes and not so much of a typical assessment activity. Its graphical user interface offers users a more enjoyable experience when compared with the more usual quizzes layout.

By being organized in scenarios (Figure 1) it is easier for teachers to arrange the questions by topics and also define different learning paths based on user behavior at each scenario. This allows teachers to adapt quizzes to the knowledge level of each student.

Another key feature of Epik quizzes is the possibility of integration with existing LMS that support the IMS LTI standard. For example on Moodle this can be done by creating an activity as an external tool where the URL of the quiz game as to be given. When the quiz activity is started from



(a) Example of a concept presentation

(b) Anakin asks for a hint help

(c) Irina helps Anakin

(d) Help is presented to Anakin

Figure 2: An example of collaboration sequence.

the LMS, information regarding the course and users is sent to Epik. This information is used to find a session for the player and the scores and other information regarding the session can be stored within the context of the LMS course. Furthermore, at the end of the game information concerning each players' performance can be sent to the LMS and be used as assessment. The integration with an existing LMS enables not only the quiz game distribution but also, even more important, the reuse of the didactic contents stored within the LMS environment, and thus simplifying the game development process.

But what we considered to be the more distinctive feature of Epik quizzes is the possibility to integrate collaborative mechanisms that enable the interaction between users of the same team. Epik quizzes include various forms of perception of the environment that contributes to increased collaboration and competition between players. In the context of a quiz game, collaboration is promoted primarily through the use of "helps". As already mentioned the game is organized as a sequence of scenarios, in collaborative mode players are organized as teams and the progress on the game is only possible when all members concluded a scenario (or timeout). To achieve this, team members that have difficulties on some of the activities of a scenario, can ask for help from other team members. The "helps" may be in the form of pre-defined hints[1] that can be given by team members that have successfully completed the associated activity. When the hint is received the player has a time limit to answer, if he answer correctly the player that helped receive a bonus. That is what we call the collaboration points that will contribute to the total score of the player and of the team. An example of a collaboration sequence is illustrated in Figure 2. In the example (Figure 2(b)), a player (Anakin) asks for a hint about the upper right question. The system selects a player (Irina) that already correctly answered the question, to help him. The player (Irina) that receives the "help" request choose and sends the hint (Figure 2(c)). When the hint is received the player who requested the help (Anakin) have a time limit to answer the question (Figure 2(d)), if he answer correctly the player who helped him (Irina) also receive a score bonus.

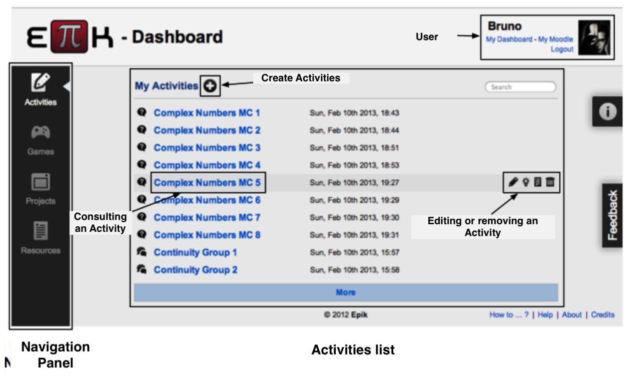
3.2 Developer interface

Epik is an online graphical framework to build quiz-based games. First the developer has to create a project where the game contents and characteristics are defined.

The Dashboard area (Figure 3) is where developers can create, edit and remove projects, activities, resources and manage games already created.

The Development environment (Figure 4) is where the game scenarios are edited and configured. In this area the developer can define the scenario layout, scores and helps associated to the activities, bonus points and also the scenarios sequence (game flow). This environment is divided

[1]Hints are pre-defined by teachers when building the game.



Complex Numbers MC 1	Sun, Feb 10th 2013, 18:43
Complex Numbers MC 2	Sun, Feb 10th 2013, 18:44
Complex Numbers MC 3	Sun, Feb 10th 2013, 18:51
Complex Numbers MC 4	Sun, Feb 10th 2013, 18:53
Complex Numbers MC 5	Sun, Feb 10th 2013, 19:27
Complex Numbers MC 6	Sun, Feb 10th 2013, 19:29
Complex Numbers MC 7	Sun, Feb 10th 2013, 19:30
Complex Numbers MC 8	Sun, Feb 10th 2013, 19:31
Continuity Group 1	Sun, Feb 10th 2013, 15:57
Continuity Group 2	Sun, Feb 10th 2013, 15:58

Figure 3: Epik Dashboard screen.

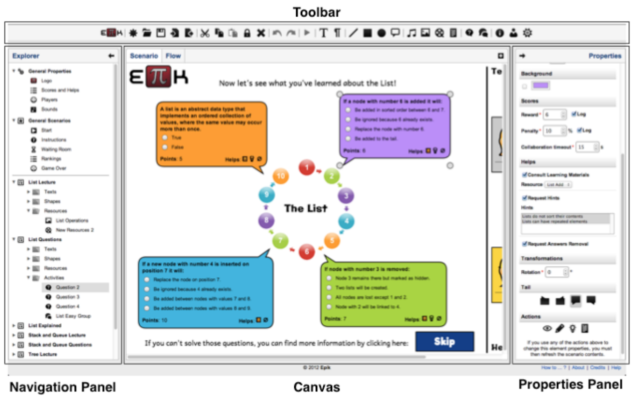

Toolbar
Explorer
Scenario
Flow
Properties
Now let's see what you've learned about the List!
The List
If you can't solve those questions, you can find more information by clicking here:
Skip
Navigation Panel
Canvas
Properties Panel

Figure 4: Epik Development environment screen.

into the following sections:

- Toolbar (top): where a set of icons to access tools to create, access, edit and remove activities, resources and other scenario components, can be found;
- Navigation Panel (left): enables the navigation between the game elements. These elements are common and body scenarios and general game properties. For each body scenarios its contents, organized in folders, are listed. When a user select any item in this panel, this element is presented in the Design Area and all its configurable properties are presented in the Properties Panel;
- Canvas (middle): is the design area where the layout of the scenario is built;
- Properties Panel (right): in this area the configurable fields, of currently selected element in the Navigation Panel, can be accessed and edited.

The development of a Epik game goes through several phases: the creation/selection of activities and resources; create a design and parameterization of the general properties; creation of scenarios and their contents; and, finally, generating the game. Note that these phases do not have to be performed in this order and some of them can be performed simultaneously.

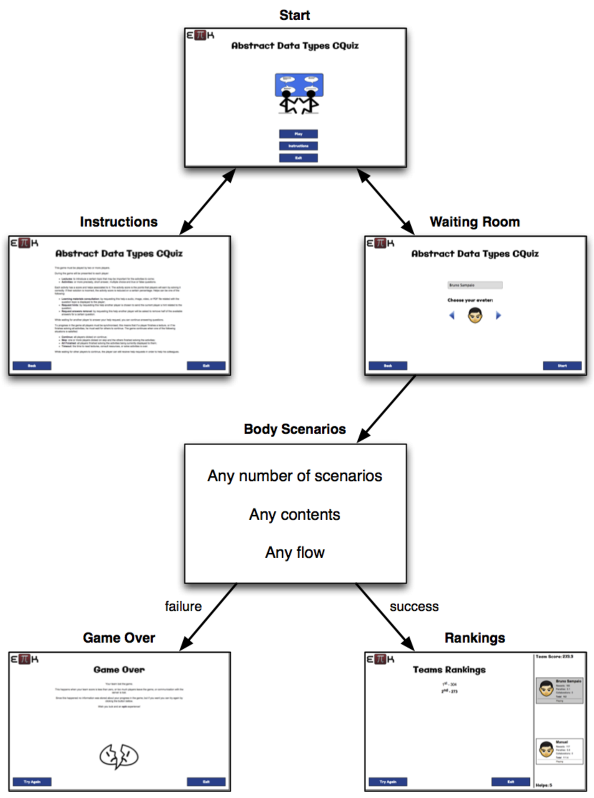


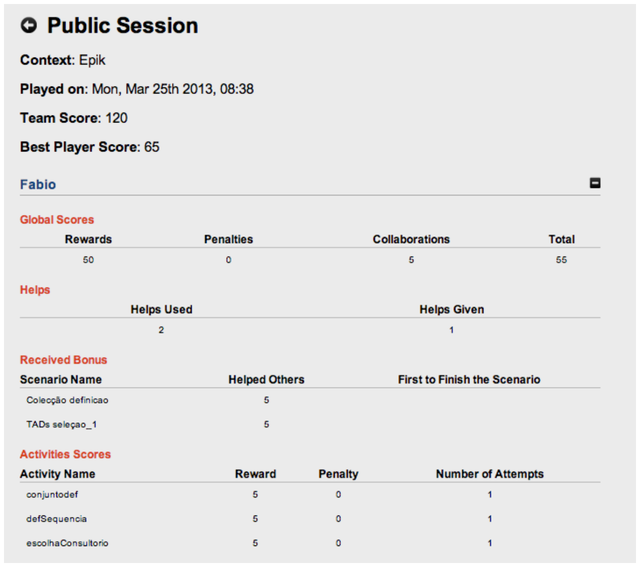
Figure 5: Epik scenarios, game flow.

4. EPIK QUIZ GAME STRUCTURE

Every Epik quiz game has a sequence of scenarios that follows a common flow always starting on the ”Start” scenario, as is illustrated in Figure 5.

At this scenario the player must fill the name and choose an icon (that will the player avatar throughout the game). After that, the player must select a button in order to proceed the game flow. Note that, in case of a collaborative quiz (multiplayer mode), before starting, the players go to a waiting room until the team is completed (number of players defined by the game developer). In both modes, multiplayer or singleplayer, the player can go the ”Instruction” scenario that offers a set of information concerning the actual game. The game can end in one of two scenarios the ”Game Over” scenario and the ”Ranking”/score scenario. At the middle we have the ”Body” scenarios, these are created and configured by the game developers (at the Design Environment) as well as the sequence that should be followed between them (game flow). The ”Body” scenarios can be activities or concept scenarios. On the first ones is where the activities (quizzes) are created and configured but there we can also have geometric forms and resources like text, images and videos usually used to present the activities. The second ones can contain only text, geometric forms and resources and are mainly used to present the concepts concerning the next activities.

After creating and defining all the game structure scenarios and flow, a game can be generated and can be played within the Epik framework. During a game session all the information and actions of the players, including the collaboration between them, will be managed by Epik. At the



Public Session

Context: Epik

Played on: Mon, Mar 25th 2013, 08:38

Team Score: 120

Best Player Score: 65

Fabio

Global Scores

Rewards	Penalties	Collaborations	Total
50	0	5	55

Helps

Helps Used	Helps Given
2	1

Received Bonus

Scenario Name	Helped Others	First to Finish the Scenario
Colecção definicao	5	
TADs seleçao_1	5	

Activities Scores

Activity Name	Reward	Penalty	Number of Attempts
conjuntodef	5	0	1
defSequencia	5	0	1
escolhaConsultorio	5	0	1

Figure 6: A recording session screen.

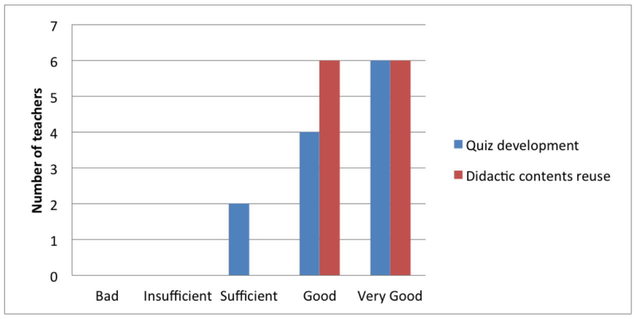


Figure 7: Evaluation of Epik features

end of a session, if the game was completed, the information generated about the performance of the players is stored and will be available for consultation, by the teachers, in the form of a recording session (Figure 6).

5. EVALUATION OF EPIK QUIZZES

The features of Epik framework, which distinguishes it from others, are its graphical environment for quizzes development and its total integration with the LMS (that support the IMS LTI standard), namely Moodle. The Epik quizzes may reuse didactic contents from LMSs, and may be distributed through the Epik framework or through an LMS as a learning activity.

The Epik evaluation was made by a group of 12 teachers of different education areas and levels without any previous experience with this framework. As can be seen in Figure 7, the Epik features were considered as "good" or "very good" by the majority of the inquired teachers. In this evaluation most teachers considered the collaborative Epik quiz features as "good" or "very good", as illustrated in Figure 8.

The environment features of Epik quizzes were evaluated by a group of 37 students, most of them of computer science courses. In this evaluation the students considered these features as "good" or "very good", as can be seen in Figure 9. Moreover, in general, the quizzes were characterized with

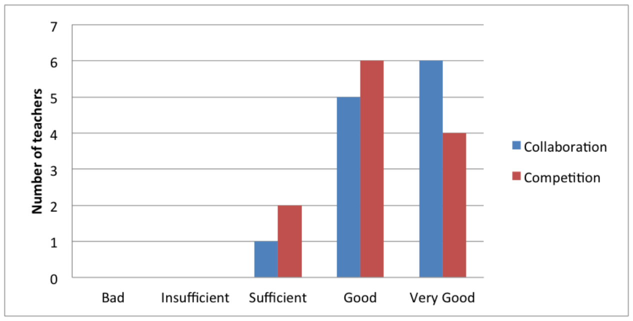


Figure 8: Evaluation of Epik quiz collaborative features

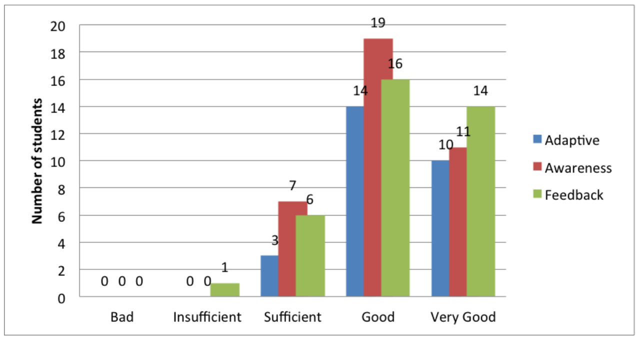


Figure 9: Evaluation of Epik quiz environment features

positive words for most students (see Figure 10).

In order to compare Epik framework with the other frameworks described in Section 2, we present Table 3 which describes the Epik features. Note that: Epik quizzes are easy to integrate and distributed through existing LMS; have collaboration and competition features; provides a pleasant environment, that offers awareness features and is adaptable to student' knowledge.

6. CONCLUSIONS

Quizzes are usually used in computer science courses. Additionally, the actual student generations are fascinated by the gaming world, because of the competition and also because of the natural embedding teamwork, which can provide collaborative learning in an entertaining way. So, Epik collaborative quizzes, where a group of students works to-

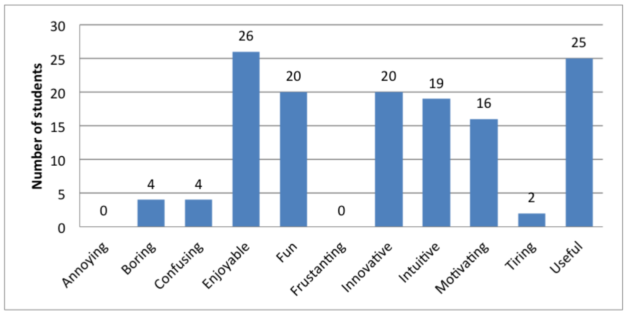


Figure 10: Characterization of Epik quizzes

Table 3: Features of Epik quizzes

General Features	
Quizzes Distribution	Learning activities on LMS; Epik framework;
Student Assessment	Yes

Environment Features	

Feedback	Yes
	Question feedback;
	Final feedback;

Adaptive	Yes questions organized into scenarios; game paths;
Awareness	Yes player score, bonus, penalties; teamplayer score;

Collaborative Features	
Collaboration	Yes question helps (hints and 50/50)

Competition	Yes
	individual and team scores;
	collaboration bonus;
	Epik game ranking;

gether in order to solve a problem, is an asset in the learning process, particularly in computer science.

Besides the collaboration and competition allowed the organization of questions in scenarios, embedding didactic contents, questions with "helps", and a perceptual and attractive environment, are characteristics of Epik quizzes that distinguish them from the other quizzes. Based on our evaluation, we can state that these features were very well received by teachers and students. Moreover, students consider Epik quizzes as learning activities very enjoyable, useful, innovative and fun.

Regarding the Epik development environment and its functionalities, it is desirable to integrate new question types and define a set of tools, in the Epik graphical environment, in order to facilitate the question editing task, or even allow the association of a file with questions, as is possible in some of the existing frameworks. However, it should be noted that Epik allows the reuse of educational content of the LMS, namely Moodle. This, coupled with its graphical development environment, makes the Epik quizzes development an intuitive and easy task, as was demonstrated in evaluation.

7. REFERENCES

[1] Class maker: a web-based testing service. http://www.classmarker.com/. Accessed: 2013-06-30.

[2] Educational technology and mobile learning. http://www.educatorstechnology.com/2012/04/free-tools-to-create-and-administer.html. Accessed: 2013-06-30.

[3] Epik: Edutainment by playing and interacting with knowledge. http://epik.di.fct.unl.pt/epik/. Accessed: 2013-06-30.

[4] Moodle. http://moodle.org. Accessed: 2013-06-30.

[5] Proprof: Build and tests knowledge. http://www.proprofs.com/. Accessed: 2013-06-30.

[6] Quest base. http://www.questbase.com/. Accessed: 2013-06-30.

[7] Quiz revolution. http://www.quizrevolution.com/. Accessed: 2013-06-30.

[8] Quiz rocket. http://www.quizrocket.com/. Accessed: 2013-06-30.

[9] Socrative: a smart student response system. http://www.socrative.com/. Accessed: 2013-06-30.

[10] Teacher tools for creating quizzes or polls. http://list.ly/list/L7-teacher-tools-for-creating-quizzes-or-polls.

[11] J. Burguillo. Using game theory and competition based learning to stimulate student motivation and performance. *Computers and Education*, 55(2):566 – 575, 2010.

[12] T. Connolly, M. Stansfield, and T.Hainey. An application of games-based learning within software engineering. *British Journal of Educational Technology*, 38(3):416–428, May 2007.

[13] M. Dickey. World of warcraft and the impact of game culture and play in an undergraduate game design course. *Computers and Education*, 56(1):200 – 209, 2011.

[14] R. Eck and J. Dempsey. The effect of competition and contextualized advisement on the transfer of mathematics skills a computer-based instructional simulation game. *Educational Technology Research and Development*, 50(3):23–41, 2002.

[15] W. Huang and J. Tschopp. Sustaining iterative game playing processes in dgbl: The relationship between motivational processing and outcome processing. *Computers and Education*, 55(2):789 – 797, 2010.

[16] K. Kreijns, P. Kirschner, and W. Jochems. The sociability of computer supported collaborative learning environments. *Educational Technology and Society Journal*, 5(1):8–22, May 2002.

[17] U. Munz, P. Schumm, A. Wiesebrock, and F. Allgower. Motivation and learning progress through educational games. *Industrial Electronics, IEEE Transactions on*, 54(6):3141–3144, 2007.

[18] M. Papastergiou. Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Computers and Education*, 52(1):1 – 12, 2009.

[19] B. Sampaio, C. Morgado, and F. Barbosa. Collaborative quiz development with epik. 5th International Conference on Education and New Learning Technologies (EDULEARN13), pages 506–514. IATED, July 2013.

[20] S. Slusser and R. Erickson. Group quizzes: An extension of the collaborative learning process. *Teaching Sociology*, 34(3):249–262, 2006.

[21] J. Torrente, A. del Blanco, E. Marchiori, P. Moreno-Ger, and B. Fernandez-Manjon. e-adventure: Introducing educational games in the learning process. In *Education Engineering (EDUCON), IEEE*, pages 1121–1126, 2010.

[22] N. Zea, J. Sánchez, F. Gutiérrez, M. Cabrera, and P. Paderewski. Design of educational multiplayer videogames: A vision from collaborative learning. *Advances in Engineering Software*, 40(12):1251 – 1260, 2009.

Automated Grading and Tutoring of SQL Statements to Improve Student Learning

Carsten Kleiner
University of Applied Sciences&Arts
Ricklinger Stadtweg 120
30459 Hannover, Germany
carsten.kleiner@hs-hannover.de

Christopher Tebbe
University of Applied Sciences&Arts
Ricklinger Stadtweg 120
30459 Hannover, Germany
christopher.tebbe@hs-hannover.de

Felix Heine
University of Applied Sciences&Arts
Ricklinger Stadtweg 120
30459 Hannover, Germany
felix.heine@hs-hannover.de

ABSTRACT

In this paper we present a concept and prototypical implementation of a software system (aSQLg) to automatically assess SQL statements. The software can be used in any introductory database class that teaches students the use of SQL. On one hand it increases the efficiency of grading students submissions of SQL statements for a given problem statement by automatically determining a score for the statement based on different aspects. On the other hand it may also be used to improve student learning of SQL statements by enabling them to continuously (re-)submit their solutions and determine improvements in quality by comparing the automatically determined scores. In order to keep the administrative overhead for using it minimal we have implemented the software in a way that it may be plugged into any course/learning management system with minimal overhead. We have used it in conjunction with WebCAT as well as our own proprietary course management system. Student feedback collected after its first usage in a database class shows promising results for future usage of the system.

Categories and Subject Descriptors

K.3 [**Computers and Education**]: Computer and Information Science Education, Computer Uses in Education; H.2.3 [**Database Management**]: Languages

General Terms

Computer science education, information systems education, distance learning, SQL

Keywords

Automated grading, automated assessment, tutoring, learning management system, SQL

Koli Calling '13, November 14-17 2013, Koli, Finland
Copyright is held by the authors. Publication rights licensed to ACM.
978-1-4503-2482-3/13/11 ...$15.00
http://dx.doi.org/10.1145/2526968.2526986.

1. INTRODUCTION AND MOTIVATION

Apart from database schema modeling, learning to use the SQL query language in a correct and efficient manner is among the most important goals of an introductory database system course. Whereas quality of modeling (at least on the conceptual level) seems almost impossible to assess in an automated or at least computer-assisted manner, the correctness and efficiency of SQL statements seems more appropriate to automated or computer-supported grading.

Automated or computer-supported grading is particularly important in introductory database system courses, since on one hand the number of students enrolled is rather large and on the other hand funds tend to be low and proficient students to perform grading and support lab sessions are difficult to find. Thus in order to improve efficiency of the educational institution as well as making individual assessment feasible, computer assistance is very helpful in this context.

Another aspect mandating a web-based system to be employed in computer science education in general is the possibility to provide students with space- and time-independent access to the outcome (result and contextual feedback) of exercises they submitted. At first this improves the attractiveness of the particular institution for students; also it may improve student learning outcomes since students can decide on their own when and how to work on exercises. It can also be useful to improve performance of students in distance learning classes by including electronic tutors as described in [12] for database courses. Thus using a web-based automated assignment grader is highly recommended.

For all these advantages to play a major role, it is not sufficient to develop and use a tool for a database system course alone. It is rather important to employ a single system that may be used in several different courses throughout the curriculum. By doing so, many classes can benefit from the aforementioned advantages. Also usage of the system by the students becomes more efficient and less error-prone since they are already used to the general functionality and look-and-feel of the software in place. Consequently, we never aimed at building a new course management system (CMS). We rather implemented it in a way that it may plugged into existing CMS with minimal effort. We have used it as a plug-in to WebCAT ([5]) as well as our university's proprietary CMS.

We have developed a tool called aSQLg which can perform an automated assessment of student submitted SQL queries. It may on one hand be used as a tutoring tool to improve the quality of the student results. This is achieved by them trying to improve their scores with repeated submission of hopefully better solutions based on the feedback. On the other hand it can also be used for improving efficiency in assignment (or even exam) assessment; in this case the number of possible submissions by a student can be restricted, thus delivering a fair assessment of the quality of each solution without the (in this case undesired) tutoring capabilities. The tool had been used in first experiments based on real database course assignments with a few students at first. After achieving some maturity it has been used in our regular introductory database system classes in 2012 and 2013 in two different departments of our university. Even though the number of students enrolled is too small (lower three digit number in total) to derive statistically significant results important positive and negative feedback has been collected and will be discussed here.

This paper is organized as follows: after reviewing related work in the areas of automated grading as well as support for computer-supported learning of SQL we will present the concept which forms the foundation for our implementation in section 3. The concept will then be illustrated by giving some example statements and results which have been used in the real course in section 4. Thereafter we briefly explain the implementation which further illustrates the general functionality in section 5. We present some interesting aspects from the student evaluation in section 6 before we finally conclude with a summary and ideas for future work.

2. RELATED WORK

Publications related to our work may be roughly categorized into two areas, namely improving learning of SQL on one hand and automated grading and assessment on the other hand. Almost no work to the same level of detail as ours is known in the intersection of the two which is one of the major contributions of this paper. We briefly review relevant literature from each of the two areas below.

The first group of related publications focuses on the learning process of developing SQL queries by students. In [15] an intelligent SQL tutoring system is described. The system has been extended to use a web-based front-end and is focused on improving student learning by giving them real-time feedback on the submitted SQL queries. It supports them in working on the queries until they are correct. Similarly [12] describes a tutoring system for database system courses which is also focused on improving student learning by immediate web-based feedback. In contrast to [15] it seems to also support other aspects of database systems basics apart from SQL queries. Since the concept we propose in this paper is also targeted towards automated grading within a CMS rather than only focusing on assisting student learning, the tools described in the previously mentioned references could complement our concept. Ideas from these tutoring systems could be used to improve our feedback component in the future.

There are also a number of papers focusing on the grading aspect of SQL queries: the work described in [17, 18] introduces an assessment strategy for SQL queries that may be used in web-based tutoring and assessment systems. It is designed in order to improve student learning since the way students learn is greatly influenced by the way the assessment is performed. The system focuses on interactive query refinement rather than a mere grading of a given solution as in our case. Our grading approach is more complex and detailed though. In [8] the authors describe a tool (SQLify) for partially automated grading of SQL queries using an elaborate algorithm for assessment. This tool focuses heavily on peer-reviews and interaction in order to improve student learning. Thus it still requires a lot of manual work as opposed to our software. In [4] the authors sketch a system architecture which seems to include both a (newly designed) tutoring as well as a scoring component. No details on the grading component are explained though since the short article focuses on the overall system architecture.

In [7], an application based on the GNU SQL Tutor (see [1]) is described. It contains both a tutorial part and an exam part. The grading part is not explained in detail in the paper. The grading score seems to be binary for a single question.

SQL-KnoT [6] features question templates, which are used to generate actual questions increasing the variability. We believe this feature to be highly valueable in our setting and plan to integrate similar functionality.

To test whether a given student's SQL statement is a valid answer to a given problem, most tools (including our aSQLg), compare the outcome of the statement with a reference solution. A different approach for this problem is followed by SQLator [20]. This tool uses various heuristics to detect equivalence between the solution and the reference statement. The drawback is that possbily correct solutions are not detected. An intermediate aproach is followed by [9], who uses result comparison first to check correctness, and semantic comparison by heuristic reformulation rules to give hints to the student.

On the other hand at institutions with a large student to instructor (or TA if present) ratio one would probably prefer to focus on the automated grading portion rather than on the interactive approach to student learning since this increases efficiency. Therefore we advocate the inclusion of the SQL grading component into a general purpose web-based grading tool such as WebCAT ([5]) or general purpose CMS such as Moodle. There are also other learning management systems described in the literature: [16] introduces a generic automated grading system which may be used for different types of courses. Good experiences are reported for different programming classes; the system is not web-based though, but rather requires a full client application. Also it is not obvious on how it could be extended to cater for the specifics of SQL grading without additional information. Other similar systems for programming or other computer science courses are described in [10, 11, 14]. In [19] there is an interesting overview of general purpose as well as course specific learning management systems (LMS) that had been introduced in the literature. The overview does not only include course management but also lists a lot of systems focusing on improving student learning along with an impressive list of references. Any of the systems listed in the online submission and automated assessment section of that paper could in principle be used as foundation for implementation of our concept.

3. AUTOMATED SQL GRADING CONCEPT

We will now present the concept of how our SQL grading is performed in detail. The concept is visualized in figure 1 as a flow diagram starting with loading all statements, which have been provided. It finishes with setting the points for the graded assignments and delivering them to the student. The complete flow of all actions during the grading process is as follows:

1. Load all SQL-statements
2. Check one statement for forbidden elements (optional)
3. Check if the statement equals reference solution (optional)
4. Check for syntactical correctness (points)
5. Check statement cost (points)
6. Check correctness of result of the statement (points)
7. Check statement style (points)
8. After all statements processed: add points & generate report
9. Additional manual grading of statements (optional)

10. Update total grade

At the beginning of the grading process all statements of an assignment are loaded into the plug-in. Optionally it is possible to load a reference solution, too. After all statements have been loaded, the statements are analyzed individually, one statement at a time. The first thing checked is, whether a statement contains forbidden elements or not. This step is optional and can be selected and configured by an instructor. The filter uses a set of allowed or forbidden elements, so whitelists or blacklists may be used for filtering. To get all elements of a statement a SQL-parser called JSqlParser [2] is used. The parser reads a statement and creates a tree-structure of all elements. Every element is represented by a corresponding Java class. All values in a statement are saved in variables of types like Integer, String and so on. Such classes are used to check whether an element of the statement is allowed or not. JSqlParser is used in some later steps, too. If a statement is filtered out by the statement filter, it is ignored in subsequent steps and the next statement gets checked by the filter.

If a statement is allowed, the statement is compared to the reference solution. If they match the statement is correct, gains the maximum points and the grading is complete for this statement. Because using a reference solution is optional, this step is performed only when it is available.

If a statement does not match, the next step is performed. The test of syntactical correctness of a statement is performed using the database. We do not use the JSqlParser here as it did not accept all possible queries according to the specific database system dialect used (Oracle) in our class turning out to be a major source of frustration on the student side. Syntactical correctness is determined by using the query cost of a statement. If the result does not contain an error, the syntax of the statement is correct and the syntactical correctness points for the statement are awarded. If the syntax contains errors, the statement is discarded. If the syntax is correct the query cost which has already been retrieved is evaluated. The instructor can choose a limit of the maximum allowed cost, to prevent the execution of a long running statement which could block the rest of the statement grading like a denial of service attack. If the query cost of the statement is not acceptable, the check for correctness is skipped.

Evaluating the cost of a statement makes sense only, when it is at least partly correct. That's why the syntactical correctness is checked first. The number of points awarded for efficiency depends on how close the query cost of the students statement is to the reference solution or to a given value from the configuration file. When the cost is not to high the correctness of the statement can be checked using the statement in listing 1 which should return no results.

```
( (<student-statement>)
    UNION
  (<reference-statement>))
MINUS
( (<student-statement>)
    INTERSECT
  (<reference-statement>));
```

Listing 1: Statement used to compare the student statement with the reference solution

In the practical experiments with the students it turned out that such a strict correctness check led to a lot of frustration. This happened because students had problems finding the exact correct solution without any support. Thus we introduced two additional steps in correctness checking which helped the students correct their previous errors. At first before executing the statement in listing 1 we checked if the datatypes of the student and the reference solution match and are in the correct order. If that was not the case the student is informed by a corresponding message. After the execution in order to be able to verify correct usage of sorting there was an additional check if the sorting of the student had been the same as in the reference solution. This check can be activated for each individual problem as it is not required for all statements. These changes are subsumed in figure 1 in the box check correctness.

Checking the correctness of a student statement using the statement 1 is only possible if a reference solution is available. If not, the student's statement is executed in the database and the result set is saved for manual grading. If the statement is used, the result sets of the student statement and the reference solution are compared to each other. If they match, the statement gets the maximum correctness points possible. Currently partial credit for correctness cannot be assigned by the automated grading step. If that is desired, an additional manual grading step by an instructor may be added.

The last automated grading step is the style check of a statement. In general it is very difficult to assess the style of a SQL statement as there are no generally accepted rules on SQL style in contrast to e.g. Java. Thus we decided to use a set of rules which describe what a SQL statement should look like these have been taken from different sources. aSQLg is delivered with a small list of rules describing a general style. This list does not cover a complete style description, hence a complete list of rules has to be generated by the instructor. The number of points awarded during this step depends on how many style rules are followed by the student statement.

Now all checks for the statement have been executed. If there are statements left, the next statement is chosen to be analyzed by our grader. If the last statement has been checked the total points of the automated grading part are calculated. Additionally reports about the single grading steps are generated, which contain errors that may have occurred and other information including the number of points received by each statement in every step. The reports are shown to the student in the regular GUI of the CMS used. The number of available points for a statement in each step may be set in a configuration file. The four grading steps for syntactical correctness, cost of the statement, result correctness and style are evaluated. The weight of the different grading steps may as well be set in the configuration file. If maximum points for one of the grading steps is set to 0, the step is skipped during grading.

After the report generation and point calculation the statements can be optionally graded manually by an instructor. This step may be included directly into the CMS, e. g. by a special web form. The instructor has total access to the results of the automatic grading steps performed. The main task of the instructor normally is to check the correctness of the students statement, if the correctness could not be verified during the automated part. The instructor may award additional points to the student.

One point to consider is the danger of SQL-injection attacks on the database used to analyze and execute student statements. Malicious students could try to attack the database. In our concept this is not really an issue, because of two things: all students have read access to the database schema used for the queries anyway and no student query is ever directly executed on the system. They are just executed in the context as explained in listing 1. Thus in combination with correct setting of access rights to system tables by the database administrator students cannot perform an insider attack.

Some students may try to get full credit without learning and working on the statements. For example, if a task of an exercise demands a complex statement, which just returns one number as

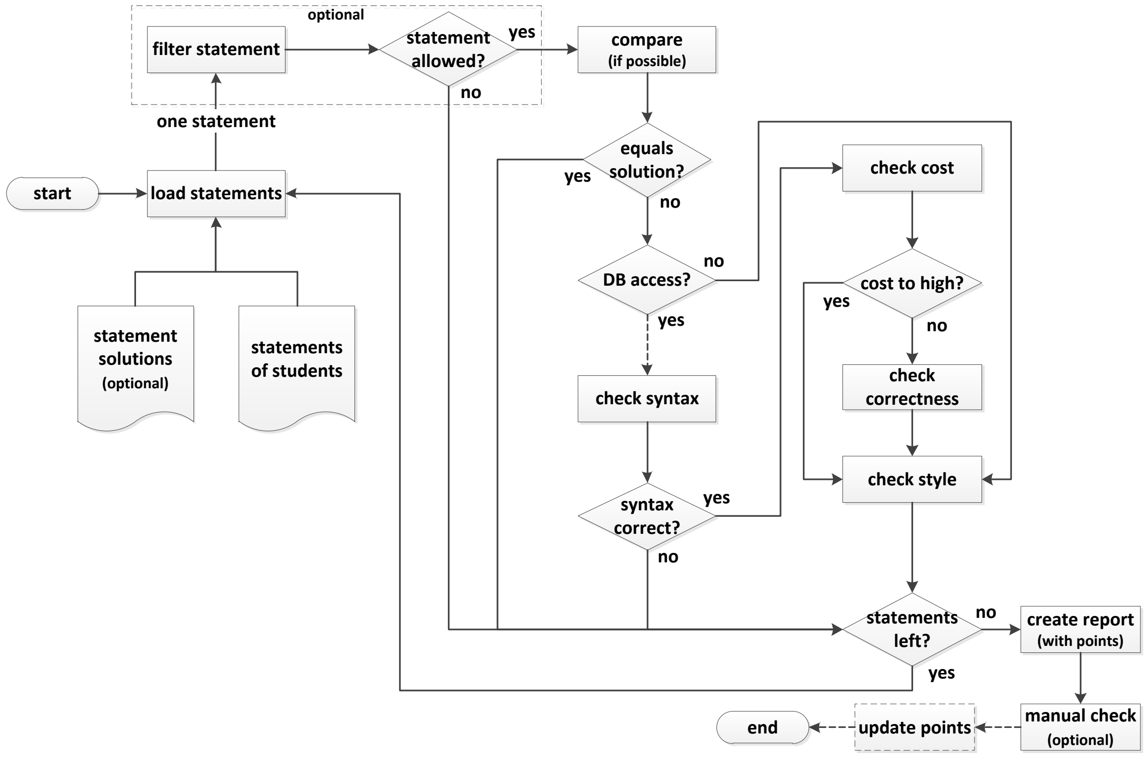


Figure 1: Flow diagram of the different steps in our SQL-statement Grader (based on [13])

result, a student could decide to cheat. When the statement result is 5 the student could use the statement "SELECT 5 FROM DUAL", which would return the correct result, without really working on the task. Countermeasures are needed to prevent this. That is why the filter is used at the beginning of the analyses process. Using the filter, usage of the table "DUAL" can be prevented. Alternatively execution of a statement on different datasets is possible which can also prevent such simple cheating attempts.

There is no semantic analysis of the provided solutions in comparison to the reference solution (i. e. comparison of query trees) so far. This would facilitate assigning partial credit for almost correct solutions. This feature is on the list of open issues for a future version of the system and would be an addition to the correctness check.

The described concept covers the whole grading process of SQL-statements. Nearly all checks are automated and statements containing forbidden elements can be identified and discarded. Only checking the results of partly correct statements has to be done manually. The syntax check and the filter need manually created lists to work properly, but the execution is done automatically. The only thing not mentioned so far is the identification and prevention of plagiarism, which figured out to be to complex to implement during the project.

4. SOME EXAMPLES

In this section, we present a real-life example showing how the grader was used during a first year database course. Part of this course was an introduction to SQL, which was accompanied by tutorials and about 40 SQL exercises of increasing difficulty.

The chosen example refers to an exercise from the beginning of the SQL tutorial. It uses the well-known Oracle example table `employees`. The task was to select every employee (first and last name as a single string) and the year of the hire date. The result had to be sorted by hire year and last name. A valid solution was:

```
SELECT first_name || ' ' || last_name name,
TO_NUMBER(TO_CHAR(hire_date, 'YYYY')) hired
FROM hr.employees
ORDER BY hired, last_name;
```

We now describe a simplified log of a student trying to solve the task. The first upload to aSQLg contained the following SQL statement:

```
Select First_Name||' '||Last_Name,
to_char(Hire_Date, YEAR)
from hr.employees
order by Hire_Date, Last_Name;
```

The syntax check of the grader failed, delivering a message to the student including the original database error: `ORA-00904: "YEAR": invalid identifier`. Due to the incorrect syntax, all further checks were skipped. In the following attempt, the student fixed the syntax error:

```
... to_char(Hire_Date, 'YYYY') ...
```

Now the syntax check succeeded, so the grader could proceeded with the cost check, which turned out to be uncritical. In the next step, the column count was compared with the correct solution, which also succeeded. However, the data type check failed, because column two returned a character value while a numeric value was expected. The following message was shown:
`Datatype of column 2 is wrong.`

```
Expected:  NUMBER, your solution:  VARCHAR2
```
The student went on with this solution:

```
... to_number(to_char(Hire_Date, 'YYYY')) ...
```

This change corrected the problem, so the grader now confirmed that the column count and data types were correct. The row count and row contents were also correct. The final error message was stating that the result rows were not sorted as expected, because the `order by` columns were still wrong (sorted by the whole `hire_date` instead of only the year). In the final submission, the student responded to this hint:

```
Select First_Name||' '||Last_Name,
to_number(to_char(hire_date,'YYYY')) AS datum
from hr.employees
order by datum,Last_Name;
```

Finally, the student was informed that the solution had been fully correct and that the full score had been recorded. Only warnings remained about the naming of the columns of the result, because of the differences to the proposed solution.

Summarizing, the example shows how the detailed messages generated by aSQLg can guide a student step-by-step towards the correct solution of a given exercise.

5. IMPLEMENTATION OVERVIEW

Our system consists of six modular components. Together all of this components implement the grading environment, facilitate statement grading, perform the grading steps and the generation of reports.

The goals of the implementation architecture are:

1. Modular structure
2. Extensibility of SQL dialects supported
3. Possibility of reuse for modules in other projects
4. Adaptable for different e-learning platforms with minimal programming effort

Figure 2 shows the components and their relationships. This structure makes it possible to easily adapt the statement grader to other environments than WebCAT by changing only a single component.

In the sequel we will briefly describe each of the components.

5.1 aSQLg Core

The aSQLg Core embeds the plug-in into the hosting environment system, e. g. WebCAT. This component configures the other components and implements the overall flow control for SQL grading.

The configuration of all other components is done in the order Reporter, StatementLoader, StatementViewBuilder, StatementFilter and StatementTester. While the first step sets up the reporting engine, the next two steps do the necessary groundwork to transform statements into an internal structure in order to be able to grade them. The StatementTester finally performs the actual grading steps with help of the CheckStyle component.

5.2 Reporter

The Reporter component collects information, that is generated by the other components, and creates report files. This component is used by most of the other components.

Information is generated for two receivers. The main receiver is the student who receives general information on the grading process, error notes and warnings for his statement as well as the awarded points. The instructor and assistants receive all this information on demand as well. In addition they have the option to add comments and points in the manual grading step. Finally instructor or system administrator receive detailed information about configuration errors and database failures.

There are two types of messages used to submit all of the above pieces of information. Plain messages have a simple text or XML content. Referenced messages also have a text or XML content but in addition contain information about the code position, to which the message refers. The latter makes it possible to show a code fragment within the results and mark the position where an error occurred - a helpful feature for grading assistants.

After the completion of the grading process the reporter generates an XML formatted report. This report can be converted by an XSLT stylesheet. By default the report is converted to XHTML, but the WebCAT plug-in package comes with a specialized transformation file to embed the reports into the WebCAT user interface.

5.3 StatementViewBuilder

For grading of SQL statements, in addition to the plain syntax, one needs a structured representation of the statement. For instance one of the tokens may belong to the blacklist of forbidden elements when used as a table name whereas it may be allowed as column name. Thus the StatementViewBuilder component builds a structured object view of a statement in form of a tree.

The information about the semantics of a token can be gained by parsing the statement. Our implementation uses a combination of JSqlParser [2] with our own simple extension. By implementing a different StatementViewBuilder interface this base is easily changeable. For each token in the statement a node in the StatementView tree is generated. This node contains the string representation of the token, as given in the input statement, a syntax type and a semantic type. The main syntax types occurring are keywords and parameters. The semantic types used vary from table, over column to numbers and texts.

5.4 StatementFilter

Giving students the possibility to submit any statement for grading by a central server, may incur a security risk as well. Even if not intended a statement might be harmful for the system. E.g. in the case where the instructor requests the statement `DROP Student-Table` a student could send the statement `DROP Instructor-Table`. Our StatementFilter component analyzes statements submitted in order to prevent such insider attacks. Denial of service attacks by inefficient statements are handled by the StatementTester later on.

The base for the StatementFilter is the previously generated StatementView. The filter can be configured by either using a whitelist of allowed or a blacklist of forbidden elements. Such elements will be detected by the StatementView. If one of it exists, the statement will be marked illegal, is never executed and a message to the student is generated.

5.5 StatementTester

The StatementTester is the central component of the grading process. It has two main functions, statement loading and execution of the statement.

The loading step includes reading SQL statements (as strings), using the StatementViewBuilder to form a StatementView and finally point allocation. All statements are loaded from files sub-

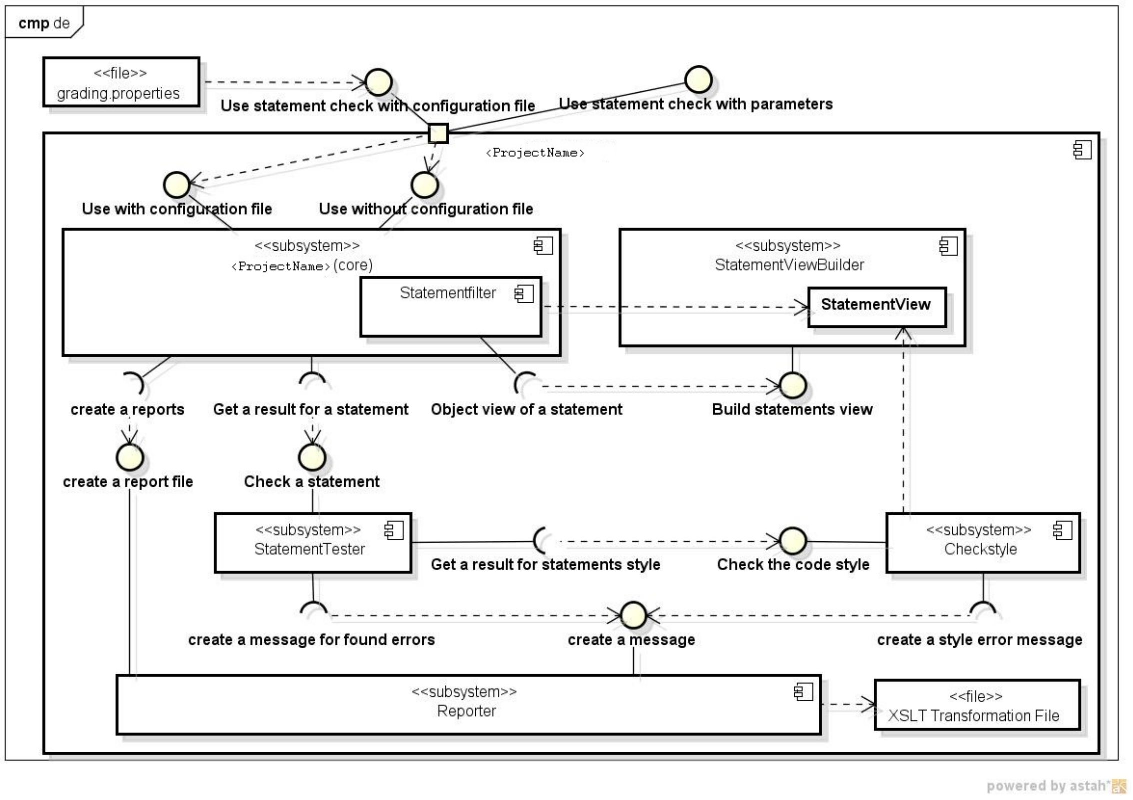


Figure 2: Overview of the components for implementation of aSQLg

mitted by students. Statements can have a reference statement, a solution given by the instructor. Student statements are compared against the instructors statement.

Allocation of points may consist of the following components: syntax check, query cost check, result check, style check and manual grading. For each of these aspects a maximum number of points may be defined by the instructor. Apart from the correctness components (syntax and result) where only full or no credit is possible for an individual statement, partial credit is possible.

After all the groundwork the actual grading is performed for legal statements. As the grading depends on the particular database system we use a "default" StatementTester class which may be extended to cater for specific systems. We use an OracleStatementTester extension in our classes. The default implementation may also be used as a fall back. It provides checks that can be used if there is no specific SQL dialect to be used.

The default StatementTester implements the general flow of the check as explained in section 3. The result check for SELECT statements is performed by comparing the results of the students statement and the instructors statement. If they are identical, the step is passed and full credit awarded. The syntax check in the generic tester is done by parsing the statement with the zql parser [3]. If the statement can be parsed, the syntax is correct, otherwise the parser throws an exception with the information of the line and column where a syntax error exists. This information is added to the reporter data to be provided to the student later on. The last step is the style check. It uses the CheckStyle component.

The OracleStatementTester uses the possibilities of the Oracle database and checks the statement for the Oracle specific SQL dialect. Here the syntax check is performed by using functions for calculating query costs. These functions return an error, if the statement contains a syntax error. The query cost check uses the result from the previous syntax check, because it is computed there anyway. The result and style check are performed using the implementations of the default StatementTester.

5.6 CheckStyle

Base for the CheckStyle component is the StatementView with its syntactic and semantic information. The instructor may define a style rule file. Each rule defines to which syntax and/or semantic type it is applicable. Furthermore rules can be defined for special keywords. As there is no commonly accepted *good* style for SQL queries so far, this part is completely configurable by the instructor. A foundation observing some of the most commonly rules for *good* style is provided as a reference.

6. EVALUATION

As mentioned earlier aSQLg has been used in our introductory database system classes in two different departments in 2012 and 2013. SQL statements have not only been assigned in aSQLg but classical paper assignments have also been used. On one hand that reduced the risk associated with employing a completely new and unproven tool. On the other hand that put the students in the position to better compare the two approaches and make informed judgments about the usefulness of automated SQL grading. In addition to the regular student course evaluation an online survey specifically targeted at the evaluation of the usage of aSQLg has been administered. In total 114 students (about 80 % of students en-

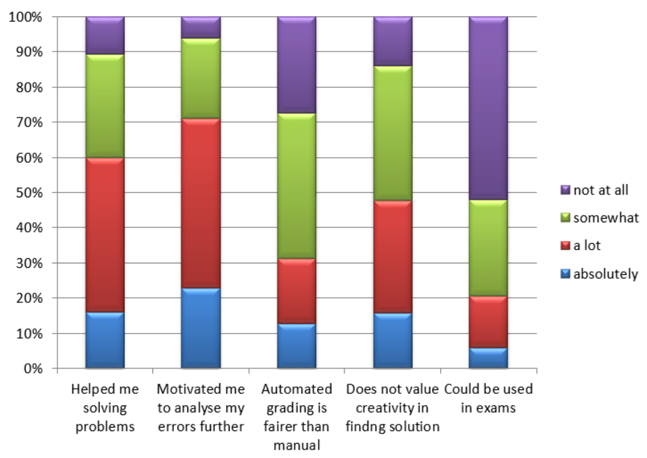


Figure 3: Exemplary results from student survey

rolled in the class) submitted answers to this specific survey which consisted of 16 questions.

Some interesting results from the survey are displayed in figure 3. They show that about 60% of the students felt supported a lot or more by aSQLg in finding a solution (and only just over 10% did not feel supported at all). This supports the claim that there is potential for being used as a tutoring system as students believe in the tool helping them find solutions. Consequently almost 40% reported that the immediate feedback was the most helpful feature of the tool.

Motivation by students to dig deeper into problems when they are encountered is increasingly difficult to achieve; this in our experience is particularly true for SQL and database classes in general. In the case of aSQLg 70% of the students were motivated a lot or more by the tool to further analyze their errors in more detail. This is an extremely good and promising result. Maybe that has also led to the better student judgment of the whole course when compared to previous years.

Still most students (about 70%) expected manual grading to be fairer than automated grading, probably because they (just over 50%) did not like that just the result of the submission counts as opposed to the way of determining it which might be taken into account during manual grading. In a future version of aSQLg we would like to (at least partially) replace the result-oriented assessment with a semantic assessment. That kind of assessment would be able to compare the query tree of the ideal solution structurally against the submitted solution and thus value the way of solving the problem better even if there is a minor error which may render the result completely wrong. This would also remedy the most significant negative aspect of the system (30%, cf. Fig. 4).

Just over 10% of the students feared that students would tend to work alone instead of in teams as a consequence of using aSQLg and just over 15% stated that an advantage of the tool would be reduced need for personal instruction (cf. figure 5). Also about 25% said that reduced personal contact to the instructor might be the most significant drawback. These results show that as far as tutoring is concerned automated grading can be a good addition but will and should in general not replace personal lab sessions. Again students feared most that grading would be reduced to a comparison of results instead of valuing the way that has been used to find a solution, see above.

On the positive side from a student's perspective the immediate feedback to their solutions has been the most important significant

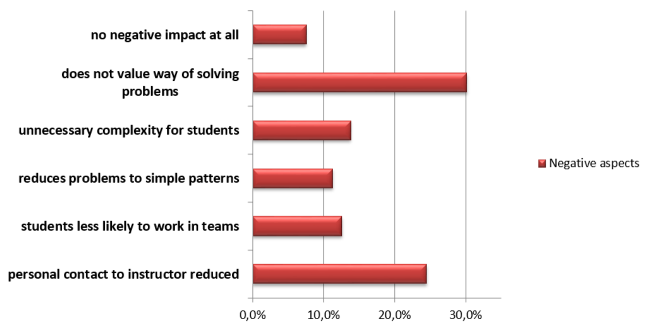


Figure 4: Most significant negative aspect of automated grading (percentage of students agreeing)

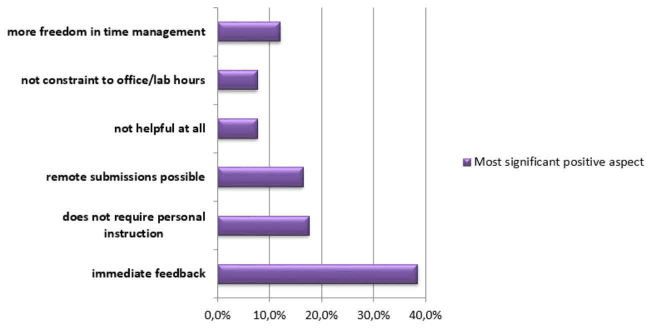


Figure 5: Most significant positive aspect of automated grading (percentage of students agreeing)

aspect (almost 40%). Also aspects relating to distance education and electronic self-guided learning such as no need for personal instruction (18%), remote submission feature (17%) and more freedom in time management (12%) have been named often by the students. This supports the claim about the positive impact of aSQLg for electronic tutoring and distance education.

Finally, even in its first two instances the tool worked pretty good already with very few false negatives (would be very frustrating for students) and almost no false positives. Nevertheless students would not like to see it being used in exams (just over 50% said *not at all suitable*). That is probably because as long as there is at least a minimal chance of a false negative assessment students fear to be penalized in their grades for an erroneous software. Almost 60% of the students would like to see aSQLg be used in advanced database classes again and about 40% would like to use automated grading in other classes, too.

7. CONCLUSION AND FUTURE WORK

In this paper we have presented a concept for automated assessment of SQL statements. The proposed algorithm uses the following aspects of the statement submitted in order to determine the score: syntactical correctness, efficiency of statement, correctness of results and style. Apart from these automatically assessed dimensions where points for each part are configurable it is also possible to assign points in a manual grading step as well. Finally in order to be able to practically use the software and prevent harm from the system by students submitting malicious solutions we have added the following features. There is the possibility to use black- or white-listing for allowed parts of the statement. Moreover statements beyond certain configurable efficiency bounds are not executed in order to prevent denial of service attacks.

We have already used the system with individual students in a lab environment as well as in introductory database classes in two different departments at our school in 2012 and 2013. In efforts to get personal feedback and be able to make the system as student-friendly as possible, valuable input by the students has already been used in improvement steps of the system. We also received several of the ideas on how to attack the system in these lab sessions.

The system may both be used as an electronic tutor as well as for efficient course management. In the first case it guides the students to better results by them trying to increase the automatically assigned score. It also simplifies to take part in lab sessions for students who cannot or do not want to attend sessions on campus. For the second case it should be used in conjunction with a CMS. To achieve fair scores for the students in this setting the number of possible submissions should be restricted. Still manual grading (at least partially or to double-check the automated results) is possible within this framework if desired by the instructor.

The gain in efficiency by using the system as compared to pure manual assessment is obvious. This is because a completely manual grading process is still possible and just using the correctness check for instance is already an improvement over manual checking. The improvement in student learning by being able to submit result candidates multiple times and improve on the solution by comparing automatically computed scores which are shown as immediate responses has been the second goal of the system. This goal seems achieved partly already when looking at the feedback in figure 3. Also the student responses improved from 2012 to 2013 showing that the improvements in the system have been valued. Nevertheless the system is not yet perfect as some false negatives show, we will further have to work on that. Similarly assignment of points to different grading steps for different problems and at different points in a course needs to be further elaborated.

To further improve on the tutoring capabilities it would be very interesting to integrate our assessment system with one of the tutoring systems described in section 2. Thus one would obtain an even more valuable SQL learning tool for student self-study. The tutoring capabilities in addition to the score related feedback would be a perfect combination for self learning. Thus distance learning would be even more simplified as students can submit the solutions from any place and use the responses to improve their results an their own.

In addition in the near future we will plug the software into the new university-wide CMS lonCAPA. As explained above architecture and design simplify moving to such different hosting systems. Also we plan to publish the code once the system has reached production maturity for use at other institutions and also to get more feedback from a wider audience.

Finally the correctness check could be significantly improved by performing a semantic comparison of student and reference solution. An extended statement parser is needed for this check; this will be part of a future version of aSQLg .

We would like to thank Andreas Stöcker and Florian Fehring for supporting implementation and student evaluation for aSQLg .

8. REFERENCES

[1] GNU SQL tutor - website. http://www.gnu.org/software/sqltutor/. (visited on 16.09.2013).

[2] JSqlParser - website. http://jsqlparser.sourceforge.net/. (visited on 23.07.2011).

[3] ZQLParser - website. http://sourceforge.net/projects/zql/. (visited on 18.08.2011).

[4] A. Abelló, E. Rodríguez, T. Urpí, X. B. Illa, M. J. Casany, C. Martín, and C. Quer. LEARN-SQL: Automatic Assessment of SQL Based on IMS QTI Specification. In *ICALT*, pages 592–593. IEEE, 2008.

[5] R. Agarwal, S. H. Edwards, and M. A. Pérez-Quiñones. Designing an adaptive learning module to teach software testing. *SIGCSE Bull.*, 38:259–263, March 2006.

[6] P. Brusilovsky, S. Sosnovsky, M. V. Yudelson, D. H. Lee, V. Zadorozhny, and X. Zhou. Learning SQL Programming with Interactive Tools: From Integration to Personalization. *Trans. Comput. Educ.*, 9(4):19:1–19:15, Jan. 2010.

[7] A. Cepek and J. Pytel. SQLtutor. In *Professional Education 2009 – FIG International Workshop Vienna*. FIG Fédération Internationale de Géomètres, 2009.

[8] S. Dekeyser, M. de Raadt, and T. Y. Lee. Computer Assisted Assessment of SQL Query Skills. In J. Bailey and A. Fekete, editors, *ADC*, volume 63 of *CRPIT*, pages 53–62. Australian Computer Society, 2007.

[9] R. Dollinger. SQL Lightweight Tutoring Module – Semantic Analysis of SQL Queries based on XML Representation and LINQ. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010*, pages 3323–3328, Toronto, Canada, June 2010. AACE.

[10] C. Douce, D. Livingstone, and J. Orwell. Automatic test-based assessment of programming: A review. *ACM Journal of Educational Resources in Computing*, 5(3), 2005.

[11] X. Fu, B. Peltsverger, K. Qian, L. Tao, and J. Liu. Apogee: automated project grading and instant feedback system for web based computing. In J. D. Dougherty, S. H. Rodger, S. Fitzgerald, and M. Guzdial, editors, *SIGCSE*, pages 77–81. ACM, 2008.

[12] C. Kenny and C. Pahl. Automated tutoring for a database skills training environment. *SIGCSE Bull.*, 37:58–62, February 2005.

[13] C. Kleiner. A concept for automated grading of exercises in introductory database system courses. In *Proceedings of the 7th International Workshop on Teaching, Learning and Assessment of Databases (TLAD)*, 2009.

[14] L. Malmi, V. Karavirta, A. Korhonen, and J. Nikander. Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *ACM Journal of Educational Resources in Computing*, 5(3), 2005.

[15] A. Mitrovic. An intelligent sql tutor on the web. *I. J. Artificial Intelligence in Education*, 13(2-4):173–197, 2003.

[16] R. E. Noonan. The back end of a grading system. In D. Baldwin, P. T. Tymann, S. M. Haller, and I. Russell, editors, *SIGCSE*, pages 56–60. ACM, 2006.

[17] J. C. Prior. Online assessment of sql query formulation skills. In T. Greening and R. Lister, editors, *ACE*, volume 20 of *CRPIT*, pages 247–256. Australian Computer Society, 2003.

[18] J. C. Prior and R. Lister. The backwash effect on sql skills grading. In R. D. Boyle, M. Clark, and A. N. Kumar, editors, *ITiCSE*, pages 32–36. ACM, 2004.

[19] G. Rößling, M. Joy, and et al. Enhancing learning management systems to better support computer science education. *SIGCSE Bulletin*, 40(4):142–166, 2008.

[20] S. Sadiq, M. Orlowska, W. Sadiq, and J. Lin. SQLator: an online SQL learning workbench. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '04, pages 223–227, New York, NY, USA, 2004. ACM.

Communication Patterns in Collaborative Software Engineering Courses: A Case for Computer-Supported Collaboration

Antti Knutas
Lappeenranta University of Technology
P.O. Box 20
FIN-53850 Lappeenranta, Finland
+358-0294-462-111
antti.knutas@lut.fi

Jouni Ikonen
Lappeenranta University of Technology
P.O. Box 20
FIN-53850 Lappeenranta, Finland
+358-0294-462-111
jouni.ikonen@lut.fi

Jari Porras
Lappeenranta University of Technology
P.O. Box 20
FIN-53850 Lappeenranta, Finland
+358-0294-462-111
jari.porras@lut.fi

ABSTRACT

Collaboration has become an important teaching method in software engineering and there are several computer supported collaboration tools to aid the development and learning process. However, most studies have concentrated on intra-group studies. We believe that computer supported collaborative learning tools can also aid software engineering students to have beneficial inter-group collaboration. In this research the communication patterns in three collaborative software engineering courses were analyzed with the method of social network analysis. It was found out that students do collaborate, but mostly along pre-established social connections. The main reason for this was the difficulty in matchmaking and discovering others who were struggling with the same problems. Our proposal is to study how students in similar learning scenarios benefit from computer supported collaborative tools that increase networking opportunities. The findings presented in this paper provide a baseline for comparison when performing social network analysis in future studies.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education – *collaborative learning.*

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education.*

General Terms

Measurement, Documentation, Experimentation, Human Factors.

Keywords

Collaborative learning, computer supported collaborative learning, social network analysis, software engineering education

1. INTRODUCTION

University education and information technology teaching are going through a time of change. Learning is changing to be more interactive and the importance of collaborative learning and teamwork has grown [21]. At the same time intensive courses and team-based rapid development methods are growing more popular in software engineering education. In these approaches the goal is not only to have the students cooperate in groups, but to help each other achieve their learning goals by collaborating, for example by sharing newly learned knowledge with each other and then applying it to improve their group work. These methods have been proven to work in tertiary level education in both domestic and international studies [8, 25, 26].

However, there has been little research in how the presence of multiple groups in the same workspace affects the patterns of collaboration and if there are methods of arranging the group work in a manner that groups can benefit from each other's presence. There has been research into computer supported collaborative learning (CSCL) tools that enhance communication within classrooms, but again little research in inter-group communication. This suggests towards there being a research gap in applying CSCL learning tools to inter-group collaborative learning, which we believe to be possible and beneficial.

Collaborative learning in intensive courses (the Code Camp course series) has been studied previously in Lappeenranta University of Technology [2, 25, 26] and collaborative teaching courses in general have been studied generally [16, 24], but there have been no detailed research into inter-group communication. From observations in earlier studies it can be seen that intra-group communication occurs and some students feel that it is a beneficial part of the course [2], but more exact communication patterns and how they affect information traversal in student groups is still unclear. This gave the motivation to perform a more rigorous study on how the student communication actually occurs during the courses. Three Code Camp courses were selected for observation in order to map and analyze the student communication. The main research questions are:

1. How students utilize different communication channels during courses for collaboration?
2. Which kinds of patterns of collaboration emerge during the course, especially between different student groups?
3. Are there any available resources present in the classroom environment that could be changed to encourage more comprehensive communication or cooperation?

Koli Calling '13, November 14–17, 2013, Koli, Finland.
Copyright 2013 ACM 978-1-4503-2482-3/13/11…$15.00.
http://dx.doi.org/10.1145/2526968.2526987

In order to identify emerging communication patterns in student communication in the course, the student communication that occur during courses must be mapped first. Information for the study was gathered with individual surveys, recording time-lapse video for analysis and team interviews, after which patterns of collaboration were analyzed by modeling the communication patterns with the help of graph theory. The study aims to find repeating patterns with the help of social network analysis and to identify communication patterns, which could be improved with the help of CSCL tools. If any repeating patterns are found, this study can provide requirements for the next step of research, which is to implement a new CSCL system to address any found issues. The research results from this study also provide a baseline for comparison when social network analysis is applied to the improved courses.

The rest of the paper is organized as follows: the section two covers related research, the section three covers the research methods, how the research was performed and the research results, the section four analyses and discusses the results and the section five presents the conclusions.

2. COLLABORATIVE LEARNING METHODS IN SOFTWARE ENGINEERING EDUCATION

Collaboration has become an important subject of in education [13] and has been established as an essential part of 21st century skills [6]. In a collaborative learning environment, students can experience new approaches to thinking from their peers and can obtain a clearer perspective of a topic by expressing their understanding [11]. In a study by Chen et al. [7] computer supported collaborative learning tools were applied to a collaborative classroom setting, where both face to face and electronic interactions occurred, the tools used enhanced classroom communication across different groups. The intertwining of online and face-to-face collaboration was shown to unify and strengthen the collaborative learning experience [7]. However, care must be taken in applying the CSCL (computer supported collaborative learning) tools in classroom, taking into account existing social structures and pedagogical approaches [3].

Collaboration in general has been shown to be beneficial to learning software engineering [25] and that it improves both the motivation and student performance in learning programming [19]. The use of computer supported collaborative learning tools specifically in software engineering education courses is also established and it has been shown that students can collaborate directly using computer collaboration tools [9] with beneficial results, share information using messaging and annotation services [17] and collaborate in groups using groupware software [5] or wikis [20]. However, we found no evidence of studies concentrating on the research of how introducing CSCL tools affect inter-group interactions. This research gap could be addressed by researching how collaboration social networks in the classroom function and if there are any problem spots that could be addressed with the introduction of CSCL tools.

The problem of analyzing classroom interaction can be approached with social network analysis. Social network analysis (SNA) is an interdisciplinary technique for the analysis of social networks [22], where social relationships are viewed in the terms of network theory. In social network analysis communication between individual or social units are mapped into a communication matrix and then visualized in graphs. In graph theory there are different mathematical tools available, which can be used to for example estimate the relative influence of nodes in the graph or analyze the graph by the nodes' connection patterns [1, 14]. In this research case the communication patterns of different groups could be analyzed by modeling students as nodes and mapping student and intra-group communication as node edges.

This kind of analysis has been applied to collaborative learning by [27] to model collaboration in distance learning groups and was applied with additional qualitative analysis to CSCL classroom learning scenarios [18]. They found out that it is possible to apply SNA to CSCL scenarios, but Martinetz et al. [18] concluded main difficulties were related to the speed of processing. More specifically, it is difficult to gain results fast enough to provide corrective feedback during the progress of the course. Additionally, the process depends on the expertise of the researcher and it is difficult to automate.

3. COMMUNICATION PATTERNS IN INTRA-GROUP STUDENT COLLABORATION

The study about observing student communication patterns in software engineering course was carried out during three five days long collaborative courses. At the start of the course the students were first divided into work groups and were assigned a programming task with a deadline, but were allowed rather freely to decide how to achieve their goal. While the students worked towards their goal, their inter-group communication was observed both by a researcher and by automated tools, which was later analyzed with SNA analysis methods.

The Code Camp style intensive courses last five days, with the course structure clarified in the Figure 1. The course starts with a technical introduction and task assignment in a traditional lecturing environment. Usually at this point no groups are formed, though there is some informal chatter between some students and some study groups have been known to agree to form a group. After the first day's lectures and the task assignment, a social event is held where the students are allowed to get to know each other more and form groups for the next day. The following days concentrate on actual implementation work and during the last day there are student presentations and a friendly competition between the project works.

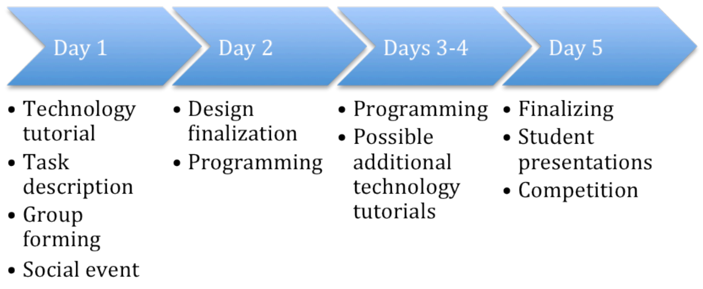


Figure 1. Code Camp course outline

During the course most of the information was first made available through a lecture, but the course had a supporting wiki page for distributing and coordinating information. Course material was made available there, including lecture slides, software tools and schedules. All students had edit access for the wiki and there was a course requirement of each team creating a wiki page for group coordination. From the student group's point of view these wiki pages were used as communication hubs, with team members updating their project status and uploading course deliverables there. The wiki had no access restrictions; so all participants of the course were free to examine all available materials.

There were two levels of support during the course: Collaborative and teacher-provided. During the course, the students and the groups were allowed to freely communicate with each other and share code to solutions with face-to-face communication or with online tools. In addition to being present in the classroom constantly during the course, the teachers entered most often asked questions into the Q&A wiki page, linked to code examples and uploaded course materials like lecture slides.

3.1 Research Setup

The observation of the communication patterns was performed by three different methods: Direct observation of the students by having one researcher present in the classroom and observing the conversations, time lapse video monitoring of the groups interacting in the classroom and recording wiki activity on the web server. After the course had ended, both the students and the teaching personnel were asked to fill a survey about their usage of resources to gain additional data to supplement the direct observations. Additionally in the second and third observed courses the teams were interviewed using a qualitative approach in order to gain better insight to the interactions and to try to detect interactions that were considered but were not taken. The questions that were asked in the interviews were:

- What went well?
- What didn't go well?
- How did you cooperate with other teams?
- Were there any reasons why you were not able to collaborate with the other teams?

The interviews were performed using a semi-structured format. When the teams answered to the questions, further questions based on the teams' answers were presented, especially if they mentioned any factor that affected teamwork or communication positively or negatively. After the graph visualization was completed, the interviews were studied in order to gain perspective how each individual group perceived the effect of communication on their work. Additionally, any factors that affected communication, intergroup cooperation or teamwork were noted and collated.

The main source for the interaction data was video monitoring. The classrooms were monitored with two web cameras that covered the observed groups and captured single images at a pace of an image per second, or six images per second when motion detected in the room. When these single images are converted into video frames they form a time-lapse video that allows the entire five day event to be viewed at a rapid speed. The following data was recorded from the video for further analysis:

- Time of communication
- Type of event (lecture, active group work, break)
- Initiating group member
- Receiving group (or teacher)

The following additional data was gathered with the surveys:

- Group the student most often preferred to collaborate with
- How often the student asked the assistant for help
- How often the student used online resources
- How often the student collaborated with other teams

After each course the videos were first analyzed and the interactions recorded and then collated. The surveys and researcher's observation notes were used to set context for each interaction. After collating the interactions graph theory and social network graphing software are used to form social network interaction graphs of the physical interactions occurring in the classroom.

3.2 Methods of Analysis

The patterns of collaboration were analyzed by modeling the communication patterns with the help of graph theory. Each interaction, the interaction context and reason for the interaction were recorded from the available raw material. The lists of interactions were collated into a directed graph, where the nodes represent individual students and the edges represent communications between the nodes. The graph was analyzed by inputting it into the graph analysis software Gephi [4] and using the visualizations produced by the software to identify influential nodes, strength of cooperation between groups and repeating patterns of collaboration between the nodes. The analysis software uses the Force-Atlas algorithm to map the nodes, taking into consideration the relative importance (size) and connection strengths of the nodes [10].

In order to gain more understanding into the arrangements of the nodes into the graph, influence analysis was performed on the graphs using the PageRank indexing algorithm [23], which can also be used to measure influence of nodes in social networks [12, 28]. For example the influence of Twitter users has been researched with this SNA approach [15, 29]. The algorithm calculates a probability distribution for arriving to a specific page in a graph representing the links of a set of hypertext documents. The values returned by the algorithm are normalized so that the sum of values is 1. An essential part of the PageRank algorithm is the dampening factor, which causes pages that have few links to important pages to be valued more highly than pages that have a wider array of more random links [23].

The PageRank algorithm is expressed in the Equation 1, where the $PR(A)$ is the PageRank of Page A, $PR(Ti)$ is the PageRank of pages Ti which link to page A, $C(Ti)$ is the number of outbound links on page Ti and d is a dampening factor which can be set between 0 and 1, but is usually set to 0.85. PageRank or $PR(A)$ can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web.

$$PR(A) = (1 - d) + d\,(PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn))$$

Equation 1. The PageRank algorithm. [23]

3.3 Observed Communication Patterns

Three courses were observed during a one-year period, with a total of 42 students spread across fifteen student groups. The studied courses were optional and attracted students with varying backgrounds from different majors, but are generally master's level students.

In the first course eight groups were formed at the start of the course, seven of which completed the course and were selected for analysis for a total of twenty students. 44% of the participating students had attended an Code Camp –style events before this course. One group was left out from the study, because the members left the course before the end of the second day and did not participate in the activities.

Students actively collaborated both inside their groups and between the groups during the observed courses. The groups formed communication patterns that were modeled into graphs and are presented later in this section. Each node represents a student and the edges connecting the nodes represent collaboration-related communications that occur between each individual student. The thicker the connecting line in the figure, the more often the group member participated in the communication, with the relative strength of the communication rated from one to three. For example, in the Figure 2 the connection between D1 and D3 is level three, D3 and A1 connection level two and the D1 – F1 connection level one. The relative sizes of the nodes are based on the PageRank algorithm, which is a measure of a node's influence in a graph. The colors and the sizes of the nodes, from blue to red and small to large, indicate the nodes' relative PageRank values.

All of the seven student groups who completed the course were analyzed further. The communication patterns for the groups in the first course are presented in the Figure 2. An alphabet depicts each group and a number each group member. For example D3 means member 3 in the group D. The different student groups formed communication clusters, with certain groups having stronger communication ties with each other. It can be seen that the nodes D1, D3, A3 and C3 forms an especially important communication center, relaying often information between the nodes' own group and two other groups. This correlates with the observer's observation that the D group was influential and several other members visited the table, asking for programming advice. The student D3 also visited his friends' tables, relaying information about recent solutions. The other noticeable pattern is that one student is particularly active in communicating with other groups and relaying the information to their own group, as in the case of nodes F2 and B1. It can also be seen that some groups overall form communication hubs, becoming central to the graph. The groups A and D are influential in the classroom communication social network, with the E and C group acting as intermediaries for information.

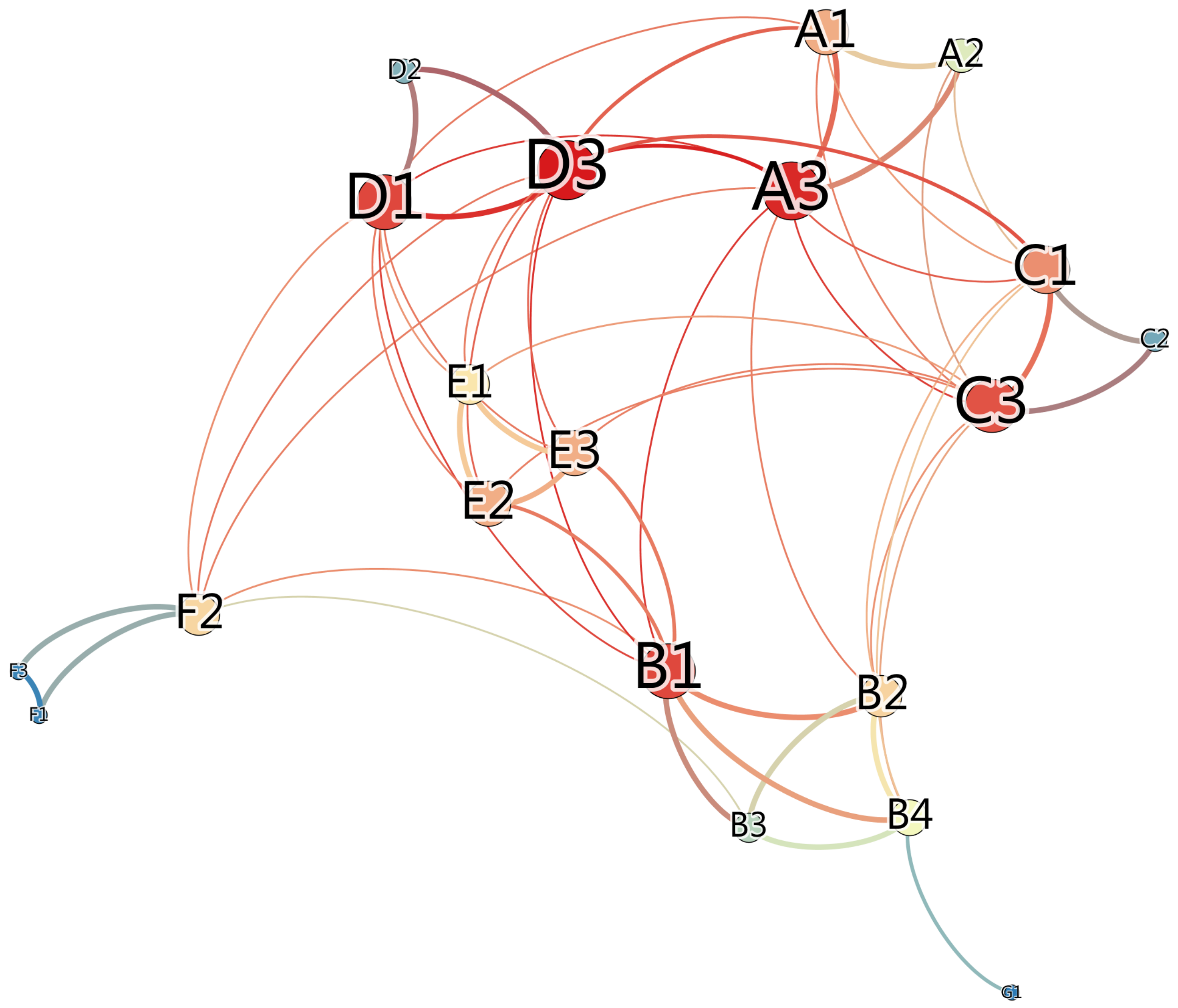


Figure 2. Student group communication graph, first course

The participants were polled for use of other communication resources, in addition to observing their classroom communication patterns. The communication frequencies are presented in the Table 1. The communication frequencies are collated to three levels: Rarely, daily or several times per day. Similarly the course success is divided to three tiers: Top third, the middle tier and last third. It can be seen that some of the most influential groups communicated a lot with the assistant and from observing the classroom it could be seen that often the information given by the assistant was spread to other students through physical communications that followed the presented communication network. All groups used online resources often in addition to communicating with each other.

Table 1. Group communication frequencies, first course

Group	Asked for help from the assistants	Used online materials	Communicated for ideas or technical from other groups	Course success tier
A	Several times per day	Several times per day	Several times per day	2nd best
B	Daily	Several times per day	Daily	2nd best
C	Rarely, once or twice	Several times per day	Several times per day	2nd best
D	Several times per day	Several times per day	Several times per day	1st
E	Rarely, once or twice	Several times per day	Several times per day	2nd best
F	Rarely, once or twice	Daily	Daily	2nd best
G	Several times per day	Several times per day	Several times per day	3rd best

The second course had five student groups, four of which were selected for observation for a total of sixteen students. 24% of the participating students had attended a Code Camp –style event before this course. One student group was excluded because they only partly attended the event. In the second course the communication graph, presented in Figure 3, forms a circle instead of a set of clusters. Two major groups, the groups A and C, form a tight and influential cluster, which mostly connect to two separate groups. Again the tight cluster correlates with tight sociability: In the social event the groups got to know each other better and the students A3, A2, C2 and C4 spent a good part of discussing during the social event. However, when inspected from the perspective of utility, the collaborations are not as effective as they could be: The groups C and D both worked on the same mobile platform and collaborated little, while the groups A and C collaborated on completely different concepts and platform. Similarly, the groups B and C worked on similar problems and only some of the students collaborated. As in the previous course, some groups, A and C in this case, form communication cluster and in some groups there is a distinct person, like D1 or B4, who communicates with this influential cluster.

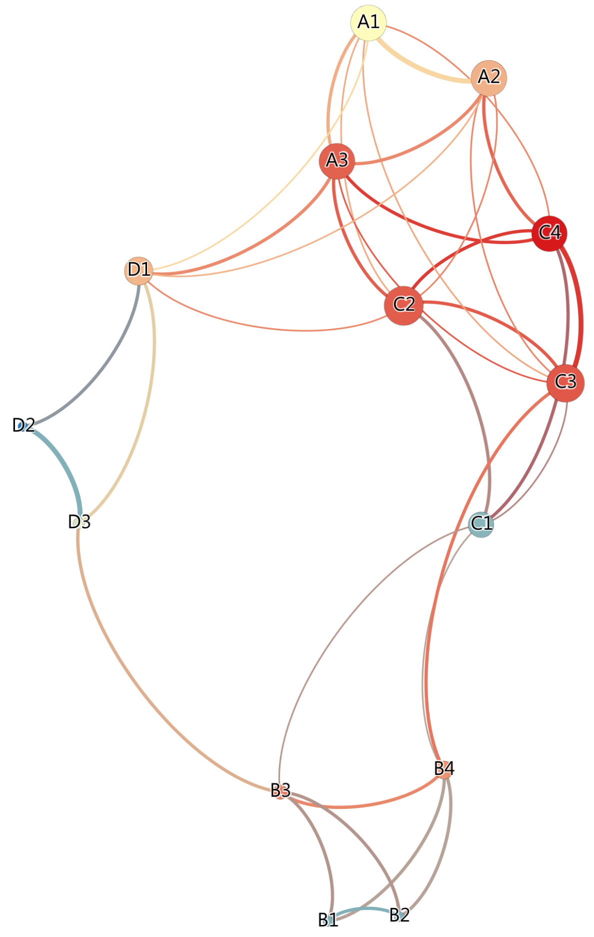


Figure 3. Student group communication graph, second course

The students participating for the second course were also polled. The communication frequencies for the second course are presented in the Table 2. The course success was very similar for all participating groups in this instance, with difficulties in finding clear differences between groups. Unlike in a previous course, the communication between groups was less frequent and one influential group reported that they received ideas and technical help from other groups less often. The other influential group C repeated the pattern observed in the first course and communicated often with the course assistant, sharing technical help with the other groups. Of all the groups, the groups B and C were physically closest to each other.

Table 2. Group communication frequencies, second course

Group	Asked for help from the assistants	Used online materials	Communicated for ideas or technical from other groups	Course success tier
A	Daily	Several times per day	Rarely, once or twice	2nd best

B	Rarely, once or twice	Several times per day	Daily	3rd best
C	Daily	Several times per day	Daily	3rd best
D	Rarely, once or twice	Several times per day	Rarely	1st

In addition to asking for numerical answers, the poll was expanded with optional textual answers asking about most important communication factors that helped the entire group to progress. The answers included good communication amongst the team, resourceful team members, meetings, online help and resources, friendly working environment, advice and ideas from other groups and sharing information. The most commonly mentioned sentences were related to information sharing and friendly working environment or team spirit.

The student groups participating in the second course were interviewed as teams in addition to being polled. The most comments involving collaboration mentioned it being allowed to collaborate with other groups useful, but difficult to initiate. Several students mentioned that collaboration was difficult to initiate with more distant groups, because it was difficult to know what they were currently working on. They said that another issue was about knowing when to establish collaboration. Programming is mostly a quiet activity and it was difficult to tell when the person did not want to be disturbed. However, at the same time several interviewees said that they could have welcomed more requests for collaboration, but other people did not initiate them.

The third and the final studied course had fewer students, because of its position at the end of the semester and had only three student groups, all of which were selected for observation for a total of seven students. 80% of the students in this course had participated to Code Camp –style events before, with overlap from the second observed course. The communication graph that student collaboration communication forms is presented in the Figure 4. Again in here it can be seen that the most influential students form a tighter communication cluster, with one student and one group at the edges. However, some students are more left out from the tight core of communication, with only faint and occasional instances of collaboration. It should be noted that the students A2, B1 and B2 correspond to students A1, A2 and A3 in the previous code camps. It is a pattern that can also be seen repeating in other courses that are outside this study: Students tend to network and tend to collaborate with each other, whether formally in the same group or not.

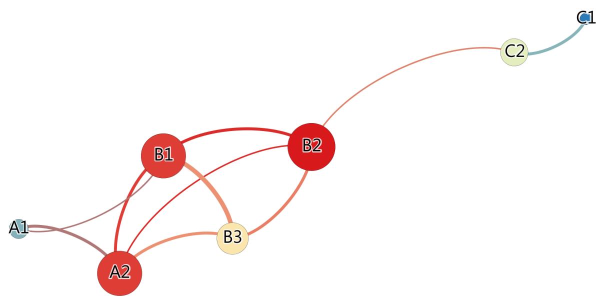


Figure 4. Student group communication graph, third course

The communication frequencies for the third course are presented in the Table 3. While the course had fewer participants than the previous ones, the differences in success were more noticeable. It can be seen from the observations that the most successful groups, A and B, used online resources often in addition to communicating with each other.

Table 3. Group communication frequencies, third course

Group	Asked for help from the assistants	Used online materials	Communicated for ideas or technical from other groups	Course success tier
A	Rarely, once or twice	Several times per day	Daily	2nd best
B	Daily	Several times per day	Daily	1st
C	Daily	Daily	Rarely, once or twice	3rd best

The textual part of the poll asked about most important communication factors that helped the entire group to progress. Student replies included team spirit, other people, shared ambition, working cooperation, version control software and online resources. Cooperation in work and team spirit were mentioned most often.

Like in the second course, students participating in the third course were team interviewed. The interview results were similar as well: The major difficulties to communication according to team interviews were the diversity of topics and the difficulty in establishing contact to people the students did not know beforehand.

Finally, the usage of communication resources was observed and polled in all three courses in aggregate. As presented in the Table 4, the overall usage of all communication resources in all of the courses was high. Most of the resources were in usage daily or more often, except for group-to-group communication in the second course. A possible explanation for this was the diversity of allowed topics, so that the groups had less to contribute to each other.

Table 4. Aggregate communication frequencies

Course	Asked for help from the assistant daily or more often	Used online materials daily or more often	Communicated with other groups daily or more often
1st	61%	100%	67%
2nd	50%	100%	33%
3rd	71%	100%	71%

3.4 Analyzing the Communication Patterns

From the different communication patterns it can be seen that the students collaborated a good amount, both within the groups and to a certain extent between the groups. There were repeating patterns discovered in the student communications. First observation is that students willingly collaborate between groups despite a wealth of available online resources and a slightly competitive atmosphere being present in the classroom. When interviewed and observed in the classroom and the social events, it was revealed that often the first or strongest collaboration occurs between people who know each other from previous social contexts. This collaboration can expand when more people see this center of activity and join in. The second most common provided reason for initiating communications was pre-established knowledge that the person might have useful information or be able to help with a problem.

The use of online resources, collaboration networks and communication frequencies between the groups differed between the three courses. The nodes in the first collaboration graph formed loosely connected clusters, while in the second course the pattern was a rough circle, with less interconnectedness but stronger collaboration in a single cluster. The probable reason for this is the difference in the task assignment: In the first course all the groups were assigned a similar task on a same platform and in the second course the groups were allowed to pick their own implementation, platform and topic within a specific theme. The same reason can explain higher communication frequency between groups in the first course. Lastly, in the third course there was a tight cluster of collaboration that was as strong as the collaboration inside of one of the groups. In interviews the provided reason for this collaboration was existing friendships between many of the collaborators from different groups. It appears that a certain minimum amount of groups is required in order to have varied collaboration between groups. One possible pattern that did not occur in the study is one in which two strong competing centers of cooperation form, with the groups essentially split in two separate blocs.

In all of these courses one or two core groups of collaborators were discovered. In the first course they were groups A and D, in the second course groups A and C and in the third course groups A and B. These groups are characterized by a certain enthusiasm, willingness to communicate and were well graded and placed well at the competition that was held at the end of the courses. The groups that communicated often with each other also used other communication resources, like online materials both on the course pages and other web materials. Other groups benefited from interactions with them, since the groups' willingness to communicate extended to sharing information they had discovered. However, not all groups were equally connected to these strong centers of collaboration and did not equally benefit from it.

4. DISCUSSION ON CLASSROOM COLLABORATION COMMUNICATION PATTERNS

Our study showed that overall the communication patterns appear to follow pre-existing, physical social networks that were established outside the context of the course. The question is whether this is an optimal case and whether the classroom or online environments can be changed to encourage forming new, beneficial social connections. One of the major issues that were mentioned in the team interviews was the difficulty of knowing whether it is appropriate or beneficial to establish communications. This is the major reason that inter-group collaboration follows pre-existing social connections, because the barrier to establish communication is less difficult. Informal communication channels between friends also provide more information about group project work status, which makes it easier to initiate collaboration.

The analysis was performed using three different approaches, which are video and direct observation, interviews and polls. Results from the different sources were used to complement each other. For example, direct observation and interviews provided context data while the video observation covered the event completely from start to finish for most accurate graphing data. Student participation in the data gathering was high, with every observed group participating in the interviews in the last two courses and 95% of the students completing the online surveys. With the high percentage of data collection and accurate observation records from the entire duration of the events it can be said that the structure of the graphs are an accurate abstraction of what occurs in the classroom. The study would have been more conclusive if it had included more courses in a wider range of universities. However, the studied communication channels in the three observed courses already provided notable patterns of communications for analysis. Also, some of the patterns repeated during each of the courses, suggesting towards them being common patterns to Code Camp –style courses in environments that are similar to the study environment.

The main pattern that repeated in all of the courses was a strong center of collaboration that formed around one to three groups. The fact of strong center of collaboration was contrasted with the interview reports of difficulties in matchmaking when looking outside one's group. This suggests that not all groups find or are not able to work with the strong center of collaboration and that collaboration does not equally benefit all participating groups.

We propose that the discovered issues can be addressed by offering computer supported collaborative tools that support wide student-initiated collaboration. A common issue during the courses was that the students did not realize that they had similar problems, which caused hesitation in initiating communication. Software tools can be used to publicize commonly encountered problems and to find people who are struggling with the same problems for collaboration. Tools like these could help students find each other without spending time on discovering partners and accidentally disturbing people who are concentrating on individual problem solving. An additional benefit would be that the problem and the following conversation would be recorded for other participants to view later in the course if they struggle with a similar problem. For example question and answer sites with reward systems have seen wide use in the field and could be also applied inside classroom. Additional tools, like projectors or mobile clients, could be used to publish unanswered questions and the most useful solutions.

Traditional online courseware tools are now commonly seeing use in classroom environments, but their usage focus is often to provide course literature, assignments and accept returns. While they do allow things like peer review of assignments, this style of collaboration is teacher controlled and usually more slowly paced. This study shows that the patterns of collaboration in the Code Camp style of courses could be improved and that additional tools should be provided to encourage student-initiated collaboration. Improved software tools could give more opportunities for collaboration within classrooms by easing sharing, providing matchmaking services and should be investigated in future research. There are several examples of web-based collaborative tools widely used in the industry. One is Github, which enables direct collaboration between several people by allowing several people to edit and version software source code concurrently. Another is Stack Overflow, which facilitates problem solving by allowing people to post questions and rate and reward the best answers with reputation points.

Computer-based communication tools do require the presence of computers in the learning environments and this could a drawback in adopting the tools. However, in software engineering courses computers are already present as development tools and using computer-based collaboration tools will most likely have a lower barrier for adoption than in other fields of education. Also, software engineering students are already familiar with using online tools for working. There is a possibility that their current development or planning tools do not have to be replaced, just extended. There are plugins or workflows that support collaboration and collaborative learning both within and between groups and these tools can be introduced into the course workflow and be pre-installed into the working environments.

5. CONCLUSION

In this study we applied social network analysis to intensive collaborative software engineering courses using recordings, polls and interviews as source material. We presented and analyzed the communication collaboration patterns that form during intensive collaborative software engineering courses. It was found out that the students do collaborate outside their groups on problems, but the patterns of collaboration follow pre-established social connections and not all groups equally benefit from the collaboration. The main method of collaboration was seeking out these social connections, like well-known classmates or friends and discussing with them, whether they were working on the same problem or not. The main result of this study is discovering the form of communication patterns that are established during the courses. These patterns and discovered issues in matchmaking can provide the basis for designing CSCL tools to improve collaboration. Additionally the results can be used to validate and compare improvements to communication patterns when applying social network analysis to future courses that use CSCL tools.

While students mostly collaborate along pre-existing social connections, almost all of the groups in the observed courses used a major amount of online resources and used computers for planning from the start. Our proposed solution for improving collaboration in these already computer-supported work processes is introducing of online collaboration tools and groupware solutions to that are already well established elsewhere in the industry, instead of the more often used classroom online tools. These online collaborative tools, such as Stack Overflow or Github, can fit the fluid nature of the event better. The classroom tools are often aimed for delivering preplanned course material according to a curriculum.

During the courses students mainly communicated face to face, but because of privacy and monitoring issues there was no possibility to gain detailed information about the students' electronic communications. However, in further social network analysis research where a centralized CSCL tool is designed and used for comparison, these limitations can be overcome by collecting statistics from the collaborative tools.

6. REFERENCES

[1] Abraham, A. and Hassanien, A.E. *Computational Social Network Analysis: Trends, Tools and Research Advances*. Springer, 2010.

[2] Alaoutinen, S. et al. Experiences of learning styles in an intensive collaborative course. *International Journal of Technology and Design Education*. 22, 1 (2012), 25–49.

[3] Baker, M. et al. Integrating computer-supported collaborative learning into the classroom: the anatomy of a failure. *Journal of Computer Assisted Learning*. 28, 2 (2012), 161–176.

[4] Bastian, M. et al. Gephi: An open source software for exploring and manipulating networks. *International AAAI conference on weblogs and social media* (2009).

[5] Bravo, C. et al. Integrating educational tools for collaborative Computer Programming learning. *Journal of Universal Computer Science*. 11, 9 (2005), 1505–1517.

[6] Bruns, A. Produsage. *Proceedings of the 6th ACM SIGCHI conference on Creativity &amp; cognition* (New York, NY, USA, 2007), 99–106.

[7] Chen, W. et al. What do students do in a F2F CSCL classroom? The optimization of multiple communications modes. *Computers & Education*. 55, 3 (Nov. 2010), 1159–1170.

[8] Davies, W.M. Intensive teaching formats: A review. *Issues in Educational Research*. 16, 1 (2006), 1–20.

[9] Duque, R. and Bravo, C. Analyzing Work Productivity and Program Quality in Collaborative Programming. *The Third International Conference on Software Engineering Advances, 2008. ICSEA '08* (2008), 270–276.

[10] Force-Atlas Graph Layout Algorithm: URL *http://www.medialab.sciences-po.fr/publications/Jacomy_Heymann_Venturini-Force_Atlas2.pdf* (2012). Accessed 12 June 2013.

[11] Gillies, R.M. Teachers' and students' verbal behaviours during cooperative and small-group learning. *British Journal of Educational Psychology*. 76, 2 (2006), 271–287.

[12] Java, A. et al. Modeling the spread of influence on the blogosphere. *Proceedings of the 15th international world wide web conference* (2006), 22–26.

[13] Johnson, D.W. and Johnson, R.T. Learning Together and Alone: Overview and Meta‐analysis. *Asia Pacific Journal of Education*. 22, 1 (2002), 95–105.

[14] Knoke, D. et al. *Social network analysis*. Sage Publications, second edition, 2007.

[15] Kwak, H. et al. What is Twitter, a social network or a news media? *Proceedings of the 19th international conference on World wide web* (New York, NY, USA, 2010), 591–600.

[16] Laat, M. de et al. Investigating patterns of interaction in networked learning and computer-supported collaborative learning: A role for Social Network Analysis. *International Journal of Computer-Supported Collaborative Learning*. 2, 1 (Mar. 2007), 87–103.

[17] Lan, Y.-F. and Jiang, Y.-C. Using Instant Messaging and Annotation Services to Improve Undergraduate Programming Courses in Web-Based Collaborative Learning. *Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09* (2009), 236–241.

[18] Martínez, A. et al. Studying participation networks in collaboration using mixed methods. *International Journal of Computer-Supported Collaborative Learning*. 1, 3 (Sep. 2006), 383–408.

[19] McDowell, C. et al. The effects of pair-programming on performance in an introductory programming course. *SIGCSE Bull.* 34, 1 (Feb. 2002), 38–42.

[20] Minocha, S. and Thomas, P.G. Collaborative Learning in a Wiki Environment: Experiences from a software engineering course. *New Review of Hypermedia and Multimedia*. 13, 2 (Dec. 2007), 187–209.

[21] Okamoto, T. Collaborative technology and new e-pedagogy. *IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings* (Sep. 2004), 1046 – 1047.

[22] Otte, E. and Rousseau, R. Social network analysis: a powerful strategy, also for the information sciences. *Journal of Information Science*. 28, 6 (Dec. 2002), 441–453.

[23] Page, L. et al. The PageRank citation ranking: bringing order to the web. 1999.

[24] Pais Marden, M. and Herrington, J. Supporting interaction and collaboration in the language classroom through computer mediated communication. *World Conference on Educational Multimedia, Hypermedia and Telecommunications* (2011), 1161-1168.

[25] Porras, J. et al. Better programming skills through Code Camp approach. *16th EAEEIE Annual Conference on Innovation in Education for Electrical and Information Engineering, Lappeenranta* (2005), 6–8.

[26] Porras, J. et al. Code camp: a setting for collaborative learning of programming. *Adv. Technol. Learn.* 4, 1 (Jan. 2007), 43–52.

[27] Reffay, C. and Chanier, T. Social Network Analysis used for modelling collaboration in distance learning groups. *Intelligent Tutoring Systems* (2002), 31–40.

[28] Scott, J. *Social network analysis*. SAGE Publications, third edition, 2012.

[29] Weng, J. et al. TwitterRank: finding topic-sensitive influential twitterers. *Proceedings of the third ACM international conference on Web search and data mining* (New York, NY, USA, 2010), 261–270.

Pedagogy of 1:1 Computing in Colombia: A Case Study of Three Rural Schools

David Silva
Stockholm University, DSV
Department of Computer and Systems Sciences
Forum 100, 16440, Kista, Sweden
maildavidsilva@gmail.com

Matti Tedre
Stockholm University, DSV
Department of Computer and Systems Sciences
Forum 100, 16440, Kista, Sweden
first.last@acm.org

Mikko Apiola
University of Helsinki
Department of Computer Science
P.O. Box 33, University of Helsinki, Finland
mikko.apiola@helsinki.fi

ABSTRACT

The utilization of one-to-one computing (each students is equipped with a personal device) has slowly started to become available into educational contexts in developing countries. One crucial challenge in relation to the understanding and developing of one-to-one computing related learning and teaching practices in developing educational contexts is the lack of context-situated educational research on the topic. This study participated in addressing the lacks in educational research by exploring the pedagogical strategies utilized in three rural schools in Colombia. The results consist of rich descriptions of teachers' pedagogical approaches. The results show high empowerment in teachers' efforts in developing and contextualizing one-to-one related pedagogical approaches into their teaching. Many of the teachers' approaches were found to be well aligned with a number of constructivist and student-centered pedagogical approaches. This study adds important new educational results to the ongoing scientific discussion about education with one-to-one computing

Categories and Subject Descriptors

K.3.1 [**Computer Uses in Education**]: Computer-assisted instruction (CAI), Collaborative learning

General Terms

Experimentation, Human Factors

Keywords

1:1 computing, ICT4D, TEL, OLPC, Escuela Nueva, Pedagogy

Acknowledgments

This paper is based on the M.Sc. thesis "*Exploring educational and pedagogical strategies of one-to-one computing in Colombia: A case study of three rural schools in Colombia*," Stockholm University, Department of Computer and Systems Sciences (DSV), by David Silva.

Koli Calling '13, November 14–17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526988

1. INTRODUCTION

The use of information and communication technology (ICT) in the classroom has recently become an everyday practice, especially in the developed world. One approach to utilizing ICT in education is one-to-one computing (each student is equipped with a personal laptop). One-to-one computing has become common in many developed countries, and has recently been increasingly advocated for many developing countries, too [18, 27]. However, fruitfully embedding one-to-one computing in the classroom has not always been straightforward—neither in a developed country context nor in a developing country context [36].

One crucial challenge of one-to-one computing in learning environments in developing contexts is the paucity of context-situated educational research on the topic. Education is known to be very context-dependent [38]. Although there are case studies and technical studies on educational technology in various developing countries, the learning and teaching side is often left with less attention—which is a common line of critique in both developing and developed country contexts [8, 22, 34, 37]. There exists a low number of research studies, in developing country contexts, on the pedagogical strategies that successful educators apply in their one-to-one projects. Thus, there is a serious lack in basic as well as applied education research, which both are needed to better understand different educational contexts and to develop the 1:1 related learning and teaching praxis (cf., [21]).

This research study addressed that gap in knowledge by exploring, in three rural schools in Colombia, the pedagogical strategies adopted by experienced teachers who utilize one-to-one computing as a learning tool in their regular classroom teaching. That exploratory aim was approached through two concrete objectives: 1) to identify teachers' pedagogical approaches, and 2) to understand the reasons for teachers' choices of those pedagogical approaches. The research questions for this study were:

- RQ_1 : What pedagogical approaches do teachers use with 1:1 computing in three rural schools in Colombia?
- RQ_2 : What are the teachers' reasons for adopting those pedagogical approaches?

2. RELATED RESEARCH

One-to-one Computing and Pedagogy

One-to-one computing refers to the idea of equipping each student with a personal computer. The idea of technology enhanced education dates back to the beginning of modern computing and has been actively used in school projects since the 1960s. The research project ACOT, Apple Classrooms Of Tomorrow, was one of the first attempts to research and propose teaching strategies with computers in education [1]. Microsoft in turn, implemented a laptop program, the Anytime Anywhere program, in the United States.

There is quite some research on one-to-one computing in industrialized countries, and that research branch is still growing [28]. Inan and Lowther [19] studied 1:1 computing in 195 schools in Michigan, U.S. The results showed some changes in learning paradigms. Another study, conducted in the UK, identified two common use patterns for ICT in the classroom: supporting learning of the topic, and using the computer as a tool for presenting student work [6]. Yet another study, from Sweden, indicated that teachers' willingness to utilize laptops depended on the way they plan their teaching [29]. A number of 1:1 related success factors have been identified [2], [14, pp.21–22].

The One Laptop Per Child (OLPC) Foundation was established in 2005 and it used to be best known for the '$100 laptop project'. OLPC Foundation is a non-profit organization with the aim of providing laptops to pupils in every part of the world [34]. The organization focuses specifically on providing computers to students from disadvantaged socioeconomic backgrounds [12]. The notable technical characteristics of the XO laptops include their screens' readability in sunlight, robust construction, low power consumption, use of mesh networks for Internet access, and low production price [13].

The OLPC foundation's pedagogical position is heavily influenced by the views of the South African-born computing and education pioneer Seymour Papert, who stresses learning by exploring, independent thinking, and through playin (e.g., [18]). One-to-one initiatives are often paired with a discourse of other educational reforms, which advocate student-centered, project-based, problem-based, collaborative, and creative learning [36].

Those views of learning follow a long tradition in education. After the decades-long paradigm shift in learning from behaviorism to constructivism, a number of student-centered pedagogical approaches have emerged. Among those pedagogical approaches are Problem Based Learning (PBL), Project Based Learning, and Progressive Inquiry (e.g., [4, 15, 17, 20]). All those three pedagogical approaches follow the constructivist views of learning. They advocate the use of realistic, open-ended projects as learning tasks, and they see the teacher's role updated: the teacher's role is to act as a coach and a facilitator of the learning process in contrast to giving direct instruction as in behavioristic learning.

"Escuela Nueva" Model of Colombia

In the past few decades, Colombia has aimed at reforming its rural primary school pedagogy [23]. The reform is named *Escuela Nueva* ("new school"), and it has been adopted in Colombia as well as in other Latin American countries [5]). The reform has received positive evaluations [32].

One evaluator of the Escuela Nueva reform listed four essential characteristics of the reform [23]. First, all teachers undertake adequate preparation to lead a multi-grade classroom, to utilize new learning materials, and to support both individual and co-operative learning. Second, instructional materials, textbooks, and teacher guides are provided to schools to support self-guided learning. The students' textbooks are designed to further support self-directed learning at their own pace, without sustained teacher supervision. Third, teachers are given the chance to share knowledge with colleagues from other schools. Fourth, it is stressed that students must have an active role in the learning process, where they work both independently and in small groups, and using creativity to achieve their learning goals.

The learning activities in the Escuela Nueva pedagogy often take the form of context-based projects, relating to the rural and agricultural surroundings including indoor and outdoor activities [23]. Students are also encouraged to be involved in politics and school management. One of the goals of the project is also improvement of reading and writing skills, and to have a close relationship with the surrounding community [33].

Table 1: Case Studies in Developing Countries on One-to-One Pedagogy

Pedagogical Focus	*Location*	*Sources*
Self-determination theory, constructivism	Tanzania	[3]
Storytelling, student-centered learning	Tanzania	[10]
Project-oriented, self-determined learning	Ethiopia	[16]
Formal and informal learning	Sri Lanka	[24, 25]
Analysing the conversion from traditional teaching to one-to-one using activity theory	Ethiopia	[18]

Case Studies in Developing Countries

There are a variety of case studies on one-to-one computing in developing countries, described in recent research literature. A number of those case studies also discuss pedagogical issues: Table 1 lists the pedagogical elements and countries of those studies.

The first case study in Table 1, conducted in Tanzania, was performed through a workshop approach, where seventh grade students studied general health care topics [3]. The researchers developed the workshop to have a context-conscious approach, together with a combination of modern educational and pedagogical theories such as intrinsic motivation, constructionism, constructivism, and explorative learning. The workshop started with 32 lessons of ICT introduction and self-exploration, after which the teachers designed the lessons to increase in difficulty and self-directed learning. Constructivism was further approached by group problem solving, role-playing, dialogic teaching, and by use of experimental learning principles. The study showed positive educational results and increased student motivation. Instead of presenting a general pedagogical method the study proposed various best practices with the XO laptop in education.

Of the studies in Table 1, another especially relevant study was done on the traditional pedagogy of Ethiopia [18]. The study used Engeström's extended model of activity systems [11] to study the shift from traditional Ethiopian classroom education to a one-to-one learning mode. The initiative used XO laptops together with a special software package, the Melepo software, which was designed to "*...transmit digitized Ethiopian school books in an interactive one-to-one format*" [18]. The study proposed that in order to fundamentally change the education to a constructivist one-to-one model, it is essential to consider the situational context, such as social and cultural elements [18]. The Melepo software succeeded to circumvent some of those issues, and it was argued that it would be beneficial to further emphasize working with the tool [16]. The authors also clarified that the current textbooks are not adequate to help teachers and students take advantage of one-to-one computing education [18].

A more recent study of six rural schools in Sri Lanka, where XO's were used, found that local language content could shift the pedagogy towards student-centered approach if more local-language content is developed [24]. The initiative was challenged due to many computer breakdowns, which forced the teachers to re-organize their teaching so that students shared computers. Teachers pointed out, that a well functioning internet connection is very important in order to be able to utilize learning materials over the internet.

3. METHODOLOGY

The aim of this study, exploration of actual teaching approaches, required rich and detailed information on a limited number of cases. The methodology literature recommends case studies as one effective approach for that type of research [7]. The cases were three schools in rural Colombia. As the objectives required in-depth information on teachers' approaches and the reasons behind them, semi-structured interviews and classroom observation were selected as methods for data collection. The method choice was similar to the method choice in related studies [3, 10, 24, 18].

Data Collection.

The data collection was performed between February and March, 2013. This study utilized qualitative data collection and analysis methods following Creswell's and Denscombe's guidelines for qualitative research [7, 9]. Semi-structured interviews were conducted in group and individual modes, and observations of two classrooms were done in the form of researcher as a silent observer, which as a method of observation is not highly obtrusive. A total of ten teachers participated in this study, out of which nine were interviewed and two were observed during one day each while teaching using XO and Classmate laptops.

Subjects.

The participants in this study worked as teachers either in the primary or secondary level, where they integrated 1:1 computing in their teaching of primary education, informatics education, biology education, and agricultural education. The teachers were selected through chain-referral sampling, which is appropriate for the exploratory nature of this study. The initial sample started by contacting the OLPC Foundation, which provided contact information to OLPC-equipped schools. The first and following schools were contacted through e-mail and phone, which resulted in meetings with the school principals who, through their coordinators, selected teachers that taught with the laptops.

Research Materials.

Audio recordings of the interviews were recorded using a mobile sound recorder. Field notes were taken throughout the research period, and they were supplemented with photos of the classrooms and schools. The digital analysis tool Dedoose was used for data analysis.

Data Analysis.

Translation of the material was done by the authors. For analysis, the study followed Creswell's data analysis spiral [7] as a generic guideline for stages of qualitative analysis [31]. The analysis deviated from Creswell's spiral at the 'account' level of the spiral. At that stage the results of each case were instead analyzed for their correlation with findings of related studies and through triangulation of findings between interviews and observations.

Research Ethics.

Informed consent was taken from each of the participants, and all participants were kept anonymous through the use of pseudonyms. Although this study is limited to the teachers, students, who were minors, were indirectly involved. Hence, special care was taken to conceal any and all information about students to protect their anonymity.

4. RESULTS

This study investigated teachers' pedagogical approaches in three schools: Fundación Formemos, Santa María del Rio, and Normal Santa Teresita. The schools are located in rural areas near Bogotá, the capital city of Colombia. The students in the three schools use either XO laptops (OLPC Foundation's 'hundred dollar laptop') or Intel's Classmate laptops in their classrooms as part of their education.

Santa María del Rio

Santa María del Rio school is a public school, which has pupils from primary up to high school levels. Santa María del Rio is located in the rural area of Chía, one hour drive from capital Bogotá. Pupils of Santa María del Rio come mostly from low-income homes, although the school is surrounded by exclusive rural high schools. The school received 215 XO-1 laptops and an unknown amount of Classmate computers in 2009, but due to the lack of sustainability of the initiative, in 2013 the school only had 40 working computers left.

Fundación Formemos

Fundación Formemos is located in the rural highlands of the municipality Tena. The school has a strong focus on teaching sustainable agriculture and providing children of internal refugees with education possibilities. It is both a boarding school and a day school with around 200 students. The school is state-funded, and the ages of students are between children of six years and young adults of 16–19 years. An estimated number of 117 XO-1 laptops were given to the school in 2009 (the exact number was not known). Currently the one-to-one initiative lacks continuity and the school has to deal with many technical problems.

Normal Santa Teresita

The school Normal de Santa Teresita is surrounded by the rural highlands of the municipality Quetame near the department Meta. The school is located in the center of the small town Puente Quetame, which is currently being rebuilt due to earthquakes. Normal de Santa Teresita is a fairly small school with 200 students. The school provides education to children in all ages up to high school, and also offers basic teacher education. The school was given 250 XO-1 laptops in 2009. Like the other schools, because of the lack of sustainability and technical support of the project, the school currently only has 26 working XO laptops.

4.1 Results - Santa María del Rio

The school's classrooms were well lit, and the walls decorated with student function maps such as classroom representatives, curriculum maps, and upcoming student responsibilities of, for instance, classroom maintenance. The fifth grade teacher sat between the children and the classroom whiteboard with a small bookshelf of educational material to the side of her desk.

Students shared the XO-1 computers in groups of three and sat in groups close to the walls because that was where the power sockets in the room were located. The school previously used the school's computer labs to study informatics, but since receiving 215 XO laptops in 2008, the school has used the laptops in all areas of the curriculum. The school does not always have a functioning wireless connection, and the connection was out of order during this study, too.

The school pedagogy encouraged a project-based approach, in which teachers bring forward a theme to students, from which students can develop projects. Students could freely choose a project based on the proposed theme and their own interests. One of the teachers explained this further: "*Depending on whether the student's proposal is within the theme, they can openly choose within range of sub-themes that are related to the knowledge that they will study.*" The purpose behind the freedom of choice of the project was to encourage the students' creativity.

The next step was to develop the students' project proposals to be in accordance to the related subjects. This was done together with the teacher and also in cooperation with teachers from different subjects. One teacher explained that the collaboration between teachers focused on integrating parts of curriculum subjects into the students' projects so that educational goals and state official education standards were fulfilled. Another teacher continued to state that the outcome of this step was to build a conceptual model of the proposed project.

The next step in the teachers' pedagogy was to aid the students to gather information around their topics, using the computers, so that the students could construct their project. One teacher said, "*...students are instructed to build something out of the information* [...] *we call this step construction.*" That teacher also helped students with their constructions in case they were having problems using the computers. Student could choose any method of construction whether using physical models, digital models, or setting up a theatrical play.

The teachers tried to ensure student activity by reviewing the student's progress often and by having active communication with them. Moreover, one teacher noted that the children controlled each other's activeness as they worked in groups. The students presented their digital projects to the teachers with slide shows using presentation software they learned in informatics education. Regarding evaluation, the school employed two different approaches: one where the teachers evaluated their strategies, and one where students evaluated their acquired knowledge and their used tools. Books were not an essential source of information to construct the student projects, instead they were used simultaneously with other material. Students had few compulsory school books as the school was state-funded for children of low-income families. Yet, whenever the school's Internet connection was down, school books were used more. In these cases, the teachers provided the students with digital school literature from their personal digital library. One of the teachers explained his material management approach:

> *I have a library with material about my subject, which I store on a memory card* [...] *The material includes whatever I need to include on the current subject. Afterwards, I distribute the material on to each of the students' computers, but obviously I do not do their entire work, I only give them enough material so they can develop their own work.*

The pedagogy has resulted in many student projects related to Chía, which is the municipality where the school is located. One of the teachers, for instance, noted that "*we have seen many student projects related to Chía's history*". Another gave an example of a project about tourist locations in Chía: "*This year, students from eleventh and twelfth grade are working on a Wikipedia project, to inform about tourist locations in Chía...*" The teacher continued to explain the construction of the Wikipedia page: "*... each week a group of students makes story cards about a location and also updates the Wikipedia page, creating a history of posts.*"

Children in the lower grades used the XO laptop's games to learn basic mathematics. One teacher explained a strategy to teach dimensions and geography with the XO laptop: "*The subject of the continents was explained by discussing that we are on a continent within a continent, that our continent belongs to America. They looked in the maps in the computer and talked about geographic dimensions. The digital maps allowed them to describing all those things.*" The same teacher continued to explain a recent digital project of airplanes types: "*... the students made models of airplanes; they also made a helicopter with a motor and presented it with lots of slides.* [The informatics teacher] *taught them another program to work with slides.*"

During the classroom observation, it was noticed that the majority of students had problems with the batteries and chargers of the computers. The teacher used much of her time helping students with the technical problems. The teacher also had to mediate student brawls over taking shifts with chargers in order to get a working power socket. Moreover, there were disagreements over whose turn it was to use the computer within the group—the education was not one student per one computer, as originally planned. Meanwhile, the teacher was receiving student reports over their progress. At one occasion, one student from a group that had difficulties finding a functioning power source got up from his chair and sneaked to a power socket that was used by another group. Once there, he changed the charger to his own on the sly. The group that had used the power source soon noticed what had happened when their laptop signaled low battery, whereupon the saboteur was quickly traced. A loud argument between the groups followed in which they blamed each other of sabotage, as well as of not sharing the power source. Similar scenarios occurred several times, especially during the first hour when the students were setting up the laptops.

4.2 Results - Fundación Formemos

Fundación Formemos has a strong focus on environmental and sustainable education. The school had adapted their pedagogy around the Escuela Nueva model. One of the teachers explained the schools pedagogical approach:

> *Well, here we work with the Escuela Nueva pedagogical model, which allows working in groups for collaborative work through guides, starting from the pre-concepts and ideas of the student. We start from there and then go to the teacher's theoretical explanation, and we leave out some questions from those explications and we work on understanding the questions through practice. It is in this practice that the XO laptops are used and integrated.*

The students worked in self-regulating groups of six where they rotated between different roles and responsibilities such as leader, time keeper, and evaluator. The classroom walls were used for information about classroom responsibilities, maps of current projects, and information about important upcoming events. Each classroom had one teacher and around twenty students. The classrooms included a small library with educational sources and reference books such as dictionaries.

The teachers built the courses and a sort of guides for students in accordance with the national educational goals. The activities were contextually formed; one teacher explained, "*The Escuela Nueva pedagogy gives students the opportunity to work with his reality and the location where they are active.*" He continued that in addition to creating the guides the teachers also searched the Internet for tutorials of activities that were suitable for their theme as very little training was given to the teachers on suitable activates to use the laptops. Another approach to create guides was to use models from the learning resource center that the Ministry of Education provides, which teachers found easy to combine with the laptops.

One of the teachers explained that students were not obligated to use the computer to complete assignments or projects: the methods depended on the theme. There again, speaking to the other teachers made it clear that the laptops were used in all areas, mostly in the lower grades. In those grades, teachers instructed their students to use the pre-installed software for drills and for practicing basic mathematics. Another teacher said that although students did independent group work, teachers instructed the children to stick to subjects-specific applications, such as mathematics applications when they are studying mathematics.

One teacher showed how he used the measurement function in the XO laptops in his agricultural classes, as his group of students studied the percentage growth of plants. Another approach was to take a series of photographs and videos to study the growth of plants. In addition, in that teacher's class the laptop cameras were also used to take pictures of unknown plants and insects that were later studied in Wikipedia. In larger projects, the teachers planned student work in cooperation with the ecology, pastoral, and education ministries because of the schools' focus on environmental agriculture. The teacher described that students completed these types of projects by presenting, in an annual workshop, photos and videos of them doing their fieldwork.

One of the interviewed teachers was newly hired, and still learning how to integrate laptops with education. She explained that she built each of her guides for students' self-directed studies in a form that allowed her to give a theoretical talk about that theme. Afterwards, students could start using the guide to find more information about the topic using the laptops. Another teacher explained a different way of using the computer when she taught the anatomy of the human body to her primary level students:

> *...pictures of the human body are downloaded first from the Wikipedia. The details of the body pictures depends on the students' ages, so with the younger student we only study the trunk, head, arms, and legs while with older students studies we add ears, nose, and mouth, while the eldest study organs and senses like touch, smell, and sight...*

That teacher concluded by explaining that the childrens' task was to add parts to the body using the paint software on the laptops.

4.3 Results - Normal Santa Teresita

As the teacher responsible for laptops explained his/her work at the school, it became clear that the institution had severe difficulties integrating the laptops to education. The reasons were mostly due to numerous technical problems with the computers: problems with chargers, batteries, and broken screens and keyboards had resulted in a low number of available computers. The technical difficulties were even more evident during a one-day classroom observation as most of the children's computers were in very bad shape. A noticeable number of students sat alongside the classroom walls, to have access to the power sockets and the majority of students had technical problems with the computers.

Despite the technical issues all of the school's primary teachers worked with the laptops in every classroom, as it fits the schools integral pedagogy. One of the teachers explained the school's pedagogy by describing a current project:

> *We do not work with specific subjects. We follow an approach called integrating knowledge, so what we do is design a bi-semester project. I am, for instance, working on a project called 'Justin is a chicken and Mary is a hen' where I include all the content of the first quarter, without having to talk about: 'this part is mathematics..., this part is civics..., or this part is humanities...' No, what we do is we integrate. In this way, Justin and Mary project is a pretext to erase all curriculum subjects.*

The children usually worked in a group of six as they performed collaborative work but could be allowed to work individually depending on the current task. However, the lack of working computers forced students to share computers and work in groups. The classroom that was observed had walls colorfully decorated with information and models from the students' current project. The classroom was equipped with a whiteboard, a desktop computer, and an electronic whiteboard. During the classroom observation one assignment for students was to draw Justin the chicken, and write a story of how and what the students had learned in a previous lesson. Several children proudly showed their drawings and stories on the computer.

The teachers created the project ideas and received support from the rest of the primary education staff to include important educational parts. However, as the project advanced, students could engage in developing the project further. The teachers started their lessons by writing a problem on the whiteboard. In the case of the example teacher above, the problem was related to her current Justin and Mary project. One of the example problems she used to connect the computer to projects in her class was to investigate the cost of maintaining the chicken. By this approach, the teacher instructed her students to approximate the costs using a spreadsheet program. The teacher explained that the chicken lived in a nearby farm, that students had made two field trips to visit them, and that children had calculated the expenses of their field trips with the spreadsheet program. The project also entailed contextual factors, such as caring about the rights of the two chicken and understanding the environmental effects of their visits to the farm.

Another common approach linked with the pedagogy of the school was to make use of graphical applications to build conceptual maps of the current project. The teacher also explained that the computers were easier to bundle with pedagogy after children had lost some interest in the laptops' computers games, as at the beginning her students used the computer to play games. One teacher told that a commonly used strategy to develop knowledge about dimensions and mathematics was to use the 'micro worlds' application and mathematical games on the computers.

During the project work, the teacher assured students' activeness by walking around between students. The teacher said that keeping the class focused was quite difficult being the only teacher in the classroom, but she was also helped by other students. During the classroom observation the teacher talked to students about their work and at the same time walked back and forth between groups, aiding them with technical issues.

The school could not require students to acquire school books for their classes, which meant that teachers had to find alternative sources of knowledge that were useful for their current projects. Similar to the teacher at Santa María del Rio, quoted earlier in this text, one teacher explained that she searched for information in school books and in the web, and in the case of the digital material she distributed the sources in .pdf format to the students' computers.

Although the laptops had wireless connection, the wireless router in the school main building did not have the capacity to reach the classroom building, where the laptops were used. However, the teachers complemented this part by bringing students to work with the Internet on the connected computers in the system laboratory.

5. DISCUSSION AND CONCLUSIONS

The three cases showed a number of important similarities and differences. First, the project-based pedagogical models were similar between the three schools, and they aimed at integrating 1:1 computing with various kinds of project work. Second, all schools employed some kinds of contextualized education in their teaching with 1:1 computers. Third, teachers employed pedagogical approaches of the constructivist kind as well as of the instructivist kind. Fourth, teachers used different strategies for managing digital learning material. Fifth, teachers all identified control over the classroom as important, and they employed various strategies for controlling the classrooms. Sixth, teachers used intriguing strategies for presenting students' work.

Similar project-based pedagogical models

The most significant common feature between the schools was their similar pedagogical approaches. First and foremost, they all used varieties of project-based learning, and shared similarities with the Escuela Nueva model. The Escuela Nueva model was, indeed, intended for rural schools with scarce economic resources [33]. The integration of the curriculum subjects in each of the three schools was also aligned with the Escuela Nueva model of integrated subjects.

There again, there were slight differences in teachers' classroom instruction. In Santa María del Rio, students were entirely free to choose any method to construct their projects. On the contrary, the teachers in the other two schools used a more strictly limited, instructional approach. Those teachers, for instance, planned their projects and regulated students' learning more through, for example, models, field trips, and laptop exercises. Another difference was how the students in Santa María del Rio followed a more self-directed learning approach as the students chose projects based on their interest, which is well suited for facilitating intrinsic motivation [35]. The only restriction in Santa María del Rio seemed to be that students were required to present their construction with digital slides. However, it is quite clear that compared to the other schools, students in Santa María del Rio were expected more metacognitive skills, self-regulated learning, and active learning, as they frequently evaluated their learning and their learning tools.

The pedagogies of the schools also shared similarities with inquiry-based learning. In Santa María del Rio, teachers helped students to conduct research prior to their construction, while students in Normal Santa Teresita solved problems with the information provided by the teacher. Inquiry-based learning was not as evident in Fundación Formemos, where teachers instructed their students with pre-designed guides. Another similarity based on the classroom observation and visits to the schools was that the classrooms had a similar classroom design. The three schools were arguably influenced by the Escuela Nueva model in this aspect. However, students worked with the laptops in groups of two and three, instead of groups of six members with roles, like the Escuela Nueva model advocates [33]. This finding is more in line with Piaget's and Vygotsky's social constructivism as students are learning together while sharing the laptops [30]. That decision was, however, not a conscious decision but it came out of necessity: classrooms had insufficient number of power outlets. Many computers were also in bad shape in Santa María del Rio and Normal Santa Teresita, which was another reason why students shared laptops. And due to those difficulties, none of the schools actually practiced one-to-one computing, which, by definition, offers one laptop per student.

Several additional important observations were made in all the three schools. Firstly, in all the three schools, teachers were empowered and independent, and they had developed a number of practical ways of their own to utilize the laptops in their teaching. Secondly, teachers in all the three investigated schools applied their own contextual adaptations of a number of pedagogical background theories. Problem-based (PBL) approaches [17, 20] were evident in the teachers' work. Also progressive inquiry-based learning, which is based on implementing scientists' methods to learning [15] were evident in many of the cases, as well as project-based learning, which is considered to unify problem-based and progressive inquiry-based learning [4].

Thirdly, the teachers' approaches used realistic, open-ended cases and project goals, and stressed collaboration and creativity among their projects. In practice, the pupils were given a lot of responsibility and self-direction in their project works, which spanned over a multiple of school topics. Fourthly, while one-to-one computing activities were utilized to teach distinct school topics, such as mathematics, informatics, agriculture, and anatomy, the computers were widely utilized in cross-disciplinary pupils' projects. Fifthly, in addition community outreach was one important aspect in many of the projects. Sixthly, in many cases pupils' knowledge-construction processes were not limited to one tool, such as the laptop, but pupils were encouraged to construct and express knowledge on any platform of their preference.

Context based education

Contextualized education was a clear pattern between the schools. In Santa María del Rio, the students' interests were probably the reason why much of the student work had been context-related. In Fundación Formemos the agrarian focus, with or without the collaboration of external partners, had guided the teachers to work with tasks based on the reality of the students. In Normal Santa Teresita, the teachers project 'Justin is a chicken and Mary is a hen' integrated different disciplines to teach the students how to take care of chicken. In other words, all the cases shared a strong thread of education supported by the surroundings of the schools. The strongest relation to agrarian education was arguably found in Fundación Formemos and Normal Santa Teresita, possibly because these schools were more rurally located than Santa María del Rio was.

It can be argued that the context and project-based education in Fundación Formemos and Normal Santa Teresita shared similarities with the experimentally implemented pedagogy in a previously reported Tanzanian school case, where one educational goal in that school's OLPC project was to teach the pupils general health care [3]. This finding supports the view that contextual grounding, or meaningful relationship between teaching and students' life, is an important factor for 1:1 computing, which has been suggested in earlier research [36, 18].

Different pedagogical strategies with the laptops

In each of the cases, the teachers discussed different types of pedagogical strategies in their one-to-one mode of education. One teacher in Santa María del Rio explained how he used the laptops to develop a Wikipedia page with tourist information. Another teacher explained a strategy to teach geography by using the laptop's map application to describe Colombia's position in relation to the South and North American continents. In addition, teachers reported that students in general were asked to present their finished work using slideshows. In some strategies the laptop was a supportive tool for learning, whereas in others it was the main vehicle for learning.

Similarly, in Fundación Formemos the laptop was used as a tool of measurement in agriculture education, but also used to creatively edit pictures of the human body to learn about the human anatomy. The same pattern was found between the different approaches of teachers in Normal Santa Teresita. One example combined various subjects using the laptops as a glue that bound the subjects together: teachers instructed students to learn to calculate the costs of chicken farming using spreadsheet software. Another approach to develop knowledge of spaces was to explore applications of micro worlds in the computer. Our findings propose that teachers employ, with the laptops, strategies of both instructional and constructional variety.

Digital distribution of educational material

Teachers from the three schools reported that they download educational material from the Internet. However, the schools' strategies were slightly different from each other, especially in the case of Fundación Formemos. In Santa María del Rio, the teachers built digital libraries and distributed them to the students that needed material for their projects. The teachers in Normal Santa Teresita similarly downloaded educational material to support students' projects by distributing the material to the children's laptops. The strategy to build a digital library in Santa María del Rio was also slightly different to Normal Santa Teresita's, as the teacher downloaded material to support each project. Teachers' strategies in Fundación Formemos differed from the other schools regarding the aspect of distributing digital material, as the teachers partly develop their 'guides' using material from the web-based education resource center provided by the Ministry of Education. In teaching human anatomy in Fundación Formemos, the teachers distributed the downloaded Wikipedia pictures of the human body so that students learned by editing the material.

Controlling learning

One of the critiques of one-to-one computing has been that games and other entertainment material can distract students [34]. The three schools in this study shared similar approaches to monitoring and regulating student participation. The teachers trusted their students to announce inactivity of other students. Moreover, the teachers screened inactivity by frequently inquiring students about their progress. In Fundación Formenos, one of the six group members had the responsibility to make sure that the other group members were active.

Presenting student work

One of the most interesting findings, from our point of view, were the teachers' strategies to present student work in Fundación Formemos and Santa María del Rio. In Fundación Formemos, students presented, in annual workshops, photos and video recordings from their fieldwork in large-scale agricultural projects. In Santa María del Rio, students were often asked to prepare and present their finished projects using slide shows. One major difference is how Santa María del Rio had built a support structure to facilitate students' slide presentations. The informatics teacher is responsible for teaching students how to use slides, and the same teacher is responsible for supporting students when they face difficulties with preparing their slideshows.

5.1 Conclusions

The first implication of this study is that it did not support the critique of one-to-one computing's absent pedagogical guidelines [37]. Teachers in this study have instead found different ways to integrate the computer into curriculum activities, where the computer is used both as a tool and as the main vehicle of learning. It is possible that teachers in other parts of the world have also found ways of integrating the computers in their own ways. It is, however, more likely that the large South American OLPC rollouts in, for instance, Uruguay and Peru have increased the availability of educational material for the region [22]. In addition, the results of this study indicate that the Colombian Escuela Nueva reform with its rural educational features and student-centered pedagogy have further supported the integration between computers and education. This raises the question whether initiatives in Africa and Asia could be benefited by considering elements similar to the Colombian Escuela Nueva model.

A second implication of this study is concerned with the schools' strong focus on context related school projects. This raises the question how much one-to-one computing initiatives should take the context of the students into account in pedagogical planning. It can also be questioned how important context related student work is to one-to-one initiatives. This finding supports the claims about the importance of context related activities in one-to-one computing [18]. This study indicates that contextual elements should be given great focus in teacher education and in classrooms.

A third implication of this study is concerned with how teachers have adopted strategies where the laptop is shared, which is of course in contrast with the very idea of one-to-one computing. However, other researchers [27] have also indicated that one-to-one computers are in fact shared in some developing country contexts due to insufficient power sources. This study shows that the poor quality of the computers and absent technical support are additional reasons why it is likely that computers are shared in groups in similar contexts. Moreover, the poor quality of the computers have been discussed in [25], and the lack of technical support in [36]. The consequences of group sharing deserves to be studied further as the laptops are not designed for group use. Likewise, the shortage of ICT devices in developing countries have been shown to increase gender injustice in schools [26].

This study's main empirical contribution to literature is its rich presentations of strategies of three shools, which utilize one-to-one computing. It is possible that teachers in similar one-to-one initiatives could be inspired by the teachers' strategies in this study. One strategy, which could likely be successful is for instance to 'outsource' the technical support to the schools' most ICT competent teacher, or, like in Santa Maria del Rio, to the informatics teacher. The benefit from such a strategy is that all teachers do not not need to possess deep technical knowledge. Various well-working support models are many, and good solutions might be found in similar studies elsewhere.

6. REFERENCES

[1] Changing the conversation about teaching learning & technology: A report on 10 years of ACOT research. Technical report, Apple Computer, Inc., California, USA, 1995.

[2] A. Andersson and Å. Grönlund. A conceptual framework for e-learning in developing countries: A critical review of research challenges. *The Electronic Journal on Information Systems in Developing Countries*, 38(8):1–16, 2009.

[3] M. Apiola, S. Pakarinen, and M. Tedre. Pedagogical outlines for OLPC initiatives: A case of Ukombozi school in Tanzania. In *Proceedings of IEEE Africon 2011 Conference*, Livingstone, Zambia, 13th–15th September 2011.

[4] B. J. S. Barron, D. L. Schwartz, N. J. Vye, A. Moore, A. Petrosino, L. Zech, J. D. Bransford, T. Cognition, and T. G. at Vanderbilt. Doing with understanding: Lessons from research on problem- and project-based learning. *The Journal of the Learning Sciences*, 7(3&4):271–311, 1998.

[5] V. Colbert, C. Chiappe, and J. Arboleda. The new school program: More and better primary education for children in rural areas in Colombia. In H. M. Levin and M. E. Lockheed, editors, *Effective Schools in Developing Countries*. The Falmer Press, London, UK, 1993.

[6] M. Cox, M. Webb, C. Abbott, B. Blakeley, T. Beauchamp, and V. Rhodes. ICT and pedagogy: A review of the research literature. ICT in Schools Research and Evaluation Series 18, Department for Education and Skills, 2004.

[7] J. W. Creswell. *Qualitative Inquiry and Research Design: Choosing Among Five Approaches*. Sage Publications, Thousand Oaks, CA, USA, 3rd edition, 2007.

[8] L. Cuban. *Oversold and Underused: Computers in the Classroom*. Harvard University Press, Cambridge, Mass., USA, 2001.

[9] M. Denscombe. *The Good Research Guide for Small-Scale Social Research Projects*. Open University Press, Berkshire, England, 3rd edition, 2007.

[10] M. Duveskog, M. Tedre, C. Islas Sedano, and E. Sutinen. Life planning by digital storytelling in a primary school in rural Tanzania. *Educational Technology & Society*, 15(4):225–237, 2012.

[11] Y. Engeström, R. Miettinen, and R.-L. Punamäki. *Perspectives on Activity Theory*. Cambridge University Press, Cambridge, MA, USA, 1999.

[12] B. A. Ezumah. *Toward a Successful Plan for Educational Technology for Low-Income Communities: A Formative Evaluation of One Laptop Per Child (OLPC) Projects in Nigeria and Ghana*. PhD thesis, Howard University, 2010.

[13] S. Fox Buchele and R. Owusu-Aning. The one laptop per child (OLPC) project and its applicability to Ghana. In *Proceedings of the 2007 International Conference on Adaptive Science and Technology*, pages 113–118, Accra, Ghana, December 10–20 2007.

[14] Å. Grönlund, T. Englund, A. Andersson, M. Wiklund, and I. Norén. Unos uno Årsrapport 2012. Annual Report 2012, Örebro Universitet, 2013.

[15] K. Hakkarainen. Emergence of progressive-inquiry culture in computer-supported collaborative learning. *Learning Environments Research*, 6(2):199–220, 2003.

[16] H. Härtel. Low-cost devices in educational systems: The use of the "XO-laptop" in the Ethiopian educational system. *Deutsche Gesellschaft für Deutsche Gesellschaft für Technische Zusammenarbeit (GTZ) GmbH*, January 2008.

[17] C. E. Hmelo-Silver. Problem-based learning: What and how do students learn? *Educational Psychology Review*, 16(3):235–266, 2004.

[18] M. Hooker. *1:1 Technologies/Computing in the Developing World: Challenging the Digital Divide.* The Global e-Schools and Communities Initiative (GESCI), Nairobi, Kenya, 2008.

[19] F. A. Inan and D. L. Lowther. Laptops in the K-12 classrooms: Exploring factors impacting instructional use. *Computers & Education*, 55(3):937–944, 2010.

[20] D. H. Jonassen. Toward a design theory of problem solving. *Educational Technology Research and Development*, 48(4):63–85, 2000.

[21] K. Juuti and J. Lavonen. Design-based research in science education: One step towards methodology. *Nordic Studies in Science Education (NorDiNa)*, 4:54–68, August 2006.

[22] K. L. Kraemer, J. Dedrick, and P. Sharma. One laptop per child: Vision vs. reality. *Communications of the ACM*, 52(6):66–73, 2009.

[23] P. McEwan. Evaluating multigrade school reform in Latin America. *Comparative Education*, 44(4):465–483, 2008.

[24] P. Mozelius, K. Rahuman, and G. Wikramanayake. A Sri Lankan one-to-one computing initiative and its impact on formal learning in primary school, Sri Lanka. *Hearld Journal of Education and General Studies*, 1(1):16–21, 2012.

[25] P. Mozelius, K. Rahuman, and G. Wikramanayake. Two years of one-to-one computing in Sri Lanka - the impact on formal and informal learning in primary school education. In T. Bastiaens and G. Marks, editors, *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education (ELEARN)*, pages 1192–1201, Chesapeake, VA, USA, 2012. AACE.

[26] W. M. Olatokun. Gender and ICT policy in Africa: Issues, strategies and policy options. *Information Development*, 24(1):53–65, 2008.

[27] J. Pal, R. Patra, S. Nedevschi, M. Plauche, and U. Pawar. The case of the occasionally cheap computer: Low-cost devices and classrooms in the developing world. *Information Technologies and International Development*, 5(1):49–64, 2009.

[28] W. R. Penuel. Implementation and effects of one-to-one computing initiatives: A research synthesis. *Journal of Research on Technology in Education*, 38(3):329–348, 2006.

[29] H. Pettersson. Datorn som hjälpmedel - ett kreativt redskap, en distraktion eller en väldigt dyr skrivmaskin? Master's thesis, Malmö Högskola, Malmö, Sweden, 2012.

[30] A. Pritchard and J. Woollard. *Psychology for the Classroom: Constructivism and Social Learning.* Routledge, London, UK, 2010.

[31] J. J. Randolph. *Multidisciplinary Methods in Educational Technology Research and Development.* HAMK University of Applied Sciences, Hämeenlinna, Finland, 2008.

[32] E. Schiefelbein. *In Search of the School of the XXI Century: Is the Colombian Escuela Nueva the Right Pathfinder?* UNESCO Regional Office for Education in Latin America and the Caribbean, 1991.

[33] E. Schiefelbein, R. Vera, H. Aranda, Z. Vargas, and V. Corco. *En Busca De La Escuela Del Siglo XXI: ¿Puede Darnos La Pista La Escuela Nueva De Colombia?* Universidad Pedagógica Nacional, 1986.

[34] V. W. Setzer. A critical view of the "one laptop per child" project, dept. of computer science, university of são paulo, brazil., February 26 2009.

[35] G. A. Straka, editor. *Conceptions of Self-Directed Learning: Theoretical and Conceptual Considerations.* Waxmann, Münster, Germany, 2000.

[36] M. Tedre, H. Hansson, P. Mozelius, and S. Lind. Crucial considerations in one-to-one computing in developing countries. In P. Cunningham and M. Cunningham, editors, *Proceedings of IST-Africa 2011 Conference*, Gaborone, Botswana, May 11-13 2011.

[37] T. Unwin, editor. *ICT4D: Information and Communication Technology for Development.* Cambridge University Press, Cambridge, UK, 2009.

[38] D. Wiliam. Comments on Bulterman-Bos: What should education research do, and how should it do it? *Educational Researcher*, 37(7):432–438, 2008.

A JavaScript Library for Visualizing Program Execution

Teemu Sirkiä
Department of Computer Science and Engineering
Aalto University, School of Science
Espoo, Finland
teemu.sirkia@aalto.fi

ABSTRACT

In this poster, we present a JavaScript library which can be used to create educational program visualization applications for multiple programming languages. By using modern web technologies, visualizations can be embedded in web pages, allowing them to be used with all modern web browsers on different platforms.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Human factors

Keywords

program visualization, introductory programming, JavaScript

1. INTRODUCTION

Program visualizations can help students to understand the correct execution model of the given program. Various program visualization applications do exist but they are often tightly bound to a specific programming language and are standalone applications.

There are few JavaScript-based program visualization applications or frameworks which support integrating the visualizations seamlessly into web pages together with course materials.

It is also difficult to share best practices of program visualization because the visualization applications for different programming languages are completely independent. A common toolkit to build such applications does not exist.

In this poster, we present the current status of a new JavaScript library which provides a framework for building educational program visualization tools, and discuss some of the key concepts behind this library.

Koli Calling '13, November 14–17, Koli, Finland
ACM 978-1-4503-2482-3/13/11 ...$15.00
http://dx.doi.org/10.1145/2526968.2526990

2. THE LIBRARY

Instead of a complete program visualization application, the library provides an API for creating language-specific visualizations. The library contains the common basic functionality such as controlling the animation step-by-step, moving backwards and forwards, showing explanation texts as well as the most common visualization steps such as fetching values, evaluating operators and assigning values to variables.

The current capabilities of the library can be described in terms of the two-dimensional engagement taxonomy 2DET [6]. On the direct engagement dimension, the library supports *Controlled viewing* of a program visualization. On the content ownership dimension, the library supports only the lowest level, *Given content.*

2.1 Embedding in Online Materials

Electronic materials are becoming more popular, in part because it is possible to include interactive elements together with the text.

As the library is implemented in JavaScript instead of Java or Flash, the library works with the modern browsers ranging from mobile and tablet devices to desktops. The library could be used as a platform to implement visualization applications in the same vein as UUhistle [7] and Jeliot 3 [4].

The JSAV library [2] has a similar purpose as it provides a JavaScript-based general framework for creating animations. The main difference between JSAV and this library is that JSAV is meant for algorithm visualization instead of program visualization and therefore the abstraction level of JSAV is higher.

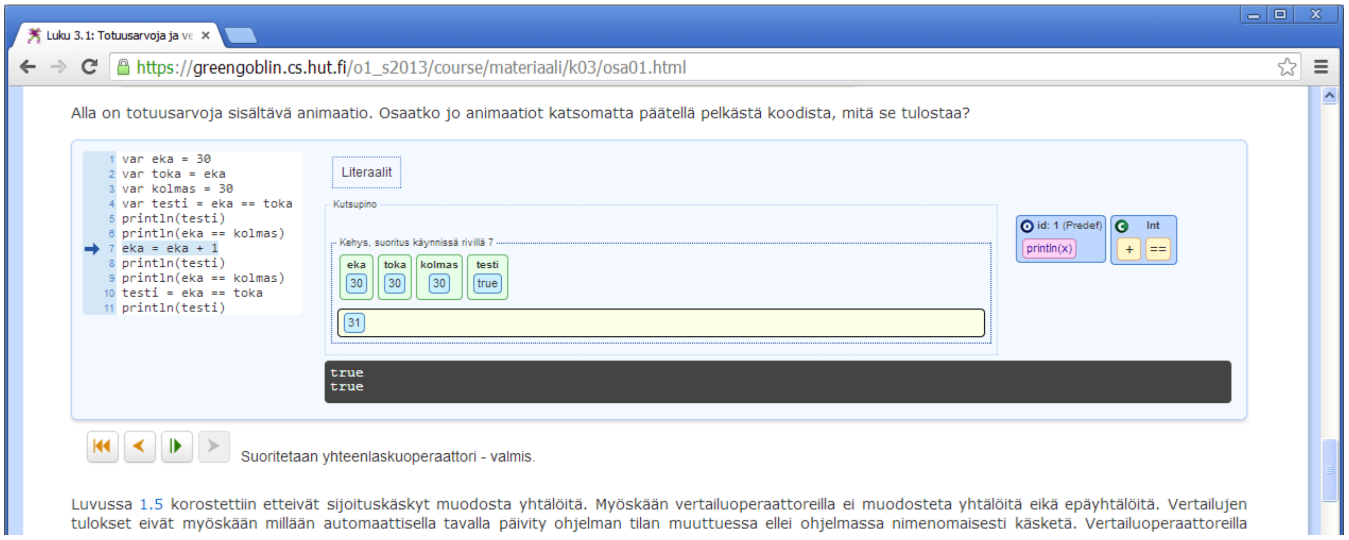
Online Python Tutor [1] is an example of an existing program visualization application whose visualizations can be embedded in web pages. However, it does not provide a general toolkit for creating visualizations.

ViLLE [5] works with modern browsers, but instead of providing embeddable visualizations, ViLLE is an online learning platform.

2.2 Multiple Languages

The goal of our library is to be language neutral while still providing a toolkit for creating language-specific visualizations as opposed to ViLLE which is ostensibly a language-independent system but it supports only a small subset of multiple languages.

The code to be visualized is transformed into a language-independent intermediate language that the library can execute. Jeliot 3 uses a similar approach to visualize Java code by using its own intermediate language called MCode [3].



Luku 3.1: Totuusarvoja ja ve

https://greengoblin.cs.hut.fi/o1_s2013/course/materiaali/k03/osa01.html

Alla on totuusarvoja sisältävä animaatio. Osaatko jo animaatiot katsomatta päätellä pelkästä koodista, mitä se tulostaa?

```
var eka = 30
var toka = eka
var kolmas = 30
var testi = eka == toka
println(testi)
println(eka == kolmas)
eka = eka + 1
println(testi)
println(eka == kolmas)
testi = eka == toka
println(testi)
```

```
true
true
```

Suoritetaan yhteenlaskuoperaattori - valmis.

Luvussa 1.5 korostettiin etteivät sijoituskäskyt muodosta yhtälöitä. Myöskään vertailuoperaattoreilla ei muodosteta yhtälöitä eikä epäyhtälöitä. Vertailujen tulokset eivät myöskään millään automaattisella tavalla päivity ohjelman tilan muuttuessa ellei ohjelmassa nimenomaisesti käsketä. Vertailuoperaattoreilla

Figure 1: A part of the course materials (in Finnish). A visualization related to Booleans is embedded in the middle of the chapter. Students are expected to realize, among the other things, that the value in the variable `testi` remains the same although the value in variable `eka` is just about to change.

The language-neutral approach benefits the applications using the library, because they can use the new features without the need to implement the same feature into each application separately.

2.3 Custom Visualizations

In the most existing tools, a developer can not change neither the layout nor the visual style of visualizations.

As the library does not create or define the layout, the developer using the library can decide how the visualization will look like. The structure of HTML elements and CSS stylesheet will create the visual appearance and therefore it is quite easy to change the style.

By changing the layout or the style, the developer can customize the animation to suit its specific purpose. For example, unnecessary views (such as the stack) can be left out. The same code can be visualized in many ways showing more or fewer details.

3. USE CASE

This fall, a new basic programming course started at Aalto University. The course materials are online as a kind of an electronic textbook which contains text, visualizations and exercises.

The visualizations (see Figure 1) are built by using the library and currently the materials contain over 50 animations of programs written in Scala. The style of the animations is similar to that in UUhistle.

The material works in modern browsers, including mobile and tablet devices, although the display resolution of the mobile devices might not be optimal for this kind of material.

4. FUTURE WORK

As the development of the library has just begun, there is plenty to do in order to provide a good toolkit for creating program visualization applications for different programming languages.

Our goals include better support for multiple programming paradigms, in particular functional programming in addition to imperative and object-oriented programming.

Currently the library supports only the controlled viewing of given content in terms of the 2DET. The support for higher levels of learner engagement both in the form of direct engagement and content ownership are being explored.

5. REFERENCES

[1] P. J. Guo. Online Python Tutor: Embeddable Web-Based Program Visualization for CS education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[2] V. Karavirta and C. A. Shaffer. JSAV: the JavaScript algorithm visualization library. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, ITiCSE '13, pages 159–164, New York, NY, USA, 2013. ACM.

[3] A. Moreno. The Design and Implementation of Intermediate Codes for Software Visualization. Master's thesis, University of Joensuu, 2005.

[4] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing Programs with Jeliot 3. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '04, pages 373–376. ACM, 2004.

[5] T. Rajala, M.-J. Laakso, E. Kaila, and T. Salakoski. ViLLE: a language-independent program visualization tool. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88*, Koli Calling '07, pages 151–159, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[6] J. Sorva, V. Karavirta, and L. Malmi. A Review of Generic Program Visualization Systems for Introductory Programming Education. *Trans. Comput. Educ.*, To appear.

[7] J. Sorva and T. Sirkiä. UUhistle: A Software Tool for Visual Program Simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 49–54, New York, NY, USA, 2010. ACM.

How Do Students Learn to Program in a Connected World?

Jian Shi
School of Electronics & Computer Science
University of Southampton
Southampton, Hants, UK
+44 23 8059 6000
js9g09@ecs.soton.ac.uk

Su White
School of Electronics & Computer Science
University of Southampton
Southampton, Hants, UK
+44 23 8059 6000
saw@ecs.soton.ac.uk

ABSTRACT

Computer scientists have been interested in the recurrent problem of teaching introductory programming for years. New undergraduates come from varied backgrounds with different prior experiences, so it is a complex task to ensure an equal learning outcome for each student. A large body of relevant literature exists. However, this mainly dates from an era when practical based learning activities were only supported by paper resources. It may therefore be interesting and perhaps helpful to teachers to gather insights into the contemporary practices of undergraduates' accessing information sources when learning to program.

This poster presents a study exploring the learning practices of students enrolled on introductory programming courses. A mixed methods approach is designed to triangulate the gathered quantitative and qualitative data in an effort to expose practices, beliefs and attitudes to learning from a perspective of identifying what the student does. The poster will present interim findings and discuss the challenges and potential advantages of working within a mixed methods research framework.

Categories and Subject Descriptors

K.3.2 [**Computers and education**]: Computer and information science education – *computer science education*

General Terms

Human Factors.

Keywords

Perception, education, learning programming, experience, mixed methods, Nominal Group Technique.

1. INTRODUCTION

Every year thousands of students arrive at university who will be taught introductory programming. The backgrounds and prior experience of these students will vary, and the role of the academic is to strive to ensure wherever possible an equal outcome for all; establishing is a sound foundation for future study. The introduction needs to be technically robust, establish and build sound learning practices and be taught at a level and in a manner which sustains motivation across the cohort irrespective of prior experience or expertise.

The research questions for this study are:

1) What effect do a student's background, attitude, and beliefs have on their subsequent approaches and practices when learning to program?

2) Whilst learning to program to what extent do different students*?*
   - a) Gain or lose confidence and motivation.
   - b) Modify their approaches to learning as they progress?

3) How do students integrate their formal and informal learning activities? To what extent do students
   - a) Rely on formally provided materials and exercises?
   - b) Develop individual approaches to learning which integrate real world and online activities and materials?

2. BACKGROUND

Learning to program is an activity which may be very different from the learning tasks with which most new undergraduate students are familiar. This is perhaps the reason why academics in computer science have become so concerned with this initial educational challenge. Usually universities recruit students who are judged to be academically capable of successfully completing their studies. The task therefore is to ensure that students are taught and learn in such a way that they achieve this potential, acknowledging that some students may choose to change the direction of their academic career or for external reasons may not be able to sustain the motivation or the workload which is necessary for ultimate success.

In terms of understanding, this study adopts a framework of teaching strategies and modeling student learning suggested by Biggs [1], in which academics should focus on "what the student does" (Level 3) to optimize the pedagogy strategy.

3. METHODOLOGY

A range of established approaches in research are used in order to study student behaviors and teaching interventions better. Quantitative methods are widely used to gather base data, whilst qualitative approaches enable the researcher to categorise subjective factors such as thoughts, beliefs and attitudes. Since this study is aiming to understand "what the student does" while learning to program, gathering just quantitative data would not be

Koli Calling '13, Nov 14-17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526991.

sufficient. Qualitative evidence should be collected as much as possible to support research outcomes. Therefore, a mixed methods approach which has the advantage of anticipating and remedying the potential gaps and risks in our data collection should be adopted.

Interviews and focus group discussion are common approaches used to gather qualitative data, although typically such tools are also used to gather smaller amounts of short statement/comment style qualitative data. Interviews and Nominal Group Technique (NGT) [2] are also common approaches used to gather qualitative data. The NGT is one form of structured "brainstorming", aiming to ensure that each participant obtains an equal discourse rights and actively support and cultivate independent thinking during the entire survey process. Typically, it is used as a wide spread decision making method for generating ideas into one particular sequential list.

A mixed methods approach for the whole study has been designed combining quantitative and qualitative methods listed above. The respondents were AY2012-13 students selected from 1st year undergraduate introductory programming in the University of Southampton. An initial survey was designed to provide basic data in answer to some of the research questions. Subsequent weekly cohort wide surveys could help the researcher note and collect the data about learners' learning progress, as well as some contemporaneous observation. Having identified some interim findings from a subsequent end of term survey and NGT study, additional data is scheduled to be collected from the students using individual and focus group interviews.

4. DATA COLLECTION

4.1 Pilot Study

For the purposes of obtaining an overview of the approaches of undergraduate students when they are introduced to programming at universities and trailling future formal surveys from the responses, a preliminary pilot study in a smaller scale was conducted with AY2011-12 students. The questions covered standard teaching processes in the UK universities – lectures, assessments and feedback. Plus, students' personal information should be considered due to the diversity

4.2 Initial Survey

A survey of the AY 2012-13 cohort was conducted in the form of questionnaire in September 2012 prior to teaching beginning. It aimed to gain a general understanding of how much prior experience our new students had and their initial attitudes towards learning to program, including whether they were motivated and their beliefs about how difficult they assumed programming would be. The collected responses provided basic evidence for the first research question. Among the 169 completed responses, 40 participants (23.7%) stated that they were total novices. Approximately 95% respondents believed that they were motivated (in which two thirds were highly motivated), and had high expectations.

4.3 Weekly Survey

This short online survey was scheduled between October 2012 and December 2012 in students' weekly practical labs, aiming to trace students' learning progress. This survey was designed to investigate responses relevant to the second research question. In order to maximise the response rate, questions were designed to be concise and easy to answer. Students would only need to simply tick any that apply.

Having analyzed the responses, it can be summarized that their most challenging topics are "Testing & Debugging", "Array & API" and "Overloading".

4.4 End of Term Survey

There are 46 completed responses from an online questionnaire survey conducted to investigate to what extend the Web impacts on the students' learning to program once the students have finished the introductory module.

The most popular online platform was a public search engine, while the formally provided learning environment comes second. Participants tend to use the Web the most when they were introduced to a new theory or looking for help when they got stuck. In terms of the reason, participants believe that the Web is convenient to use, and it is the best place to find information.

4.5 NGT Study

This NGT study focused on 1st year undergraduates' detailed perspective when learning to program, especially identifying their learning approaches. Since this survey is conducted by adopting the Nominal Group Technique, participants were asked to provide their opinions on A5-size index cards and have them ranked by using the Zapper voting device.

Interesting findings have been gained in some level according to participants' responses:

- Little use was made of the printed notes and set textbook, because it is felt that this learning approach is much slower than searching the Web.
- Most participants use the Web when they get stuck while learning to program. It is believed that the Web could help to avoid asking stupid or obvious questions and save time.
- If the participants failed to obtain the information they wanted from the Web, they might ask their friends and have a discussion.
- It is worth mentioning that participants experienced loss of motivation and feel frustration if they failed to solve the problems from the Web and their friends.

At the time of writing, the responses are still under analysis. If necessary, some individual interviews will be undertaken to gather much more in-depth insights from our students.

5. CONCLUSION

Evidence so far suggests that there have been some changes in routine practice among learners. It will be interesting to discover whether the approaches are consistent across achievement and experience levels. Insights into successful strategies may assist in designing learning activities to augment programming assignments.

6. REFERENCES

[1] J. B. Biggs and C. Tang, *Teaching for quality learning at university*, 4th Ed. Open University Press, 2007, pp. 16–20.

[2] R. B. Dunham, "Nominal Group Technique: A Users' Guide." 1999.

Teaching Artificial Intelligence Using a Web-Based Game Server

Stefan Friese
University of Duisburg-Essen
Universitätsstr. 9
45141 Essen, Germany
stefan.friese@icb.uni-due.de

Kristian Rother
University of Duisburg-Essen
Universitätsstr. 9
45141 Essen, Germany
kristian.rother@icb.uni-due.de

ABSTRACT

Games are a classical field of application for concepts of artificial intelligence (AI). We outline a didactic concept for teaching AI to Business Information Systems students at the university level and a web-based game server that was implemented to support this concept. Our results show an improved engagement of students as well as a flexible technical basis for AI education and AI research which can be applied in various learning and research contexts.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: [Computer Uses in Education]; K.3.2 [**Computers and Education**]: [Computer and Information Science Education]

General Terms

Game Server, AI Education

Keywords

Artificial Intelligence, Games, Teaching, Server

1. INTRODUCTION

The concepts for the course described in this paper are aimed at teaching AI based on games to graduate students in Business Information Systems at a German university. Using games to teach CS and AI is a common concept [8, 6, 2, 10]. An increased engagement of students is expected, and games provide an environment of a certain level of complexity and a well-known set of rules.

2. GAME CHOICE CRITERIA

The game choice is important. It depends on the AI techniques to be taught and the properties of relevant real-world problems. In our case, these are economic planning and decision problems. A suitable level of complexity is required and the game should not be well known to start students on equal footing. The following table lists our criteria based on classification schemes in [7, 5, 9] (completeness of information, randomness), [1] (alea, agôn, mimicry, ilinx) and [3] (pace, player structure, teleology, time representation, savability, determinism):

- **Completeness of Information** Perfect information games were excluded (perfect information is less common in economical decision problems).
- **Randomness/Alea** Purely random games are unsuitable (no player influence), but a certain amount of randomness, e.g. a random initial state, may fit.
- **Competitiveness/Agôn** A competitive game should be chosen to fit economical environments. This matches our course setting of competing teams.
- **Mimicry** A player being an imaginary character or not had no influence on the game choice.
- **Psychomotor skills/Ilinx** We exclude games requiring physical skills as robotics is beyond our scope.
- **Pace** Real-time games imply a higher level of complexity regarding protocols and communication. As many real-world problems in our scope can be represented in a turn-based way, we favor these games.
- **Player structure** Two adversarial players seem to be a good choice, because testing and evaluation of the implemented agents is easier for smaller sets of players.
- **Teleology** To be able to simulate tournaments in a reasonable amount of time, the game should be finite.
- **Time Representation** As we set a focus on turn-based games already, the representation of time defaults to discrete steps.
- **Savability** It should be possible to interrupt and resume matches as this simplifies testing agents.
- **Determinism** For evaluation, deterministic effects of actions are desirable. Matches should be repeatable.

3. THE GAMES OF ROK AND TENCARDS

Our choice was the trick-based two-player card game Rok, a classical but rather unknown game matching our criteria. A match starts with a bidding phase, followed by a playing phase. Both players have a hand that is only known to them and a board where the top cards are visible. Points

Koli Calling '13, Nov 14-17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526992

are awarded for won tricks and certain cards in a trick[1]. Rok has similarities to Bridge, Skat and Doppelkopf.

Learning a new game and applying AI concepts to a complex scenario is a hurdle for students from a conceptual and technical perspective. Hence we developed tencards, a custom-tailored, simple but non-trivial introductory game which prepares students for the more complex Rok task. The rules of this two-player game are extremely simple: There are ten cards numbered one to ten. Each player has three cards, while the remaining ones form a hidden stack. Each player plays a card from his hand or a stack card. The player who played the higher card wins the trick and continues. The game is won by the first player who wins two tricks. Despite the simplicity, non-trivial situations can occur.

4. IMPLEMENTING A GAME SERVER

The technical part of our concept consists of the development of an online game server. Existing game servers serve special needs (e.g. ggp server for general game playing, ruling out incomplete information games) or their architecture does not fit the requirements (e.g. Cubeia Firebase).

The server should support arbitrary games with a clean abstraction between game logic, visualization and general operations and have little dependency on the client system. Matches should be possible between human players and/or AI agents. Human games are relevant to test AI agents and to gain data of human behavior for techniques like CBR. As we focus on a solution usable beyond the borders of our own university, there should be a language-independent interface for external AI players. Individual matches and tournaments should be possible and the server should be easy to integrate with our Prolog learning environment EPPU [4].

4.1 Technical Overview

The server has been implemented using declarative concepts of Prolog where possible, alongside with Java-based server functionality (using Apache Tomcat), a web-based GUI (HTML, CSS, JavaScript) and a PostgreSQL database.

The core of the game engine is a set of game rule descriptions implemented in Prolog. This language is well-suited to describe the state transitions caused by the players' actions. On top of the rule description of a game, there is a declarative description of its visualization (called a view). Views determine how a match state should be presented to the players, independent of any actual GUI technology. The Java server transforms them into actual JavaScript code.

4.2 Communication and Integration

The engine is able to communicate directly with Prolog agents developed by local students in EPPU. A second interface has been implemented to call remote AI players implemented in arbitrary languages using a network connection. This way it is possible to implement an agent e.g. in Java and run it on a remote host. An AI agent is stateless and has to be able to answer to requests containing a formal description of the match state and a private information store.

5. RESULTS

The two-step concept with two games in our AI project eased students into agent development first but left enough time to focus on the more complex card game Rok. The availability of a web-based system enables a quick evaluation of the performance of an agent and debugging during the development process thus shortening the feedback loop and enabling easy remote collaboration. These benefits lead to a working hypothesis that the performance of the developed agents (measured in wins) should be higher after the introduction of the game server, ceteris paribus.

The agents were evaluated in large tournaments where all agents (including those from earlier semesters) played against each other in sets of 4000 matches to rule out random effects. The results are promising: All match sets between old and new agents were won by the new agents with the exception of one extraordinary good old agent. However we need more empirical evidence to claim an effect of the game server because the number of participants was low (41 in six semesters) and isolating the effect is difficult. Qualitative, oral student feedback corroborates the working hypothesis.

The game server has also proven to be a useful tool in research because it allowed us to quickly prototype new games and to prototype and test agents for different games (especially Rok, Bridge and Poker).

6. REFERENCES

[1] R. Caillois. The definition of play and the classification of games. In K. Salen and E. Zimmerman, editors, *The Game Design Reader: A Rules of Play Anthology*, pages 122–155. MIT Press, 2006.

[2] P. Drake and K. Sung. Teaching introductory programming with popular board games. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 619–624. ACM, 2011.

[3] A. Espen, S. S. Marie, and S. Lise. A multidimensional typology of games. In *Level Up Conference Proceedings*. University of Utrecht, November 2003.

[4] S. Friese. Measuring of and reacting to learners' progress in logic programming courses. In *Proceedings of ITiCSE'10*, pages 152–154, 2010.

[5] O. M. Halck and F. A. Dahl. On classification of games and evaluation of players–with some sweeping generalizations about the literature. In *Proceedings of the ICML-99 Workshop on Machine Learning in Game Playing*, 1999.

[6] P. Hingston, B. Combes, and M. Masek. Teaching an undergraduate ai course with games and simulation. *Technologies for E-Learning and Digital Entertainment*, pages 494–506, 2006.

[7] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94:167–215, 1997.

[8] S. Kurkovsky. Engaging students through mobile game development. In *ACM SIGCSE Bulletin*, volume 41, pages 44–48. ACM, 2009.

[9] A. Macleod. Selecting games for artificial intelligence research. In *The Second Annual International Workshop in Computer Game Design and Technology*, 2004.

[10] A. McGovern, Z. Tidwell, and D. Rushing. Teaching introductory artificial intelligence through java-based games. In *AAAI Symposium on Educational Advances in Artificial Intelligence, North America*, 2011.

[1]Complete rules on `http://udue.de/gameserver`

Brain-based Programming – A Good Concept For Schools?

Peter Antonitsch
Alpen-Adria-Universität Klagenfurt
Universitätsstraße 65-67
9020 Klagenfurt, Austria
+43(463)27003517
peter.antonitsch@aau.at

Barbara Sabitzer
Alpen-Adria-Universität Klagenfurt
Universitätsstraße 65-67
9020 Klagenfurt, Austria
+43(463)27003517
barbara.sabitzer@aau.at

ABSTRACT

"Brain-based Programming" is a teaching concept based on neurodidactical principles and was originally developed for introductory programming courses at university level. As the results of a pilot project show it seems to be a good concept because the learning outcomes were higher than in parallel courses. Hence we are attempting to transfer the concept of brain-based programming to schools based on the findings of the pilot project at our university. This paper aims at presenting the original concept and basic ingredients as well as adaptions considered necessary for a successful implementation at school.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]

General Terms
Human Factors, Languages.

Keywords
Programming, computer science education, brain-based learning.

1. INTRODUCTION

Learning to program is hard. How can it be made easier? This question often appears in relevant literature and neurodidactics may give an answer. This research field was originally introduced in order to support students with learning difficulties. So why not use it to reduce the learning difficulties in programming courses? It offers proposals for effective learning that considers how the brain works, some of them deriving from teaching approaches like constructivism or progressive pedagogy [1]. This is the basis for the new concept of brain-based programming that was developed during a project of the same name and successfully tested in an introductory programming course at the university. The paper presents this concept as well as some results and describes an adapted version for schools.

2. BRAIN-BASED PROGRAMMING

2.1 The Project

The project "Brain-based Programming" was initiated in order to improve the learning outcomes of the bachelor course "Introduction to structured and object-based programming". The course consists of two parts, each 90 minutes per week: a lecture course where the concepts of programming are introduced, and a practical course, where they are put into practice. In the practical course the students have to solve several worksheets containing Java programming exercises and pass two written exams. Usually, many students fail (50% -70%) and have to repeat the course.

Koli Calling '13, Nov 14-17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526993

Starting from an analysis of the errors made in these exams, a questionnaire as well as informal interviews with students and lecturers we developed the concept of brain-based programming. We changed the lesson design, used different teaching and learning methods and provided the students with supplementary brain-based worksheets and tasks. In the last winter term the concept was tested and evaluated in one of seven parallel practical programming courses. The results of the pilot project were promising. (For a detailed description of the project see [1]).

2.2 The Concept

Brain-based Lessons

The lessons were structured considering the functioning of the brain and the different memory phases. First of all, at the beginning of the semester, the students were divided in three groups depending on the level of their individual programming experience: *professionals* with advanced programming skills and *amateurs* with basic skills, "worked" as peer-tutors or –teachers and supported the real *beginners* who solved more "mini-exercises" according to their needs and competencies they had to improve.

This type of learning requires an open setting with free work phases, considering many neurodidactical principles, for example learning through recall (peer tutoring, pair programming), automatic pattern recognition (discovery learning, learning etc.) [1].

The weekly units were divided in three phases that were not strictly limited in time.

Questions (work in groups with peer-tutor): Open questions of the weekly lecture should be clarified.

Discovering: Students worked in groups on reading exercises, step-by-step instructions or worked-out examples in order to (re)discover the topics of the lecture in their own learning rhythm.

Lab: Students solved tasks by pair programming.

Brain-based Tasks

The tasks used in the experimental group were supplementary to the regular worksheets and contained approximately the same amount of exercises. Contrary to the other courses the students in the "brain-based" group had free choice concerning the exercises and also the type of tasks. The worksheets, which are being assembled to a booklet, contain the following exercise types:

Reading exercises, e.g. complete program code, step-by-step instructions or short tasks with solutions, foster discovery learning and benefit from the brain mechanism of pattern recognition

Competence-oriented tasks enhance learning by following the principle "practice makes permanent" [2].

Detail-orientated exercises are assembled into a complex program at the end of the course. This is based on the principle that "the whole is more than the sum of its parts" (Aristotle, 384-322 BC). The brain simultaneously processes parts and wholes [3].

The evaluation of the pilot project shows that the concept of brain-based programming was successful: 70% of the students in the experimental group passed the exam whereas the average in the parallel groups was 42%. Regarding methods and tasks, in particular, discovery learning (step-by-step videos, mini-exercises with solutions, reading exercises) and cooperative learning (peer tutoring, pair programming) were considered to be very useful.

3. BRAIN-BASED PROGRAMMING II– ADAPTATION FOR SCHOOLS

3.1 Steps towards Brain-based Programming

To adapt the design of the pilot study to meet specifics at school level, the following aspects have to be considered as well: Introductory programming courses at university level consist of two distinct parts, separated over time: A series of lectures explaining the basic concepts in a theoretical way, and a series of practical lab courses, which allow for repetition and deeper study of the material. At school, phases of initial theoretical input and practical application usually are mixed within the same learning unit.

Second, at university level a well-defined learning setting based on Java and the development environment Eclipse was used. At school most probably different kinds of programming language and/or development environments are in use. Comparing, for instance, Eclipse with the Greenfoot-environment makes clear, that corresponding representations of the programmable performer are prone to influence the learning progress.

Third, while grading in university courses depends mainly on the results of an intermediate and a final exam, assessment at school has to be based on constant monitoring of the learners' performance. Commingling of learning and assessment based on self-responsible learning can become critical as it is still not common school culture to allow for learning from mistakes.

Moreover, and related to the last issue, (most) learners at school age are at a different level of personal development than university students. This concerns the ability to think in abstract terms including the availability of appropriate problem solving strategies, but in particular the development of general social and personal skills like self-responsibility.

3.2 A First Proposal for Schools

Considering all of the above, the most basic concern for the extension of the study to introduce brain-based programming is to provide comparable learning settings. The programming languages in use are Java, C# and PHP, all of which are part of the C-language family and provide rather similar basic control- and data structures. Based on this the didactical concept is as follows:

As some of the learners can be expected to lack experience in algorithmic thinking, learning to program has to cover two phases. During the shorter first phase learners should get in touch with algorithms by following and describing a sequence of step-by-step instructions what can be done without using digital devices. In the second, longer phase algorithmization should lead to automation by handing program code to a computer, similar to the pilot study.

As both phases have to consider periods of input and periods of practicing, the three-step structure of the pilot-study will be slightly altered by replacing "Questions" with "Instructions" and extending "Discovering" to "Questions and self-organized Discovering" but keeping the final step of "Lab". Most important, the second step has to include activities to train self-organized and self-responsible learning on the side of the learners.

To allow for sufficient monitoring of the learners performance by the teachers, a peer-tutor system seems reasonable. Supposing that some of the learners do have experiences with algorithmic thinking and algorithmization beforehand, the first phase is ideal to introduce this concept to the learners and rely on the established tutoring system during the second phase of programming/coding.

Programming, in turn, can be done by pair programming mixed with classical forms of cooperative learning described by the catchphrase "think – pair – share", which means that given a certain task first all learners have to find a solution or something nearby by themselves, then compare the found solutions pair wise and present a joint solution to the whole learning group.

Finally, monitoring of the learners performance will be augmented by short written assessments following a strict assessment plan handed to the learners at the beginning of the learning phase. This assessment plan contains the learning goals, a corresponding schedule. Hence it is a strict guideline to structure the learning process as needed by most of the learners at that age.

4. CONCLUSION AND OUTLOOK

The measures described above regard the shortcomings of the previous approaches to brain-based related programming by providing tighter structures for the learning processes by the teacher, thus allowing phases of input, phases of practicing and phases of internalizing to take turns in an effective way, and by varying task representation to include the effect of multimodality, which can go as far as leaving off the software in use and switch to offline forms of representation, thus giving special prominence to the competence of algorithmic thinking which is fundamental for writing programs.

These measures shall introduce a first notion of brain-based programming, which, referring to findings from the pilot study at university, has much potential to be a good concept for informatics education at school as well. Nevertheless, as is pointed out by [4, 5, 6], the fundamental competence of algorithmization could be acquired much earlier, even at primary level, which would make it possible to map the structure of the pilot study at university with less adaptations to programming at secondary schools

5. REFERENCES

[1] Sabitzer, B.; Strutzmann, S. 2013, in press. Brain-based Programming. *Proceedings of IEEE Frontiers in Education*, October 2013, Oklahoma City, Oklahoma, US.

[2] Sousa, D. A. 2006. *How The Brain Learns*. Third edition. Corwin Press, Thousand Oaks, California.

[3] Caine, G., & Caine, R. 2007. Natural learning: The basis for raising and sustaining high standards of real world performance. Position Paper: *Natural Learning Research Insitute*.

[4] CSTA: Computational Thinking in K–12 Education Teacher Resources, 2nd edition. 2011. http://csta.acm.org/Curriculum/sub/CurrFiles/472.11CTTeacherResources_2ed-SP-vF.pdf.

[5] Antonitsch P., Gigacher C., Hanisch L.: Imagination, Algorithmization, Automation. Paper submitted to the 8thWIPSCE,Aarhus/Denmark,Nov. 11-13, 2013.

[6] Antonitsch P.: How to Consider Informatics in Primary Education, in press: 25 Jahre Digitale Schule in Österreich. Proceedings of eEducation Summer Conference, August 2013, Klagenfurt (2013)

.

Brain-based Teaching in Computer Science – Neurodidactical Proposals For Effective Teaching

Barbara Sabitzer
Alpen-Adria-Universität Klagenfurt
Universitätsstraße 65-67
9020 Klagenfurt, Austria
+43(463)27003517
barbara.sabitzer@aau.at

Stefan Pasterk
Alpen-Adria-Universität Klagenfurt
Universitätsstraße 65-67
9020 Klagenfurt, Austria
+43(463)27003517
stefan.pasterk@aau.at

Sabrina Elsenbaumer
Alpen-Adria-Universität Klagenfurt
Universitätsstraße 65-67
9020 Klagenfurt, Austria
+43(463)27003517
sabrina_elsenbaumer@gmx.at

ABSTRACT
Brain-based teaching is neither a method nor a concept. It is rather a way of teaching that tries to support the learning and memory process in all phases from lesson design over input and practice up to the transfer of knowledge and competencies in real situations. The proposals for brain-based teaching come from neurodidactics or educational neuroscience that combines findings of brain and memory research, didactics, pedagogy and psychology. This paper aims at presenting concepts and methods that can facilitate learning and proposals for designing computer science lessons by considering the functioning of the brain and the memory.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]

General Terms
Human Factors.

Keywords
Computer science education, brain-based learning, neurodidactics.

1. INTRODUCTION
Brain-based teaching is not new - and it is no panacea. It is neither a method nor a concept. It is rather a way of teaching that tries to support the learning and memory process in all phases from lesson design over input and practice up to the transfer of knowledge and competencies in real situations. Proposals for brain-based teaching come from neurodidactics, an interface between brain and memory research, didactics, pedagogy and psychology. This research field confirms learning theories and teaching concepts like constructivism [1, 2] or progressive education [3]. This paper presents some examples of brain-based lesson design and teaching methods as well as their application in practice.

2. BRAIN-BASED LESSON DESIGN
Summarizing recommendations of neurodidactical research the following facts shall be kept in mind in lesson planning: Biological facts that cannot be changed but considered, environmental facts that can be created, personal facts that can be influenced, and brain and memory functions that can be supported [5]. Brain-based lesson design tries to consider as many factors as possible. Concerning the lesson structure they all approaches have two important principles in common: Alternating attention levels have to be considered and learners have to be active.

Koli Calling '13, Nov 14-17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526994

Sousa [6] e.g. suggests nine components that should be considered (certainly not all of them fit in all lessons): anticipatory set, learning objective, purpose, input, modeling, check for understanding, guided practice, closure and independent practice. Furthermore he proposes to divide each learning episode in two prime-time phases of about 20 minutes, one at the beginning containing the most important information, and one at the end of the lesson for a closure. The downtime between these phases should be used for practice. The model of Learning under Self-control follows the phases of attention and memory, too, and proposes the following structure:

1. Activation phase (about 10 minutes)
2. Core-information phase (about 5-10 minutes)
3. Consolidation phase 1 (about 5 minutes)
4. Repetition phase 1 (about 5 minutes)
5. Consolidation phase 2 (about 10 minutes)
6. Repetition phase 2 (about 10 minutes) [7].

Caine and Caine [2] propose three fundamental lesson phases:

1. Orchestrated immersion (creation of a learning environment that fully immerses the students);
2. Relaxed alertness (optimal state of learning that combines high challenge and expectations with low threat, confidence, competence and intrinsic motivation.
3. Active processing and personal engagement of students.

3. BRAIN-BASED TEACHING METHODS
One of the key findings of neurodidactics is that knowledge cannot be transferred but has to be newly created in the brain of each student. That means that learning is always an active process where knowledge has to be constructed. This corresponds to the approaches of progressive pedagogy and constructivism [1, 3]. The teaching and learning methods proposed in these approaches contain some neurodidactical principles and they are effective [8].

Discovery Learning: Instead of getting detailed instructions from a teacher, discovery learning focuses on "[…] teaching things to oneself, in order to solve one's own problems." [9].

Social or Observational Learning [10] means learning by imitating and, is based on the so-called mirror neurons that are active whenever observing and imitating others.

Learning by Doing wants the students to be active, which is more effective than teacher-centered instruction [6, 11, 12].

Learning by Teaching: "Whoever explains, learns!" [6]. By teaching others learners recall information from memory, which restarts the whole memory process and enhances retention [11, 13].

COOL – Cooperative Open Learning: The COOL teaching concept, initiated by a team of Austrian teachers, is based on the Dalton Plan and its key principles *freedom of choice*,, *cooperation* and *budgeting time* (self-responsibility of the learning process). During COOL lessons the students work independently on written assignments. They can decide on when, where, with whom and how to solve the tasks until a predefined deadline. This requires a change in the role of teachers, who act as coach or tutor [16].

4. BRAIN-BASED TEACHING PRACTICE

4.1 Brain-based Methods In Creative Projects

One regional part of the Austrian teacher support program IMST[1], "Informatik kreativ unterrichten" (Creative Informatics Teaching) aims at fostering creativity in computer science education by funding creative school projects. The majority of these projects use brain-based teaching methods. One example is described here.

The project "Let's make the adolescents talk", which was carried out in a vocational school for computer scientists, is based on Learning by teaching. The students had to create short creative podcasts or webcasts about different topics like algorithms, object orientation or databases. Besides the aspect that learning by teaching others is very effective the project leader assumed that for adolescents it might be easier to understand the explanations of their peers who use the "same" language. During the design and the production of these "micro learning-modules" the students were very motivated and active, discussed the topics and tried to help each other to understand the concepts. With each step in the project they could also increase their self-esteem, an important factor in learning [14], because they were proud of their products [15]. Furthermore, the multimedia learning modules supported the memory process by taking benefit of the modality effect [16].

4.2 Implementation of a Brain-based Concept

One example of a teaching concept integrating neurodidactical principles is "Brain-based Programming" [4]. This concept has been developed in order to improve the learning outcomes of introductory programming courses at university level and implemented last winter term in one of seven parallel courses (90 minutes/week). It is mainly based on *COOL* (Cooperative Open Learning) using the method of pair programming, *Discovery Learning* with reading exercises and step-by-step-instructions as well as *Learning by Teaching*. Students with good programming skills (*professionals*) or some competencies (*amateurs*) "worked" as peer tutors and supported there al *beginners* in small learning groups during the first two parts in the lessons: the *question* and the *discovery phase*. In the third phase, the lab, all students worked in pairs and tried to solve different programming exercises of their choice in the sense of pair programming. In this way all students were active, always according to their individual preconditions and competencies [4]. The results of the pilot study show that considering brain-based principles in teaching programming can improve learning and that it is worth continuing and extending this concept.

5. DISCUSSION AND OUTLOOK

Brain-based teaching and learning is not a panacea as some commercial publications may claim, but it can help students in learning difficult and complex subject matters. Many proposals of neurodidactics are not new but refer to well-known teaching concepts like constructivism and progressive education and teaching methods like cooperative and discovery learning or learning by teaching. But they are scarce and further empirical research is necessary. The pilot project of "Brain-based Programming", for example, could demonstrate that a neurodidactical approach on different levels (lesson structure, classroom setting, teaching methods and material) can be successful. In a follow-up project that is being planned now we want to test the adapted concept in schools and have a closer look at specific aspects like the use of pattern recognition in classroom or the impact of cooperative methods from the point of view of neurodidactics. Future research should also study the effectiveness of different brain-based methods.

[1]IMST = Innovations Make Schools Top

6. REFERENCES

[1] Gülpinar, M. A. 2005. The Principles of Brain-Based Learning and Constructivist Models in Education. *Educational Sciences: Theory & Practice* 5 (2), Nov. 2005, 299-306.

[2] Caine, G. and Caine, R. 2007. *Natural learning: The basis for raising and sustaining high standards of real world performance.* Position Paper: Natural Learning Research Institute. Retrieved on 19/07/2013 from: http://www.seeing learning.com/wp-content/uploads/2013/05/Position.pdf.

[3] Bruer, J. T. 2007. In search of... brain-based education. In *The Jossey-Bass Reader on the Brain and Learning,* K. W. Fischer, M. H. Immordino-Yang, Ed. John Wiley & Sons.

[4] Sabitzer, B., Strutzmann, S. 2013, in press. Brain-based Programming. In *Proceedings of IEEE Frontiers in Education, October 2013*, Oklahoma City, Oklahoma, US.

[5] Sabitzer, B.and Antonitsch, P. 2012. Of Bytes and Brain. – Informatics Meets Neurodidactics. In *INTED 2012 Proceedings*, Gómez Chova, Candel Torres, López Martìnez, (Eds.) IATED, pp. 2003-2012, Barcelona.

[6] Sousa, D. A. 2006. *How The Brain Learns.* Third edition. Corwin Press, Thousand Oaks, California.

[7] Theurl, P. 2009 "Lernen unter Selbstkontrolle" – Entspannung und Kontemplation in Schule und Unterricht. In *Neurodidaktik: Grundlagen und Vorschläge für gehirngerechtes Lehren und Lernen, Second edition,* U. Herrmann, Ed. Beltz Verlag, Weinheim und Basel, Germany.

[8] Hattie, J. 2009. *Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement.* Taylor & Francis, London, New York.

[9] Baldwin, D. 1996. Discovery learning in computer science. In *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*. ACM, New York, pp. 222 - 226.

[10] Bandura, A. 1976. *Lernen am Modell.* Klett, Stuttgart.

[11] Elsenbaumer, S. 2013. *Neurodidactics in Practice. A Practical Approach to Introducing Informatics into a Primary School in a Brain-based Way.* Unpublished Diploma Thesis, Alpen-Adria-Universität Klagenfurt.

[12] DuFour, R., DuFour, R., Eaker, R. and Many, T. 2006. *Learning by Doing. A handbook for professional learning communities at work.* Solution Tree, USA.

[13] Grzega, J. (n.d.). *Lernen duch Lehren.* Retrieved May 26, 2013, from http://www.ldl.de/.

[14] Geake, J. G. 2009. *The brain at school: Educational neuroscience in the classroom.* Open University Press, Berkshire, New York.

[15] Kölblinger, I. 2013. *Lassen wir die Jugend sprechen (Let's make the adolescents talk).*Unpublished project report.

[16] Sabitzer, B., Pasterk, S. and Elsenbaumer, S. 2013. *Informatics is COOL – Cooperative and COmputer-assisted Open Learning.* Submitted to WIPSCE Conference, November 2013, Aarhu.

Computational Thinking in CS Teaching Materials: a pilot study

Erik Barendsen
Radboud University Nijmegen
ICIS, PO Box 9010, 6500 GL Nijmegen
The Netherlands
e.barendsen@cs.ru.nl

Idzard Stoker
Radboud University Nijmegen
ICIS, PO Box 9010, 6500 GL Nijmegen
The Netherlands
i.stoker@student.ru.nl

ABSTRACT

This poster reports on research in progress. We develop a coding scheme to analyze teaching materials with respect to Computational Thinking (CT) content. In this pilot study, we apply the coding scheme to a sample of Dutch materials for Computing Science. The framework turns out to be useful for both global and in-depth analysis of CT content.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Literacy*

General Terms

Human Factors, Algorithms

Keywords

Computational thinking, computing education

1. INTRODUCTION

It has been widely recognized that digital literacy is important in our present day information society. The term *Computational Thinking* (CT), coined by J. M. Wing [6], refers to a set of analytical skills needed to apply computers in an effective way. According to Wing, CT consists of a number of mental processes and tools originating from computer science, such as recognizing algorithmic and data aspects in a problem. CT can be regarded as 'CS in context'.

Several operational definitions of CT have been suggested, notably by the Computational Thinking Task Force of CSTA [2]. In addition, CSTA proposes a 'vocabulary' of basic CT subjects (data collection, data analysis, data representation, problem decomposition, abstraction, algorithms & procedures, automation, simulation and parallelization), together with a suggested development throughout the K-12 grades and examples of so-called *CT Learning Experiences.*

Koli Calling '13, November 14–17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526995

The study reported in this paper is part of a bigger design research project on Computational Thinking in Dutch secondary education, aiming at development of teaching materials, assessments and instruction strategies suitable for the Dutch context. In the first phase of the project we analyze the current state of affairs of CT in Dutch CS education. This phase includes an analysis of teaching materials and teachers' pedagogical content knowledge (PCK, [5, 4]).

This study focuses on the teaching materials. We expect to find aspects of CT in CS text books, albeit probably not in a coherent way, as CT was usally not explicitly intended as a learning outcome at the time of writing. We take the CSTA operationalization as a starting point. We intend to refine the categories to allow for a deeper analysis and comparison of teaching materials. The refinement will also be used to conduct PCK interviews with teachers at a later stage.

Our research questions are as follows. (1) How can the CT content of teaching materials be characterized? (2) Which aspects of CT are present in the materials used in Dutch CS education? (3) Which variation with respect to content matter can be recognized in the CT parts of the materials?

We carried out an exploratory study on a set of modules taken from established Dutch CS text books: *Algorithms and programs, Programming languages, Relational databases, Information modelling, Organization and projects*. We also analyzed a new module *Model checking* and a module on programming for first-year general science students. The latter was included because it was constructed as a series of tasks with a context-based approach [3, 1] in mind.

2. METHOD

2.1 Development of the coding scheme

The list of nine CT categories of CSTA was extended to a coding scheme in two steps. In the *first step* we analyzed the examples in the *CT Learning Experiences* of [2]. We attached short descriptions describing the CT content in more detail. When all learning experiences were characterized, we reduced the short descriptions within each category to a limited number of subcategories. In the *second step*, we used the refined coding scheme to code a sample of module fragments. The codes were checked for completeness (are all CT elements covered?) and mutual exclusion (is there any overlap?). Where necessary, subcategories were added.

2.2 Analysis of the teaching materials

The resulting coding scheme was used to analyze the modules. We coded the texts, assigning subcategories to relevant text fragments. We determined the relative frequencies of categories and subcategories in our sample. Then we performed a more in-depth qualitative analysis, determining the variation within each (sub)category.

3. RESULTS

3.1 Coding scheme

In the first step, we found 7 fragments in the category *Abstraction*, 8 in *Algorithms & Procedures*, 6 in *Automation*, 5 in *Data Analysis*, 4 in *Data Collection*, 3 in *Data Representation*, 1 in *Parallelization*, 2 in *Problem decomposition*, and 7 in *Simulation*. After reduction of the descriptions, we obtained the subcategories in Table 1. The subcategories displayed in italics were discovered in the second step, in which we used the intermediate coding scheme to code 5 fragments of the modules.

category	*subcategories*
Data Collection	Collecting data *Selecting relevant data*
Data Analysis	Drawing conclusions Finding patterns Making sense of data
Data Representation	Arrange data for analysis Organize/represent data
Problem Decomposition	Breaking down tasks *Merging subtasks*
Abstraction	Finding characteristics Creating models
Algorithms & Procedures	Making sequential steps in a specific order Understanding and changing algorithms Making decisions in algorithms *Implementing algorithms*
Automation	Recognizing different forms of automation Recognizing the advantages of automation
Simulation	Creating pseudo-code Creating models of processes Experimenting
Parallelization	Combine/merge activities

Table 1: Coding scheme

3.2 Analysis of teaching materials

The modules turned out to be quite complementary with respect to CT content. This is not surprising since each module focuses on a specific CS subject, and only touches upon the CT aspects directly related to that subject. We give some examples of the quantitative results.

Remarkably many codes have been assigned to *Programming for Science*: 111 of 234 codes, almost 50%. This contrasts the size of the module: *Programming for Science* has 20250 words, whereas the textbook modules together have 43056, more than twice as many. Thus, the density of codes in *Programming for Science* is much higher than that of the textbook modules.

We also see that *Programming for Science* has the highest coding score on *Algorithms & Procedures.* The module *Algorithms and programs* was also assigned relatively many codes in this category. Both modules focus on programming. It is remarkable that no codes in this category were assigned to *Programming languages*, although this module also concerns programming. Table 2 shows the occurrences of subcategories in the two modules. It appears that the CT-contents of the modules are highly complementary.

subcategory	*A&P*	*PfS*
Implementing algorithms	0	23
Making decisions in algorithms	0	2
Making sequential steps in a specific order	17	6
Understanding and changing algorithms	1	38

Table 2: Subcategories of Algorithms & Procedures compared

A full paper will elaborate on the findings of the in-depth qualitative analysis.

4. FUTURE WORK

We intend to further document the coding scheme and examine the inter-coder reliability of the instrument. Moreover, we plan to analyze more (international) texts using the resulting scheme.

One could imagine that the coding method can be turned into an instrument to be used not only in a descriptive way (to analyze CT content), but also in a more normative way (assessing CT content of new materials).

The module *Programming for Science* shows a remarkably high density of CT content. A plausible explanation is the explicit context-based approach. We are planning to pursue this direction in future research, making use of the insights about context-based eduction obtained in the science education community.

5. REFERENCES

[1] J. Bennett and J. Holman. Context-based approaches to the teaching of chemistry: What are they and what are their effects? In J. K. Gilbert, O. De Jong, R. Justi, D. F. Treagust, and J. H. Van Driel, editors, *Chemical education: Towards research-based practice*, pages 165–184. Dordrecht: Kluwer, 2002.

[2] CSTA. Computational thinking teacher resources, second edition, 2011. Retrieved from `http://csta.acm.org/Curriculum/sub/CompThinking.html`, July 2013.

[3] J. K. Gilbert. On the nature of 'context' in chemical education. *International Journal of Science Education*, 28(9):957–976, 2006.

[4] S. Magnusson, J. Krajcik, and H. Borko. Nature, sources, and development of pedagogical content knowledge for science teaching. In J. Gess-Newsome and N. G. Lederman, editors, *Examining pedagogical content knowledge*, pages 95–132. Dordrecht: Kluwer, 1999.

[5] L. S. Shulman. Those who understand: Knowledge growth in teaching. *Educational researcher*, 15(2):4–14, 1986.

[6] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

Why is Big-O Analysis Hard?

Miranda Parker, Colleen Lewis
Harvey Mudd College
301 Platt Blvd
Claremont, CA, USA 91711
mparker, lewis@cs.hmc.edu

ABSTRACT

We are interested in increasing comprehension of how students understand big-O analysis. We conducted a qualitative analysis of interviews with two undergraduate students to identify sources of difficulty within the topic of big-O. This demonstrates the existence of various difficulties, which contribute to the sparse research on students' understanding of pedagogy. The students involved in the study have only minimal experience with big-O analysis, discussed within the first two introductory computer science courses. During these hour-long interviews, the students were asked to analyze code or a paragraph to find the runtime of the algorithm involved and invited students to write code that would in run a certain runtime. From these interactions, we conclude that students that have difficulties with big-O could be having trouble with the mathematical function used in the analysis and/or the techniques they used to solve the problem.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms
Human Factors

Keywords
Big-O, runtime analysis, algorithmic complexity

1. INTRODUCTION

Big-O is used in computer science to estimate the upper-bound of an algorithm's runtime [2]. We assume that big-O is important to students' ability to write efficient code. However, from our experience, students appear to have a fair share of difficulty with this subject. Big-O has been shown to be the most difficult topic for college students at the introductory programming level [3]. However, it's also seen as the least relevant topic at this level, and thus does not get all of the attention it deserves in terms of understanding why it is difficult [3].

We created an interview protocol designed to investigate students' understanding of big-O analysis. During the interview the students were asked to analyze code or a textual description to find the runtime of the algorithm involved. We also invited students to write code that would operate in a certain runtime. Afterwards their answers and actions were qualitatively analyzed in order to gain insight into their understanding of big-O.

Koli Calling'13, November 14-17, 2013, Koli, Finland.
http://dx.doi.org/10.1145/2526968.2526996
ACM 978-1-4503-2482-3/13/11.

2. ANALYSIS

In the analysis we focus on portions of an explanation Ethan (a pseudonym) gave for why he feels he does not understand big-O analysis. This was during the second attempt of a problem that asked him to write a function that runs in O(*log*(n)) time. He initially passed on the problem, but since we had extra time during the interview he chose to reconsider it. He worked towards a solution to the problem while he was mentioning what parts of big-O were hard for him.

From this discussion, we develop the idea that difficulty with big-O derives from two sections of understanding: the mathematical function in the analysis and the technique used to solve the analysis, be in plug-and-chug or reductive thinking [1].

2.1 Episode One

2.1.1 Data

01 Logs, they're always involving logs.
02 **Logs** are like the **least friendly thing**.
03 Like, with you know **n squared I could easily point to life** and
04 say that's an example of something being squared, but
05 a **log** is really **less tangible**, you know?
06 like, and even like trig functions, like sine, cosine, tangent, you can say
07 'Oh, triangles.'
08 Like uh I don't know log and uh
09 also I think part of it is just me.

2.1.2 Analysis

From the transcript presented above, it can be concluded that Ethan can have different levels of difficulty with different mathematical functions. This can be deduced from his differentiation in difficulty between squares and logs. He sees logs as "the least friendly thing" and "less tangible." This helps to provide context for why Ethan initially did not answer the logarithmic runtime problem, since he could tell from the problem statement what mathematical function it involved and he knew that he did not completely understand that function.

2.2 Episode Two

2.2.1 Data

01 Um at least **it helps me**
02 when I put the number six in there and
03 see, actually sort of count it and
04 reason it in my head with a **tangible** number and
05 then put it in **variable form**.

2.2.2 Analysis

Ethan's plug-and-chug technique for solving the problem may lead to difficulties in understanding the problem. He feels a sense of comfort, expressed in line one of this episode, with using this technique. He likes it because it uses "a tangible number." This is connected to the idea of having difficulties with mathematical functions. This is because plugging a number into a coded function can be more or less helpful in the big-O analysis depending on the mathematical function. For example, it may be easier to notice a number being squared than a number that has the log taken of it. However, just because a student understands a mathematical function does not imply that the student also has a valid and dependable technique for solving for the runtime. In other words, even if Ethan understood logarithms, his chosen technique for solving a problem might still give him difficulties with the runtime analysis.

2.3 Episode Three

2.3.1 Data

01 I know loops and recursion and stuff has n attached to them, but
02 I don't know how to mix and match them, and
03 I don't know what **corresponds** with what and
04 what logs **correspond** to.
05 I know there's some type of **correspondence** between **a type of programming thing** and **logs**, or whatever.

2.3.2 Analysis

Ethan desires a connection between the abstract (big-O analysis) and the concrete (algorithms, structures, etc.). He recognizes that certain programming structures or algorithms have certain runtimes, expressed in line one of this episode. However, he does not know all of these correspondences, and admits as much for logs in line four. A student could plausibly understand logarithms in a math context but not relate logarithms to inherently binary structures in a computer science context.

2.4 Analysis Summary

Ethan's interview led us to hypothesize about two possible areas that students could have difficulty in when learning big-O analysis. We are led to this conclusion through Ethan's discussions of tangibleness (leading to the plug-and-chug technique) and correspondence (the reductive thinking technique), which point to key parts of big-O analysis that, if misunderstood, could increase the difficulty of runtime analysis from the student's perspective. Ethan's dislike of logarithms carried through all of these areas, but that does not imply that the mathematical function and the solution technique are one and the same in terms of difficulty. The mathematical function interacts with the solution technique, including plug-and-chug and reductive thinking techniques, to create difficulty with big-O analysis, as seen in Figure 1.

3. CONCLUSION

From the analysis, we conjecture that two things work together to affect a student's understanding of big-O analysis: the mathematical function used in the big-O analysis and the technique the student uses to find the solution to the problem, be it plug-and-chug or reductive thinking [1].

In terms of the mathematical function, students seemed to have different experiences in solving a big-O analysis problem depending on what mathematical function was involved, such as *log*(n) or n^2. This was most evident when the student was asked to write a function that ran with a certain big-O runtime.

Additionally, there were various techniques the student used to find a big-O runtime, some of which produced more correct answers for a student than others. In some cases, students would plug values into the algorithm and then try to extrapolate the runtime from the number of steps the algorithm took to produce the return value. In other cases, students had an easier time with a problem when they could determine a pattern in the algorithm that they seen before, such as a certain set of recursive calls, and associate it with a certain runtime.

The data suggests that the mathematical function and the technique used in solving the problem are connected, since the technique that a student uses may produce a wrong answer depending on the mathematical function that is involved. For example, some students found it much easier to detect a polynomial pattern than a logarithmic pattern.

This study takes the first step towards understanding how students reason about big-O. Although only a few examples are provided, these examples of why big-O is difficult can still make a difference in the pedagogy of this topic. Furthermore, this research can easily be expanded to explore more areas of big-O with which students struggle.

4. REFERENCES

[1] Armoni, M., Gal-Ezer, J. and Hazzan, O. 2006. Reductive Thinking in Undergraduate CS Courses. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education* (ITICSE '06). ACM, New York, NY, USA, 133-137. DOI=10.1145/1140124.1140161

[2] Dasgupta, S., Papadimitriou, C.H. and Vazirani, U.V. 2006. *Algorithms*. McGraw-Hill.

[3] Schulte, C., and Bennedsen, J. What Do Teachers Teach in Introductory Programming? 2006. In *Proceedings of the second international workshop on Computing education research* (ICER '06). ACM, New York, NY, USA, 17-28. DOI=10.1145/1151588.1151593

An open approach for learning educational data mining

Ilkka Jormanainen
School of Computing
University of Eastern Finland
P.O.BOX 111
FI-80101 Joensuu, Finland
ilkka.jormanainen@uef.fi

Erkki Sutinen
School of Computing
University of Eastern Finland
P.O.BOX 111
FI-80101 Joensuu, Finland
erkki.sutinen@uef.fi

ABSTRACT

The Open Monitoring Environment (OME) allows a teacher to monitor, model and, thus, understand, the learning process based on the real data rising from an educational robotics class. The OME uses a novel educational data mining approach where teachers are empowered to create rules to extract pedagogically and contextually meaningful patterns of actions from a raw data flow. The OME has been tested in various educational robotics settings and our results indicate that the data mining approach in the OME is easily accessible even for users who are not computer science experts. We propose that the OME could be utilized in computer science education as a platform for empirical, hands-on approach for teaching and learning educational data mining.

Categories and Subject Descriptors

K.3.2 [**Computers and Education]** Computer and Information Science Education – *computer science education, information systems education.*

General Terms

Design, Experimentation, Human Factors.

Keywords

Educational data mining, robotics, learning analytics

1. INTRODUCTION

Educational data mining and learning analytics have recently become important trends in educational technology research and industry. It is evident that one of the catalysts that create a need for analyzing learning data deeply is the changing role of modern technology in regular classrooms. Technology-enhanced learning environments, such as educational robotics or tablet-based learning tools, often emphasize project-based working methods, and learning processes in such settings generate vast amounts of logging data. There is a need to have tools to explore data, and more importantly to understand reasons behind phenomena in the classroom. As a result of our recent research on analyzing learning data rising from educational robotics classrooms, we have developed an Open Monitoring Environment (OME) that automatically gathers data from the learning setting and empowers a teacher to processes this data to collections and visualizations that are relevant for the current learning context. The OME is driven by an open data mining process in its core, and all steps of the data mining are accessible and under control of a teacher using the system. Taking into account the positive results from our previous research with the OME, we propose in this paper a hands-on approach for learning and teaching basics of educational data mining with the OME.

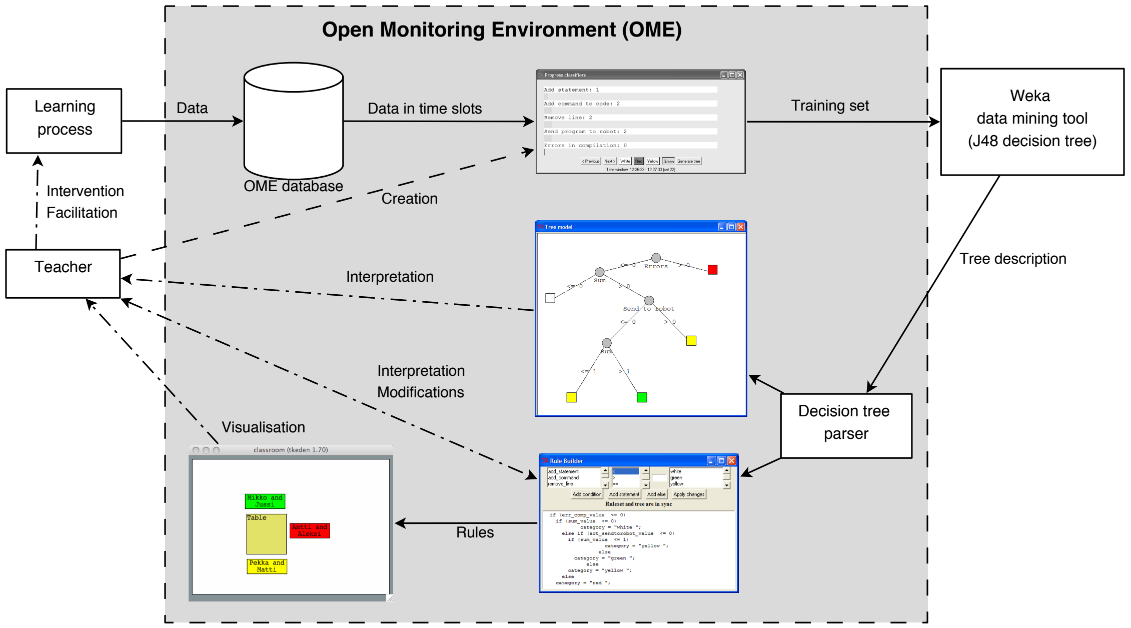
Koli Calling '13, Nov 14-17 2013, Koli, Finland
ACM 978-1-4503-2482-3/13/11.
http://dx.doi.org/10.1145/2526968.2526997

2. BACKGROUND

Data mining tools have a recognized status as a part of modern learning environments. Most of the work in data mining in educational systems contributes to student assessment and course adaptation (for example [2] and [3]). Clustering and classification are mostly used data mining techniques in learning environments because of the nature of the problems. However, learning environments where data mining processes and results would be explicitly visible to the users are rarely reported in the research literature. Rather than that, the data mining takes place in a black box. Systems presented in literature are usually based on approach where a domain expert manually labels data sets and builds models describing the learning activities in advance. This is a time-consuming and error prone process especially in exploratory learning environments with open-ended problems, because prior definitions of relevant behaviors are necessarily not available for labeling data and training the model [1], and a substantial amount of data may be required to build a model. We have shown in our previous research that the OME produces valuable insight into the progress of the learning activity even on relatively small datasets [4].

3. OPEN MONITORING ENVIRONMENT

The core idea of the OME (Figure 1) is to help the teacher to detect the right moments for intervention in a robotics class. In this way, the OME potentially helps the teacher to build his or her intervention strategies. The OME features an open data mining approach, which automates data collection and preprocessing and allows the teacher to combine data streams from the robotics classroom with his or her own observations arising from the learning process. The OME provides tools for classifying events (essentially, creating training set), and creating a classifier and evaluating the outcome. In this way, the environment encourages the teacher to model the learning process in the way that best suits his or her preferences and the current learning setting. We have shown that the open data mining approach of the OME works in educational robotics context [5]. There are also indications that the OME approach is suitable for analyzing data from other learning settings, such as tablet-based science teaching. We believe that the open data mining supports any kind of learning setting as long as suitable logging data is available and pedagogically meaningful connections can be made between data and events in the classroom.



```
Add statement: 1
Add command to code: 2
Remove line: 2
Send program to robot: 2
Errors in compilation: 0
```

Time window: 12:26:33 - 12:27:33 (set 22)

Ruleset and tree are in sync

```
 if (err_comp_value  <= 0)
   if (sum_value  <= 0)
           category = "white ";
     else if (act_sendtorobot_value  <= 0)
       if (sum_value  <= 1)
                   category = "yellow ";
                 else
         category = "green ";
             else
       category = "yellow ";
     else
   category = "red ";
```

Figure 1. Components of the Open Monitoring Environment (modified from [4]).

4. OME IN EDM TEACHING

Educational data mining (EDM) tools provide interesting possibilities for more in-depth exploration of learning processes. The tools used to teach basics of EDM should be powerful but at the same time easy enough to learn and operate. The students necessarily do not have skills to modify the functionality of the tools. In many cases, the tools provide mainly pre-defined set of rules and models [5] and modifications are impossible to apply without a complicated and time-costly process. Empirical approach of the OME and hands-on experiments with real data sets in learning environments help the students to gain understanding about causations that follow their choices in different parts of the data mining process. Based on the positive results of previous research [5], we envisage that the OME principles could be applied also when teaching basics of educational data mining and learning analytics. The OME uses the following identifiable steps in the data mining process.

1. Collecting data from the learning process
2. Preprocessing the data
3. Classifying the data into a training set
4. Creating a classifier with the training set
5. Judging students' progress against the created classifiers

The steps are rather general, and they can be found in many data mining driven learning environments. By following these steps, the OME provides a simple and straightforward way to learn the basic educational data mining concepts with simple classification schemes, such as J48 descision tree or BFTree.

The small data sets in the OEM make possible to iterate the steps many times during a learning session and the users have an immediate view how their choices with classification affect to the outcome of the data mining algorithm. The OME also provides functionalities for "replaying" previously collected and saved data. This feature was built for analyzing learning data subsequently. This approach could be utilized if real-time learning settings, such as educational robotics classes, would not be available. The OME is not fixed to work in educational robotics classes only, but any technology-based learning environment that produces suitable data about students' activities can be used. The open nature of the OME ensures that it can be easily to modified and deployed in different contexts.

5. CONCLUSION

We have presented the Open Monitoring Environment that allows a teacher working in a technology-based learning environment to adopt roles of software developer or domain expert in his or her work. In this way, the OME essentially mixes teaching and development activities together and allows a flexible and highly contextualized learning environment for observing students' activities. The fundamental steps of an educational data mining processes can be found in the OME. We propose that the OME could be utilized not only as a tool for helping teachers in robotics classes, but also as a learning environment for educational data mining courses when learning and teaching how to use simple classifying schemes. The simplicity of the environment and rapid iterations of the OME could bring added value for educational data mining teaching.

6. REFERENCES

[1] Amershi, S., and Conati, C. 2009. Combining Unsupervised and Supervised Classification to Build User Models for Exploratory Learning Environments. *Journal of Educational Data Mining* 1, 18–71.

[2] Gobert, J., Sao Pedro, M., Baker, R., Toto, E., and Montalvo, O. 2012. Leveraging Educational Data Mining for Real-time Performance Assessment of Scientific Inquiry Skills within Microworlds. *Journal of Educational Data Mining* 4, 111–143.

[3] Kristofic A., and Bieliková, M. 2005. Improving adaptation in web- based educational hypermedia by means of knowledge discovery. *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia, HYPERTEXT '05*, 184–192.

[4] Jormanainen, I., and Sutinen, E. 2013. Role blending in a learning environment supports facilitation in a robotics class. *Journal of Educational Technology & Society*, to appear.