The collection of the CSERC submissions 2019

This package contains the submissions for CSERC 2019. It contains the submissions that were accepted for paper presentation as well as the poster presentation.

Please note that CSERC publishes post-proceedings. This means that not all the reviewers' comments are processed yet.

Program Chairs
Ebrahim Rahimi, Open University, The Netherlands
Dave Stikkolorum, University of Leiden, The Netherlands

Table of Contents

Tuesday 19 November - Paper Session (6) [10:30 - 12:30]

Experience Report: Creating Tutorial Materials by Integrating Drawing Tablet and Video Capturing/Sharing

ABSTRACT

We report the experience of adopting an innovative technique for creating tutorial videos which complement lectures and facilitate students' learning. Our technique relies on: **1)** preparing starter pages consisting of code fragments or writings/figures on a drawing tablet; **2)** illustrating complex ideas on the drawing tablet; **3)** recording all computer desktop activities (e.g., development of code on a programming IDE, illustration on the drawing tablet); and **4)** sharing the recorded tutorial videos with students online. Our technique has been adopted in creating tutorial series for four Computer Science and Engineering courses, ranging from the first year to the third year. Analytics of these online tutorial videos is presented to show the average amount of time which each registered student spent on watching them. Course evaluation results indicate that our technique is perceived as effective for achieving the course learning outcomes. Comparison of students' performance on complex topics (arrays and loops) also indicates a positive impact of our approach.

KEYWORDS

Large Class; Laboratory Assignments; Tutorial Videos; Computational Thinking; Instructional Technologies

1 INTRODUCTION

It is challenging to teach complex computational thinking [13, 19, 20, 28] (e.g., arrays, loops, object-oriented thinking) and software design principles (e.g., design by contract, object-oriented design patterns leveraging polymorphism and dynamic binding) in undergraduate courses, where: students have limited prior exposure to the course content; and the class size is typically large (e.g., 400+ for first-year courses, 150+ for second-year courses, and 100+ for third-year courses in our home department). On the one hand, many students encounter obstacles to full comprehension of course content because the class size restricts the instructor's intentional pauses and student interactions during lectures.

On the other hand, it is very common for a Computer Science (CS) or Engineering course to have a weekly laboratory (lab) component, but there is a significant gap between lecture materials and the pre-requisites (on concepts and skills) for completing the weekly lab assignments. There are two inherent limitations of in-class instructions, making it difficult to implement, among lectures, a logical decomposition of the taught subjects. First, lecture hours are *fixed*, so that very often we have to interrupt the discussion simply because the class time has run out. Second, lecture hours are *limited*, so that in-depth discussion and illustrations of certain technical insight cannot be accommodated in class.

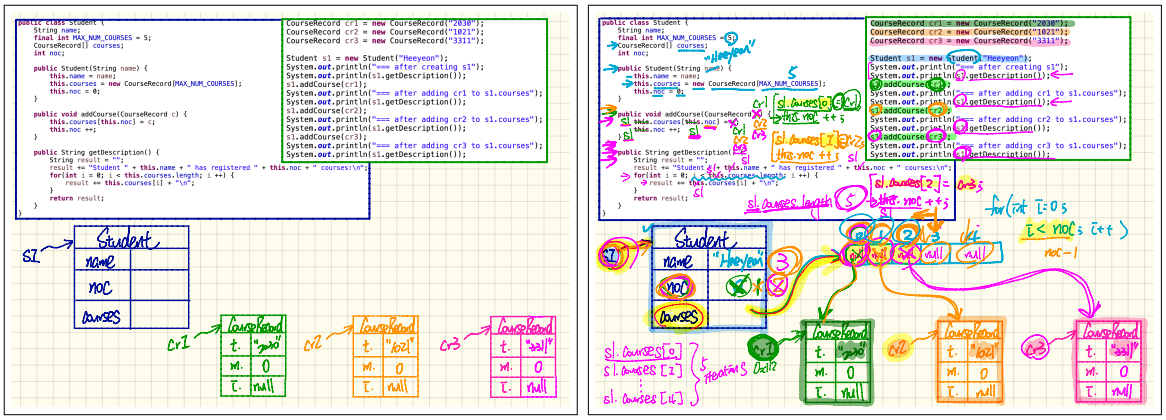
How can we make the in-depth and detailed illustrations accessible to students for their self-paced study outside the classroom, so as to help them complete the weekly lab assignments? We propose to address the two limitations of lectures by creating series of tutorial videos, each of which: **1)** being sequentialized according to the suitable logical order (as judged by the instructor); and **2)** making in-depth remarks and illustrations on concepts and examples. For **2)**, such remarks and illustrations reflect the instructor's insight into the taught subjects, and are thus a valuable aid to student learning. To create these tutorial videos, we have adopted, in four undergraduate CS and Engineering courses, the integrated use of: **1)** a drawing tablet for illustrating concepts and code examples; **2)** a program for recording all desktop activities, such as the slide presentation as well as illustrations on the tablet and programming IDEs; **3)** a high-end studio microphone ensuring decent sound quality of the tutorial videos; and **4)** online access to recordings and notes for students to review before attempting their lab assignments.

The main contribution of this paper is a technique for creating tutorials videos on complex ideas. Our proposed technique is much more than recording the occasional and lightweight annotations on a slide show. Instead, our technique requires the instructor to plan and prepare starter artifacts (e.g., code fragments, figures) on the drawing tablet prior to starting the recording, which is more effective than setting up these artifacts on a conventional in-class/in-office blackboard or whiteboard "on the fly". Our proposed technique is novel in that it relies heavily on explaining and illustrating complex ideas on a drawing tablet, and that it relies on recording the process of building up complex examples (e.g., static software architecture, dynamic runtime execution) from scratch. Such explanations and illustrations represent the insight into the taught subjects, as well as its thinking process which, thanks to the recording, students can review as needed and thereby learn from. As an example, Figure 1b (p2) shows an annotated fragment of object-oriented code at the end of our illustration. The reasoning process of moving from Figure 1a to Figure 1b, through recording, can be reviewed by students whenever they need.

The rest of the paper is organized as follows. Section 2 summarizes topics of the courses in which we adopted the proposed approach. Section 3 introduces our proposed approach. Section 4 summarizes the tutorial series created using our approach. Section 5 proposes a general pattern for structuring tutorial videos and gives an example. Section 6 outlines the equipment requirements. Section 7 presents analytics of our online tutorial videos, as well as performance comparison and student feedback. Section 8 discusses the related works. Section 9 concludes the paper.

CSERC'19, November 2019, Larnaca, Cyprus

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn



(a) Before Annotations Began (26:33)

(b) After Annotations Ended (1:17:50)

Figure 1: Illustrating a Code Fragment on a Drawing Tablet (26:33 – 1:17:50 of https://youtu.be/gsZo6XyzJ6s)

2 TEACHING CONTEXT

We adopted our approach in four undergraduate CS and Engineering courses in the academic years of 2017 to 2019. Examples course topics are summarized below.

CS1A Mobile Computing and *CS1B OOP: From Sensors To Actuators* are the second-semester courses for, respectively, CS and Engineering[1] students at the first year. There are 400+ students registered in each of the two courses. In both CS1A and CS1B, students learn about basic computational thinking and object orientation, but through different means. In CS1A, students develop Android mobile apps using the Android Studio IDE (Integrated Development Environment), and visualize the effects of their Java programs on physical tablets. In CS1B, students use Phidget interface boards connected to hardware equipment such an LED light bulb and Theremin glove. Example topics covered in both courses are: **1)** elementary programming (variables, data types, assignments); **2)** conditionals; **3)** loops; **4)** primitive 1D/2D arrays; and **5)** object orientation (attributes, methods, classes, and class associations).

CS2 Advanced Object Oriented Programming (with 150+ students) is the first-semester course for both CS and engineering students at the second year. Students in CS2 are required to develop, test, and debug Java programs in the Eclipse IDE. Example topics covered in CS2 are: **1)** unit testing; **2)** code reuse and subtyping via inheritance; **3)** polymorphic assignments and dynamic binding; **4)** recursion; and **5)** asymptotic upper bounds (i.e., the big-O notation) of programs.

CS3 Software Design (with 100+ students) is a required course for third-year CS and Software Engineering students. Example topics covered in CS3 are: **1)** the Design-by-Contract (DbC) method for constructing object-oriented software (using loop invariants and variants, method preconditions and postconditions, and class invariants); **2)** the information hiding design principle (exemplified by the Iterator design pattern); **3)** object-oriented design patterns leveraging polymorphism and dynamic binding (e.g., composite, visitor, observer); and **4)** introduction to program verification.

[1]The majority of students in CS1B are from Computer Engineering and Software Engineering, whereas others are from Mechanical Engineering and Civil Engineering

3 THE PROPOSED APPROACH

We propose an approach as visualized in Figure 2 for preparing self-paced tutorial materials (i.e., videos, illustration notes, and program source code) which facilitate both the instructor's teaching and students' learning of complex ideas (e.g., computational, design, abstract). We discuss the proposed approach from two perspectives.

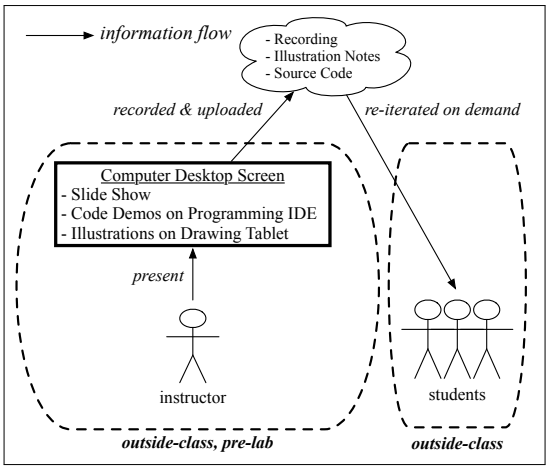


Figure 2: Approach: Effective Outside-Class Learning

First, the *instructor* chooses a set of relevant topics and creates a series of tutorial videos by presenting and illustrating those topics on their personal computer. Formats of presentation and illustration include the conventional slide show, code demonstrations on some

programming IDE, as well as tracing runtime execution of program code and explaining complex logic on a drawing tablet. The entire presentation and illustration occur as various desktop activities, which are are recorded, edited, and uploaded to an online sharing platform accessible to students (e.g., before attempting their weekly lab assignments). Given that illustrations on the connected drawing tablet become part of the desktop activities, the instructor is able to record *thorough* discussion of pre-selected topics (e.g., a code fragment, an example to be solved from scratch) using coloured annotations. The use of a drawing tablet for illustration, although analogous to the in-class use of a whiteboard or blackboard, has the advantages of greater visibility and being "re-playable" by students.

Second, *students* follow the tablet-focused presentation on the screen of a computer or a mobile device. This means that visibility is ensured (as opposed to the case of a whiteboard or blackboard which may have limited visibility due to the size of classroom, the position of students' seats, *etc.*). Students may also download the notes from the instructor's drawing tablet (exported as PDF documents) to review the illustration process.

4 TUTORIAL SERIES

In this section, we summarize how the proposed technique was adopted to create tutorial materials for the four undergraduate courses (see Section 2 for examples of the topics and learning outcomes of the course). Twelve series of 148 tutorial videos (with a total duration of approximately 59.5 hours) have been created for the purpose of students' learning. For each tutorial series, the ordering of videos with various lengths corresponds to a logical decomposition of the taught topics.

We divide these tutorial series into two categories: **1)** study materials for lab assignments (Section 4.1); and **2)** preparation materials for lab tests (Section 4.2). For completeness and review, all of our tutorial videos have been anonymized and re-uploaded to this channel: https://www.youtube.com/channel/UC6Ova_3A9A-iOb9vtpoZyzg/playlists. Links to specific playlists and videos will be referenced in the following two subsections.

4.1 Study Materials for Lab Assignments

Each of the four undergraduate courses has scheduled weekly lab sessions: CS1A and CS1B have 3 hours, whereas CS2 and CS3 have 1.5 hours. Weekly lab assignments are meant for students to acquire the required practical skills (e.g., programming, object-oriented thinking, design patterns). Thus, the level of difficulty of these weekly assignments should not target on any single scheduled session. Instead, challenging assignments are released at least one week before their due dates.

However, given the limited number of lecture hours[2], it is challenging to fill the conceptual gap between the covered topics in class and the pre-requisites of each lab assignment. Our proposed approach attempts to solve this problem by creating self-contained tutorials on the relevant topics:

- For CS1A, students are exposed to the Android Studio programming environment on Day One of the semester to develop working apps deployable on a tablet. Assuming that students have no prior experience on programming in Java, we created the first tutorial series [3], which uses the development of a simple Body Mass Index (BMI) calculator to illustrate aspects of an event-driven controller, a graphical user interface (with simple buttons and menu boxes), and an object-oriented model. As we progress the course, three further tutorial series were created to elaborate on: **1)** separating controller and model [4]; **2)** declaring a reference-typed attribute [8]; and **3)** declaring an array whose element is reference-typed [7].
- For CS1B, students are given weekly programming assignments, expected to be completed prior to their scheduled lab sessions. Each week students are assigned four to five tutorial videos to study (each of which guiding them through the reasoning process of developing fragments of code). The lab assignments are designed in such a way that students finishing the assigned videos are able to complete the actual lab assignments independently (or with minor assistance from the TAs or online forum). The tutorial series [12] contains 46 videos with the following design of roadmap:
  - Lab 1 (Videos 01 to 08): Simple Console Applications using Primitive Variable Assignments
  - Lab 2 (Videos 09 to 17): Simple If-Statements using the Boolean Data Type and Logical Operations
  - Lab 3 (Videos 18 to 19): A Simple Bank Account Application using Nested If-Statements
  - Lab 4 (Videos 20 to 24): Syntax and Semantics of *for*-Loops and *while*-Loops, Using Breakpoints and Debugger in the Programming IDE to Reveal Defects
  - Lab 5 (Videos 25 to 28): Basics of Arrays – Initialization using Loops and Tracing
  - Lab 6 (Videos 29 to 33): Deciding if Array Elements Universally/Existentially Satisfy Given Properties
  - Lab 7 (Videos 34 to 39): Object Orientation – Classes, Methods, Object Creations, and Method Calls
  - Lab 8 (Videos 40 to 46): Understanding and Implementing Associations between Classes
- For CS2, lab assignments require the use of classes from the Java collection library. To help students gain hands-on experience, as well as understanding the data structures of these collections (e.g., `ArrayList`, `HashTable`), we created a tutorial series [5] for them to review before attempting the lab assignments.
- For CS3, the composite/visitor design patterns are expected to be used in one of the labs and projects. Due to the limited number of lecture hours, we cannot guide students through the process of implementing these two advanced design patterns. Instead, we created a tutorial series [2] which implements and debugs a simple language processor using the two design patterns. For some labs and the project, we adopt a programming framework [23] which restricts all students to work under a given API, while being allowed to design their own programming modules that implement the common API. This programming framework is challenging for students to use due to its sophisticated architecture. To help students get started, we created a tutorial series [10] which

[2]There are two lecture hours per week for CS1A and CS1B, and three lecture hours per week for CS2 and CS3.

guides them through the use of the framework: architecture, extension, regression testing, and debugging.

In all CS1B, CS2, and CS3, we require students to apply the common software engineering practice of managing their projects using a revision control system such as Github. We created a tutorial series [6] to help them initiate a private account and workspace on their computers, as well as understand the workflow of common operations (e.g., `clone`, `commit`, `push`, `pull`).

4.2 Preparation Materials for Lab Tests

An important learning outcome of CS1A, CS1B, and CS2 is being able to write *runnable* programs (upon which students are assessed through automated unit tests). We emphasize to students that when they write an essay, if there are grammatical mistakes, it can still be interpreted by a human. Computer programs, on the other hand, just cannot be run (and hence unknown runtime behaviour) when they contain compile-time syntax or type errors.

In order to help students (especially those in CS1A and CS1B who have little prior programming experience and discipline) write compilable code during in-lab computer tests, we created a tutorial series [11] to guide them through the process. For CS1A and CS1B, we show how to write valid Java methods, given: **1)** an API **2)** a console application tester; and **3)** expected console outputs. For CS2, we show how to write valid classes/methods, given a set of unit tests.

Furthermore, in order to help students apply the above code-writing process to solve real problems:

- For CS1A and CS1B, we created a tutorial series [9] on going through the code and thinking process for solving twelve practice problems (involving arrays and loops).
- For CS2, we created a tutorial series [1] on developing a complete Birthday Book application (using two parallel arrays with methods for insertions, removals, and lookups).

5 A PATTERN FOR TUTORIALS

The majority of our tutorial videos (Section 4) conform to the following general pattern:

(1) **Present the Problem.** This can be done by using slide show to present the general problem to be solved, by using a programming IDE to demonstrate what is ultimately expected from the final (e.g,. software) product, or even by just pointing to what has been achieved in the previous tutorial video(s).
(2) **Sketch the Solution.** This part emphasizes the high-level thinking process, which can be illustrated on the drawing tablet, which may be pre-set with starter pages containing, e.g., code fragments, formulas, writings, figures.
(3) **Develop the Solution.** This is typically done in a programming IDE, or any software tool that is applicable to the course being taught.
(4) **Discuss the Solution.** This is to be done on starter pages on the drawing tablet. These starter pages may be set up either before the recording starts, or after Step 3[3] (by copying and pasting snapshots of parts of the solution developed). As the discussion progresses, we annotate on the starter pages to gradually build towards the solutions or conclusions.

[3]This alternative would require editing of the recordings.

The above pattern requires the instructor to determine what concepts/examples they will illustrate in the same series of tutorial videos, and then created the starter pages on the drawing tablet accordingly. Some steps (except tablet illustrations) may be omitted, and the order of choreographing these components may be adjusted according to the subject being taught.

As an example of instantiating the above pattern, consider Video 42 from the Java Tutorial Series for CS1B [12]: https://youtu.be/gsZo6XyzJ6s. This tutorial video shows how to implement `Student` objects, each of which storing an array of `CourseRecord` objects:

- 00:00 – 03:03: Summarize classes and methods developed in the previous videos, and briefly mention the extension to be completed in the current tutorial video.
- 03:04 – 26:32: Delay the sketching of solution and develop the programming solution on Eclipse right away.
- 26:33 – 47:05: On the drawing tablet, trace the developed code line by line, by visualizing object creations and method calls. This part was actually recorded separately and appended to the previous recording, so that it was possible to take snapshots of the code developed between 03:04 and 26:32, and to paste them to starter pages on the tablet.
- 47:06 – 50:49: On Eclipse, extend the code by introducing a second version of the implemented methods.
- 50:50 – 58:04: On the drawing tablet, sketch the idea about the second version of implementation.
- 58:05 – 59:57: On Eclipse, execute the second version of implementation, faulty due to a null pointer.
- 59:58 – 1:07:00: On the drawing tablet, illustrate how the runtime exception occurs and go back to Eclipse to fix the code accordingly.
- 1:07:01 – end: On the drawing tablet, justify why the final implementation works in two boundary cases: empty array vs. fully-occupied array.

Contrast Figure 1a with Figure 1b on page 2 to see how much illustrations of complex ideas has been performed in the above tutorial video. There are many more instantiations of the above pattern that can be found from our series of tutorials videos (Section 4).

6 ADOPTING THE APPROACH

Figure 3 summarizes how to assemble the various equipment to implement the proposed approach. Here we describe what we use, but the interested reader may choose other equipment with the same functionality.

Install the following software programs on your teaching computer (e.g., a MacBook): **1)** a presentation program (e.g., a PDF or PowerPoint reader) for your slides; **2)** a programming IDE as applicable to your course (e.g., Android Studio, Eclipse); **3)** a screen recording program (e.g., Active Presenter [24] for recording all desktop activities on the computer; and **4)** a program for projecting the screen of your drawing tablet (e.g., the free QuickTime Player).

Connect the following hardware to your computer: **1)** a high-end studio microphone (e.g., Blue Yeti [14]) using a USB cable; and **2)** a drawing tablet (e.g., iPad Pro) installed with an app for annotations

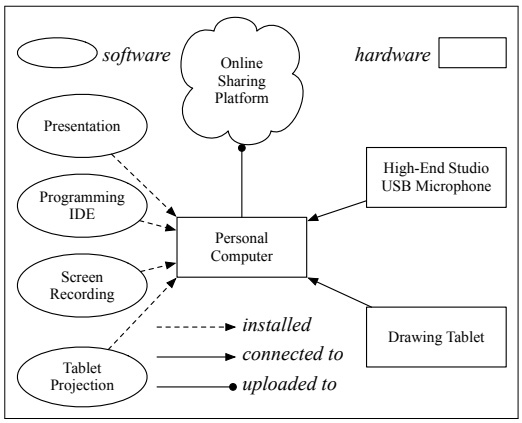


Figure 3: Adopting the Approach: Schematic View

(e.g., GoodNotes, Notability). For **2)**, a wired connection to the USB port is recommended for stability throughout the recording session. To project the screen of the drawing tablet to your computer desktop, if you use the QuickTime player and an iPad Pro, start a "New Movie Recording" and select your iPad as the camera.

When ready to start your tutorial recording, start the screen recording program and choose the connected microphone as the input device. When each recording session is finished, stop the screen recording, export it to an acceptable form (e.g., MP4), upload it to an online video sharing platform (e.g., YouTube), add it to the relevant playlist, and publish the link to students. The annotation app on your drawing tablet should allow you to export the annotated notes (e.g., Figure 1b, p2) as a PDF file.

7 EVALUATIONS

7.1 Student Engagement

Table 1 summarizes on YouTube, as of April 2019 when all courses were completed: **1)** the average number of minutes which each registered student spent watching the videos[4]; and **2)** the average completion rate (i.e., the ratio of average watch time to the duration of the tutorial series in question) accordingly. The average time is calculated based on one iteration of CS1A (Winter 2018 with 357 students[5]), one iteration of CS1B (Winter 2019 with 459 students), two iterations of CS2 (Fall 2017 with 99 students and Fall 2018 with 134 students), and three iterations of CS3 (Fall 2017 with 82 students, Fall 2018 with 88 students, and Winter 2019 with 95 students).

In Table 1, the measures of average watch times, and of completion rates accordingly, are arguably underestimates: apathetic students (e.g., those who never watched any of the videos) are not excluded. Consequently, a "good" student (e.g., those who attempted to watch these videos) in these courses should have a higher completion rate. Such engagement is confirmed by the majority of students in the (informal) midterm and (formal) end-of-semester course evaluations, expressing that these tutorial videos are helpful.

[4]We exclude [11], which is not directly related to computational thinking.
[5]There was an abnormal drop on the number of students due to a labour disruption.

Course	Series	Avg. Watch Time (Min)	Completion Rate
CS1A	[3]	304.58	86.52%
	[4]	42.48	45.34%
	[8]	31.25	65.04%
	[7]	75.83	18.33%
CS1B	[12]	365.36	21.15%
CS1A,B	[9]	35.33	14.23%
CS2	[1]	108.67	41.21%
	[5]	28.00	34.50%
CS3	[10]	58.9	48.59%
	[2]	25.1	22.80%
CS1B,2,3	[6]	35.06	43.62%

Table 1: Average Watch Time and Completion Rates

7.2 Improvement on Performance

Our proposed approach to making tutorial videos is meant for helping students understand complex computational thinking. One example is writing procedural code (in Java) using primitive arrays and loops. Table 2 shows the results of two in-lab computer tests in Winter 2018[6] from CS1A (taught by us, and our tutorial series on practice test solution [9] was supplied prior to the lab test) and CS1B (not taught by us, and no tutorial videos were supplied).

Course	Tutorials?	# of Students	Avg. Performance
CS1A	Yes	201	57.7%
CS1B	No	439	43.75%

Table 2: Performance Comparison: Arrays and Loops

As can be observed from Table 2, our tutorial solutions [9] had a positive impact on CS1A students. Although the two tests in Table 2 used different questions, the level of difficulty of tasks in CS1A (e.g., given as inputs two sorted arrays, return a new sorted array that merge them) is significantly higher than that of tasks in CS1B (e.g., given as inputs a list of numbers and an integer n, return a sublist whose values are larger than or equal to n).

7.3 Student Feedback

The anonymized online course evaluations[7] of CS1A, CS1B, CS2, and CS3 indicate that our tutorial videos (Section 4 and Section 5), despite their length, are perceived by students as being effective:

- "*... Personally, coming into the course knowing nothing about java, his online tutorials allowed me to understand the course material to the best of my ability. It was easy to follow, very in-depth with the explanations and most importantly, had relevance to the lab and course syllabus.*" [CS1B]
- "*... He [the author] puts in a lot of effort to get the students involved in the course material and also makes very long tutorial videos for us to really understand and concepts.*" [CS1B]

[6]The lab test for CS1B had more participants because it was taken prior to a labour disruption, whereas the lab test for CS1A was taken during the remediation period.
[7]We do not include numerical ratings because it is hard to distinguish between the impact of our in-class instruction and that of our tutorial series.

- "[The best things about this course are] Just the way he explains everything in the tutorial videos. Tracing code line by line makes it so helpful and easy to understand[.]" [CS1B]
- "*... video tutorials are extremely useful (although very long and time consuming to watch), and he took the time to explain concepts thoroughly and in detail which was helpful to complete labs and further my understanding.*" [CS1A]
- "*The tutorial videos were also great because he led us step by step of the way of a very new and complicated android application development process.*" [CS1A]
- "*The instructor did his best for understanding the course materials. Specially, all of his tutorial videos were very helpful to me to fulfill learning outcomes.*" [CS1A]
- "*The tutorial series and the recording system help me a lot in this course.*" [CS2]
- "*... really great to have the tutorial videos for the labs, it really helped us to where we needed to start for the lab.*" [CS3]

8 RELATED WORKS

We report a novel technique for creating tutorial videos which complement lectures and facilitate students' learning. Some recommendations for creating engaging tutorial videos from [16] — "continuous visual flow" and "the instructor speaks ... with high enthusiasm" — correspond to the guiding principles of creating our tutorial series. However, also as indicated in [16], videos shorter than six minutes are more effective for students' engagement. Some students in CS1B (close to 20% of those who completed the online evaluation) complained that the lengthy videos increased their course workload. Nonetheless, it was only a much smaller group of students (less than 5%) expressing that these videos are unnecessary or useless for them to achieve the course learning outcomes.

Our tutorial videos offer new, more sophisticated examples, which cannot be completed in class, rather than repeating demonstrations done in class [25]. Consequently, given that our tutorial videos are meant for thoroughly demonstrating sophisticated examples, they are not comparable to short "podcast highlights" to full-length lecture footage [21]. Moreover, unlike many other attempts of making better tutorial videos [17, 18, 22], our approach is meant for teaching heavy-weighted complex computational thinking [13, 19, 20, 28] by requiring the careful setup of starter artifacts (e.g., code fragments, figures, writings) on a drawing tablet for illustrations, which are recorded for students to review outside class. Our videos were perceived as effective (see Section 7.3 for examples) by the majority (more than 60%) of students in CS1B.

Stanford Online [27], designed for online distance learning, offers similar tutorial support. The use of a drawing tablets for intensive illustrations, as well as constant switches between the various desktop activities, is not typical for courses there. Similar to Stanford Online are the online course repositories such as Coursera [15] and Udemy [26], but these are meant to be commercial, unlike our intention. Moreover, the use of intensive tablet illustrations is also not typical in these commercial courses.

9 CONCLUSION

As future work, we will: **1)** distribute a questionnaire specific to the learning experience of our tutorial videos; **2)** conduct more performance comparison on other subjects; and **3)** reflect on the proposed pattern (Section 5) by creating tutorials for other CS or Engineering courses (e.g., a course on the introduction to theory of computation).

REFERENCES

[1] The Author. 2017. Birthday Book Application in Java. https://www.youtube.com/playlist?list=PLxmJie0AEbu5iEysK1OlQBoi_CfQ8bpsm.

[2] The Author. 2017. Composite and Visitor Patterns. https://www.youtube.com/playlist?list=PLxmJie0AEbu4frG95BmvYyH8m1vg0bU9e.

[3] The Author. 2018. BMI Calculator: Model, View, Controller. https://www.youtube.com/playlist?list=PLxmJie0AEbu4VukLpZ_T-pdNyqvc1RJNA.

[4] The Author. 2018. Even-Driven Controller vs. Object-Oriented Model. https://www.youtube.com/playlist?list=PLxmJie0AEbu4sQkggQj3k_TPF4Bb2sms_.

[5] The Author. 2018. Java Collection Library. https://www.youtube.com/playlist?list=PLxmJie0AEbu7HTCOoe60HvFmVeXVQblzk.

[6] The Author. 2018. Managing Software Projects Using Github. https://www.youtube.com/playlist?list=PLxmJie0AEbu4cdt_7H1owO2-n6C_FXcmk.

[7] The Author. 2018. OOP: Array-Typed Attributes. https://www.youtube.com/playlist?list=PLxmJie0AEbu5ayf3scPtAS4SZ6R9OFDxq.

[8] The Author. 2018. OOP: Reference-Typed Attributes. https://www.youtube.com/playlist?list=PLxmJie0AEbu4C5lluKqRvKYSlZqX12joD.

[9] The Author. 2018. Solutions to Practice Test on Arrays and Loops. https://www.youtube.com/playlist?list=PLxmJie0AEbu7H1SQb6xXkZsDkDqsEayHv.

[10] The Author. 2018. Use of the Eiffel Testing Framework (ETF). https://www.youtube.com/playlist?list=PLxmJie0AEbu4frG95BmvYyH8m1vg0bU9e.

[11] The Author. 2018. Writing Java Code Based on Given Tests. https://www.youtube.com/playlist?list=PLxmJie0AEbu6Em3P7Y9ROjxeS4iseSKZJ.

[12] The Author. 2019. Java Tutorial Series for CS1B Labs. https://www.youtube.com/playlist?list=PLxmJie0AEbu6rQFQ9LGmb9thT48mGLk64.

[13] Jens Bennedsen, Michael E. Caspersen, and Michael Kölling. 2008. *Reflections on the Teaching of Programming: Methods and Implementations* (1 ed.). Springer Publishing Company, Incorporated.

[14] Blue. [n. d.]. Blue Yeti: Professional Multi-Pattern USB Mic for Recording and Streaming. https://www.bluedesigns.com/products/yeti/.

[15] Coursera. [n. d.]. https://www.coursera.org/.

[16] Philip J. Guo, Juho Kim, and Rob Rubin. 2014. How Video Production Affects Student Engagement: An Empirical Study of MOOC Videos. In *Proceedings of the First ACM Conference on Learning @ Scale Conference (L@S '14)*. ACM, New York, NY, USA, 41–50. https://doi.org/10.1145/2556325.2566239

[17] Juho Kim. 2013. Toolscape: Enhancing the Learning Experience of How-to Videos. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI EA '13)*. ACM, 2707–2712. https://doi.org/10.1145/2468356.2479497

[18] Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. 2013. Community Enhanced Tutorials: Improving Tutorials with Multiple Demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1779–1788. https://doi.org/10.1145/2470654.2466235

[19] Anna Lamprou and Alexander Repenning. 2018. Teaching How to Teach Computational Thinking. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. ACM, New York, NY, USA, 69–74. https://doi.org/10.1145/3197091.3197120

[20] James Lockwood and Aidan Mooney. 2017. Computational Thinking in Education: Where does it Fit? A systematic literary review. *CoRR* abs/1703.07659 (2017). arXiv:1703.07659 http://arxiv.org/abs/1703.07659

[21] Mia Minnes, Christine Alvarado, Max Geislinger, and Joyce Fang. 2019. Podcast Highlights: Targeted Educational Videos From Repurposed Lecture-capture Footage. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, 365–371. https://doi.org/10.1145/3287324.3287465

[22] Cuong Nguyen and Feng Liu. 2015. Making Software Tutorial Video Responsive. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, 1565–1568. https://doi.org/10.1145/2702123.2702209

[23] J. S. Ostroff and C. Wang. 2018. Modelling and Testing Requirements via Executable Abstract State Machines. In *2018 IEEE 8th International Model-Driven Requirements Engineering Workshop (MoDRE)*. 1–10. https://doi.org/10.1109/MoDRE.2018.00007

[24] Active Presenter. Version 7. All-in-one Screen Recorder, Video Editor & eLearning Authoring Software. https://atomisystems.com/activepresenter/

[25] Ben Stephenson. 2019. Coding Demonstration Videos for CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, 105–111. https://doi.org/10.1145/3287324.3287445

[26] Udemy. [n. d.]. https://www.udemy.com/.

[27] Stanford University. [n. d.]. Stanford Online. https://online.stanford.edu/courses.

[28] Jeannette M. Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35. https://doi.org/10.1145/1118178.1118215

Keep Calm and Code on Your Phone: A Pilot of SuaCode, an Online Smartphone-Based Coding Course

Anonymous Author(s)*

ABSTRACT

Africa lags behind the rest of the world in terms of digital literacy skills with less than one percent of African children leaving school with basic coding skills. One cause of this gap is poor access to equipment such as computers for teaching and learning. Yet, there is a proliferation of smartphones in Africa. Seeking to leverage this opportunity, we developed SuaCode, an online smartphone-based coding course to teach programming fundamentals to Africans. We designed the course to teach coding in a visual, interactive and fun way through the building of a pong game using Processing (a Java-based programming language). In this work, we describe our experience delivering the course online to 30 Ghanaian high school and college students. At the end of the course, 7 of the 30 students completed the first part of the course, building the pong game. The reflection essays from our students showed that they enjoyed the course and coding on a smartphone was not a barrier to completing the assignments. Improvements such as having more mentors and automated feedback on the coding assignments will improve the quality of the course. Given the difficulty in accessing computers in Africa, our work shows that smartphones can be leveraged to effectively introduce students to programming concepts via an online course. We are excited about the results of this pilot and see the potential to scale the course to eventually bring coding skills within arm's reach of millions across Africa, literally into their palms thereby bridging Africa's digital divide.

CCS CONCEPTS

• **Applied computing** → **Distance learning**; **E-learning**; Learning management systems.

KEYWORDS

smartphones, mobile phones, online course, coding, introductory programming, Processing, Ghana, Africa

1 INTRODUCTION

Africa lags behind the rest of the world in terms of digital literacy skills, with less than one percent of children leaving school with basic coding skills [13]. One cause of this gap is poor access to equipment such as computers for teaching and learning, which hampers efforts to promote digital learning among the general population [9] . For example, Ghana's national student to computer ratio stood at 42:1 in 2010 [17]. Hence, computer literacy remains the preserve of the elite who can afford computers and opportunities to gain this education. Fortunately, there is a proliferation of smartphones in Africa. According to research firm Ovum, Africa's smartphone penetration rate will grow at 52.9% year-on-year [15]. In 2016, there were 293.8 million smartphone users and there is projected to be 929.9 million smartphones by the year 2021 [15]. Hence, smartphones provide a unique means to provide coding skills to the youth of Africa. Additionally, coding on a phone gives a huge opportunity to be very innovative with reference to teaching students how to code. For example, the coding curriculum can be designed around students building mobile-first programs such as games, which has been shown to be a more exciting and effective way to introduce programming[4, 12].

In this paper, we describe our experience running a pilot of SuaCode, an online smartphone-based coding course to teach programming fundamentals to a cohort of primarily Ghanaian students (Figure 1). In our previous work, we introduced students to coding in-person using smartphones (a first of its kind in Ghana) [2]. Building upon that work, we sought to assess the feasibility of students learning remotely using smartphone to scale the reach and impact of our course. Therefore, we gathered students' feedback in this study to assess this approach. This experience report describes the process of creating and running the course along with a discussion of lessons learned. The rest of the paper is organized as follows: we give an overview of our coding curriculum, describe the pilot experience, followed by the findings and discussion, related work and conclusion.

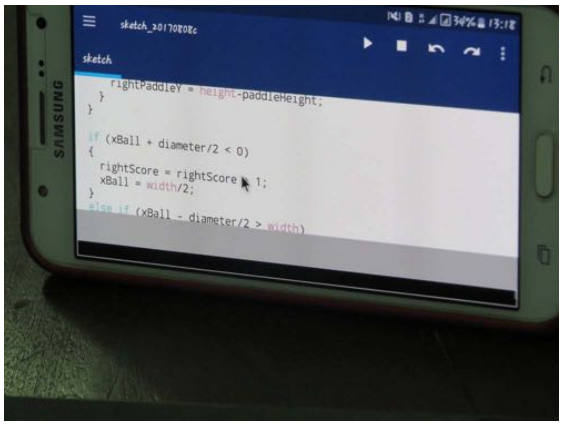


Figure 1: Snapshot of Code on Phone

2 OVERVIEW OF SUACODE

The pilot of SuaCode was implemented as an 8-week online course for high school and college students in Africa by Nsesa Foundation, an education non-profit in Ghana. The course ran from from May to June, 2018. We invited high school and college students all over Ghana to apply to take part in the SuaCode pilot via advertisements on social media (Figure 2).

Students were asked to respond to the question "Why do you want to take part in the SuaCode pilot?" The response to this question was used to select students. Specifically, two people rated the

responses on a scale of one to five and the average of the two ratings were used to select the students. A total of 117 applications were received, out of which 30 students were selected based on their motivation essays. The 30 students were all Ghanaian except one student who was from Ethiopia with (76%) being college students and 50% being female.

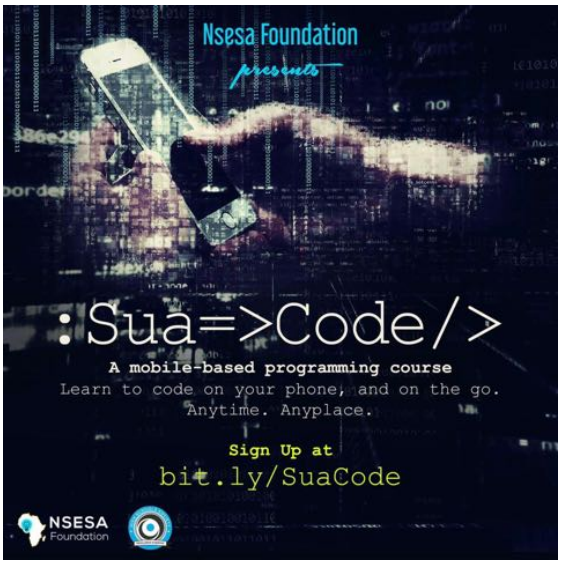


Figure 2: Poster used for advertising the SuaCode program

3 OVERVIEW OF THE CODING CURRICULUM

We designed the curriculum of our custom smartphone-based course to introduce fundamental programming concepts to students with little or no programming experience. Our curriculum was previously used in an in-person coding course in which there was a significant improvement in perceived understanding of programming concepts. The curriculum was inspired primarily by Dartmouth College's Introduction to Programming course (CS 1) [6]. We also based our curriculum on Dartmouth's computer science course, "Programming for Interactive Audio-Visual Arts" [7], and "the Coding Train" a set of programming tutorials, both of which use Processing for instruction [19].

The objectives of our SuaCode course were as follows:

(1) Introduce students to fundamental programming concepts
(2) Develop critical thinking and problem-solving skills.

Our course teaches programming fundamentals in a visual, interactive and fun way through game development using the Processing programming language [18] on the smartphone-based programming environments APDE for Android [3] or Processing iCompiler for iOS [8] with which to run the code on phones. Processing is an open-source, Java-based language which we chose to use because it enables learning of programming in a fun way since it can be used to easily create visual and interactive programs.

We also chose to use a game development paradigm because several studies have shown that it is an engaging, motivating and effective way to teach programming [4, 5, 11, 12]. Specifically, the course was structured such that students will build a pong game at the end of the course (Figure 3). This game has 2 paddles, one for each player and a ball. Once the ball starts moving, each player has one goal; to prevent the ball from exiting the vertical wall on his side. If that happens, the opponent's score increases.

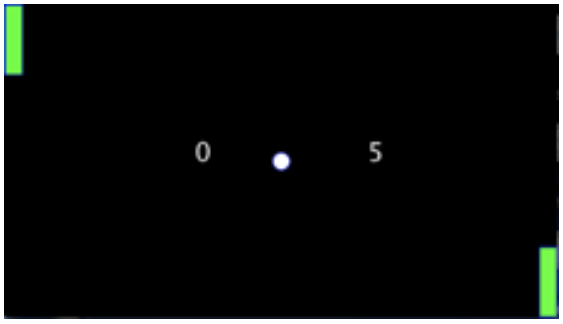


Figure 3: Pong game

Our curriculum consists of 6 lessons and it is divided into two parts. The first part introduces basic programming concepts (such as variables and functions) and the second introduces more advanced concepts (such as classes and objects). Each topic has a corresponding lesson note containing programming exercises and an end-of-lesson assignment, which incrementally built a component of the game. This arrangement was chosen to make the progression through the course more engaging. The contents of the lessons and assignments are openly available for others to use here: [1]

Completion of "Part 1" results in the building of the pong game. Part 1 consists of lessons 1, 2, 3, and 4. Lesson 1 introduces students to the Processing language and gives examples of various programs that have been created with it. The students learned how to draw and color various shapes such as line, ellipse and rectangle and fill them with colors. The assignment at the end creates an interface of the pong game consisting of different shapes and colors. Lesson 2 introduces students to variables, enabling them to move objects on the screen. At the end of the lesson, the assignment makes use of variables to represent the shapes in the pong interface from the previous assignment and then make the ball move. Lesson 3 introduces students to conditionals, enabling various actions to be performed if certain conditions are not met. At the end of the lesson, the assignment was to build upon the previous assignment and make the ball bounce if it hits the top and bottom walls of the game environment. Lesson 4 introduces students to functions, enabling them to reuse blocks of code without rewriting them. To practice, students were asked to take the previous assignment and reorganize various blocks of code with functions, and then to write

[1] https://github.com/Suacode-app/Suacode/blob/master/README.md

functions to move the 2 paddles and also make the ball bounce off them.

Completion of "Part 2" results in adding more features to the pong game such as having several balls on the screen to make the game more interesting, or optionally adding obstacles. Lesson 5 introduces students to classes, objects and object oriented programming (OOP) enabling them to write code in a much cleaner and more intuitive manner. The lesson's assignment entailed reorganizing assignment 4 using classes and objects and including an additional ball in the game. Lesson 6 concludes the course and introduces students to loops and arrays, enabling them to easily manipulate several objects. The corresponding assignment was to build upon assignment 5 and add at least 5 balls to game. Students at this point had the options to add several features to the game to make it more interesting such as increasing the speed of the ball slightly when it bounces off the paddle, changing the color of the ball randomly as it moves or when it bounces of the paddle, varying the size of the ball from its size to zero as it moves, etc.

Table 1: Programming Curriculum

Lesson	Topic	Assignment
PART 1		
1	0.0 Introduction 1.0 Basic Concepts in Processing	Make Interface
2	2.0 Variables	Move Ball
3	3.0 Conditionals	Bounce Ball
4	4.0 Functions	Move Paddles
PART 2		
5	5.0 Classes and Objects	Add Extra Ball
6	6.0 Loops and Arrays	Add More Balls

4 COURSE LOGISTICS AND EXPERIENCE

We hosted and delivered the course online via Google classroom, a free learning management system (LMS) (Figure 3). We chose to use Google classroom among several options which we considered and tried out such as Moodle, Teachable etc. because it is the only LMS that satisfied these key requirements:

(1) Free to use both for teachers and students
(2) Course notes are accessible offline
(3) Fully functional Android and iOS apps
(4) No need to set up our server, database etc.
(5) Assignments can be assigned grades
(6) Simple to use

For each lesson, the corresponding lesson note and assignment were made available as Google docs which were accessible in Google Classroom (Figure 4). Each week, students read the week's lesson note and also completed the corresponding assignment. Students read the lesson notes using the Google classroom iOS and Android apps. They then wrote their code for the assignments using APDE (Figure 4) or Processing iCompiler apps. We did not use videos as instruction materials but only lesson notes because Internet data is very expensive in Africa, and more so for our target population [1], students living in Africa.

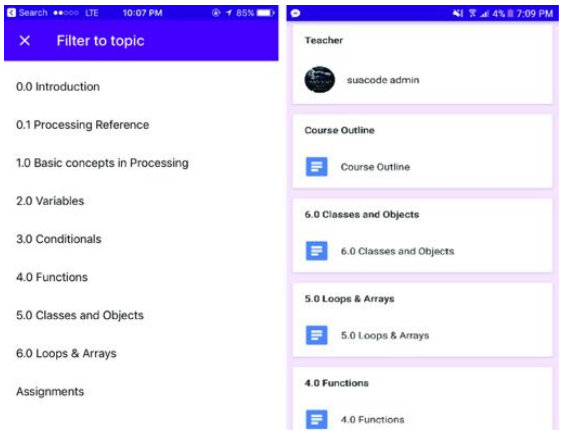


Figure 4: Snapshot of Course

Students posted questions in Google Classroom. We had two facilitators for the this pilot that answered students' questions as well as graded their assignments gave feedback for improvement. For each assignment submission, the students included a reflection essay describing among several things whether that week's lesson and it's corresponding assignment was fun, challenging, etc. They also described their experience coding the assignment with their smartphones.

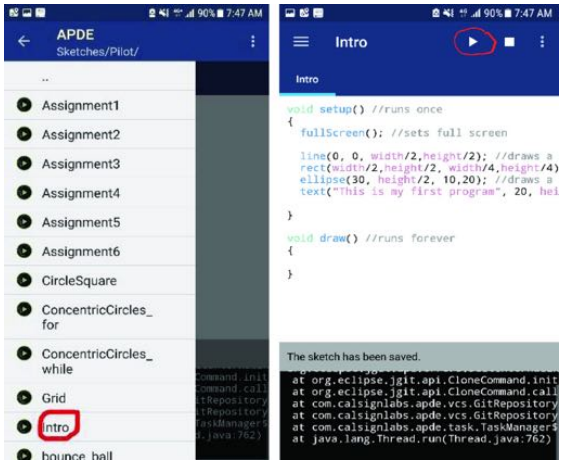


Figure 5: Snapshot of APDE Coding Environment

5 FINDINGS AND DISCUSSION

One of the main challenges we had was that students would submit their code for the assignments without checking if it met all the specifications outlined for the assignment. Because we were concerned with competency-based learning in which students are

expected to master concepts before moving to the next module, we ended up giving feedback on their code with the opportunity to resubmit. This back and forth went on generally for about three to four times for the assignments before the students' code met all the necessary specifications. This process was very labor intensive, but we felt it was necessary to ensure they understood all the concepts in each lesson. Because of this experience, we are now exploring an automatic approach for giving feedback on assignments, and plan to develop that for subsequent iterations of the course. We intend to develop some algorithms and code that will automatically review the assignment submissions of students and give feedback about the changes needed to be made, especially if their code submissions do not meet all the requirements. This solution when implemented will aid in scaling the program to more students as we plan to run more pilots.

Additionally, out of the 30 students who were selected and invited to take the course, we had 11 students making submissions for the first assignment, and mainly 7 students consistently completing the assignments. Specifically, the 7 students completed part 1 of the course (consisting of four assignments), resulting in the building of a fully functional pong game. Unfortunately, only 1 student completed assignment 5 and no student completed the second part of the course (consisting of two assignments), which sought to teach more advanced programming concepts. For those 7 students, that completed the first part of the course, it took two months rather than the previously planned one-month period. For this pilot, we use completion of part 1 as the metric for completing the course since students built the intended pong game after completing it and also we offered part 2 as an optional next step. That puts our completion rate of 23% which is above the completion rates of online course which are mostly under 10% and on average 6.5% [10]. Nonetheless, we would like to improve on the completion rates.

We sent an email to find out the reason students were not completing assignments. It ended up being that most students were taking their examinations during the time the pilot was running, making it difficult to dedicate time to completing the assignments. This insight was useful because it showed that it is necessary to make sure the program does not run during the examination period. Additionally, in the future, we plan to have mentors assigned to students to check up on them especially when they are falling behind schedule. Also, our 1-month estimate for the completion was an underestimate of the time needed to complete the course. Hence, for future sessions, the deadlines of the assignments will be planned around an 8-week time period.

Also, there were a few issues with the APDE app. Some students reported that the app crashed sometimes and they lost their code. Debugging the issue revealed that it was happening on phones running on older Android versions. We have been in touch with the developer of the APDE app andcollaborated with him to fix issues that were encountered during the program. We plan on advising our students to keep a backup of their code in a Google doc for worst case scenarios like this. Also, when code is ran, the APDE app builds and install an Android apk which takes a long time and takes up memory for the phone since several "apps" are installed with every code run. We communicated this issue with the APDE app developer and a preview mode which avoid this process and runs the code in a much shorter time has been built into the newly updated version of the app.

Additionally, students reported that coding on a phone was a bit challenging because of the small screen size, but it was nonetheless an exciting and enjoyable experience mainly because of the convenience and accessibility it offered. Also, they said it got easier over time. This summary can be seen from the following excerpts from the reflection essays on the coding experience:

- *"Coding on my phone was very fun and convenient since I had my phone all the time and I could easily work on my code."*
- *"It is very exciting coding on phone but sometimes very challenging because of the screen size."*
- *"One problem that I have identified is not being able to have a wider screen to work on. This would have made it easier for me to see the code in its entirety. This might just be a matter of getting used to using my phone to work on my assignments."*
- *"It was really convenient, honestly. I didn't have to necessarily sit behind a desk to do it so I could do it when I was on my bed, eating, even using the bathroom. So it was fun and convenient coding on my phone."*
- *"It wasn't very different from doing it on a computer. In fact, it was more convenient as I didn't have to worry about carrying a computer everywhere."*
- *"It was really cool actually. Initially, it was not comfortable because I am used to working on my PC. However, once I started working on it, I released that it was very convenient since I can work anywhere even when I am taking a taxi, at home, work anywhere. So it was really cool. It is much better than being on social media."*
- *"I have gotten more used us my phone to code so it is no longer a difficulty."*
- *"In terms of writing the code, I found it easier than in a laptop because I could type faster. However, the small screen size doesn't allow one to view a large portion of the code at once so there's a lot of scrolling going on and that could be distracting."*
- *"This has been a very exciting experience, and I look forward to more challenging assignments."*
- *"So while reviewing the material, I found it quite interesting and easy to understand. I'm really learning a lot! And I love this course."*
- *"The material was detailed enough and easy for me to understand with a good outline. Thank you very much for this opportunity to write my first code."*

Finally, we were successful in developing students' critical thinking, problem solving, and basic coding skills as evidenced by students completing the assignments in part 1 of the course which resulted in them building a fully fledged and functioning pong game. Also, the acquiring of these skills were confirmed in the following feedback from three of the students about the course:

- *"Suacode has been a very great experience for me. I got to learn processing and actually code on my phone. I also had help from the tutors and my fellow course mates which made it easier. I learnt a lot and I'm glad I had the opportunity to be part of the first batch of suacode initiative"*
- *"I'd also like to say I'm very glad I participated in the course, it gave me a good head start in Programming which helped*

me a lot in my intro programming class. The course was phenomenal!"

- *"SuaCode helped improve my algorithmic thought process. I had lots of practice with thinking in a step by step process and working through challenges."*

6 RELATED WORK

There have been a number of previous works that sought to introduce students to fundamental coding concepts via mobile programming [14, 16, 20]. Nonetheless, our approach is unique for two reasons. First, we introduce students with no or very limited prior experience to programming using the smartphone as a coding environment. Secondly, we deliver our course online rather than in-person. Therefore our approach presents an interesting opportunity to scale the reach and impact mobile programming initiatives.

In a study by Tillman and coauthors [20], a mobile programming environment, TouchDevelop was developed to introduce middle and high school students to programming. However, TouchDevelop was designed for Windows Phones. This is a limitation since 87.5% of smartphones worldwide operate on the Android OS [21] and also as our survey revealed that 90% of our study participants used Android phones. Our work, on the other hand, utilizes apps on iOS and Android which allow students with these smartphones to code with our course which is a useful advantage. In the same study, the authors also noted that in programming on a smartphone, actions that require fine navigation such as making structural corrections are awkward on a touchscreen where precision is limited by the size of a finger [20]. This is a challenge which students in our program also reported. Other limitations such as the smaller screen, limited battery life and relatively slower processors affect the smartphone-based programming experience [2, 20]. These challenges will need to be addressed to improve the learning experience of smartphone users.

In another study [16] the authors developed a scaffolding to support programming by university students in Kenya and South Africa who were taking an introductory course in programming using Java. In contrast to our approach, the participants in this study used desktop computers in their classroom learning and only transitioned to a mobile programming interface during the experiments. Thus it did not capture the true experience of learning programming on a phone. Besides, Mahmoud and Popowicz advocate for the use of mobile phones to teach introductory programming to computer science students [14] and our approach serves this purpose.

Another innovation in our program is the delivery of the course remotely via an online system. A study by Wang [22] identified that lack of instant instructor feedback as one of the key challenges to learning programming online. We tried to mitigate this by having facilitators provide feedback to students on assignments during the course. Wang further recommends two solutions, among others, to improve the online learning experience. First, by adding multimedia content such as videos to make the instruction more engaging and understandable. Second, to create a sense of community among students to help them support each other. These suggestions can be found in our pilot in various forms. We do not include videos in our materials as stated earlier because Internet data is expensive in Africa [1], . Nonetheless, our lesson notes contain images where necessary to better explain concepts which is the next best option. Also, we encouraged students to help other students and we saw that happening but not as strongly as we hoped. We plan to encourage that more in subsequent runs of the program.

7 CONCLUSION

In this work, we described our experience delivering a smartphone-based coding course online to 30 students primarily based in Ghana. Seven of the 30 students completed the first part of the course. The reflection essays from our students showed that they enjoyed the course. Additionally, the seven students built the pong game and coding on the smartphone was not a hindrance. However, improvements such as having more mentors for the students and automated feedback will significantly improve the quality of the course and potentially improve the course completion rate. Overall, the course met the objectives of introducing students to fundamental coding concepts, critical thinking and problem solving skills. We are excited about the results of this pilot and we see the potential to scale the course to eventually bring coding skills within arm's reach of millions across Africa, literally into their palms thereby bridging Africa's digital divide.

REFERENCES

[1] Quartz Africa. 2019. The cost of internet access is dropping globally but not fast enough in Africa. Retrieved June, 2019 from https://qz.com/africa/1577429/how-much-is-1gb-of-mobile-data-in-africa/

[2] Annonymous. 2018. Project iSWEST: Promoting a culture of innovation in Africa through STEM. In *2018 IEEE Integrated STEM Education Conference (ISEC)*. 104–111. https://doi.org/10.1109/ISECon.2018.8340459

[3] APDE [n. d.]. APDE - Android Processing IDE. Retrieved Jan, 2019 from https://play.google.com/store/apps/details?id=com.calsignlabs.apde

[4] Tiffany Barnes, Eve Powell, Amanda Chaffin, and Heather Lipford. 2008. Game2Learn: Improving the Motivation of CS1 Students. In *Proceedings of the 3rd International Conference on Game Development in Computer Science Education (GDCSE '08)*. ACM, New York, NY, USA, 1–5. https://doi.org/10.1145/1463673.1463674

[5] Jessica D. Bayliss and Sean Strout. 2006. Games As a "Flavor" of CS1. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '06)*. ACM, New York, NY, USA, 500–504. https://doi.org/10.1145/1121341.1121498

[6] CS1 [n. d.]. Dartmouth CS 1. Retrieved Jan, 2019 from http://www.cs.dartmouth.edu/~cs1

[7] CS2 [n. d.]. Programming for Interactive Audio-Visual Arts. Retrieved Jan, 2019 from http://aum.dartmouth.edu/~mcasey/cs2/

[8] iCompiler [n. d.]. Processing iCompiler. Retrieved Jan, 2019 from https://itunes.apple.com/us/app/processing-icompiler/id648955851?mt=8

[9] ITU 2018. ITU releases 2018 global and regional ICT estimates. Retrieved Jan, 2019 from https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

[10] Katy Jordan. 2014. Initial trends in enrolment and completion of massive open online courses. *The International Review of Research in Open and Distributed Learning* 15, 1 (Jan. 2014). https://doi.org/10.19173/irrodl.v15i1.1651

[11] Stan Kurkovsky. 2009. Engaging Students Through Mobile Game Development. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)*. ACM, New York, NY, USA, 44–48. https://doi.org/10.1145/1508865.1508881

[12] Scott Leutenegger and Jeffrey Edgington. 2007. A Games First Approach to Teaching Introductory Programming. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*. ACM, New York, NY, USA, 115–118. https://doi.org/10.1145/1227310.1227352

[13] Digitalist Magazine. 2015. Africa Skills Gap: Rising To Meet The Digital Challenge. Retrieved Dec, 2017 from http://www.digitalistmag.com/improving-lives/2015/08/19/africa-skills-gap-meet-digital-challenge-03292180

[14] Qusay H. Mahmoud and Pawel Popowicz. 2010. A mobile application development approach to teaching introductory programming. In *2010 IEEE Frontiers in Education Conference (FIE)*, Vol. 00. T4F–1–T4F–6. https://doi.org/10.1109/FIE.2010.5673608

[15] Vincent Matinde. 2016. Africa 2017: Smartphone penetration, Open Data and less online freedom,. Retrieved Jan, 2019 from https://www.idgconnect.com/idgconnect/opinion/1022805/africa-2017-smartphone-penetration-online-freedom

[16] Chao Mbogo, Edwin Blake, and Hussein Suleman. 2016. Design and Use of Static Scaffolding Techniques to Support Java Programming on a Mobile Phone. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, New York, NY, USA, 314–319. https://doi.org/10.1145/2899415.2899456

[17] Ghana Ministry of Education. 2010. Education Sector Performance Report 2010.

[18] Processing [n. d.]. Processing Foundation. Retrieved Jan, 2019 from https://processing.org/

[19] The Coding Train [n. d.]. The Coding Train. Retrieved Jan, 2019 from http://thecodingtrain.com/

[20] Nikolai Tillmann, Michal Moskal, Jonathan de Halleux, Manuel Fahndrich, Judith Bishop, Arjmand Samuel, and Tao Xie. 2012. The Future of Teaching Programming is on Mobile Devices. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '12)*. ACM, New York, NY, USA, 156–161. https://doi.org/10.1145/2325296.2325336

[21] Derek Walter. 2016. Report: Nearly 90 percent of smartphones worldwide run Android. Retrieved Jan, 2019 from https://www.greenbot.com/article/3138394/android/report-nearly-90-percent-of-smartphones-worldwide-run-android.html

[22] Wendy Wang. 2011. Teaching programming online. International conference on the future of education. In *Internation Conference on Future of Education*. Florence, Italy.

Autism: Implications for Inclusive Education

with respect to Software Engineering

Sylvia Stuurman
Sylvia.Stuurman@ou.nl
Open Universiteit, the Netherlands

Harrie J.M. Passier
Harrie.Passier@ou.nl
Open Universiteit, the Netherlands

Frédérique Geven
Frederique.Geven@senevita.nl
Senevita, the Netherlands

Erik Barendsen
E.Barendsen@cs.ru.nl
Radboud University, the Netherlands

ABSTRACT

Within Computer science and Software engineering, the prevalence of students with a diagnosis of autism spectrum disorder is relatively high. Ideally, education should be inclusive, with which we mean that education must be given in such a way that additional support is needed as little as possible.

In this paper, we present an overview on what is known about the cognitive style of autistic individuals and compare that cognitive thinking style with computational thinking, thinking as an engineer, and with academic thinking. We illustrate the cognitive style of autistic students with anecdotes from our students.

From the comparison, we derive a set of guidelines for inclusive education, and we present ideas for future work.

CCS CONCEPTS

• **Social and professional topics** → **People with disabilities**; • **Applied computing** → *Education*.

KEYWORDS

Autism, Inclusive education, Cognitive thinking style

ACM Reference Format:
Sylvia Stuurman, Harrie J.M. Passier, Frédérique Geven, and Erik Barendsen. 2019. Autism: Implications for Inclusive Education: with respect to Software Engineering. In *Proceedings of CSERC'19.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Inclusive education will only work when education benefits *all* students, as opposed to education for who is not disabled, with additional programs for the disabled [54]. This statement has implications for software engineering education with respect to autistic students. In this paper, we explore these implications.

Need for inclusive education. The prevalence of autism continues to rise. In the USA for instance, between 2000 and 2014, the prevalence of autism increased from 1 in 150 (about 0.67%) to 1 in 59[1] (almost 1.7%). It is difficult to interpret these numbers, because there is no standardization of autism survey methodology. Also, it is an open question whether autism is more often diagnosed because of shifting definitions, because of more attention, because of a society that becomes more difficult to live in for who is autistic, or because there are more environmental factors that cause autism. The rising prevalence of autism is probably due to the rising numbers of individuals with high functioning autism, who are most able to go to university [66]. We may therefore conclude that we will continue to see a rising number of university students with a diagnosis in the autism spectrum.

Students with a form of autism often need support to achieve success at universities, while they do not lack intellectual capacities [11]. In the US and Australia, youth with autism have the highest risk of being completely disengaged from any kind of post-secondary education or employment [43]. Compared with other disabilities, youth with autism have the lowest rates of employment and education participation. Similar figures were found in Sweden and Canada [43]. These figures are even more startling when one realizes that one of the theories about autism is that it represents high, imbalanced intelligence, a 'disorder of high intelligence' [12]. Moreover, IT companies begin to see the advantages of autistic personel [2]. Therefore, it is really worthwhile to investigate how we can make education more inclusive with respect to autistic students.

More and more, autism is seen as 'being different' as opposed to a disorder [28]:

> In our opinion, high-functioning autism should neither be regarded as a disorder or a disability nor as an undesirable condition per se, but rather as a condition with a particular vulnerability. Autism can also have desirable and enabling consequences, both to the individual and to society.…For what are now disabling traits of these people, may, in a differently constructed social environment, become 'neutral' characteristics.

Inclusive education that helps autistic students, therefore, is not only needed because of the rising number of autistic students (who really need support of some kind), but it would have benefits for

CSERC'19, The 8th Computer Science Education Research Conference, November 18–20

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://www.cdc.gov/ncbddd/autism/data.html
[2] see, for instance *CEO's finally get it. Staff on the autism spectrum are a huge asset*, Wired, https://www.wired.co.uk/article/companies-employing-autistic-individuals

all: inclusive education would promote a specific kind of diversity, and diversity has a positive outcome on cognitive skills for all students [7]. This need for diversity has, for instance, also been explained with respect to personality type [10, 62]). Especially important is that successful teams show diversity in personality [46].

With respect to diversity, we want to stress that increasing the possibilities for autistic students to successfully finish their studies does not have a negative impact on the number of women, in our opinion. The low number of women in Computer science has been attributed to the 'masculine culture' that Computer science has been 'drenched' into [15]. This 'male culture' is often associated with autism [29]. The diagnosis of autism however, is itself heavily 'gendered', in such a way that researchers payed much attention to everything considered 'male', discarding everything considered 'female' [29]. As a result, more males received a diagnosis, and a sex ratio with far more males than females was considered the 'natural' ratio. More and more, implicit biases in the process of diagnosing and measuring autistic traits have been made explicit, and more and more, women are diagnosed too. The ratio male-female is now considered to be 2:1, and still, autistic females may be missed by current diagnostic procedures, which would bring the 'real' ratio closer to 1:1 [50].

In fact, one could say that the same mechanism that associated Computer science with 'masculine' did associate autism with 'masculine', and both associations form a disadvantage for women. Therefore, we do not make Computer science more 'masculine' by addressing the cognitive style of autism in Computer science education. In fact, we hope to help in 'de-gendering' both autism and Computer science by the type of inclusiveness we adrress.

Cognitive aspect of inclusive education. Autism is characterized by a different cognitive style (which we elaborate on in Section 5). On the one hand, this different cognitive style has advantages. For instance, one can see an increasing demand for people within the autism spectrum, from, for instance, IT firms [3]. On the other hand, because Software engineering education is geared to a 'non-autistic' cognitive style, it might be more difficult than necessary for students with an autistic cognitive style to complete their education. A really inclusive education would be geared to both autistic and non-autistic students.

Mismatches between cognitive skills for Software engineering and the autistic cognitive style, often are skills that whoever is not autistic will take for granted. As such, they form 'blind gaps' that educators in Software engineering might have with respect to autistic students. With 'blind gaps', we mean to take a skill for granted (because non-autistic students often acquire that skill intuïtively), that should not be taken for granted (because for autistic students, it takes explicit effort to acquire that skill). In this paper, we take a theoretical approach to detect these 'blind gaps'. To count as inclusive, education should cover these 'blind gaps'.

In this paper, we only analyze *cognitive skills*. Excluded from our attention here, is, therefore, support with independent living and social skills, with planning and time management, which is often attended to by universities [59]. We also exclude attention to cooperation, because we think that cooperation in education for students in the autism spectrum deserves a separate study.

Our goal is to create a set of guidelines for inclusive education for students within the autism spectrum, based on the differences between the cognitive style that characterizes autism and the cognitive style that is needed within Software engineering.

Structure of this report. This report is structured as follows. In Section 2, we give a short introduction into autism (as we will explain, we use 'autism' as a synonym of 'autism spectrum disorder'), and about the prevalence within Software engineering. We explain our research method in more detail in Section 3. Section 4 contains related work, about support for autistic students in general.

We characterize the autistic thinking style in Section 5. In Section 6, we compare the thinking skills that are needed within Software Engineering with the autistic thinking style that we established in Section 5. We show which of those skills might form a hindrance for autistic students. We illustrate some of these findings with anecdotal evidence that we have gathered from our students.

Section 7 describes the guidelines we derive from the comparison. In Section 8, we discuss the explorative nature of this paper, and argue why our exercision is worthwhile. In Section 9, we draw our conclusions and present ideas for future work.

2 AUTISM, BACKGROUND

Autism is described in the Diagnostic and Statistical Manual of Mental Disorders (DSM) as a pervasive development disorder [14]. Diagnostic criteria are persistent deficits in social communication and social interaction, and restricted, repetitive patterns of behavior, interests, or activities.

Autism, Asperger, ASD. Autism has been 'discovered' by Leo Kanner [32], although other people described similar characteristics earlier (in particular Hans Asperger [74]). At first, what Asperger and Kanner described has been classified as two different disorders, although within the same 'autistic spectrum': the Asperger syndrome and classical autism [72]. Later, more variants were discerned, such as PDD-NOS (Pervasive Developmental Disorder Not Otherwise Specified) or high-functioning autism (classical autism with normal to high intelligence). In the fifth version of the DSM [14], this distinction has been abandoned, because most researchers agree that the distinction is not useful. At this moment, one speaks of 'autism spectrum disorder' (ASD). When we speak of autism, in this article, we refer to ASD.

During the years, the diagnosis of autism (and with it, the meaning of the word 'autism') has seen major shifts in type of symptoms [63, 74]. The most recent shift is to view sensory and perceptual issues as the main characteristic [6, 27, 60].

Neurodiversity. Autism is described and treated as a psychiatric disorder and for a long time, research has been directed to both prevention and cure for autism. In recent years, a shift can be seen to view autism in individuals with average or above average intelligence not as a disorder, but as a difference, which nevertheless requires adaptations from the rest of society [33] (in the same manner as for, for instance, left-handedness). In other words, autism is, in this view, not seen 'as a disorder or a disability nor as an undesirable condition per se, but rather as a condition with a particular vulnerability and with particular strengths' [28]. This view of autism as a difference has a name: neurodiversity [44]. When we

hold this view on autism, it becomes even more important to try to educate our students in such a way that autistic students have a chance to get their degree.

To avoid the contrast 'autistic' versus 'normal', that emphasises the abnormality of autism, many use the word 'neurotypical' for non-autistic persons. In this report, we will use that word too.

Prevalence in Software engineering. Individuals within the autism spectrum more often have a profession that requires technical skills than neurotypical individuals [57], and are more inclined to follow STEM (science, technology, engineering and mathematics) studies [66]. This fact makes it plausible that autism has a higher prevalence among Software engineering students than among the general public.

Measuring the prevalence of autism among students is difficult. It is not possible, for obvious reasons, to diagnose each student as part of an investigation. When one asks students whether they have a diagnosis, one misses those students who are autistic but never have been diagnosed, and one misses students who do not want to disclose their diagnosis.

In countries where students receive special education services when they have a diagnose of autism, one may measure the prevalence of those services. Based on this estimation, Wei et al. found a higher prevalence of autism in the STEM sciences than in non-STEM disciplines [66].

Other estimations are possible as well. There are, for instance, measurements of autistic traits, such as the Autism-Spectrum Quotient (ASQ or AQ) [5]. This measurement rates individuals relative to the mean of the population, with respect to autistic traits that are measured through a self-report questionnaire. The ASQ can be used as a coarse-grained estimation of the prevalence of autism. Note, however, that the ASQ is based on self-reporting, and as such cannot be regarded as an alternative to a formal diagnosis.

The ASQ has a normal distribution [53]. When depicting the ASQ in the form of a bell curve, people who are probably within the autism spectrum occupy a small portion of the extreme of the cuve. One half of all people have more autistic traits than average.

Using the ASQ on a population of university students, White et al. found that more than 50 percent of the high ASQ students where computer science majors. Of the students who did not score a high ASQ, only 28 percent were computer science majors [68]. We may conclude, therefore, that in general computer science students scorefairly high on the ASQ. Baron-Cohen found that computer scientists score higher on the ASQ than scientists in medicine and biology (and mathematicians score higher than computer scientists) [5]. This means that self-perceived autistic traits are more prevalent in mathematicians and computer scientists than in other scientists.

3 METHOD

Our goal is to derive a set of guidelines with respect to inclusive education. To do that, we compare traits of the autistic cognitive style with cognitive skills that are needed in Software engineering, .

Although there are many competing theories about autism, they agree on the general ideas about what the autistisc cognitive style is. It is, however, a difficult task to summarize the knowledge within such a vast field without delving into the details of all competing theories. In the first place, there are those characteristics of the cognitive style that can directly derived by the description in the DSM. Theories that try to explain autism, show how this theory explains various cognitive aspects of autism. Such cognitive aspects form part of the autistic cognitive style that belong to autism, according to experts. Finaly, one of the authors is a practitioner, with comprehensive experience as a therapist for autistic adults. She checked whether we were complete.

The autistic cognitive style that we present here is therefore grounded in how experts see autism.

During the past two or three years, students sometimes sent us a summary of the difficulties that they faced during their study, that, according to their idea, are associated with their autism. We cannot use these observations as 'proof', because they are anecdotical by nature. However, they illustrate the points we make very well, so we included them where applicable, to make some cognitive characteristics more clear.

Deciding the cognitive skills that belong to Software engineering is no exact science. As a start, we tried to find an operational model of computational thinking, because that would give us the most concrete means to compare aspects of cognitive thinking with the autistic cognitive style. We added thinking as an engineer and academic thinking to these cognitive skills. We can never be certain that we are complete, but at least we have a beginning.

4 RELATED WORK

In a systematic literature review, Gelbar et al. found that case studies in a university setting indicate the presence of anxiety, loneliness, and depression and the need for supports for autistic students. They also found a lack of studies indicating which support is needed, and what works [22].

Fleury et al. [19] inventarised what is known about Academic performance of students with ASD (Asperger Syndrome; the study was done before all forms of autism are called ASD, autism spectrum disorder). They found the following characteristics:

Reading Students with ASD were found to be quick in the mechanics of reading (recognising words), but in general have problems comprehending text.

Writing Hand-writing is often difficult for students with ASD. Also, planning and organising a text is a difficult skill for students with ASD.

STEM STEM studies are popular with students with ASD, in particular mathematics, science, and computer science. However, within these studies too, they face difficulties with language comprehension and executive functioning. Mathematic achievements for students with ASD ranges from modest weaknesses to mathematical giftedness.

They also inventarised instruction strategies for students with ASD.

Priming: Preparing a student in advance for what is coming, for instance before the start of the study, the start of a course, a task or a meeting.

Peer support: Peers are taught how to support students with ASD.

Video modelling: Examples of a skill that is taught are videotaped.

Explicit Strategy Instruction: Explicit strategies are given for a task (for instance, writing, or solving mathematical problems): routines to follow.

Self management: Students monitor their own behaviour and performance through self-tests.

Graphic organizer: A visual chart is used to organize a student's knowledge or ideas.

Facilitate skill generalization: To help generalization of a skill, a skill is trained in various contexts.

Kurth et al. found that students with autism have areas of strength in concrete, procedural academic tasks. They were less successful in performing abstract and inferential tasks, including passage comprehension, writing passages, and solving applied math problems (e.g. word problems). They also found that academic achievements of autistic students were better in an inclusive setting than in a special education setting [37] This means that adjusting the education in such a way that all students are able to follow (true inclusive education) works better than leaving education as it is and have separate additional classes for autistic students.

Twenty six university students within the autism spectrum were compared with 158 neurotypical university students (from various universities within the UK) in a study in which they were asked to self-report their strengths and 'challenges' [23]. The challenges mentioned by autistic students concerned the need for guidance and clear instructions, not knowing how to pace, absorption in one subject, processing time, organizational skills, attention problems, group work and supervisor relationships, visualising abstract concepts, motivation/procrastination, critical/creative thinking and research/data analysis. Theit strengths, as self-reported, were academic and critical writing, the ability to work long hours, to anderstand complex ideas, and memory.

When comparing students with and without a diagnosis of autism, autistic students' self-reported strengths more often contain [40]: attention to detail, logical reasoning, focus, systemizing, consistency, visual skills, creative solutions, retentiveness, repetitive tasks, numbers, auditory skills, and concentrativeness. Neurotypicals score themselves higher on organizing ability, verbal skills, emotional control, flexibility, social skills, multitasking, empathy and team work.

Strategy instruction has been proposed for students within the autism spectrum, for the subjects of reading, writing and mathematics. [67]. Strategies teach knowledge of procedures (i.e., how to do something). Strategy instruction teaches the rules, processes, or the order of the steps that are applied systematically that lead to a problem solution. Strategy instruction proved helpful, in these areas.

Interestingly, what is called strategy instruction resembles the procedural guidance that is proposed as helpful in general for software engineering education [45]. Here, we see that what is helpful for students with an autistic thinking style, might deliver better education in general.

5 AUTISTIC COGNITIVE STYLE

In this section, we try to define an autistic cognitive style, based on literature, to be able to compare that style with the cognitive skills that are related with Software engineering.

Autism can be characterized by a cognitive style [24]. Research in parents and siblings of autistic children suggests that this cognitive style has a normal distribution, with autistic individuals on the 'autistic' end, while the remainder of the distribution is 'neurotypical'. Neurotypical family members of autists, while in the 'neurotypical region', are close to the 'autistic region' of the bell curve [25]. That means that more people have an autistic cognitive style than only those with a diagnosis of ASD.

Weak Central Coherence. One of the characterizations of autistic cognitive style is 'weak central coherence' [21]. Strong central coherence means: being able to process information into a higher level meaning, at the expense of details. In contrast, weak central coherence means more attention to detail as to the whole. For instance, someone with ASD will in general be faster to spot a mistake in an architectural blueprint, and many people with ASD are especially good at software testing [65]. On the other hand, it will be more difficult for them to grasp the essence of a text.

Recent research suggests that weak central coherence in autistic people is not a global processing deficit, but a local processing bias: when permitted free choice, they show a reduced preference to report global properties of a stimulus, but when they are instructed to report global properties, they are as able to do so as neurotypicals. A better description of 'weak central coherence' is, therefore, preference for local processing ('strong local processing'), or a disinclination of global processing('central processing avoidance') [35].

Thus, a focus on details is one of the aspects of an autistic cognitive style. The fact that this preference for local processing is not (only) a voluntary choice, has been elaborated in the theory of enhanced functional processing [42], which states that autistic perception is locally oriented (visual and auditory) and has enhanced low-level discrimination. The 'involuntary' aspect seems to be the fact that switching from local to global is hard, for autists [55].

People within the autism spectrum are, for instance, less fooled by some visual illusions than neurotypicals, because of the strong local processing [24]. Here, we see that the autistic cognitive style is bottom-up, in contrast to the top-down thinking style that neurotypicals often use.

Explicit rules for categorization. This preference for local processing may be the reason behind the enhanced discrimination skills found in autism (discrimination is the ability to respond to differences in stimuli) [9]. Enhanced discrimination skills may form a hindrance for the task of categorization (the action or process of placing concepts or objects into classes or groups). Each individual object is perceived different from all other objects, which makes it difficult to create classes [56].

On the other hand, when taught a rule for categorization, autistic children are at least equally capable of categorization as other children [9]. Autistic cognitive style is thus characterized by strong discrimination skills, and by the need for (explicit) rules for categorization.

Autistic individuals have to *learn* categorical information because they miss the automatic mechanisms that allow neurotypicals to form prototypical representations of information spontaneously. Therefore, abilities that rely on the formation of prototypes, such as facial recognition, emotional expression, and the organization of information into different categories, are affected in autism. Individuals with autism must develop their own idiosyncratic strategies to perform categorical organization and discrimination tasks [69]. For instance, when asked to sort books, autistic individuals more often sort by color or size than neurotypicals [52]. The reason is that categorization of the contents of books is more difficult than categorization based on color.

For autistic individuals, it is difficult to discern which details are the most salient (for instance, those details that are socially important).

Context blindness. Another way of looking at the autistic cognitive style is to see it as context blindness [64]. Context blindness explains, for instance, why autistic people have such a hard time processing ambiguous information. Their brain does not use context to process information, which means that ambiguousness cannot be resolved by context. A preference for unambiguous language (logic, mathematics) is, therefore, also one of the aspects of an autistic cognitive style.

The positive side of context blindness, is that people within the autism spectrum make more consistent decisions: they are more likely than neurotypicals to represent the value of each attribute or option in isolation, rather than being influenced by the other items in a choice set. [17].

Rational reasoning. People within the autistic spectrum tend to prefer deliberate, rational reasoning ('system 1 thinking' [31]) to intuitive, fast reasoning ('system 2 thinking') , probably because their brain does not support intuitive reasoning as much as the brains of neurotypicals [8].

Weak generalization. The memory style of autistic individuals is a 'look-up table memory style', versus an 'interpolation' memory style in neurotypicals [49]. Autistic people use precise information, and will have difficulties with generalization, while neurotypical people learn by generalization. Generalization is the ability to reason inductively, to broaden something specific into something more general, by focusing on similarities.

Categorization (which we discussed before) is a form of generalization. Generalization is poorly developed in individuals with autism [48]. Also, there is a link with the focus on details, which prevents seeing what is the same between situations as opposed to what is different.

People in the autism spectrum tend to make decisions on the basis of (too) limited evidence (they tend to 'jump to conclusions') [30]. This is because autistic individuals try to understand the world by applying rules. Jumping to conclusions means that they presume rules based on too little information. In other words, with an autistic cognitive style, it is difficult to form abstractions (generalization as forming categories), and one tends to form rules, based on data, too soon (jumping to conclusions).

Systemizing. Austistic people show a high 'Systemizing quotient'. Systemizing is the drive to analyze systems or construct systems, to analyze the variables in a system, and to derive the underlying rules that govern the behavior of a system. Autistic people show a higher degree of systemizing than neurotypicals [4]. Systemizing differs from categorization and generalization: systemizing means that one forms structure bottom-up, from the details, analyzing data and constructing rules that explain the data (deductive resoning), while categorization and generalization means to form structure using a top-down approach (inductive reasoning).

Executive functioning. 'Executive functioning' is an umbrella term for those functions that are needed to reach a goal: planning, working memory, impulse control, inhibition, shifting attention, and the initiation and monitoring of action. In some of these areas, people within the autistic spectrum show impairments, in particular [26]:

Planning and organization are difficult for people within the autistic spectrum. They have poor time management, and difficulties in prioritizing, coordination and sequencing of activities.

Mental flexibility is impaired. Switching to a new train of thought, for instance, is a difficult task for people within the autistic spectrum. When task instructions do not contain an explicit indication of the rules to be applied, and do not explicitly state that a rule switch will occur, results show rather consistent cognitive flexibility deficits in autism [60]: it is difficult, when you are autistic, to detect that the rules have changed. Another aspect of mental flexibility is the ability to handle exceptions to a rule. People with an autistic cognitive style are good at conditional reasoning, but have problems with exceptions to a rule [47].

6 COGNITIVE STYLE AND SOFTWARE ENGINEERING

In this section, we discuss the thinking skills that are needed within software engineering, and compare them to the aspects of the autistic cognitive style that we reviewed in the previous section. The cognitive skills that we discern are: computational thinking, thinking 'like an engineer', and academic skills.

6.1 Computational thinking

Thinking like a computer scientist is coined as 'Computational thinking' by Jeanette Wing [70].

Computational thinking has a long and rich history [58], with, for instance, Dijkstra who stated that for algorithmic thinking, one should be able to transform informality into formality, that one should be able to form ones own formalisms and concepts, and that one should be able to go back and forth between various levels of abstraction [13]. Another example is Knuth, who stated that computational thinking involves representing reality, the reduction of a problem into simpler problems, abstract reasoning, information structures, attention to algorithms, managing complexity, and reasoning about causality [34].

Computational thinking is composed of at least three components [70]: algorithmic thinking, 'the thought processes involved in formulating problems so their solutions can be represented as

computational steps and algorithms [1], abstraction (some see abstraction as the base of computing [36, 71]), and decomposition (the divide and conquer approach to problem solving).

Because a model to operationalize computational skills has been established [2], we compare these operationalizations with what we established about autistic thinking. This model discerns abstraction, generalization, algorithmic thinking, modularity, and decomposition.

Abstraction. Abstraction is operationalized by:

(1) *Separate the important from the redundant information*
(2) *Analyze and specify common behaviors or programming structures between different scripts*
(3) *Identify abstractions between different programming environments*

Separating the important from the redundant information is in direct conflict with the detail-focused cognitive style of autistic thinking. Separating important from redundant information is a form of categorization, and generalization. To be able to categorize, people with an autistic cognitive style need explicit rules.

Even when explicit rules have been given, it is a difficult task to recognize which information is redundant, when the superfluous information is described in various, slightly different, ways. In addition, context blindness plays a role: deciding which information is important, is only possible when one is aware of the context.

This aspect of abstraction plays a role in, for instance, problem analysis. It also plays a role in deciphering assignments, in understanding what the important aspects of an assignment are. Also, OO-modeling will probably be difficult without explicit guidance on how to capture a problem domain into a model.

Analyzing and specifying common behaviors or programming structures between different scripts is more concrete. People within the autism spectrum are more inclined to spot the differences than the commonalities. However, if explicit rules are given, this task will probably be doable for students with an autistic cognitive style. One may, for instance, show how to detect (almost) duplicate code, and how to create functions or methods to catch the commonalities.

Identifying abstractions between different programming environments (what is meant here, is to learn to work with various environments, such as Eclipse or IntelliJ), is, like the first aspect, in direct conflict with an autistic cognitive style, for the same reason. Discerning the details from the abstractions in programming environments demands categorization skills, that need explicit rules, for students with an autistic cognitive style.

It is probable that students with an autistic cognitive style will encounter more difficulties when asked to work with a new software tool or programming environment. In the words of a student:

> "For this assignment, I had to master too many new (for me) concepts: new environments (OS: Linux, IDE: Qt Creator), new language (C/C++). The teacher spends (almost) no time on these new concepts, so I guess that this comes naturally for other students."

Summarizing, to teach abstraction to students with an autistic cognitive style, we need to pay additional attention. We may try to formulate rules to follow, to perform abstraction. These rules should be accompanied by exercises. Abstraction is a skill that cannot be taken for granted in the presence of an autistic cognitive style.

Generalization. Generalization (transferring a problem-solving process to a wide variety of problems) is operationalized by:

(1) *Expand an existing solution in a given problem to cover more possibilities/cases*

As we have seen, weak generalization is one of the characteristics of autistic thinking. One of the strategies of students with an autistic cognitive style is to systemize: to form structures 'bottom-up', and, doing so, find rules that might be used to generalize. Another strategy seems to be to 'jump to conclusions': to form rules from (too) few data.

Another view on generalization is that it is the ability to transfer a solution from one context to another. When it is clear what is context and what is the solution, this might not pose problems for students with autistic thinking, but the problem is, of course, that differentiating context from the essence is difficult. In most occasions, it is not made explicit what part of a case is context and which part is the essence, or even what the context of a problem is.

This means that, for instance, 'learning by example' will be difficult for students with an autistic cognitive style, unless it is made explicit what the essence of the example is. Also, because with an autistic cognitive style, one tends to jump to conclusions, examples may very easily put students on a wrong track.

In the words of a student:

> "Often, descriptions of assignments are unclear, but for me, it is even more difficult when there is no clear structure in the assignments. If I have to bridge a too wide gap between conceptual knowledge and practical knowledge, I get overwhelmed, and then I cannot think any more. If, on the contrary, we start with small assignments, each training one particular aspect, and later on, we combine these aspects in assignments, everything goes well."

Summarizing, generalization is difficult for students with an autistic cognitive style. As a remedy, teachers can try to be explicit about what constitutes context, and can explain explicitly which parts can be transferred into other contexts. Also, it is important to realize that 'learning by example' does not work for students with an autistic cognitive style. When giving examples, one has to spell out what the essence of each example is, and preferably give explicit rules or guidelines to follow.

Algorithm. Algorithm (writing step-by-step specific and explicit instructions for carrying out a process), operationalized by:

(1) *Explicitly state the algorithm steps*
(2) *Identify different effective algorithms for a given problem*
(3) *Find the most efficient algorithm*

Explicitly stating the algorithm steps suggests a procedural approach for algorithms. Creating algorithms in a procedural way

combines well with a bottom-up thinking style and in particular with the preference for rational reasoning. To teach students how to follow a top-down approach, rules and guidelines are needed: the rules and guidelines from Felleisen [18] might help students with an autistic cognitive style to create algorithms with the end goal in mind, in a more top-down approach.

However, it is important how the problem that the algorithm should solve is formulated. Context that seems so obvious in the eyes of the teacher that it is left out, can form a hindrance for students with an autistic cognitive style.

If the problem is formulated with all context explicit, we see no conflicts with an autistic cognitive style with respect to identifying different effective algorithms for a given problem and finding the most efficient algorithm, in particular when given a clear definition of what is meant with 'efficient' (for instance, the fastest, the least code, and so forth).

Summarizing, bottom-up algorithmic thinking is probably one of the strengths within autism. Rules and guidelines for a more top-down way of working are a welcome help, and it is important to be explicit in the formulation of problems and exercises.

Modularity. Modularity (encapsulating elements such that they can be used independently), is operationalized by:

(1) *Develop autonomous code sections for use in the same or different problems*

In this case, context blindness is a double-edged sword.

On the one hand, context-blindness makes it easier to develop code that can be used in any context: developing autonomous code sections might be a strong point in students with an autistic cognitive style.

On the other hand, the same applies as in the case of algorithmic thinking: it is important how the problem that the code should solve is formulated. Also, context-blindness may lead to implicitly assuming a specific context, without realizing that. Specifying explicit pre- and postconditions can help.

In the words of our students:

> "What do you mean with 'making a selection'? Do you mean a choice (= selection)? Or do you mean a set (= selection). Dutch ! = Java. The question has more than one meaning.
>
> 'A selection of columns' can mean, at the level of the user, that a specific column should be chosen to be read, but also, that the program should use a specific set of columns. This is because 'selection' can both point to the process of choosing, as to the choice itself."

Summarizing, the context and formulation of the problem should be very clear. It is advisable to ask to explicitly specify pre- and postconditions.

Decomposition. Decomposition (breaking down problems into smaller parts that may be more easily solved), operationalized by:

(1) *Break down a problem into smaller/simpler parts that are easier to manage*

Here, the problem of discerning the essence from additional context may also form a problem. The lack of central coherence and problems with executive functioning (planning and organization) may for a hindrance with respect to decomposition. In order to decompose a problem into smaller steps, one should be able to see the problem as a whole in the first place (instead of as a sum of details), and one should be able to discern the essence from less important details.

Decomposition is, in essence, a form of top-down thinking, and we have seen that the strength in autistic thinking is bottom-up. Decomposition can also be seen as part of executive functioning (it has to do with planning and organization). As we have seen, there are impairments here.

Summarizing, for the skill of decomposition, there are several hindrances for students with an autistic cognitive style. As in the case of abstraction, these students probably need more explanation and more exercises to master this skill.

6.2 Thinking as an engineer

Engineers seek optimal solutions to problems. Engineers should be able to explain why a particular solution to a problem is best [51].

Frank [20] discerns three categories in engineering: *aims* (engineering design is directed toward the creation of new technological components), *knowledge, processes and tools* (which means that a knowledge base should be created, models and laws should be applicated, heuristics should be used), and *thinking*. In thinking, he discerns:

Synthesis. If synthesis is defined as 'an aspiration to understand how' (as Frank does), an autistic thinking mind will not have difficulties with this skill. If, on the contrary, it is defined as the skill to create a whole from parts, we may expect a need for additional rules of thumb, and exercises.

Concrete thinking. The preference for local processing, for thinking in details, and the weak generalization, means that concrete thinking is a strong point in an autistic cognitive style. Concrete thinking is the default thinking style for students with an autistic cognitive style.

Systems thinking. Systems thinking can be translated as looking at the whole instead of at the parts. As we have seen, seeing the whole is a difficult task for someone with an autistic cognitive style. Students with this thinking style will require rules of thumb, and exercises, to learn to see the whole, and to pay respect to the whole.

Advance toward the desirable. When thinking about how to reach the goal, it must be clear what the goal is in the first place. It is important to make the goal explicit, for students with an autistic style of thinking.

As we have seen, executive functioning applies to what is needed to reach a goal. Executive functioning is weak in autistic people. To determine how to reach the desirable, rules of thumb will be needed: explicit guidelines.

Optimal solution. What is optimal should be made clear, or students should be taught how to define optimal themselves. If that is clear, thinking about what is optimal can be done by rational reasoning, which is a strong point.

6.3 Academic thinking

Academic thinking comprises, at least, critical thinking and higher-order thinking.

Critical thinking. (being able to make an evaluation or judgment [39]) Critical thinking skills have been formulated as follows [16]:

Interpretation (to comprehend and express the meaning or significance of a wide variety of experiences, situations, data, events, judgments, conventions, beliefs, rules, procedures, or criteria)
As we have seen, interpretation is difficult for students with an autistic cognitive style, unless the context has been made clear.
On the other hand, context-blindness also has an advantage with respect to interpretation: students with an autistic cognitive style will not automatically assume a context, and may, therefore, see different interpretations, that are equally valid. Forming rules as a string point will also help with interpretation.

Analysis (to identify the intended and actual inferential relationships among statements, questions, concepts, descriptions, or other forms of representation intended to express belief, judgment, experiences, reasons, information, or opinions)
Context blindness can both be a hindrance (when context is held implicit) and an advantage (when assuming a context hinders others to see alternative ways for analysis). On the other hand, the focus on rules, and the ability to form rules, is a strong point when analyzing a text. The rational reasoning aspect of the analysis process is a strong point as well.
A bottom-up thinking style may lead to a different analysis than a top-down thinking style. With respect to analysis, one can therefore see diversity in thinking style as positive. reasoning will be more deductive than inductive.
Analysis can be thought of as a form of systemizing: finding the rules, the patterns, in a given situation, and applying them. As we have seen, systemizing is a strong point in autistic thinking.

Inference (to identify and secure elements needed to draw reasonable conclusions; to form conjectures and hypotheses; to consider relevant information and to reduce the consequences flowing from data, statements, principles, evidence, judgments, beliefs, opinions, concepts, descriptions, questions, or other forms of representation)
Weak generalization may be a hindrance in the inference process; rational reasoning is a strong point.

Evaluation (to assess the credibility of statements or other representations that are accounts or descriptions of a person's perception, experience, situation, judgment, belief, or opinion; and to assess the logical strength of the actual or intended inferential relationships among statements, descriptions, questions, or other forms of representation)
Assessing the logical strength of relationships among statements etc. will be a strong point, in an autistic cognitive style.
With respect to assessing the credibility of perceptions, experiences, and so forth, students with an autistic cognitive style will use the same rational reasoning, without respect for context. This may be both a hindrance and an advantage.

Explanation (to state and to justify that reasoning in terms of the evidential, conceptual, methodological, criteriological, and contextual considerations upon which one's results were based; and to present one's reasoning in the form of cogent arguments)
For students with an autistic cognitive style, we see no problems.

Self regulation (self-consciously to monitor one's cognitive activities, the elements used in those activities, and the results educed, particularly by applying skills in analysis, and evaluation to one's own inferential judgments with a view toward questioning, confirming, validating, or correcting either one's reasoning or one's results)
The lesser mental flexibility in autism might lead to a more rigid thinking style, that might form a hindrance with respect to self-regulation.

Higher-order thinking. Occurs when a person takes new information and information stored in memory and interrelates and/or rearranges and extends this information to achieve a purpose or find possible answers in perplexing situations [39].

Because of the 'look-up table memory style', this may be one of the strong points of students with an autistic cognitive style. However, because the answers are often found using a 'different', bottom-up thinking style, the answers may sometimes be unconventional in the eyes of others.

When abstraction and generalization are needed, we refer to what we concluded about those skills.

6.3.1 Academic writing. Most students struggle with academic writing. It is, therefore, interesting to check whether some aspects of academic writing are especially hard for students with an autistic cognitive style.

Teachers often fail to explicitly describe what good academic writing style comprises[38].

Academic writing demands skills at various levels:

- Selecting/evaluating information sources: finding information in library and internet, and understanding which information is relevant;
- Synthesizing the ideas/arguments from other sources with one's own ideas/arguments;
- Referencing: conventions of citation, avoiding plagiarism, knowing why, when and whom to reference, understanding referencing as a method of a. providing evidence, acknowledging the work of others in the field, giving greater authority to one's own ideas, constructing knowledge;
- Writing ideas/arguments up into a structured,coherent text: structuring, language skills (spelling, grammar, rhetorical strategies, cohesion), using appropriate terminology, style, conventions, participating in specialist discourse, understanding rhetorical processes needed for the construction of knowledge [73].

All these skills should be explicitly taught to students. For students with an autistic cognitive style, we see several skills that will be especially difficult:

Understanding which information is relevant All students should be taught how to find relevant articles (for instance, by starting to glance over abstracts), but this is especially true for students with an autistic cognitive style: thinking in details is a hindrance when searching for relevant articles, and finding thousands of possibilities.

Making meaning with unfamiliar discourse Again, this is difficult for all students, but with an autistic cognitive style, context blindness might pose an additional problem. Students should be taught explicitly that concepts may have a (slightly) different meaning in another context, and they should be taught how to recognize the context in which a concept is used. Of course, this should be accompanied by exercises.

Structuring Structuring an academic text is partly a matter of convention, that can (and should) be taught explicitly. Partly, it depends on what is the most important part of a section, a paragraph, or a sentence, and to base the structure on that (following rules that can be taught).
As we have seen, deciding what is important is difficult with an autistic cognitive style: this should be taught, accompanied by exercises to train this skill.

7 GUIDELINES

Now, we are able to formulate a first set of guidelines for inclusive education for students within the autism spectrum, with respect to cognitive style, based on what we found in literature about autistic cognitive style and the thinking style in software engineering. These guidelines have been confirmed by anecdotic evidence from autistic students telling us about the problems they encounter, but we would like to gather more data.

Explicit Context. In general, texts should be formulated in such a way that one needs as little context as possible to understand what is meant. An autistic thinking style means that texts are read 'as is', and are processed as though there is no context. That means that texts will be difficult to follow when it is assumed that the reader will automatically fill in which context is presumed.

To make context explicit seems simpler than it is: one omits context because the context is presumed unconsciously. As a teacher, one has to put oneself in the shoes of someone who will read the text, with only the text as guideline for what it means, and nothing else.

Especially in the case of assignments and exams, it should be made very clear what one expects from a student.

On the other hand, teachers should supply guidance in teaching students how to read material without explicit context. There should be support, for instance, for how to find the implicit context in an academic source, for how to interpret a scientific article.

Explicit guidelines. In many areas, one should give explicit guidelines.

For instance, one cannot expect that everyone is able to 'learn by example'. One should point out what the salient aspects of the example are, and which general rules one may deduce from an example. Otherwise, one can expect that some students may have drawn very different conclusions, and will stick to those conclusions.

In particular where the approach is top-down (for instance, in problem analyzing and design based on such analysis), one should give explicit guidelines on how to do that. Also, one might try to find other ways to solve problems, that require a more bottom-up approach. Both top-down and bottom-up approaches may lead to good solutions.

Teachers should be aware of the fact that students with an autistic thinking style may come up with different solutions than the teacher might expect, because of their bottom-up thinking style.

Explicit guidelines are also needed where finding relevant aspects are important. For instance, one should explain how to proceed when trying to find relevant literature.

Students should also be given explicit guidelines with respect to structure texts.

With respect to 'thinking as an engineer, students with an autistic thinking style will probably show strengths. However, they will need rules and guidelines on how to pay respect to the whole system, as opposed to only parts of the system.

Exercises. When explicit guidelines are given on how to perform a task, these should be accompanied by exercises.

Consequences for education. These points have consequences in many areas. For instance, one may not take it for granted that students with an autistic thinking style will pick up what is relevant from listening to a talk. Handing out handouts with the salient points beforehand might help.

Course material should be revised. In places where examples are used to teach something, one should make explicit what the students should learn from these examples.

For many tasks, we should develop explicit guidelines. Sometimes, these guidelines may be very precise. At other times, they may state that there are no strict rules, and explain a general approach to tackle a problem.

In the very first place, teachers should be taught about autism and the autistic cognitive style, so they can see their course material and their lessons from the perspective of autistic students.

8 DISCUSSION

We formulated guidelines for inclusive education in software engineering, based on what is known about the autistic cognitive style and on the cognitive aspects of software engineering. As such, our exercise is not purely speculative, but do not have empirical evidence other then the anecdotical evidence that students sent us.

We think our effort is worthwhile nonetheless, for a couple of reasons.

First, It is very difficult to find conclusive information about hindrances in education for students within the autistic spectrum, for several reasons. In the first place, one faces the same difficulties as in estimating the number of students within the spectrum: not every student is willing to disclose his or her diagnosis, and not all students who are within the spectrum know that. Also, it is

difficult to point out hindrances you have, when you think that all fellow-students will probably face the same hindrances. To be able to point out what is a hindrance, a student would have to know how non-autistic students think, how the neurotypical cognitive style is.

Second, although Software engineering has a relatively high percentage of students within the autistic spectrum, what autism is and what the autistic cognitive style is, does not belong to the general knowledge of most lecturers. An overview of cognitive characteristics is therefore worthwhile in itself.

Third, giving explicit guidelines on how to proceed with a (complex) task, is part of the 4C/ID approach (the four component to instructional design model) to teaching complex tasks [41, 61]. It is interesting to see that many of the recommendations we give overlap with this preferred approach to teach complex tasks. It seems, therefore, that making out education more inclusive with respect to autistic students, will result in better education as a whole. Students within the autistic spectrum might be seen, so to speak, as canaries in the coal mine with respect to suboptimal education. Focusing on what such student need may be a good starting point to improve education.

9 CONCLUSIONS AND FUTURE WORK

Based on literature and on the knowledge of an expert, we formulated the autistic cognitive style. We compared these characteristics with the cognitive skills that are needed within Software engineeing. Based on that comparison, we predict possible hindrances. We formulated guidelines that might help to take away these hindrances.

Finding out how to support an autistic cognitive style is a new research area. Therefore, there are many ways in which we would like to extend this work, for instance:

Ask autistic students We want to ask autistic students about the difficulties they experience through questionnaires, and we will organize meetings with autistic students, to brainstorm about how education can be improved for them.

Coaching thesis writing Probably, writing a thesis is the most difficult part of the study for almost any student; for autistic students, this is especially true. We would like to investigate good practices in coaching autistic student while they write their thesis. This can be done by interviewing students, interviewing teachers, and by developing and trying out an approach.

Screening a course It would be worthwhile to develop a set of guidelines for education that can be used to screen a course on 'inclusiveness' with respect to autism. We would have to validate the guidelines in several ways: check whether they are concrete enough to use when reviewing a course and check whether the guidelines really help autistic students. We could interview both students and teachers to find out more about the problems they encounter.

Collaboration To support collaboration between students, both autistic and neurotypical, we would like to develop guidelines for collaboration that recognize different cognitive styles.

Explicit guidelines One of our goals is to develop explicit guidelines for processes that are inherently nondeterministic. Examples are, for instance, domain analysis or use case modeling. We would like to check whether explicit guidelines really help autistic students.

REFERENCES

[1] Alfred V Aho. 2012. Computation and computational thinking. *Comput. J.* 55, 7 (2012), 832–835.

[2] Soumela Atmatzidou and Stavros Demetriadis. 2016. Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems* 75 (2016), 661–670.

[3] Robert D Austin and Gary P Pisano. 2017. Neurodiversity as a competitive advantage. *Harvard Business Review* 95 (2017), 96–103.

[4] Simon Baron-Cohen, Jennifer Richler, Dheraj Bisarya, Nhishanth Gurunathan, and Sally Wheelwright. 2003. The systemizing quotient: an investigation of adults with Asperger syndrome or high–functioning autism, and normal sex differences. *Philosophical Transactions of the Royal Society B: Biological Sciences* 358, 1430 (2003), 361–374.

[5] Simon Baron-Cohen, Sally Wheelwright, Richard Skinner, Joanne Martin, and Emma Clubley. 2001. The autism-spectrum quotient (AQ): Evidence from asperger syndrome/high-functioning autism, malesand females, scientists and mathematicians. *Journal of autism and developmental disorders* 31, 1 (2001), 5–17.

[6] Olga Bogdashina. 2016. *Sensory perceptual issues in autism and asperger syndrome: different sensory experiences-different perceptual worlds.* Jessica Kingsley Publishers, London, UK.

[7] Nicholas A Bowman. 2010. College diversity experiences and cognitive development: A meta-analysis. *Review of Educational Research* 80, 1 (2010), 4–33.

[8] Mark Brosnan, Marcus Lewton, and Chris Ashwin. 2016. Reasoning on the autism spectrum: a dual process theory account. *Journal of autism and developmental disorders* 46, 6 (2016), 2115–2125.

[9] SM Brown and JM Bebko. 2012. Generalization, overselectivity, and discrimination in the autism phenotype: A review. *Research in Autism Spectrum Disorders* 6, 2 (2012), 733–740.

[10] Luiz Fernando Capretz and Faheem Ahmed. 2010. Why do we need personality diversity in software engineering? *ACM SIGSOFT Software Engineering Notes* 35, 2 (2010), 1–11.

[11] Nick Chown and Nick Beavan. 2012. Intellectually capable but socially excluded? A review of the literature and research on students with autism in further education. *Journal of Further and Higher Education* 36, 4 (2012), 477–493.

[12] Bernard J Crespi. 2016. Autism as a disorder of high intelligence. *Frontiers in neuroscience* 10 (2016), 300.

[13] Edsger W Dijkstra. 1974. Programming as a discipline of mathematical nature. *The American Mathematical Monthly* 81, 6 (1974), 608–612.

[14] Fifth Edition. 2013. *Diagnostic and Statistical Manual of Mental Disorders, DSM-5.* American Psychiatric Association, Washington DC, USA.

[15] Nathan Ensmenger. 2015. "Beards, Sandals, and Other Signs of Rugged Individualism": Masculine Culture within the Computing Professions. *Osiris* 30, 1 (2015), 38–65.

[16] Peter A Facione et al. 1998. Critical thinking: What it is and why it counts. *Retrieved June* 9 (1998), 2004.

[17] George D. Farmer, Simon Baron-Cohen, and William J. Skylark. 2017. People With Autism Spectrum Conditions Make More Consistent Decisions. *Psychological Science* 28, 8 (2017), 1067–1076.

[18] Matthias Felleisen, Robert B. Findler, Matthew Flatt, and Krishnamurthi Shriram. 2001. *How To Design Programs, An Introduction to Programming and Computing.* The MIT press, Cambridge, Massachusetts, Londen, England.

[19] Veronica P. Fleury, Susan Hedges, Kara Hume, Diane M. Browder, Julie L. Thompson, Kathy Fallin, Farah El Zein, Colleen Klein Reutebuch, and Sharon Vaughn. 2014. Addressing the academic needs of adolescents with autism spectrum disorder in secondary education. *Remedial and Special Education* 35, 2 (2014), 68–79.

[20] Moti Frank. 2006. Knowledge, abilities, cognitive characteristics and behavioral competences of engineers with high capacity for engineering systems thinking (CEST). *Systems Engineering* 9, 2 (2006), 91–103.

[21] Uta Frith. 1989. *Autism: Explaining the enigma.* Vol. 1989. Wiley-Blackwell, Malden MA, USA.

[22] Nicholas W Gelbar, Isaac Smith, and Brian Reichow. 2014. Systematic review of articles describing experience and supports of individuals with autism enrolled in college and university programs. *Journal of autism and developmental disorders* 44, 10 (2014), 2593–2601.

[23] Emine Gurbuz, Mary Hanley, and Deborah M. Riby. 2019. University Students with Autism: The Social and Academic Experiences of University in the UK. *Journal of autism and developmental disorders* 49, 2 (2019), 617–631.

[24] Francesca Happé. 1999. Autism: cognitive deficit or cognitive style? *Trends in cognitive sciences* 3, 6 (1999), 216–222.

[25] Francesca Happé, Uta Frith, and J Briskman. 2001. Exploring the cognitive phenotype of autism: weak "central coherence" in parents and siblings of children with autism: I. Experimental tests. *The Journal of Child Psychology and Psychiatry and Allied Disciplines* 42, 3 (2001), 299–307.

[26] Elisabeth L Hill. 2004. Executive dysfunction in autism. *Trends in cognitive sciences* 8, 1 (2004), 26–32.

[27] Grace Iarocci and John McDonald. 2006. Sensory integration and the perceptual experience of persons with autism. *Journal of autism and developmental disorders* 36, 1 (2006), 77–90.

[28] Pier Jaarsma and Stellan Welin. 2012. Autism as a natural human variation: Reflections on the claims of the neurodiversity movement. *Health Care Analysis* 20, 1 (2012), 20–30.

[29] Jordynn Jack. 2011. 'The Extreme Male Brain?' Incrementum and the Rhetorical Gendering of Autism. *Disability Studies Quarterly* 31, 3 (2011), 1041–5718.

[30] Claire Jänsch and Dougal Julian Hare. 2014. An investigation of the 'jumping to conclusions' data-gathering bias and paranoid thoughts in Asperger syndrome. *Journal of autism and developmental disorders* 44, 1 (2014), 111–119.

[31] Daniel Kahneman. 2011. *Thinking, fast and slow*. Macmillan Publishers, NY, USA.

[32] Leo Kanner et al. 1943. Autistic disturbances of affective contact. *Nervous child* 2, 3 (1943), 217–250.

[33] Steven K Kapp, Kristen Gillespie-Lynch, Lauren E Sherman, and Ted Hutman. 2013. Deficit, difference, or both? Autism and neurodiversity. *Developmental psychology* 49, 1 (2013), 59.

[34] Donald E Knuth. 1974. Computer science and its relation to mathematics. *The American Mathematical Monthly* 81, 4 (1974), 323–343.

[35] Kami Koldewyn, Yuhong V Jiang, Sarah Weigelt, and Nancy Kanwisher. 2013. Global/local processing in autism: Not a disability, but a disinclination. *Journal of autism and developmental disorders* 43, 10 (2013), 2329–2340.

[36] Jeff Kramer. 2007. Is abstraction the key to computing? *Commun. ACM* 50, 4 (2007), 36–42.

[37] Jennifer A Kurth and Ann M Mastergeorge. 2010. Academic and cognitive profiles of students with autism: implications for classroom practice and placement. *International Journal of Special Education* 25, 2 (2010), 8–14.

[38] Mary R Lea and Brian V Street. 1998. Student writing in higher education: An academic literacies approach. *Studies in higher education* 23, 2 (1998), 157–172.

[39] Arthur Lewis and David Smith. 1993. Defining higher order thinking. *Theory into practice* 32, 3 (1993), 131–137.

[40] Timo Lorenz and Kathrin Heinitz. 2014. Aspergers–different, not less: Occupational strengths and job interests of individuals with Asperger's syndrome. *PloS one* 9, 6 (2014), e100358.

[41] Jeroen J.G. van Merriënboer and Paul A. Kirschner. 2017. *Ten steps to complex learning: A systematic approach to four-component instructional design*. Routledge, NY, USA.

[42] Laurent Mottron, Michelle Dawson, Isabelle Soulieres, Benedicte Hubert, and Jake Burack. 2006. Enhanced perceptual functioning in autism: an update, and eight principles of autistic perception. *Journal of autism and developmental disorders* 36, 1 (2006), 27–43.

[43] Ann M Mulder and Andrew Cashin. 2014. The need to support students with autism at university. *Issues in mental health nursing* 35, 9 (2014), 664–671.

[44] Francisco Ortega. 2009. The cerebral subject and the challenge of neurodiversity. *BioSocieties* 4, 4 (2009), 425–445.

[45] Harrie Passier. 2017. The Role of Procedural Guidance in Software Engineering Education. In *Companion to the First International Conference on the Art, Science and Engineering of Programming (Programming '17)*. ACM, New York, USA, Article 21, 2 pages.

[46] Vreda Pieterse, Derrick G Kourie, and Inge P Sonnekus. 2006. Software engineering team diversity and performance. In *Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*. South African Institute for Computer Scientists and Information Technologists, ACM, New York, USA, 180–186.

[47] Judith Pijnacker, Bart Geurts, Michiel Van Lambalgen, Cornelis C Kan, Jan K Buitelaar, and Peter Hagoort. 2009. Defeasible reasoning in high-functioning adults with autism: Evidence for impaired exception-handling. *Neuropsychologia* 47, 3 (2009), 644–651.

[48] K. C. Plaisted. 2001. Reduced generalization in autism: An alternative to weak central coherence. In *The development of autism: Perspectives from theory and research*, J.A. J. A. Burack, T. Charman, N. Yirmiya, and P.R. Zelazo (Eds.). Lawrence Erlbaum Associates Publishers, Mahwah, New Jersey, United States, 149–169.

[49] Ning Qian and Richard M Lipkin. 2011. A learning-style theory for understanding autistic behaviors. *Frontiers in human neuroscience* 5 (2011), 77.

[50] Allison B Ratto, Lauren Kenworthy, Benjamin E Yerys, Julia Bascom, Andrea Trubanova Wieckowski, Susan W White, Gregory L Wallace, Cara Pugliese, Robert T Schultz, Thomas H Ollendick, et al. 2018. What about the girls? Sex-based differences in autistic traits and adaptive skills. *Journal of autism and developmental disorders* 48, 5 (2018), 1698–1711.

[51] John A Robinson. 1998. Engineering thinking and rhetoric. *Journal of Engineering Education* 87, 3 (1998), 227–229.

[52] Danielle Ropar and David Peebles. 2007. Sorting preference in children with autism: the dominance of concrete features. *Journal of autism and developmental disorders* 37, 2 (2007), 270–280.

[53] Emily Ruzich, Carrie Allison, Paula Smith, Peter Watson, Bonnie Auyeung, Howard Ring, and Simon Baron-Cohen. 2015. Measuring autistic traits in the general population: a systematic review of the Autism-Spectrum Quotient (AQ) in a nonclinical population sample of 6,900 typical adult males and females. *Molecular autism* 6, 1 (2015), 2.

[54] Wayne Sailor and Blair Roger. 2005. Rethinking inclusion: Schoolwide applications. *Phi Delta Kappan* 86, 7 (2005), 503–509.

[55] María Felipa Soriano, Antonio J Ibáñez-Molina, Natalia Paredes, and Pedro Macizo. 2017. Autism: Hard to Switch from Details to the Whole. *Journal of abnormal child psychology* 46, 6 (2017), 1–13.

[56] Isabelle Soulières, Laurent Mottron, Daniel Saumier, and Serge Larochelle. 2007. Atypical Categorical Perception in Autism: Autonomy of Discrimination? *Journal of Autism and Developmental Disorders* 37, 3 (01 Mar 2007), 481–490.

[57] Annelies A Spek and E Velderman. 2013. Examining the relationship between autism spectrum disorders and technical professions in high functioning adults. *Research in Autism Spectrum Disorders* 7, 5 (2013), 606–612.

[58] Matti Tedre and Peter J Denning. 2016. The long quest for computational thinking. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM, New York, USA, 120–129.

[59] Ernst van Bergeijk, Ami Klin, and Fred Volkmar. 2008. Supporting more able students on the autism spectrum: College and beyond. *Journal of autism and developmental disorders* 38, 7 (2008), 1359.

[60] Sander Van de Cruys, Kris Evers, Ruth Van der Hallen, Lien Van Eylen, Bart Boets, Lee de Wit, and Johan Wagemans. 2014. Precise minds in uncertain worlds: Predictive coding in autism. *Psychological review* 121, 4 (2014), 649.

[61] Jeroen JG Van Merriënboer. 1997. *Training complex cognitive skills: A four-component instructional design model for technical training*. Educational Technology Publications, Englewood Cliffs, NJ, USA.

[62] Daniel Varona, Luiz Fernando Capretz, Yadenis Piñero, and Arif Raza. 2012. Evolution of software engineers' personality profile. *ACM SIGSOFT Software Engineering Notes* 37, 1 (2012), 1–5.

[63] Berend Verhoeff. 2013. Autism in flux: a history of the concept from Leo Kanner to DSM-5. *History of Psychiatry* 24, 4 (2013), 442–458.

[64] Peter Vermeulen. 2015. Context blindness in autism spectrum disorder: Not using the forest to see the trees as trees. *Focus on autism and other developmental disabilities* 30, 3 (2015), 182–192.

[65] Jonathan Wareham and Thorkil Sonne. 2008. Harnessing the power of autism spectrum disorder (Innovations case narrative: specialisterne). *Innovations: Technology, Governance, Globalization* 3, 1 (2008), 11–27.

[66] Xin Wei, W Yu Jennifer, Paul Shattuck, Mary McCracken, and Jose Blackorby. 2013. Science, technology, engineering, and mathematics (STEM) participation among college students with an autism spectrum disorder. *Journal of autism and developmental disorders* 43, 7 (2013), 1539–1546.

[67] Peggy J Schaefer Whitby, Jason C Travers, and Jamie Harnik. 2009. Academic achievement and strategy instruction to support the learning of children with high-functioning autism. *Beyond Behavior* 19, 1 (2009), 3–9.

[68] Susan W White, Thomas H Ollendick, and Bethany C Bray. 2011. College students on the autism spectrum: Prevalence and associated problems. *Autism* 15, 6 (2011), 683–701.

[69] Diane L Williams and Nancy J Minshew. 2010. How the brain thinks in autism: Implications for language intervention. *ASHA Leader* 15 (2010), 8.

[70] J. M. Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.

[71] Jeannette M Wing. 2008. Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences* 366, 1881 (2008), 3717–3725.

[72] Lorna Wing. 1988. The continuum of autistic characteristics. In *Diagnosis and assessment in autism*. Springer, Springer, Boston, MA, USA, 91–110.

[73] Ursula Wingate. 2006. Doing away with 'study skills'. *Teaching in higher education* 11, 4 (2006), 457–469.

[74] Sula Wolff. 2004. The history of autism. *European child & adolescent psychiatry* 13, 4 (2004), 201–208.

Programming for teachers: Reflections on the design of a course supporting flexible learning trajectories

ABSTRACT

How to design an online flexible learning trajectory course where students are in-service teachers with varied level of programming knowledge, interests, and different application need? This paper presents the design of such a course for teachers on applied programming. The main learning objective of the course is to provide in-service teachers with insight into how programming can be used to create digital solutions. The course is practically directed and emphasizes programming as a constructive and creative tool. The course is aimed at teachers in secondary schools. The paper describes the main design choices of the course. Based on the experience with the course, the paper reflects on the challenges to design courses that do not support a single learning path for all the students, but rather aims at providing a context where students can identify and follow the learning path that is best fitting for their competencies, interests, and needs of the local practices.

CCS CONCEPTS

• **Applied computing** → **Education**; **E-learning**; *Distance learning*; • **General and reference** → Design; • **Social and professional topics** → *Computer science education*; Adult education.

KEYWORDS

Online course, Programming for teachers, Flexible learning trajectories, Applied programming, Learner-Centered Design, Continued Education

ACM Reference Format:

. 2019. Programming for teachers: Reflections on the design of a course supporting flexible learning trajectories. In *The 8th Computer Science Education Research Conference, 18-20 November 2019, Larnaca, Cyprus.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Teaching programming in schools is challenging, and can also be overwhelming if programming is not part of your educational background. The demand for incorporation of programming into the curriculum of various subjects in schools is on the rise, and teachers need some additional education and guidance regarding that. The course is aimed at in-service teachers who represent the group that has the most difficulties with this topic as they usually did not have any kind of formal programming education in their careers thus far. Some of the challenges that are discussed in this paper are: Students are in-service teachers who teach at different levels (primary, secondary, and upper secondary level), different needs for application of programming to varying levels of education, flexible course versus class size and assessment, "learning programming is hard" and "programming for all".

In this paper, we reflect on how these challenges in the everyday practice of teachers influence the design of in-service training for teachers. The discussion is based on the experience with the design on an online course: *Applied programming*.

The paper is organized as follows: Section 2 presents related work and positions the paper in the context of current teacher training in the area of programming. Section 3 presents the overall design of the course, and Section 4 discuss the results on the course based on the instructor's experience and feedbacks from participants. Lastly, section 5 outlines some implications for course designers and instructors.

2 RELATED WORK

"Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone" [12]. Wing argues that computational thinking should be included in the lower education curriculum, which indeed is in coherence with the report *Students' Learning in the Future School* by the Norwegian Directorate of Education and Training [7].

A study on student engagement in online discussions [1], concluded with the need for including pedagogical course designs in closer relation to online learning. In the course which is described in this paper, Applied Programming for Teachers, we have tried to implement didactic features of online learning by using a learner-centered design for the course structure [2], which is explained further in section 3.

To understand the place that this course takes, we need to understand the variety of courses that are delivered in an online format. Massive Open Online Course (MOOC) is a widely adopted type of course and most dominant form found today. They are aimed towards participation on a large scale with availability to anyone with access to the internet. Various reports on the effectiveness of this type of course have been made. It has been reported that adoption and drop-out rates vary, in some cases 50% or more of the participants cease their activity in the first weeks with the decline to 16% in the following weeks [4]. Also, various reasons have been reported as reasons for high drop-out numbers, some of them include lack of motivation, poor contact with the staff, long times to receive answers to queries, too constrained syllabus, and similar [6, 13, 14]. As MOOCs usually do not have targeted demographics, the assessment of the course effectiveness becomes hard other than reporting competition numbers.

CSERC '19, 18-20 November 2019, Larnaca, Cyprus

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

In recent years, several projects for teaching coding to teachers have emerged [6],e.g., Switzerland has an ongoing educational reform that requires them to conduct continued education for teachers in regards to programming and computational thinking [5]. In the context of Norwegian schools, Simula Research Laborotory[1] has been working with teachers about how programming can be implemented in classroom situations with an interdisciplinary focus similar to the course described in this paper. In contrary to Simula's face-to-face and seminar-based approach, the Applied Programming for Teachers course has adopted a web-based structure, which according to Hadjerrouit [3] offers increased learning benefits when combined with a learner-centered structure.

3 DESCRIPTION OF METHOD AND CASE

The purpose of this study is to promote how to design a course with flexible learning trajectory for students who are in-service teachers. In the following sections we will give a brief description of the challenges and how to design such a course.

3.1 Background

The message from the Norwegian government, [9] says: *"The curriculum will be renewed so that it reflects the current school life and the challenges children and young people face today."*.

Norwegian Directorate for Education and Training (Utdanningsdirektoratet) is working on renewing all the curricula in primary and secondary education, which will be implemented from 2020 onwards. The purpose of renewing *"The knowledge promise"* is to make children and young people able to meet and find solutions for todays and future challenges. They will develop relevant expertise and good values and attitudes that affect the individual, in a society characterized by greater complexity, high diversity, and speed change"[10].

To support this, the Centre for Continuing Education and Professional Development at the University offers several courses within Information and Communication Technology (ICT) Programming. These are online courses that provide teachers with insight into how programming as a subprocess of the more significant problem-solving methodology is used to create digital solutions. Using a programming language can create a solution to a problem. The program is practically directed and emphasizes programming as a useful and creative tool. The focus is on how the programming subject can be communicated to students with a focus on creativity and collaboration in task solving. The program qualifies for teaching in programming at levels 8-13 (and earlier).

The target audience is teachers who need programming skills and insight into the possibilities of coding, design, and modeling of software-based solutions. These courses give guidance of programming in schools and other subjects and activities where programming is used to support learning.

Lectures are web-based, but the emphasis is placed on social and interactive learning with weekly activities such as online lectures and regular compulsory work requirements (exercises). Lectures include interactive learning materials and videos made available to students, and online collaboration and guidance are conducted in social spaces (Slack[2] and Blackboard[3]).

The teaching is based on the curriculum consisting of both textbook [2] and online resources. In teaching, we use both text and block-based programming tools. Through the obligatory exercises, the students will try out new academic and subject didactic knowledge in their teaching.

3.2 Challenges

Through the process of designing this course and further reflection we have identified some challenges which we briefly explain in this section.

Students are in-service teachers who teach at different levels (primary, secondary, and upper secondary level). Our focus is teachers who teach in level 8-13, but we also try to meet the needs of lower-level teachers. We achieve this by including games and block-based programming.

Programming at primary and secondary schools will require other forms of programming didactics than those who wish to apply for upper secondary school programming. This requires a flexible course content that will meet the needs of all students in order to be able to implement programming in their respective areas.

Different needs for application of programming at different levels of education. Teachers from the upper secondary level teach in many different subject areas (automation in technical subjects, natural science, and history/languages, etc.). At lower levels, the students have other needs and issues related to programming and its applications. This challenge requires highly flexible course content, and students should be able to choose the direction which is relevant to them.

Flexible course versus class size and assessment. There is a need to increase the number of students taking this course. Defining a course content and having a form of assessment where the workload does not escalate when the number of students increases, is, therefore, a challenge that must be dealt with in the right way.

Learning programming is hard. *"We have significant empirical evidence that learning to program is harder than teachers might predict."*[2]. One crucial aspect is motivation and a strong desire to learn programming. *"Critical to success in learning computing is wanting to learn computing"*[2]. Learning programming languages might be more comfortable for some people, and more difficult for others. *"Becoming a good programmer is incredibly difficult, and it does not happen quickly"*[11].

The aim of this course is, therefore, to teach programming to all students at Norwegian schools in a way that increases their interest and commitment. Through this course, our task is to transfer this engagement with the students (who are in-service teachers) in a way that enables them to see the benefits of applying programing in their area while also being able to increase their student's interests. How to learn programming concepts, thus becomes an essential aspect of this course.

Programming for all [2] There is a broad range of reasons for making computing education available to everyone. These are

[1] https://www.simula.no/news/simula-educates-teachers-programming

[2] https://slack.com/

[3] https://www.blackboard.com/about-us/index.html

explained in the textbook *Learner-Centered Design of Computing Education* by Mark Guzdial[2]. To achieve the goal of making computing education available to everyone, we will need to change how we teach computing. "The most common user interface design approach, user-centered system design, emphasizes understanding the user's tasks and helping her to achieve those tasks[8]. The method of Learner-Centered Design of computing education is used for designing this course.

The term *programming for all* may not be accepted by some students as they may not see the benefit of it from their point of view. It is therefore essential to delve into research that has been done in this area and to have a proven relationship with the importance of this message. It is also important to understand what the challenges of learning programming concepts are and how to overcome these.

3.3 Learning objectives and design principles

The learning objective is a very general statement about the broader goals of the course. For the course Basic programming for teachers, the learning objectives are that students have learned and reached an understanding of the most important programming concepts. An example of this is saving and retrieving data from different media and programming structures like repetitions, making choices, etc.

The course that is discussed in this paper, Applied Programming for Teachers, will give students a deeper understanding of these concepts and how this can be applied to solve issues within different subject areas. This course focuses on the knowledge students need as teachers, and how they can ease students learning process and understanding of programming. The course is primarily intended as a way of getting ready to teach programming by focusing on applications in natural sciences, electronics, and robotics. We also want to give students insight into programming concepts as well as a general understanding of modern programming languages, techniques, and methods.

The course is designed to inspire students through examples of programming applications and tasks where the students can use this in practice. Tasks provided give the students opportunities to work interdisciplinary, be creative, and collaborate.

Designing the course Applied Programming for Teachers has been based on the following principles:

- The course should focus on students at secondary- and high school level, but it should also be useful for primary schools
- A minimum level of programming skills is defined and should be gained by all students.
- Students who already know programming at a medium or advanced level should also find this course useful and should be able to develop their skills further.
- Focus is learner-centered course design: The most critical idea in learner-centered design is respect for the learner. Learner-centered design tells us to respect students motivations to learn and what they want to learn. We expect variety in our learners and rapid change as they learn. We need to construct learning opportunities for who the learner is and wants to be, not for the expert that we computer scientists might want them to be[2]. Students at a different level should select relevant content (different trajectories). They should also be able to select the level of complexity and follow their plan to complete the course.

The course is organized in a way that students learn programming through practical applications. Lectures in modules are using web-resources (interactive materials) to extend students programming knowledge. The exercises are practically arranged. The students program micro-electronic devices or modulate natural science mathematics related issues.

3.4 Organization of the learning material

The overall topics covered. In this course ("Applied programming for teachers"), we focus on the programming didactics, applications of programming and increase the general knowledge in software development. The topics we address can be divided into the following main areas:

(1) Increase understanding of the need to learn programming. We look at research within the challenges of learning programming and look at the reasons why everyone should learn to program. The book Learner-Centered Design of Computing Education[2] is used as a textbook for students.
(2) Applications of Programming: In this section, we focus on somewhat more complicated programs and application of programming in areas such as games, simple electronics, and robotics (such as Arduino[4], Raspberry PI[5], micro: bit[6], LEGO Mindstorms[7]), computing applications and simulations in subjects such as mathematics and physics. The students get a good overview of different applications of programming, and through the project, they are allowed to immerse themselves in applications that are relevant to their teaching and subjects.
(3) Specialization in programming / general knowledge of system development process: The focus here is to acquire more detailed knowledge of constructions and structures in modern programming, know programming languages, tools and methodology, both pedagogically oriented solutions and solutions that are used professionally.

Organization of the course

Fig. 1. shows how this course has been organized. Student activities are planned in four main loops: Webinars, self-study modules, exercises, and discussion forums.

One of the main objectives of the course is to make it flexible for the students in a way that they can control their working speed. The course is providing a plan and a recommendation on how to complete the course, but students can still complete the course earlier. The content of the course is divided into six modules, and each module consists of one or several related topics. Some of the modules (modules 3 and 4) provide a variety of topics related to applications of programming. Students will choose topics based on what is most relevant to them. They can also choose tasks from exercises that are related to topics they choose to study, that is, exercises follow whatever students choose in the study modules.

[4]https://www.arduino.cc/
[5]https://www.raspberrypi.org/
[6]https://microbit.org/
[7]https://www.lego.com/en-us/mindstorms

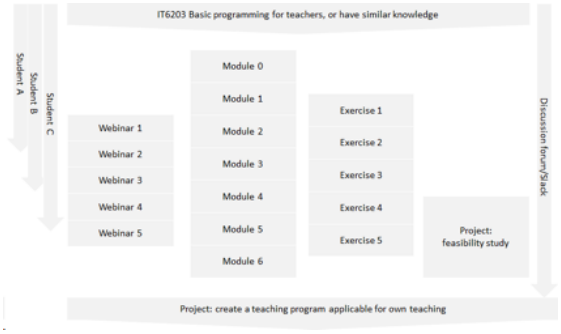


Figure 1: Organization of IT6204 Applied programming for teachers

Webinars are complementary online lectures and are meant to help students get a broader view of core topics. Also, students will have a chance to ask questions directly or discuss different subjects related to the course. All webinars are recorded and made available through the learning platform.

Module 1 - Introduction, why and how everyone should learn programming. There is a high demand for education in computing and information technology. Many students (both children and adults) are aware that they will need programming in the future. This module presents arguments for ubiquitous programming education while allowing for individual learning motivation and application of use. What do we mean when we talk about everyone having to learn to program? How can we create an educational program that works for everyone? In this module, we look at the use of a learner-centered design of computing education (student-centered) approach to reach a broad audience. Several reasons for teaching programming to everyone are discussed, and we study how the various cases lead to different choices of learning objectives and teaching methods.

Module 2 – Block based programming. This module deals with applications of programming using block-based programming. All students are encouraged to complete this module even though its content is more applicable to primary/secondary schools. The chosen technologies for this module are *micro: bit, code.org*[8]*, Scratch*[9]*, and Pocket Code*[10].

Module 3 - Principles, constructions, and structures in modern programming. In this module, students extend their programming knowledge on selected topics (relevant for natural science) to an advanced level. Also, they will learn more about algorithms and how to apply programming in natural science.

Module 4 - Understanding the software's function in electronics and robotics. In this module, we apply the application of programming in electronics and robotics. Several techniques (microelectronic technologies) are introduced, but it is up to students to choose which of these he/she wants to immerse in. Techniques introduced are relevant to teaching at all school levels.

[8]https://code.org/
[9]https://scratch.mit.edu/
[10]http://robotixedu.com/phiroresources/introduction-to-pocket-code.html

Module 5 - Game programming. Making games allows us to be high-interest, engaging, teaches foundational and transferable skills [11]. Using games in teaching can, therefore, be very motivating for the students, but it is crucial that the level is right so that they are not overwhelmed by information and technical difficulties. Game programming can be so advanced that it governs the overall IT development, both in terms of graphics and artificial intelligence. On the other hand, it can be done very quickly, and in this module, we make a game with Scratch (block programming).

Module 6 - Programming languages, tools, methodology, and testing. In this module, we address some key topics relevant to professional software development. The purpose is to familiarize students with the software development process itself, methods and techniques that are used professionally to develop computer systems. We have introduced the most central topics in this lesson and enclosed documents that deal with the topics in greater depth for those who are particularly interested in increasing their general competence in software development.

Exercise. Exercises follow the flexibility of the modules. Students choose relevant tasks from the exercises based on the topics they have select to immerse in.

Project. Student assessment will be based on exercises and project delivery. When a student has completed a minimum set of exercises, they can start to work on the project. It is, however, possible to start a feasibility study for the project before mandatory exercises have been completed.

The project is about to create a teaching program that the student can use in his/her class to teach programming.

Project delivery requirements:

- Scope: It is expected that a minimum of 40 hours per person will be worked on with the project (pre-project / previous exercises not included). Students can work individually or in groups of a maximum of three persons.
- A simple project report per person shall be provided containing hourly consumption per activity and reflections on self-learning/results.
- Project description (preliminary project): A step by step description of how the student intends to implement the teaching program
- A complete teaching program that includes how the student plan to teach programming in the classroom. This implies:
  - Program code and screen dumps
  - Video of a maximum of five minutes showing a demo of running program or robot.

Discussion forums A forum where students can discuss with each other and trainers of the course is an important part. Slack[12] is used as a tool for this purpose.

3.5 Course exercises

In this flexible course, we have designed exercises that correspond to the flexibility of the topics. The aim is that every student can have benefits of the course regardless of which level the student teaches and what level of programming knowledge he/she has. To achieve

[11]https://codakid.com/why-coding-games-is-the-best-way-to-teach-kids-computer-programming/
[12]https://slack.com/about

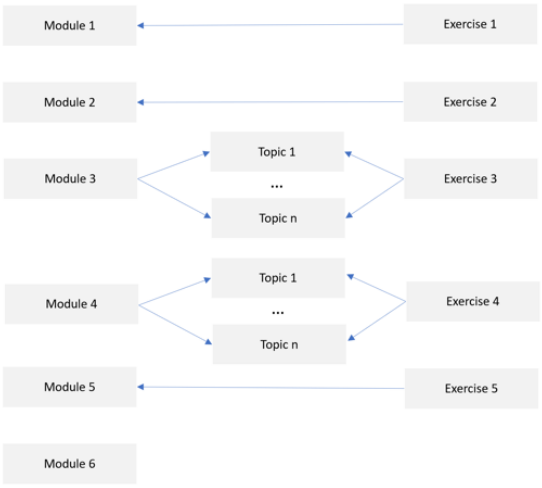


Figure 2: Flexible exercises

this, we have defined a minim set of tasks in each exercise. Each task can either be mandatory or optional. Figure 2 shows the design of exercises and their connection to topics in each module. Modules 1, 2, and 5 in the course are mandatory for all students. There is no flexibility in these modules, and therefore, tasks specified are not optional, and students must resolve the minimum set of tasks to be able to pass. However, in module 5 (Game programming), the student is supposed to create a game. The technology and complexity behind the games that students creates can be flexible. Module 3 and 4 are the main modules for application of programming and covers many areas. The specified exercises are also reflecting the same flexibility. In exercises 3 and 4, there are some mandatory tasks. Also, the students need to resolve a minimum set of optional tasks. They can select tasks depending on what topics in the module they have immersed in.

In Module 6, we address some key topics relevant to professional software development. The module is optional, and no exercise has been defined for it. Based on feedback (see figure 6 question 33), 52% of the students find this module useful. This is expected since the module is supposed to address topics only for those students who want to gain advanced knowledge to have a better understanding of the overall picture.

3.6 Course assessment

The main requirement for completing the course is that students conduct a project, as explained in the previous section. This project has two sub-deliveries. The first delivery is a feasibility study (sub-delivery one) where the students specify what programming concepts they will cover and how this can support/increase the understanding of the profession for their students. The first sub-delivery will be evaluated, and feedback is given to students before they are allowed to continue with sub-delivery two, which is the actual implementation of the teaching program they have specified in the first sub-delivery.

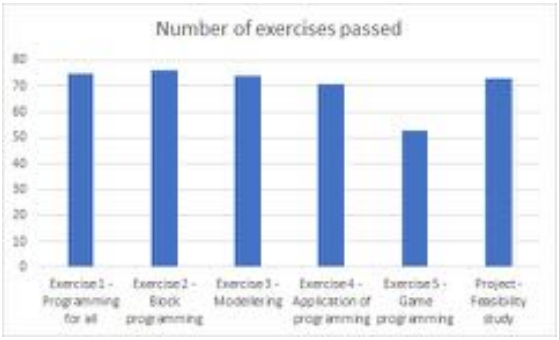


Figure 3: Overview of passed exercises

A prerequisite for starting the project is that students have to deliver a minimum of 70% of exercises in advance. Each exercise is organized in a way that reflects the diversity and flexibility of the course content. This means that students need to resolve a minimum set of questions in each exercise, but they can choose what optional questions to resolve.

Mandatory exercises force the students to study the learning materials which are relevant and interesting to students, which in turn make the basis for their project.

Figure 3 shows number of exercises passed. A minimum of 4 exercises in addition to "project feasibility study," had to be delivered. Exercises 3-4 were designed in a way that students could select a minimum set of tasks that were relevant with regards to complexity and application in their subject area.

73 students got a grade after delivering the mandatory project in the course. 80 students started the course in January. This gives a retention rate of approximately 91%. This result should be seen in the context of an online course (7,5 ECTS) where there is a clear commitment between teachers and schools to give teachers some free time to take the course. Therefore the result is excellent, but should be seen in this specific context.

4 RESULTS

4.1 Flexible Learning Trajectories

Figure 4 is showing schematic view of how flexible learning trajectories are implemented in this course. A learning path consist of common mandatory modules; flexible modules and a project delivery.

The different areas for applications of programming within electronics and natural science/mathematics in the course have been defined as follows: micro: bit, block-based (Scratch), Pocket Code, Raspberry PI, Arduino, LEGO Mindstorms, Game programming (Minecraft, PyGame), Modelling (natural science and mathematics). In table 2, we have an overview of what modules students have been using during their project.

Some examples of learning trajectories that students have followed in this course:

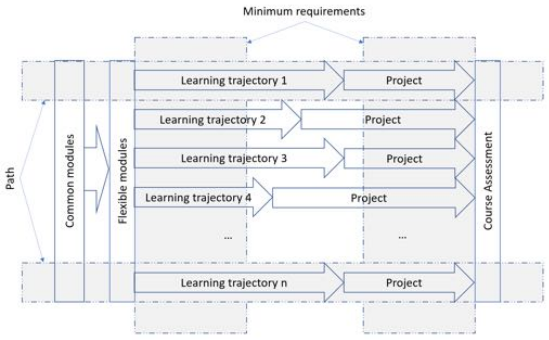


Figure 4: Schematic view of flexible learning trajectories

Example 1:

- Level
  - Upper secondary school
  - Mathematics
- Modules
  - Module 1: programming didactic
  - Module 2: Block based programming
  - Module 3: Application of programming within natural science
  - Module 4: micro:bit, robatics
- Topics
  - Modelling, Mathematics
  - 3D printing
  - micro:bit
- Project: Measurement of acceleration, speed and distance using programming

Example 2:

- Level
  - Upper secondary school
  - Technical and industrial production
- Modules
  - Module 1: programming didactic
  - Module 2: Block based programming
  - Module 3: Principles, constructions and structures in modern programming
  - Others (topics not covered in the course): C++, Programmable logic controller (PLC)
- Topics
  - Python
  - C++
  - Programmable logic controller (PLC)
- Project
  - Part 1: A basic course in "Use and programming of PLS for VG1 electrical subjects"
  - Part 2: Course in sequence control with PLC. Adapted to class level VG2 electric energy plus especially interested students in VG1 electrophysics.
  - Part 3: Basic course in text-based algorithmic programming using C ++ with emphasis on electrotechnical calculations. This course will probably fall outside the limits of the ordinary VG1 Electrical Subjects, but it should be able to be used within "Vocational Education Specialization »where the students work from the curriculum goals for later VG3 studies. (Automation and Computer electronics.)

Table 1: Application of programming in student projects

Application areas	Language								
	C++	Python	KRL	Python/C++	Block/Python	HTML/JavaScript	Block	Block/JavaScript/C++	Block/C++
App Lab							1		
Arduino	**10**								
Blue-Bot							1		
Games							1		
iPad							1		
iPad/Pythonista		1							
KUKA-robot			1						
Lego Mindstorm							**11**		
micro:bit							**21**		
micro:bit, bitbot							1		
micro:bit, Arduino									1
MineCraft							1		
PLS/Arduino								1	
PyGame		1							
Python		**9**			2				
Python/C++				1					
Pythonista		1							
Raspberry pi		2							
Scratch/Ras.Pi					1				
Spyder		2							
TinkerCad							1		
Website						1			
Totalsum	**10**	**16**	1	1	3	1	**39**	1	1

Example 3:

- Level: Elementary school
- Modules
  - Module 1: Programming didactic
  - Module 2: Block based programming
- Topics: Block based programming
- Project: Programming BlueBot to demonstrate programming concepts

Examples 1 and 2 are both on upper secondary level, but subject area and complexity they have chosen are different. Example 3 is showing a path which fulfills the minimum requirements, but is directed towards elementary school.

Table 2: Projects related to modules

	Project
Module 2, 3	2
Module 3	14
Module 3,4	16
Module 4	34
Module 4, Others	2
Module 5	3
Others	2
Totalsum	**73**

The most popular applications of programming in the project delivery has been the use of micro: bit, Arduino, and LEGO Mindstorms. Projects based on block programming are 53% (39 projects), Text-based projects are 40% (29 projects) and 7% (5 projects are mixed).

Relation between modules and student projects are also shown in table 2. Modules 3 and 4 deal with application of programming within different areas covered in the course and most projects are related to these modules. There are 2 projects in "Others" modules which are not covered in the course, but still accepted and will be considered to be added next time the course is executed.

There are both students with little or no programming knowledge and students who are advanced programmers. We have covered a wide range of topics that students can choose to immerse in. Still, some observation has been done where students have been asking for new areas that were not covered in the course (e.g., automation, programming of PLC (programmable logic controller) and 3d printing). More research needs to be done to identify all relevant subject areas in primary-, secondary- and upper secondary schools in regards to the inclusion as mentioned above of programming in specific curriculums. These areas need to be included in (or considered) the learning trajectories of the course. We have also observed that students with little or no programming knowledge and students with advanced programming knowledge have expressed that this flexible course has been useful to them since they can choose a learning trajectory that is relevant for them.

An advantage of having a class where a different level of knowledge, objectives, and interests for applications exist, is that we can use the class as a learning resource environment. We utilized Slack to enable students to interact with each other and observed a high level of participation amongst students of different skill levels. They share their knowledge, which in turn may increase the level of programming skills.

We are experiencing that there is a considerable demand for passing the programming courses, and we are expecting this demand to increase the next few years (the reason is explained in section 3.1). The main challenge of increasing the number of students in the assessment process regarding a parallel increase in workload for staff members. To accommodate the demand for more students, we need to reassess how compulsory assignments are implemented in the course, and how we can increase efficiency and general scalability.

The implementation of flexible learning trajectories in this course allow teachers to study topics which are applicable to their respective school subjects and how programming can be used to convey knowledge in a variety of topics. Reflection notes provided by the students as part of their project work show that their engagement has increased considerably during the project period, and they have discovered new areas where they can teach programming for their students. They also welcome the fact that the results of the project can be used immediately in their teaching classes, which means that they put more effort into the implementation of the project.

4.2 Course evaluation

At the end of the course, we sent a questionnaire to the students for completion. They were informed that anonymized data from the survey could be used in research related to the subject. A total of 22 students filled in the questionnaire while the approximately same number of students decided not to complete the questionnaire (stopped at first question). The result is shown in figure 5. A detail graph is shown in ??. In addition to these results, we also asked students to deliver a reflection note as part of their project deliveries. These reflections evaluate both this course and their project results, which are discussed in session 4.3.

Questions 39 and 40 are related to the project, and respectively, 71% and 67% are happy with their project delivery.

Questions 34 through 38 are related to course exercises. Most of the students (>65%) agree that exercises have been relevant and were not hard to resolve. However, we have got feedback from some students saying that exercise 3, which is related to the application of programming within natural science and mathematics (modeling), has been too hard to resolve. In this exercise, we assumed that all students have a minimum understanding of essential mathematical functions like sinus, cosine, etc.

Questions 28 through 33 are related to actual topics in each module and how relevant they were for students. Results are slightly weaker (52%-68%) but still within the range of acceptability. In module 1, we discussed programming didactics. We focused on questions like what is computational thinking, why should everyone learn to program, what are the challenges, etc. To gain a better understanding of these topics, we used the textbook Learner-Centered Design of Computing Education [2]. For this module, the satisfaction rate is 64%. Most students (68%) are also satisfied with module 2 (Block-based programming). This is also reflected in table 1 where most students have selected block based (micro: bit) as their project. The satisfaction rate for module 5 (Games programming) is lower (52%). The number of students that have delivered is lower for this exercise (see figure 3) compared to others. 24% of students have answered "Neither nor" to this question, which could mean that they have not done the exercise.

Questions 3 and 4 are related to using communication channels. 90% of students are satisfied using Slack. A total number of 1540 messages have been registered in Slack during the course. Figure 7 shows daily activities. The activity level of Slack confirms the high rate of satisfaction in the course. In addition to Slack, some students preferred to have direct communication with instructors/teaching assistants using e-mail instead of or in addition to Slack. Questions have been answered as quickly as possible to ensure that students get clarification in the shortest possible time. We would like to analyze and discuss use of communication tools used in this course

Figure 5: Course evaluation - Overview

in a separate paper. Therefore, we do not go into more details on use of Slack here.

Questions 5 and 6 are related to the use of Skype as a webinar tool. Only 50% of students were able to connect successfully to the webinars. The rest had either technical issues or was not able to join due to other reasons. At this point, we do not know how many of these students had technical issues, but based on comments during the webinars, we have been told that either they could not see the picture or they could not hear the sound. However, 64% are satisfied with the recorded version of webinars.

Questions 7 through 16 are related to the contents of the webinars. The first webinar had a focus on introducing the course and giving practical information. The last webinar focused on the project (all sub-deliveries). The other three webinars discussed different topics relevant to teachers. The satisfaction rate is lower compared to other areas of the course and the main reasons for this may be related to these conditions: Firstly, many students had technical issues connecting to these webinars (See comments on questions 5 and 6). Secondly, students have responded that they wanted a better connection between topics discussed in the modules and the topics in the webinars.

4.3 Reflection notes

In addition to results from the questionnaire discussed above, we have received a reflection note from each student who passed the course as part of their project delivery. These reflection notes are discussing two subjects: Firstly, how this course has increased their knowledge about applying programming in their subject-area and their more profound understanding of programming concepts. Secondly, students have created their own "teaching program" as the project delivery. Some of them have been able to run this program in the class for their students and have reflected on the results.

In this paper, we do not analyze all the results from these reflections, but can point to few reflections which confirms that the project has been a process for students to gain a better understanding of programming and how to apply it in their subject-area:

"What I first and foremost appreciated about this project is that it motivates me as a student and teacher as I spend time on a teaching program that is aimed at my practice...I spend time on this through the study program, but at the same time, I get a program that I can use myself in my practice. I believe that we have produced a teaching course that is very beneficial for the competency goals in the elective course programming in secondary school."

Several students are giving this type of feedback which confirms that it is motivating for students who are in-service teachers to work on a project where results which are rooted in the competence goals can be directly used in their class.

"Programming engages, it creates engagement and collaboration. In the teaching situation, we find that the students talk a lot together, and there is good work noise in the classroom. We see that this motivates the students very much. However, we also see that it requires a lot of knowledge and expertise from the educators, and sees that it is important to be well prepared. It is important to have clear learning goals and criteria for each session. The fact that the students have written a log along the way and have had to submit a description of their programming project has meant that the work the students have done has been more thorough..."

This is another type of reflection where students observe an engaged and motivated class. Another student reflects on how it went when they executed their teaching program development during the project in their class: "The main impression I am left with after the completed project period is that the task was comprehensive and demanding, but feasible, fun, and very educational. It differentiated well and gave all students, regardless of their skill level, the challenges that suited them. Some students were very self-sufficient and needed little help with the task. They showed high competence in terms of problem-solving and coding. Others needed a great deal of help with the coding itself. It was intense weeks for both students and teachers, but everyone came to the goal, and everyone felt that they were getting something they thought was impossible beforehand..."

Each student had delivered a project report where they are reflecting on how it went during project execution and when they executed their teaching program in their class. We have chosen to quote a few of these in this paper to show the type of results and what students think about the project in this course.

Some students with little or no programming knowledge experienced that the introductory course was hard, while it was easier for them to understand the programming concepts in the applied programming course.

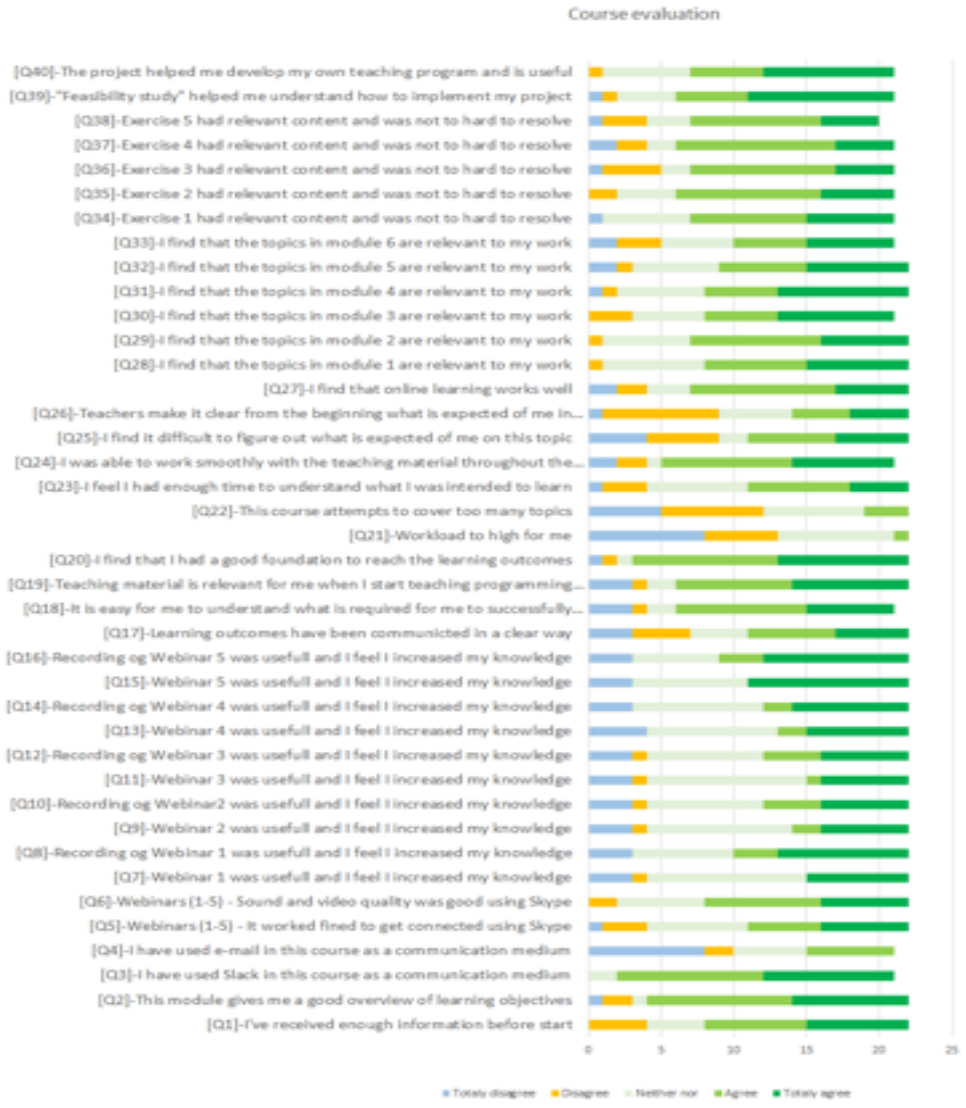


Figure 6: Course evaluation - Detail

5 LESSONS LEARNED

Multiple dimensions of flexibility in this context mean providing a fully flexible course for teachers in elementary and secondary schools in a way that helps them achieve the learning goals. E.g., selecting relevant content from the learning material and have the flexibility of choosing their topic in the project for creating their own 'teaching program'.

Teachers are usually postgraduate students with varied backgrounds and experience in computing education. One dimension is to consider this variety of backgrounds, and another dimension is that teachers are going to teach computing education to students on different levels. Teaching programming to all introduces other topics like programming didactic and gaining the understanding that computing education is important and necessary for the future.

By a variety of backgrounds as one dimension, we mean that students have different levels of programming knowledge. Some students are experienced programmers, while others do not have any programming knowledge at all. Also, teachers are teaching

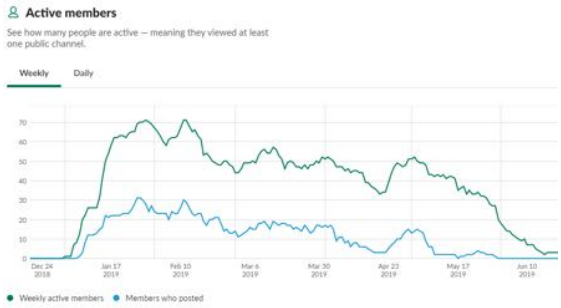


Figure 7: Slack

in a wide variety of subjects. Some examples are Mathematics, Art-, design and architecture, Media and communication, Music, dance and drama, Sports and physical education, Electricity, and electronics, Technical and industrial production, etc. [13]. This variety of background knowledge of programming and subject areas in the same classroom may lead to the design of a flexible learning trajectory course. Projects delivered by students (see table 1) reflects variety of programming applications and reflection notes in section 4.3 indicates that this flexibility is desirable. We believe that in a class-context with these varieties, a flexible learning trajectory course is a good choice.

The course was designed to provide flexible learning trajectories in some areas and mandatory in other areas of the curriculum. The first part of the course was mandatory for all students. The main goal was gaining the understanding that computing education is for everyone. This prevented students from spending much time discussing why they should learn programming and contributed to increased motivation and engagement for the rest of the course.

Module 1 and exercise 1 deal with this. Delivery of exercise 1 in figure 3 shows high student participation. the students have engaged a lot in these questions and, based on the feedback, this part of the course has created increased motivation for further learning.

Individual trajectories were planned for other topics of the course since the level of knowledge on computing education, their interest and local practice (this is an element that is different from courses for regular students) are very different. It is challenging to create enough learning trajectories to cover both students interests and what is required to know and teach in their respective subjects. However, having in mind that the aim is to increase students' interest and engagement in this field makes it easier to design the course in a way that most teachers find it useful.

Teachers need to have a clear understanding of the learning outcomes for their students. They need to have a clear understanding of the requirements of the new curricula provided by Norwegian Directorate for Education and Training in primary and secondary education. Based on this, and support from the course instructor, they can select a learning trajectory that is most appropriate for their subject.

[13] https://www.vilbli.no/en/en/no

Having a course where students can select a learning trajectory among many, makes it challenging for course instructors and teaching assistants (TA). Many different areas for the application of programming need to be supported in the course. To manage this, we have given each TA a separate area to focus on. Each TA is responsible for one or more areas of applied programming. Another challenge has been cooperation among students. Since they have different areas of interest, it can be challenging for them to get the support they need. We have introduced the use of Slack as a tool for discussions in addition to instructors and TA's actively supporting the students. Providing a set of links to relevant online Internet resources helps students find the answers to some of their questions.

5.1 Course webinars

Webinars have been used in this course as complementary lectures and a way of direct communication between students and instructors.

These types of lectures are appreciated by students, but need to be planned and executed carefully. Finding a time that suits all participants can be challenging (see section 4.2). In addition, students are spread around the country and use different types of equipment which can create technical issues with connection. When a proper solution to these issues are available, webinars can be useful in such a model.

Based on the results of the course evaluation, a tighter connection between Webinars and topics of the course is desirable. This will help students get a better understanding of each module and they will be in a better position to select a proper learning trajectory in the course.

5.2 Teaching assistants

Several TAs have been used to manage the correction of exercises in addition to answering questions and making clarifications. A course supporting flexible learning trajectories requires TAs to have a wide range of competencies and skills. Finding these types of resources could be challenging, especially if the number of students in the course increases, and the need for employment of TAs increases proportionally. To resolve this, we shared the responsibility of different topics to different TAs. E.g., some had a responsibility to answer questions/make a clarification to Arduino related discussions and exercises while others had responsibility for Raspberry Pi, etc.

We find the use of TAs with different backgrounds in programming to be appropriate in such a model.

5.3 Implications for course re-design

As the approach is learner-centered, we need to tackle the problem as a design-based-research project and use feedback from the first cohort to redesign the course where needed.

A flexible course design model as explained in this paper does not scale up well when number of participants increase. Huge number of students will result in proportionally additional work for instructors to do assessment.

Another implication in such a flexible course design is for students to find the correct learning trajectory. In-service teachers with little or no programming knowledge may need assistance to

find the right path and this challenge will increase as number of students increases in the course.

Some topics for application of programming seem to be more relevant then others (see table 1). Re-design of the course must focus on these areas which will result in reducing the flexibility of the course.

6 CONCLUSIONS AND FUTURE WORK

Designing a course that supports flexible learning trajectories introduces many challenges that need to be addressed. At the same time, it opens opportunities to increase student learning outcomes in a course where in-service teachers have varied level of programming knowledge, interests, and different application need. A learner-centered design process for computing education[2] is the basis for the design of a flexible programming course for in-service teachers. In the case of this course, this design helped creators to reach their initial idea and has provided a good structure for flexible learning trajectories. Some areas of future research on this topic are:

(1) A full evaluation of communications channels used in the course (Slack email)
(2) How to help students to reflect on their learning needs and choose the right learning trajectories
(3) How to help students to reflect on their teaching practice
(4) How to promote cooperation among students towards the establishment of a Community of Practice (CoP)

In this paper, we have explained how the course has been designed and discussed our initial reflections on results, challenges and possibilities.

REFERENCES

[1] Bodong Chen, Yu-Hui Chang, Fan Ouyang, and Wanying Zhou. 2018. Fostering student engagement in online discussion through social learning analytics. *The Internet and Higher Education* 37 (apr 2018), 21–30. https://doi.org/10.1016/j.iheduc.2017.12.002

[2] Mark Guzdial. December, 2015. *Learner-centered design of computing education : research on computing for everyone;*. Morgan & Claypool Publishers. 147 pages.

[3] S Hadjerrouit. 2005. Learner-Centered Web-Based Instruction in Software Engineering. *IEEE Transactions on Education* 48, 1 (feb 2005), 99–104. https://doi.org/10.1109/TE.2004.832871

[4] Andrew Ho, Justin Reich, Sergiy Nesterko, Daniel Seaton, Tommy Mullaney, Jim Waldo, and Isaac Chuang. 2014. HarvardX and MITx: The First Year of Open Online Courses, Fall 2012-Summer 2013. 1 (2014). https://doi.org/10.2139/ssrn.2381263

[5] Anna Lamprou, Alexander Repenning, and Nora A Escherle. 2017. The Solothurn Project - Bringing Computer Science Education to Primary Schools in Switzerland. *In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* November (2017), 218–223. https://doi.org/10.1145/3059009.3059017

[6] Fotis Lazarinis, Christoforos V. Karachristos, Elias C. Stavropoulos, and Vassilios S. Verykios. 2018. A blended learning course for playfully teaching programming concepts to school teachers. *Education and Information Technologies* (2018), 1237–1249. https://doi.org/10.1007/s10639-018-9823-2

[7] Sten Ludvigsen, Pia Elverhøi, Bushra Ishaq, Jens Rasmussen, Daniel Sundberg, Eli Gundersen, Kjersti Kleven, Mari Rege, Helge Øye, Sigve Indregard, Tormod Korpås, and Sunniva Rose. 2014. *Elevenes læring i fremtidens skole.* Technical Report. Oslo.

[8] Bonnie A. Nardi. 1993. *A Small Matter of Programming.* The MIT Press.

[9] Government Norwegian. accessed April 17, 2019. *Government message St. 28 (2015–2016).*

[10] Utdanningsdirektoratet. 2019. *What is the knowledge promise.*

[11] Utdanningsdirektoratet. 2019. *Why Don't More People Work As Programmers.*

[12] Jeannette Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35. https://doi.org/10.1145/1118178.1118215 arXiv:=

[13] Di Xu and Shanna Smith Jaggars. 2013. Adaptability to Online Learning: Differences Across Types of Students and Academic Subject Areas. *Economics of Education Review* 37, 54 (2013), 46–57. https://doi.org/10.7916/D82N59NB

[14] Di Xu and Shanna S. Jaggars. 2014. Performance Gaps Between Online and Face-to-Face Courses: Differences Across Types of Students and Academic Subject Areas. *The Journal of Higher Education* 85, 5 (2014), 633–659. https://doi.org/10.1353/jhe.2014.0028

Peer Assessment by Ranks

Insert Subtitle Here

FirstName Surname†
Department Name
Institution/University Name
City State Country
email@email.com

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

ABSTRACT

Peer assessment is a teaching technique in which students assess each other's work. It has many potential advantages. It helps students to learn and engage with the quality criteria of their subject, and to see their own work as others see it. There are problems with peer assessment as well, however, including numerous anxieties that students may have. They may have concerns about fairness, about any extra work involved, about their abilities to assess fellow students, and to be assessed by them.

We assigned a piece of coursework to twenty-one students in which their task was to rank some design concepts from a previous class. They put the designs in order of value so that they only had to judge the designs in comparison to each other, and not to some imagined universal standard that they are not familiar with. The assignment allowed students to give their answers both formally, as a ranked order, and textually, in open answers where they could explain and justify their choices. This mix permits a semi-automatic marking scheme to be applied, and we tested two such schemes. One is a standard used quite commonly in multi-choice tests, because it is simple; and the second is a refinement on the idea which was intended to give more accurate results for ranked questions. Both marking schemes are tested, and the second one is found to give significantly better results.

This style of peer assessment is thus demonstrated to be viable. It gets over some of the problems with peer assessment, and can give students a new learning experience with its own set of advantages. It can be used in cases where exact answers are not available, such as in matters of design, or other areas of expertise which cannot be reduced to a formula, but which students nevertheless need to learn in order to become professionals. There are such requirements in computing, but also in many other subjects, so the technique should be widely applicable.

CCS CONCEPTS

• Insert CCS text here • Insert CCS text here • Insert CCS text here

KEYWORDS

Peer assessment, peer feedback, automated marking.

ACM Reference format:

FirstName Surname, FirstName Surname and FirstName Surname. 2018. Insert Your Title Here: Insert Subtitle Here. In *Proceedings of ACM Woodstock conference (WOODSTOCK'18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1. Introduction and Background

Peer assessment is seen as valuable, but needing careful treatment, in areas such as teaching argumentation [4, 13], and academic writing [6, 7]. Other authors assert that more research is needed to ask just how students benefit from PA (peer assessment) [10, 20]. It is also necessary for teachers to understand their options in delivering PA [17]

Nicol et al extol the many virtues of peer assessment [12], and conclude that it should receive much greater attention in higher education.

1.1. Peer Assessment Offers Benefits to Students

Students stand to benefit from feedback in many ways, but they do not always seek it, even when it is on offer [18, 19]. Peer assessment may help, because it is an alternative way to get more feedback to students.

In a large study of 180 participants, the students recognised that assessing peers' work helped them to improve their learning. However, Ion *et al* also found that, when the feedback was returned to students, they sometimes needed mentoring and guidance to help digest the comments [8].

Judgement (as in assessment) is an important faculty in any expert profession, but we generally let students develop this skill only by having their own work judged and assessed by an expert. Professionals cannot expect always to have a better expert on hand, and they must learn to accept the judgement of their peers; and eventually clients.

We may summarise some of the benefits of peer assessment here; but there may yet be many more. Students engage with the marking scheme, internalising the criteria of value in their domain; they produce work in awareness that other students will see it, and will see other student work and learn to evaluate themselves against it,

and develop confidence in assessing as well as producing. Furthermore, judgement is crucial in domains where criteria cannot be reduced to numerical formulae. Computing is an exact domain of science and engineering, but it is not only that: it has its own needs for expert judgement at times, such as in design, and many forms of applied computing. Despite appearances then, there is a need for PA in computing just as there is in many other fields.

1.2. Problems with Peer Assessment

While feedback is perennially recognised as crucial to learning in higher education, there is also persistent dissatisfaction with it. Liu and Carless investigated why there is so much resistance to using PA, in a large questionnaire based study of over two thousand students and tutors. They advocate normalising PA in the standard course procedures, so that students have time to practice their skills special to PA [10]. In an extremely large study in Australia, Henderson and colleagues found that there were many issues, including concerns of students and tutors that the time available to generate feedback limits the quality of feedback that can be given [3, 5]. Students often feel they need more detail, in order to be sure they understand what the feedback says and can get back value from it that rewards the work they put into it when creating it.

One issue with peer feedback is that students may have exaggerated expectations if it is led and delivered by the tutors. However, if courses are designed to offer opportunities for students to provide feedback to each other, then a more sustainable model for assessment results [2]. Topping found that students "with less skill at assessment but more time in which to do it can produce an assessment of equal reliability and validity to that of a teacher" [15]. If this is explained to students it may go some way to allaying their concerns about the merits of PA.

The inexperience of students can clearly be a problem in PA. It has been found by Knight *et al*, in a large study of two thousand students [9], that it is helpful for them to start with a calibration task, in which they first learn about how to assess work, and what the criteria should be.

We may summarise the concerns that students may have with PA, by a series of questions relating to the quality of assessment. Students are uncomfortable assessing each other. Do they have the right, or expertise? Is it their role, and should they have power over their equals? Will they judge fairly? Will all other students judge equally well? Will students in their turn be assessed fairly by their peers? Does peer assessment merely miss an opportunity for genuinely valuable feedback from the teachers? Could the feedback from other students, without quality control, be misleading, or even harmful? Different students have different levels or patches of knowledge, as well as different personalities, and moods or emotions on the day; they might also have complicated social relationships with others in the class, and a recent history that may add overtones which aren't present with the teacher. Peer assessment may be done anonymously, but can anonymity be guaranteed, and if not is there a chance that students might bear a grudge toward their fellows who have assessed their work?

In addition, students and tutors have concerns about workload. Each individual student cannot be expected to peer assess all the other students in the class. If the work is divided up between students, on the other hand, then how can rising issues of unfairness be handled and moderated? Perhaps automation can help with some of the tasks inherent in managing a PA process.

1.3. Tools for Peer Assessment

There have been many attempts to build software solutions to support the peer marking process. One useful review is by Luxton-Reilly [11]. More recently there are quite innovative proposals, such as the use of social robots to help automate the experience of personal feedback [1].

The main route for automated support for PA is online, however, in website based VLEs (Virtual Learning Environments). A typical example of how higher education recommends these facilities to their academic staff is that of the University of Bristol, which uses the blackboard VLE [16]. For larger courses that are run exclusively online, as MOOCs, the demands for PA are clearly more pressing, and there are therefore some interesting attempts to solve some of the problems of PA in that sphere. In one study, which was the largest of its kind up to that date, the online education company Coursera developed algorithms to help compensate for the differing reliabilities, and any biases of online peer assessors [14].

1.4. Propose Peer Assessment by means of Ranking Exemplars

In the next section, a method is proposed to manage a PA process by setting a task to rank a small number of other students' pieces of work. This method may bring several of the benefits alluded to above, and yet avoid some of the problems.

The method was tried in a class of twenty-one students, and it is evaluated in the following sections.

2. A Ranking Method of Peer Assessment

In a game design class, students were asked to evaluate four brief design concepts that were created by students in a previous year. These four concepts (collectively a *quartet)* would give the students an insight into how their own designs might appear to others, including tutors or any future professional colleagues or clients. In this exercise of peer assessment, the students assess other student work, but did not have their own work assessed by peers in turn. This allowed them to feel secure from some of the concerns about PA they might have, but also to benefit from the experience of evaluating work from other students on the same course. The whole exercise was to take anything between a half hour and an hour, as the students chose.

The quartet of four designs were selected to give a sample of different styles of design, of varying lengths but not too long, with a range of different strengths and weaknesses. Some were more clearly better than others; but some were quite close in quality. They were authentic designs, without editing or corrections of typing errors, but they were anonymized, so that their creators could not be identified.

Students were assigned the task to read all the designs in their own time at home, online in the form of a Google questionnaire. Their answers were to two questions: a ranking question, then an open text question. The first question asked the student to rank the quartet, to put the four designs in order from strongest to weakest, according to the evaluation criteria. Students had previously been familiarized with these criteria in a practice session with four different design concepts. In this task, the design concepts may have been difficult to separate and put in order, in some cases. Students may also have only partial appraisal of the evaluation criteria, which they are still learning, after all. Therefore, the second open text question offered them the chance to explain their reasoning for the ranking they chose. This also helps the students to trust in the process, as it allowed the tutors to check their answers and give them compensation in case their ranking was not the same as the tutor's, but their reasons were nevertheless cogent and relevant. Tutors then marked the rankings according to a formal scheme, and gave marks based on that and on any comments that were given as answer to the second question.

Afterwards some general feedback was given to the class about how they all did in the task, with summary statistics. The main point of the exercise was, however, to give the students an authentic experience of assessing peer work.

There were two quartets given, as the students were already grouped into two halves for other reasons. They were groups A and B, with twelve and nine students in each one respectively. It was thus convenient to assign them two different quartets of designs, as an additional check that the assignment was consistent, and working well; and provide an opportunity to intervene and make corrections otherwise.

2.1. The simple marking scheme

The marking scheme that was used to score the rankings was the one that is typically used for such questions in VLEs, such as Blackboard for instance. In this scheme the answer gets one point for each element in the ranking that is in the correct place, according to the ranking defined by the tutor as the correct answer.

Although this is the common way to mark such questions, it is somewhat crude. It has no regard to the nature of a ranking, which places items in an order. In ordinal sequences, wrong answers may thus be wrong by a narrow or a wide margin, or distance. To exchange the top two items in the order would be to make a small mistake; but to swap the top and bottom items would be a bigger one. Both mistakes would receive the same score according to the simple scheme.

2.2. The refined (distance) marking scheme

In order to reflect the margin of error, a slightly more complex scheme was also developed. It was not used at the time of the class, but developed later, in order to improve the course fur future years. Its performance in automating scoring is compared with the simple scheme in the following sections.

The essential idea of this marking algorithm is to count errors in a student's ranking, compared to the definitive answer. Each item scores an error score of how far out of place it is. These scores add up to zero if the answer is perfectly correct. The scores can increase in multiples of two, up to a maximum of eight (assuming there are four items to rank). We obtain a final score from this total error e, by the formula $s = (8-e) / 2$. This gives an integer from 0-4, just as the simple score does. However there are several cases in which the distance method scores higher than the simple method. Table 1 shows examples.

Table 1: Examples scored by simple and by distance methods. The best ranking is in the top line, B C D A, and students below get these scores. The third student gets a higher score under the distance method.

B	C	D	A	simple	distance
A	D	B	C	0	0
B	C	D	A	4	4
C	D	B	A	0	2

With these two methods to score the formal ranking answers, it is possible to evaluate their performance on the real data that came from the coursework assignment. It is shown in the next section that the refined method is significantly better.

3. Results and Analysis

The students were in two groups, A and B, and each group was given a quartet of design concepts to rank. They were scored according to how their rankings compared to the tutor's rankings of the quartets. The students' comments on their rankings were taken into account, in arriving at a final score for the exercise, especially when a student seemed to rank the quartet very differently from the tutor. Although the rank order may be different, the student may have given some good reasons for their choices, and can gain credit by their rationale. In this way, normal academic judgement can compensate for any failings in the formal marching scheme. In the following, however, we report only the formal results, in order to see how well the two marking methods fare.

The scores for rankings are compared against the other marks that the students got, for another assignment they were set. A natural assumption is that the marks should be correlated, because stronger students will tend to do better across all assignments. The correlation will never be perfect, of course, as different assignments require and assess different skills and knowledge; but in general, a higher correlation signifies a more reliable assessment scheme.

Figure 1 shows that Group A (of twelve students) attained simple scores for their rankings that correlate reasonably well (0.38) with their other marks. Any weakness in the scoring method is most likely to be evident in the divergent scores of students achieving high marks in one measure, but low marks in the other one. These cases would appear in the charts in either the top-left or bottom-right quadrants. Figure 1 has three students top-left, who appear to perform well in their other marks, but only scored 0 (zero) in the ranking assignment. It is a concern if an assignment often gives low scores to good students.

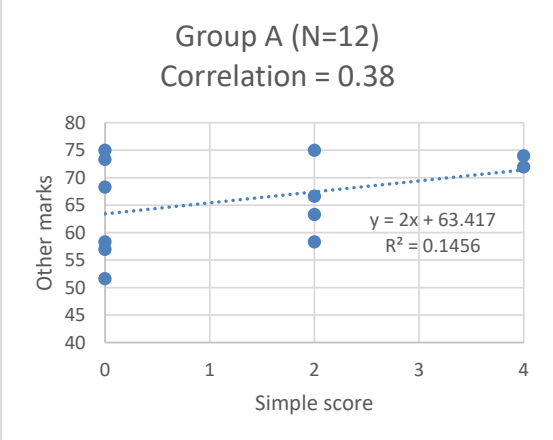


Figure 1: Some consistency is shown between the marks that students got for their other work, and the simple score for their ranks. It is not possible to get a simple score of 1. The students top-left have high marks, but only get 0 for their rankings.

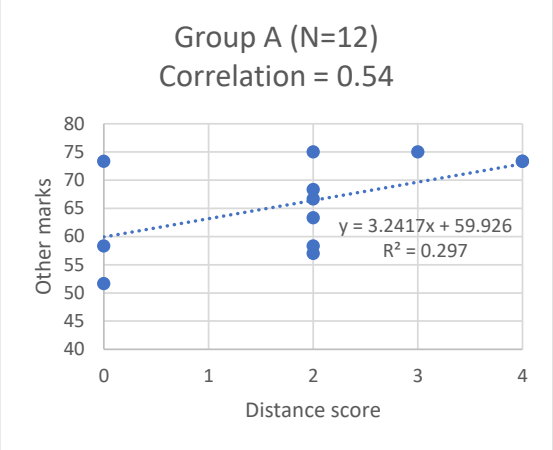


Figure 2: Using the more refined method to score rankings, based on distance errors, the correlation is significantly stronger. Some students who got 0 in Fig. 1 now get scores of 2.

When the students' rankings are scored by the distance method, however, the correlation rises to a more respectable 0.54 as seen in Figure 2. This is principally because some students, who got 0 by the simple method, get scores of 2 by the more refined distance scoring scheme.

These are reassuring results, but to confirm them, similar results are shown for the other group, B, in Figures 3 and 4. This group had the same assignment, but for a different quartet of design concepts. It was again found that the correlations in the marks were higher for the distance method (0.44) than for the simpler method of scoring (0.11). The correlations are both lower for this group than they were for Group A, but this may be because Group B has fewer students in it (nine instead of twelve).

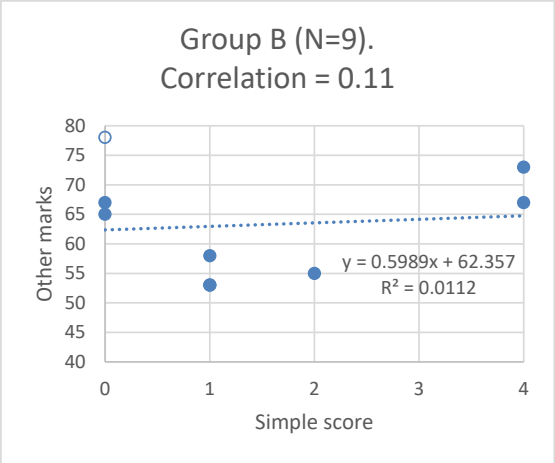


Figure 3: The other group showed less consistency. There was an outlier score of zero, for a student who otherwise got top marks (of 78).

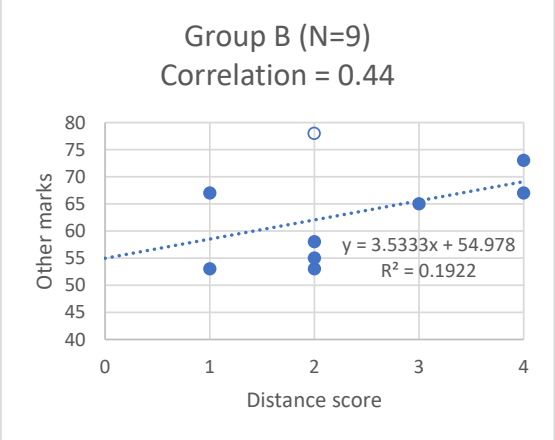


Figure 4: The outlier student from Fig. 3 now scores 2, and others have an improved score too, using this method. As with the other group in Fig. 2, there is a stronger correlation now.

In Group B, the top-scoring student in the class, with a mark in her other assignments of 78, scored only 0 (zero) by the simple marking scheme for her rankings. By the more refined scheme based on distance, though, her score is 2 instead. This is because her ranking was ACBD, when the tutor's ranking was CADB. She therefore correctly put the two better designs first, but in the wrong order (CA). Likewise, she put the weaker designs at the bottom. By the simple marking scheme she therefore scores nothing; but taking into account that her ranking put each design only a minimal distance out of place, she scores 2 out of 4.

In addition, the comments that this student gave are a good example to show how the scores by a formal marking scheme may be softened by consideration of the student's rationale. Her comment was :-

> "Concept A and C were closely tied for first place they both took novel perspectives to bullying that i wouldn't have though of, a parent and a therapist. In the end it was concept A that won out over concept C because of the elegance of the idea it was simpler that concept C and had a similar amount of value. It showed the player how to spot bullying and also gave bullies a different outside perspective to see what they were doing.
> …
> Concept B and D were hard to tell apart from last place. Concept B was even compared to another game about bullying, showing the lack of novelty. However it beat idea D because of how much value it had as a serious game…"

It is clear from this that she understood how close the better two designs are (A and C), and that the other two are also close to each other. Her reasons for distinguishing them are also good ones, and could serve as helpful discussion points for the class, in the form of feedback later. Because of this student's clear understanding of the designs, and how to evaluate them in this forced-choice exercise, she in fact was given a final score of 2 for this effort by the tutor, and not the zero score that the simple marking scheme gave her. The refined distance-based marking scheme was not used in marking the class, because it was only developed afterwards in analysis of the results, that was done to evaluate the assignment to check how well it works for future reference. However, the distance method would award 2 marks to this student, which is exactly what the tutor judged was a fair mark for her at the time.

4. Discussion and Further Work

This teaching intervention was intended to allow students a measure of peer assessment, in a way that would be fair to them both as markers and if their own work were to be marked by other students. In this case students were only marking the work of previous students, and so their own work was not being marked by their classmates. However, the analysis shows that this type of assignment would be an appropriate way to actively mark student work by their peers in the same class, if it were administered anonymously. The students have shown that they could mark each other's work fairly in this way.

Two scoring methods were tested in this study. The first, simple scoring method is one that is used in commercial VLEs, because it is easy to understand and to calculate automatically. It gave results that were acceptable for one group, Group A, but not really adequate for Group B. That might be a consequence of the group's smaller size, but it serves as a warning that the simple scoring method may be fragile.

The second scoring method, based on a concept of error size represented by natural distance across the rankings, clearly gave significantly better results. This was demonstrated by comparing the students' marks from other assignments, to validate that the ranking scores were more consistent with other student work that had been independently assessed. It therefore seems that the proposed type of assignment, based on ranking peer work, together with the more refined marking scheme, is reliable and fair to mark by peer assessment.

This type of assignment may owe its success to the way that it bases student assessments on ranking exemplars of peer work, rather than marking them on some absolute scale. The idea was that students would find it easier to rank pieces of work according to the chosen subject criteria, than to assess each piece of work as 'good' or 'bad'. They don't have to condemn any other student's work as 'bad' and don't have to feel that pressure, even anonymously. Their assessments are only based on each work's strengths and weaknesses, which people readily understand that all student work will have anyway: this is in the nature of education, after all.

Another advantage of the assignment is that it is sometimes difficult to rank works in order, because each may be stronger in some respects, but weaker in others. This will generally be the case with exemplars chosen from students who took the same classes, because they will naturally be quite close to each other in their skills and knowledge, having had the same learning experiences. The difficulty of this task thus forces students to engage critically with the exemplars, to reconsider the meaning of the evaluation criteria, and see how their own work might look to a neutral person who is judging by the same criteria. The assignment brings out many of the advantages of peer assessment, therefore.

If the assignment were to be used for the other limb of peer assessment, then further advantages would be offered. Thus, if students were to be assessed by their peers in the *same* class, albeit anonymously, they would probably be more attentive to their own work as they produce it, knowing that their peers and friends may read and judge it. This is a useful portion of motivation that tutors may use to help drive the students to perform at their best. Furthermore, it would then become feasible for students to get independent, anonymized feedback on their work from other students, and not only from the tutor. It is a clear possibility that this would induce a stronger learning effect, as students may care a good deal for hearing the opinions of their peers, in a safely private manner. Peers are directly comparable, by definition, and students would be naturally interested in an honest expression of their fellow perspectives, engaging as it does the powerful human psychology surrounding issues of social dominance and hierarchy.

More practically, it remains to be seen how easily and quickly such feedback from fellow students in the class can be organized, anonymized, and routed back to the students who created the designs. It would be useful to explore technological tools that could help, or even automate parts of the process.

Future work might also confirm that an assignment based on ranking peer work is truly fairer than absolute marking. Whether it is easier to do, or perhaps more difficult, is also of interest. If it proves to be experienced as more difficult, however, that would not in itself be a reason not to set the assignment; for it may be more difficult because it is more valuable. The marking scheme itself might be improved further. For instance, it is based on a notion of distance as number of ranking steps out of place; but a more exact notion of distance might be feasible. When two neighboring exemplars are judged as nearly equal, then the distance between them might be said to be only a fraction of a step.

Another matter for further research regards the opinions of students to the assignment. Do they feel that it is a valuable learning experience, as suggested above; and if so, in how many ways? Do they feel more confident of being able to assess work by ranking in order, rather than determining its value in some absolute sense? Do they gain any insights into the nature of their subject?

The effect on views of their own work would also be interesting to explore. Does a critical engagement with other students' work leave a lasting trace on how they see their own work? Is there any change in how they see themselves developing towards more professional standards of quality? Has there been any change in their (supported and validated) levels of self-confidence? Do they also feel more confident in being able to judge the work of colleagues, and to express their judgements considerately?

5. Conclusions

Peer assessment is often seen as potentially powerful, but it also has numerous difficulties, including the burden is places on the students who are asked to assess their peers. As well as workload, there are psychological pressures. Cognitive difficulties regard the challenge to evaluate work that the students do not yet feel confident of themselves. Emotional pressures regard the demand to pass judgement on fellow students.

The assignment task here was designed to negotiate, temper or avoid some of these difficulties. It asks students only to rank a small number of peers' works. This simplifies and focuses their cognition on the key elements of the subject domain. It also liberates them from some of the anxieties regarding peer judgement.

Results show that the assignment was able to score the students assessments reliably and fairly; but only when the more refined method of scoring was used. This method is not standard, and commercial VLEs typically use only the simpler method, which was shown here to be less reliable. Therefore, it is important to use the more accurate scoring method presented here. Even better methods may be possible, as suggested earlier, but that must be demonstrated with further research.

It would be a natural next step to take the method presented here, and apply it to return peer assessments of their creations to originating students. This should ideally be done soon after they have created them, if possible. An interesting further stage might even be to allow students to rebut the criticisms, and return those rebuttals to the peers who assessed them. That would take the notion of peer assessment into new territory, which shows just how rich the technique is, and how full of potential.

The work in this study represents a start, and only one forward step in this direction. There is much more that could be done, to develop this teaching technique for computing subjects; but also for all other subjects too, in principle.

REFERENCES

[1] Belpaeme, T. et al. 2018. Social robots for education: A review. *Science Robotics*. 3, 21 (Aug. 2018), eaat5954. DOI:https://doi.org/10.1126/scirobotics.aat5954.

[2] Boud, D. and Molloy, E. 2013. Rethinking models of feedback for learning: the challenge of design. *Assessment & Evaluation in Higher Education*. 38, 6 (Sep. 2013), 698–712. DOI:https://doi.org/10.1080/02602938.2012.691462.

[3] Dawson, P. et al. 2019. What makes for effective feedback: staff and student perspectives. *Assessment & Evaluation in Higher Education*. 44, 1 (Jan. 2019), 25–36. DOI:https://doi.org/10.1080/02602938.2018.1467877.

[4] Haro, A.V. et al. 2018. The effects of an online learning environment with worked examples and peer feedback on students' argumentative essay writing and domain-specific knowledge acquisition in the field of biotechnology. *Journal of Biological Education*. 0, 0 (Jun. 2018), 1–9. DOI:https://doi.org/10.1080/00219266.2018.1472132.

[5] Henderson, M. et al. 2019. The challenges of feedback in higher education. *Assessment & Evaluation in Higher Education*. 0, 0 (Apr. 2019), 1–16. DOI:https://doi.org/10.1080/02602938.2019.1599815.

[6] Huisman, B. et al. 2018. Peer feedback on academic writing: undergraduate students' peer feedback role, peer feedback perceptions and essay performance. *Assessment & Evaluation in Higher Education*. 43, 6 (Aug. 2018), 955–968. DOI:https://doi.org/10.1080/02602938.2018.1424318.

[7] Huisman, B. et al. 2019. The impact of formative peer feedback on higher education students' academic writing: a Meta-Analysis. *Assessment & Evaluation in Higher Education*. 44, 6 (Aug. 2019), 863–880. DOI:https://doi.org/10.1080/02602938.2018.1545896.

[8] Ion, G. et al. 2019. Giving or receiving feedback: which is more beneficial to students' learning? *Assessment & Evaluation in Higher Education*. 44, 1 (Jan. 2019), 124–138. DOI:https://doi.org/10.1080/02602938.2018.1484881.

[9] Knight, S. et al. 2019. Calibrating assessment literacy through benchmarking tasks. *Assessment & Evaluation in Higher Education*. 0, 0 (Feb. 2019), 1–12. DOI:https://doi.org/10.1080/02602938.2019.1570483.

[10] Liu, N.-F. and Carless, D. 2006. Peer feedback: the learning element of peer assessment. *Teaching in Higher Education*. 11, 3 (Jul. 2006), 279–290. DOI:https://doi.org/10.1080/13562510600680582.

[11] Luxton-Reilly, A. 2009. A systematic review of tools that support peer assessment. *Computer Science Education*. 19, 4 (Dec. 2009), 209–232. DOI:https://doi.org/10.1080/08993400903384844.

[12] Nicol, D. et al. 2014. Rethinking feedback practices in higher education: a peer review perspective. *Assessment & Evaluation in Higher Education*. 39, 1 (Jan. 2014), 102–122. DOI:https://doi.org/10.1080/02602938.2013.795518.

[13] Noroozi, 787 and Hatami, J. 2018. The effects of online peer feedback and epistemic beliefs on students' argumentation-based learning. *Innovations in Education and Teaching International*. 0, 0 (Jan. 2018), 1–10. DOI:https://doi.org/10.1080/14703297.2018.1431143.

[14] Piech, C. et al. 2013. Tuned Models of Peer Assessment in MOOCs. *arXiv:1307.2579 [cs, stat]*. (Jul. 2013).

[15] Topping, K.J. 2009. Peer Assessment. *Theory Into Practice*. 48, 1 (Jan. 2009), 20–27. DOI:https://doi.org/10.1080/00405840802577569.

[16] University of Bristol, education support unit Self and peer assessment in Blackboard.

[17] Wanner, T. and Palmer, E. 2018. Formative self-and peer assessment for improved student learning: the crucial factors of design, teacher participation and feedback. *Assessment & Evaluation in Higher Education*. 43, 7 (Oct. 2018), 1032–1047. DOI:https://doi.org/10.1080/02602938.2018.1427698.

[18] Winstone, N.E. et al. 2017. 'It'd be useful, but I wouldn't use it': barriers to university students' feedback seeking and recipience. *Studies in Higher Education*. 42, 11 (Nov. 2017), 2026–2041. DOI:https://doi.org/10.1080/03075079.2015.1130032.

[19] Winstone, N.E. et al. 2017. Supporting Learners' Agentic Engagement With Feedback: A Systematic Review and a Taxonomy of Recipience Processes. *Educational Psychologist*. 52, 1 (Jan. 2017), 17–37. DOI:https://doi.org/10.1080/00461520.2016.1207538.

[20] Zhu, Q. and Carless, D. 2018. Dialogue within peer feedback processes: clarification and negotiation of meaning. *Higher Education Research & Development*. 37, 4 (Jun. 2018), 883–897. DOI:https://doi.org/10.1080/07294360.2018.1446417.

Programming, Research and... Coffee? An Analysis of Workplace Activities by Computing Interns

Author 1 and Author 2
University
E-mail: {author1,author2}@university.edu

ABSTRACT
To overcome the skills gap between industry demands and learning outcomes achieved by graduates in higher computing education, many Bachelor programs integrate some form of internship in their curriculum; students are assumed to encounter authentic tasks and recent technologies in the workplace. In practice however, educators often do not know specifically which tasks their students perform and which technologies they use, mainly due to the distance between coach and student during the work placement and the lack of cohort-based overviews of activities performed in internships. In this study, we gathered and analyzed workplace activity data of 54 students over the course of their third-year semester-long internships in the computing industry. We performed descriptive analyses to gain insight into i) which categories of activities students performed most (programming, research and documentation) and ii) which of the activity categories they find most difficult (we were unable to tell). Subsequent text analysis gives us insight into students' perceptions of the categories used to label activities (testing, research, meetings, and academic documentation are congruous) and which technologies were used most by these students. Based on the results, we conclude it is feasible to use user-generated data to get insights into workplace activities of computing interns. The quality of this user-generated data does hamper us from drawing certain conclusions, such as which activities are perceived as most difficult. Further research is needed with improved data quality and volume in order to obtain more generalizable results.

CCS Concepts
• **Social and professional topics → Professional topics → Computing education → Computing education programs**

Social and professional topics → Professional topics → Computing profession → Employment issues

Applied computing → Education

Keywords
Computing Internships; Workplace Learning Analytics; Data-Driven Curriculum Development

1. INTRODUCTION
To overcome the skills gap between industry demands and learning outcomes achieved by graduates in higher computing education, many higher education computing programs integrate some form of workplace learning in their curriculum. The rationale is that internships better prepare students for their entry into the workplace after graduation, due to the authentic tasks and recent technologies that students encounter while working and learning in the industry. Furthermore, learning in the workplace allows students to transfer computing knowledge and skills from university to professional contexts more easily.

In practice however, educators often do not know specifically which tasks their students perform in the workplace and which technologies they use. This lack of insight is mainly due to the distance between coach and student during the work placement. Generally, only a few visits take place where the coaching teacher visits the workplace of the student. Higher educational institutes also experience a lack of cohort-based overviews of activities performed in internships, since every single teacher only coaches a small group of interns and a broader collection of workplace learning data is scarce.

Learning analytics endeavors in higher education have primarily focused on classroom-based learning. Recently, workplace learning analytics (WPLA) has become an emergent research area. WPLA opens up opportunities to analyze workplace learning data of students in computing programs in higher education and give us more insight into which knowledge and skills are necessary for computing students in their internships and hence should be covered in the curriculum beforehand. Especially for higher professional education – as is offered mainly at universities of applied sciences – workplace learning analytics seems to be crucial in data-driven or data-informed curriculum development.

Conference'10, Month 1–2, 2010, City, State, Country.

DOI: http://dx.doi.org/10.1145/12345.67890

2. THEORETICAL BACKGROUND
In this section, we discuss the theoretical background of this study focusing on the skills gap, workplace learning and technology-enhanced learning. Finally, based on this overview, we identify the goal of this study.

2.1 Skills Gap and Workplace Learning
Due to the ever-faster digitization of our society, our workplaces and our everyday life, the need for well-equipped graduates in the IT and computing domain has risen sharply [10]. According to the EU, there is a significant 'Digital Skills Gap' in Europe: business find it hard to employ both IT specialists as well as non-IT personnel with sufficient digital literacy [6, 7]. To fulfill the needs imposed by the digital transformation in Europe, two problems need to be tackled: 1) a shortage of computing graduates and 2) a mismatch between graduates' qualifications and industry requirements. In this study, we focus on the latter.

In comparison to other domains, computing graduates need a very diverse skills set, ranging from hard skills, to innovative skills to social skills [25]. In a study among recent computer science graduates and employers of these graduates, a skills gap was found to exist for many of the skills that employers require of new employees [27]. In a large study among Spanish employers, it was

found that in general, employers feel that graduates lack field-specific practical knowledge, whereas the gap for field-specific theoretical knowledge is much smaller [9].

Workplace learning within formal higher education programs is believed to help overcome the skills gap between graduates' qualifications and industry requirements. Gaining work experience during a Bachelor study, offers students an opportunity to develop both 'hard skills' (subject-matter) and 'soft skills' such as teamwork, planning and communication [11]. Graduates with prior work experience are generally considered to have a higher 'employability' [2].

Workplace learning offers students opportunities to practice job-specific functions, such as socialization, innovation and job performance [16]. An increasing number of universities is integrating some form of workplace learning into their curricula. The pedagogy of such integrations [26] and the design of such hybrid learning environments [29] have been subject to recent studies.

2.2 Technology in Education

Technology-Enhanced Learning (TEL) refers to the use of technology to support learning and teaching. TEL is often used as a synonym for e-learning but can also be used to refer to technology enhanced classrooms and learning with technology, rather than just through technology [23]. Besides e-learning, other examples of enhancing learning through the use of technology are Massive Open Online Courses (MOOCs) and Computer-Supported Collaborative Learning (CSCL). TEL provides the advantage of easier access to more information and creates flexibility in time and location of learning. Generally speaking, TEL can be used to (i) replicate existing teaching practices, (ii) supplement existing teaching or (iii) transform teaching and/or learning processes and outcomes [12].

Most studies that consider technology to support learning in the workplace, focus on the (formal or informal) learning of professionals working in industry and governmental organizations, often from a human resources point of view [19]. Examples of such technologies are (personal) knowledge management tools, e-learning platforms, mobile learning applications and social collaboration systems. Only few studies aim to design, develop and evaluate technologies that specifically support workplace learning, which is informal, contextual and social in nature [20, 22].

Recently, design propositions for Technology-Enhanced Workplace Learning (TEWL) have been developed and evaluated, and a web application was developed that implemented most of these propositions [22]. This open-source web application provides students with an interface to register their working and learning activities in the workplace in an easy-to-use way, which in turn allows for analytics (a dashboard with charts) giving them insight into their learning process.

A specific research area within TEL is Learning Analytics (LA), which provides data-driven interventions in the learning process, such as LA dashboards and predictive or recommendation functionalities within a Learning Management System (LMS). The most commonly used definition of learning analytics is the one presented at the first Learning Analytics & Knowledge (LAK) conference: 'Learning analytics is the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs' [1].

Learning analytics for workplace and professional learning is a recent area of interest. In 2016, a workshop on this topic was organized at the LAK conference [14]. One year later, a first systematic literature review on workplace learning analytics was published [17]. Only a few of the studies included in this review considered workplace learning as part of formal (higher) education. In a recent, extensive literature review of LA literature, the number of studies gathering data from informal settings (e.g. the workplace) and having a program-wide scope turned out to be very small, and even decreasing over the last few years [4].

Over the last few years, several approaches have been developed to improve education programs and curricula in a data-driven way. Data-Based Decision Making (DBDM) mainly uses hard data such as test scores to guide instructional decisions such as adaption of teaching strategies [13, 28]. Schildkamp and Kuiper published a comprehensive overview of the use of data to inform curriculum development in secondary education [18]. For higher education, in 2014, an integral approach (including a process and the design of a tool) was proposed to enable data-driven course design [5]. Specifically for higher computing education, Méndez et al. presented an overview of techniques to analyze school-based data to obtain recommendations for curriculum re-design [15] and Toetenel and Rienties analyzed learning designs of university courses in order to evaluate the impact of pedagogical decision making [24].

Research on analyzing data collected in the workplaces of student internships in the computing domain is scarce. Recently, possibilities were explored to use the data generated by the users of a TEWL application for data-driven curriculum development and requirements were elicited by evaluating the proposed solution with program managers [21].

2.3 Research Goal

For (computing) programs in higher professional education wishing to improve their curriculum in a data-driven way, it is crucial to analyze workplace data from student internships. In this study, we aim to answer the following three research questions.

When students in higher professional computing education perform an internship in industry:

- *Which activities do they perform most?*
- *Which activities do they find most difficult?*
- *Which technologies do they use most?*

The answers to these questions help to open the 'black box' of internships in higher computing education, which in turn can help universities to (re-)design their curriculum to better prepare students for professional practice.

3. METHODOLOGY

3.1 Participants

In the Fall Semester of 2018, we deployed and configured the aforementioned open-source TEWL application [22] and provided it as an opt-in instrument for a group of 183 third-year students in three Bachelor programs in the IT and Computing domain in a European university of applied sciences. It was mandatory for all students to record their worked hours; they were offered a choice between their own logbook solution (e.g. in Word of Excel) or to use our application. Out of the 183 students, 81 registered an account in our application, of which 68 entered at least one activity. We note that the participating students were self-selected, which might give rise to biased data.

Students performing an internship in these Bachelor programs have to spend at least 100 working days of at least 7.5 hours within an organization in the IT industry. Participants that registered less than four weeks of activities (equal to 20 working days, ca. 160 working

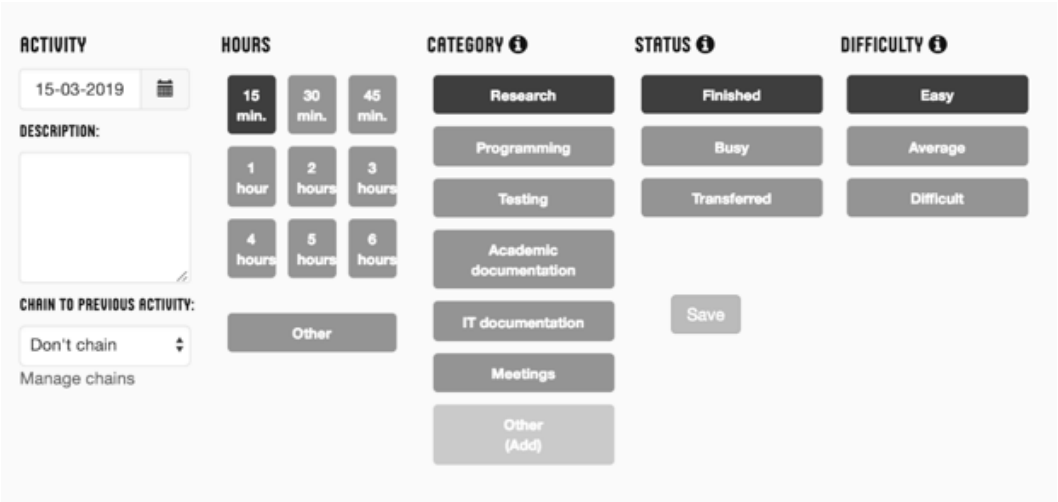


Figure 1. Screenshot of input screen of the application (with categories customized for the SE cohort).

hours) were removed from the data set, in order to have data representative of working activities for a significant amount of time with respect to the default internship of 100 days. This yielded a final set of 54 students (49 male, 5 female) that were included in this study. The average number of days per student on which activities were registered is 90 (σ=25,2 days).

All participants gave informed consent to the use of the registered data for research purposes. All data was anonymized before the analysis.

3.2 Data Collection

In the application we provided to the participants, students register their working activities in an input screen as shown in Figure 1. The relevant part of the underlying database scheme is shown in Figure 2.

Central to the database model of the application is the *LearningActivity* table of which the attributes are recognizable in the input screen of the application. A student enters a description of an activity performed during their internship and labels this with (meta)data such as a date, a category, resources used, a status and a difficulty.

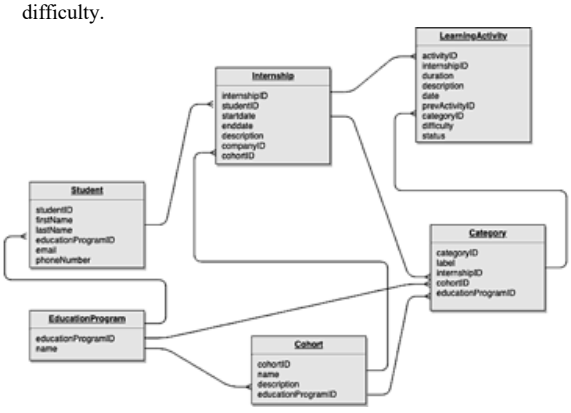


Figure 2. Core of the Entity-Relationship Diagram

For authentication, administration and customization purposes, the tables *Internship*, *Student*, *EducationProgram* and *Cohort* are included. A *Cohort* is a group of students enrolled in the same program at the same time (e.g. all Software Engineering students starting their internship in September 2018). For our study, the table *Category* is interesting as this is used to label activities into specific classes that should give insight into the nature of these activities.

Categories can be defined i) at the education program level, ii) at the cohort level or ii) user-generated by a student. For this study, all categories predefined by us were defined at the cohort-level and based on a national framework aimed at defining learning outcomes for Bachelor programs in the IT/Computing domain.

3.3 Analysis

We started our analysis with the necessary data extraction and data cleaning.

3.3.1 Descriptive analytics

As can be seen in Figure 1 and 2, students input data into a single text field (*Description*), after which they label the activity with meta data: *Duration*, *Category*, *Status*, *Difficulty* and optionally a previous activity. In our analysis, we do not use this chaining functionality and we also neglect the associated *Status* field (activities labelled as 'Busy' become available for chaining later activities).

We filtered all activities of the 54 participants from the MySQL database and exported them as comma-separated files for further analysis. We first explored the workplace data by computing several statistics on the number of activities, the number of days and the total duration of the registered activities per student; see Section 4.1. After this, the main focus of the descriptive analysis is on the *Category* field. We analyzed which categories were i) user-generated (Section 4.2), ii) most occurring (Section 4.3) and iii) most difficult (Section 4.4).

3.3.2 Text analysis

After the descriptive analyses of the meta-data of the entered activities, we performed a content analysis of the *Description* field. The analysis was focused on trying to determine the intent of the description as written by the student, by looking for commonly

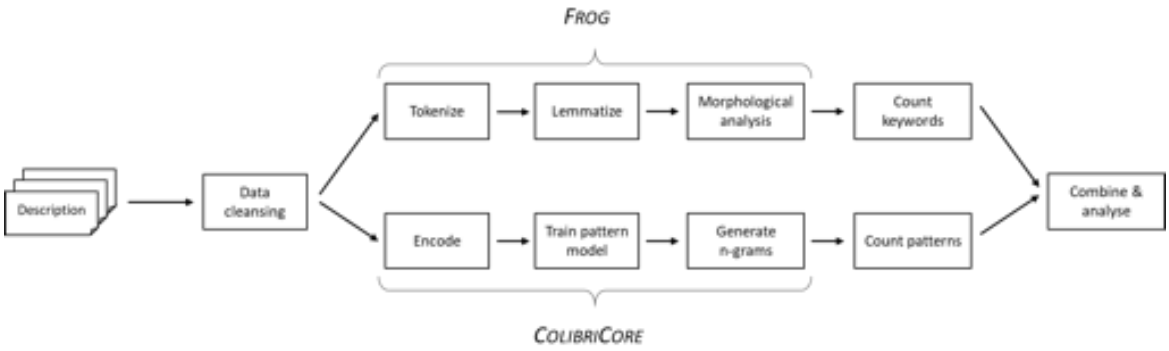


Figure 3. Text analysis pipeline.

occurring keywords. This was done for each of the pre-defined categories of activities (*Research, Programming, Testing, Academic Documentation, IT Documentation,* and *Meeting*). The descriptions were analyzed for each separate category, for all cohorts combined. The hypothesis was that the descriptions of the activity would largely match the category of the description (e.g., most students would mention something related to *Programming* in the description of a *Programming* activity), and that this would be no different between the cohorts.

To be able to check the descriptions for frequently occurring keywords, a Dutch Natural Language Processing toolkit called LaMachine[1] was used. In particular, a morphological tagger and parser named FROG [3] used to tokenize and lemmatize the words (i.e., to make sure that "program", "programming" and "programmed" are counted as a single keyword). Furthermore, we analyzed the descriptions on frequently occurring patterns of words (called n-grams) by means of the COLIBRICORE tool [8] also part of LaMachine.

The text analysis pipeline applied is shown in Figure 3. Results of this analysis are presented in Section 4.5.

A second text analysis was performed by combining all descriptions of the *Programming*, *Implementation/Configuration*, *Testing*, and *IT Documentation* activities, but separated by Cohort. These sets of descriptions were analyzed again using the same text analysis pipeline, but subsequently filtered for nouns, verbs and special named words. The resulting set of keywords was manually filtered on mentioned technologies. Each of these mentioned technologies was then used to determine the number of students mentioning using that technology (that is to say, instead of counting the frequency of that technology occurring in the descriptions, we counted the number of distinct students mentioning that technology instead). This gives us an indication of the most frequently used technologies. The results of this analysis are shown in Section 4.6.

4. RESULTS

4.1 Descriptive Statistics

The participants were students of three different Bachelor's programs (cohorts): Business Informatics (BI), Software Engineering (SE) and IT Systems & Networks (SN). In Table 1, descriptive statistics (mean and standard deviation) of the data on the workplace activities entered by the participants is shown: the number of activities registered, the total number of hours spent on all activities in an internship, the number of days with registered activities, the number of activities registered per working day and the duration of a single activity (in hours).

Out of the 54 students included in this study, six students entered activities with an average duration of 7 hours or more. Of these six, four students consistently entered a single activity per working day (average duration of a single activity ≈ 8h). We observe that even though some students might register merely a single or two activities per day, in professional practice it is seldom the case that one performs a single same activity for a longer period of time (and that for many days in a row). We presume these students did not want to put in the (extra) effort of entering several activities per day, even if this means they will not get usable insights from the analytics of these data.

As a measure of variety in types of activities performed in the workplace, we analyzed how many different categories the participants used to label their registered activities; see Figure 4. The median is eight different categories. Notably, one student only used a single category to label all activities performed during their internship, whereas another student used as much as thirteen different categories. We notice the BI students on average used the least number of categories (6.2), whereas SE students used most (8.2).

4.2 User-generated Categories

As mentioned earlier (Section 3.2), a set of categories was pre-defined for each cohort, based on the learning outcomes of the Bachelor programs:

Table 1. Descriptive statistics of the workplace learning activities entered by the participants (per student).

Cohort	#students	#activities		total duration in hours		#days		#activities per day		duration of a single activity in hours	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
BI	12	195	118.0	752.8	156.00	94.6	19.53	2.1	1.09	4.9	2.18
SE	29	204	93.5	735.6	161.08	92.5	19.40	2.2	0.90	4.0	1.23
SN	13	150	88.8	635.1	278.51	79.8	36.10	1.9	0.64	4.8	1.67
Total	54	189	100.9	715.2	200.24	89.9	25.18	2.1	0.90	4.4	1.64

[1] https://proycon.github.io/LaMachine/

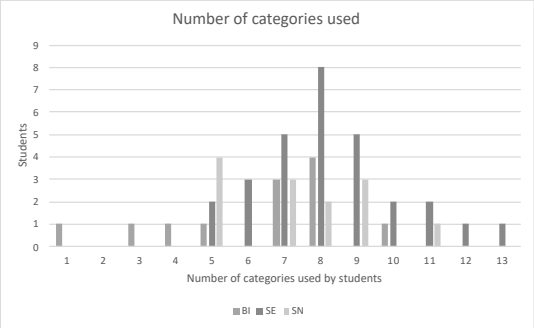


Figure 4. Number of different categories used by students to label their activities in the application.

- All cohorts:
  - Academic Documentation
  - IT Documentation
  - Meeting
  - Research
  - Testing
- BI & SN:
  - Implementation / Configuration
- SE:
  - Programming

These pre-defined categories were the categories most-used by students to classify their activities. However, the application also allows users to create their own categories. We presume that students that add customized categories feel that the pre-defined categories do not cover some of the activities they performed during their internship. We analyzed which categories were added by the computing students by extracting those from the database and classifying them into classes of activity categories. The results are presented in Table 2.

Notably, sixteen students created an 'Other' category for activities that did not fit the pre-defined categories. Moreover, five students created a 'Container category' with customized labels such as 'Internship', 'Internship activities', or 'Work'. Of course, these categories do not give much insight into which specific activities are performed by our students in the workplace. Other user-generated categories that were added while they do not provide additional insight into the work activities of computing students are categories related to the category classes 'Illness', 'Break / Free time / Drinks', 'Absent' and 'Outside of work' (or when the category label was left empty). It is interesting that students did actually register activities in such categories, since absent days and activities outside of work do not count as working hours according to academic guidelines at the university of the participants. In Table 2, all user-created categories that do not give any new insights into what students do are shaded dark grey.

The category classes that are shaded light grey actually overlap with pre-defined categories: Research, Meeting and (IT) Documentation. The students that created these categories may have overlooked the corresponding categories that were already available.

All categories that are not shaded grey in Table 2, do give us more insight into students' working activities. Most notable are the category classes 'Introduction', 'Training / course / personal development', 'Demo / presentation' and 'Setting up (technical) working environment'. Often, the internship is the first industry experience for a student, so it makes sense that many students register activities dealing with getting to know the company and the assignment, getting familiar with and setting up their working environment (such as a DTAP-pipeline) and taking in-house training or courses.

Table 2. Student-generated activity categories.

Category class	BI	SE	SN	Total #students
Other	4	8	4	16
Introduction	1	5	7	13
Illness	1	5	5	11
Training / course / personal development	4	5	1	10
Demo / presentation	2	6	1	9
Preparation	1	5	2	8
Setting up (technical) working environment		4	2	6
Container category	1	1	3	5
Planning		5		5
Break / Free time / Drinks	1	2	1	4
Scrum activities		3	1	4
Teambuilding		4		4
Research	3	1		4
University work		4		4
Meetings	1	2	1	4
Assisting others			3	3
Absent	1	2		3
(IT) Documentation	1	1		2
-left empty-	1	1		2
Proof of concept	2			2
External visit / meeting		1	1	2
Problem solving		2		2
Organizing		1		1
Administration	1			1
Deployment		1		1
Outside of work		1		1

Only seven user-generated categories were used to label more than 100 working hours of a student, of which three were a 'container category'. The other four were: 'Documentation', 'Research, 'Gathering information' (classified as Introduction) and 'Proof of Concept'.

4.3 Most Occurring Activities

To find the most occurring activity categories, we summed up the durations of all activities and grouped them by category; we merged the user-generated categories Research, (IT) Documentation and Meeting with their corresponding pre-defined categories. We disregarded those categories that had a total duration of less than 100 hours over all 54 students. In Table 3, the relative occurrence of the remaining activity categories can be found. Since we did not include all categories, the percentages do not necessarily add up to 100%. In the right-most column, the relative occurrence of the category over students of all three cohorts is shown.

Table 3. Relative occurrence of activity categories over the three different cohorts.

Category class	BI	SE	SN	Overall
Programming		44,7%		25,7%
Academic documentation	29,1%	16,4%	32,3%	22,4%
Research	27,5%	13,6%	21,1%	18,2%
IT Documentation	13,2%	13,6%	12,6%	13,3%
Implementation/ configuration	8,1%		19,1%	5,7%
Meeting	5,7%	4,2%	4,4%	4,6%
Container category	8,4%	0,2%	4,3%	2,8%
Testing	1,1%	2,7%	2,3%	2,2%
Other	1,1%	0,5%	0,8%	0,7%
Introduction	1,7%	0,3%	0,6%	0,7%
Illness	0,3%	0,5%	0,9%	0,6%
Planning		0,8%		0,5%
Preparation	0,2%	0,5%	0,3%	0,4%
Proof of concept	1,6%			0,4%
Training / course / personal development	0,3%	0,4%	0,0%	0,3%
Demo / presentation	0,8%	0,1%	0,1%	0,3%

From Table 3, we can conclude that the pre-defined categories as listed in the previous section, were the categories that were used most by students (the sum of their percentages is 92.1%). The 'Container category' class is the most-used user-generated category, which is not surprising since students presumably use such categories to label all of their activities.

We also notice that SE students spend almost half of their time programming, while BI and SN students clearly spend less time on the corresponding category Implementation / configuration. In turn, BI ad SN students spend over half of their time on the two categories Academic Documentation and Research, whereas for SE students, this combination of categories accounts for less than a third of the total time spent. Regarding IT Documentation, Meeting and Testing, all cohorts relatively spend the same amount of time: approximately 13%, 5% and 2% respectively.

For the user-generated categories, we believe the most notable insight is that BI students spend more time on both Introduction and Demo / presentations, as compared to SE and SN students.

In Figures 5, 6 and 7, we picture the ten most occurring categories for each of the three cohorts separately for a more visual indication of the distribution of hours spent on the various activity categories.

4.4 Difficulty of Activities

As shown in Figure 1, students can label the activities they register with the perceived difficulty. We allocated weights to these difficulty levels as follows: Easy=1, Average=2 and Difficult=3.

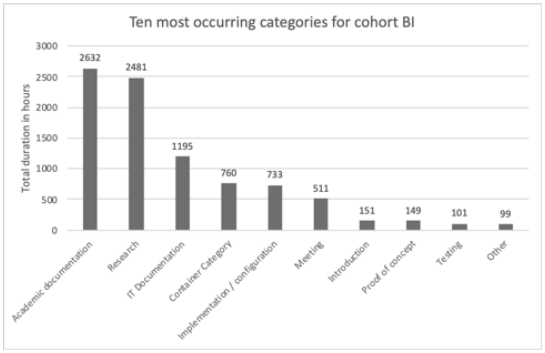


Figure 5. Ten most occurring categories for students from the program Business Informatics (BI) in total number of hours.

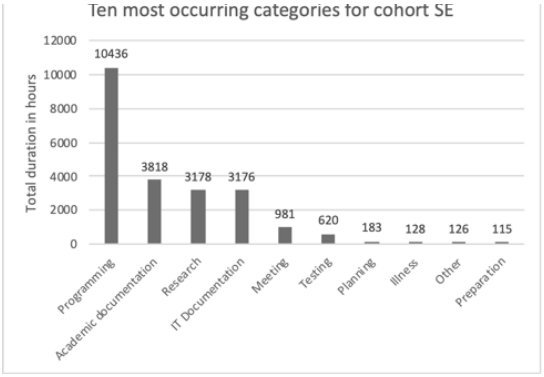


Figure 6. Ten most occurring categories for students from the program Software Engineering (SE) in total number of hours.

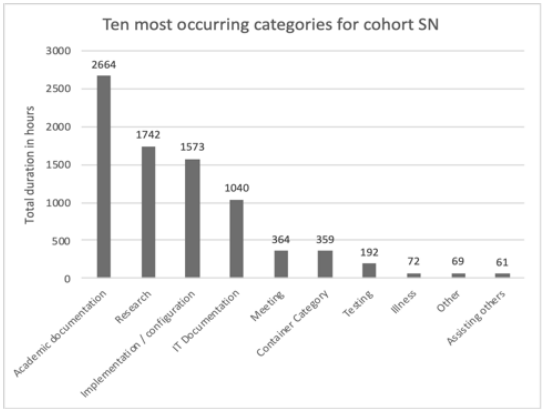


Figure 7. Most occurring categories for students from the program IT Systems & Network (SN) in total number of hours.

Table 4. Distribution of difficulty levels over all registered activities.

Difficulty	Activities	Percentage	Duration
Easy	6,101	59.8%	20,419h
Average	3,394	33.2%	14,906h
Difficult	714	7.0%	3298h
Total	10,209	100%	38,623h

First, we studied some descriptive statistics of the difficulty labels entered by students; see Table 4. We note that a mere 7% of the activities is labelled Difficult by the participants, whereas almost 60% of activities is labeled Easy.

The original goal of the TEWL application we used in this study, is to create more awareness of the learning process and to realize more of the learning potential of students working in internships [20], e.g. by showing a reflection prompt (with feedback and feedforward questions) whenever students enter an activity with the difficulty level Average or Difficult and with the Status Busy. We hypothesize that students are generally not motivated to fill out a reflection prompt and as a reaction will label their activities as Easy, leading to only a small percentage of activities labelled Difficult. Further in-depth and qualitative analysis can shed light on whether students indeed find most of their work during internships as easy as they have inputted in the application.

Next, we analyzed the filled-out reflection prompts to gain insight as to why students found specific activities difficult. A vast majority of students (83%) indicated that the reason they perceived that activity difficult was a lack of experience. A mere 3% of the filled-out prompts recorded that a lack of resources was the culprit. Finally, we analyzed which activity categories where perceived to be most difficult. We disregarded category classes that were used by only a single student and computed the average difficulty for the remaining category classes. The results of this analysis can be found in Table 5.

We see that the average difficulty over all categories for the BI cohort is 1.43, for the SE cohort 1.48 and for the SN cohort it is clearly lower with 1.26. Based on our data set, we cannot ascertain whether SN students indeed have 'easier' internships or whether they have other reasons to label more activities as Easy.

For the BI cohort, the categories Testing (2.13), Implementation / configuration (1.63) and Research (1.54) have the highest average difficulty, whereas Introduction has the lowest average difficulty (1.01). For the SE cohort, the top three is Deployment (2.00), Scrum activities (1.95) and Demo / presentation (1.90), whereas University work and Introduction have the lowest average difficulty (1.00). For the SN cohort, the top three is Implementation / configuration (1.57), Academic documentation (1.54) and IT Documentation (1.53), whereas both Preparation and Setting up (technical) working environment have the lowest average difficulty (1.00).

Interpretation of these results is not straightforward; a certain category can be labelled Difficult by merely a couple of students, or for a small sum of working hours, or even both. In that case, we cannot claim this first category is harder than another category that has a slightly lower average difficulty, but was labelled by more students and for more working hours. For example, for the SE

Table 5. Difficulty of category classes for the different cohorts.

	Total	BI			SE			SN		
Category (class)	Avg. Difficulty	Avg. Difficulty	Total Duration	Students	Avg. Difficulty	Total Duration	Students	Avg. Difficulty	Total Duration	Students
Academic documentation	1.47	1.36	2,632	11	1.52	3,818	29	1.54	2,664	13
Assisting others	1.04							1.04	61	3
Demo / presentation	1.70	1.50	69	2	1.90	28	5			
Deployment	2.00				2.00	32	2			
Implementation / configuration	1.60	1.63	733	6				1.57	1,573	10
Introduction	1.03	1.01	151	2	1.00	79	5	1.07	50	7
IT Documentation	1.40	1.24	1,195	11	1.45	1,173	27	1.53	1,038	9
Meeting	1.08	1.07	511	11	1.05	981	29	1.13	365	11
Planning	1.27				1.27	183	5			
Preparation	1.12				1.24	115	6	1.00	21	2
Problem solving	1.83				1.83	29	2			
Programming	1.71				1.71	10,436	28			
Proof of Concept	1.50	1.50	149	2						
Research	1.40	1.54	2,481	12	1.28	3,178	29	1.38	1,742	13
Scrum activities	1.95				1.95	40	3			
Setting up (technical) working environment	1.31				1.63	29	4	1.00	4	2
Teambuilding	1.50				1.50	19	4			
Testing	1.70	2.13	101	4	1.66	620	25	1.32	192	8
Training / course / personal development	1.26	1.33	24	3	1.19	101	5			
University work	1.00				1.00	47	3			
	1.44	1.43			1.48			1.26		

cohort the category Programming has average difficulty 1.71 with a total sum of 10,436 working hours for 28 different students, whereas the category Deployment has average difficulty 2.00 with a total sum of only 32 working hours for two different students. Would it be justified to conclude that Deployment is more difficult for SE students than Programming? We do not think so.
Based on the above reasoning, we conclude that the current data set does not allow us to determine which category was the most difficult for an entire cohort of students. Nevertheless, we believe the results in Table 5 can give some insight to program managers of these students and can be a starting point for further analysis.

4.5 Students' Perceptions of Activity Categories

By means of the text analysis pipeline show in Figure 3, we analyzed the descriptions entered by the students for each of their activities. We analyzed whether the descriptions were congruous to the activity's category. The analysis described in Section 3.3.2 was performed on the original Dutch texts, the keywords were translated to English only for presentation in this paper.

The descriptions of the activities in the following categories were inconclusive:

- *Implemation/Configuration*: 'implement' was mentioned in only 9.8%, and 'configure' was mentioned in only 8.8% of all 510 descriptions;
- *Programming*: 'program' was mentioned in only 5.1% of 2140 descriptions[2]; and
- *IT Documentation*: 'documentation' was mentioned in 8.1% of 800 descriptions).

Each of these keywords mentioned was also the highest ranking keyword for that category.

The results from the analysis of the categories *Meeting*, *Testing*, *Research*, and *Academic Documentation* are shown in Figure 8 to 11. Each of these categories has a keywords used in the descriptions of the activity congruous to the activity to which it is related. In the *Meeting* category, it clearly shows that 'meeting' and 'daily stand-up', mentioned in 28.8% and 16.5% of the 1450 descriptions respectively, are what the students perform most often. Moreover, meetings are more often with their 'supervisor' (10.3%) than with teachers (2.3%) or colleagues (2.2%). 'Test' is mentioned in 64.8% of all 250 descriptions of the *Testing* category and 'research' is mentioned in 37.8% of all 1810 descriptions of the *Research* category. 'Project plan, 'final report', 'reflection report', and 'personal development plan', are mentioned in 30.7%, 15.8%, 10%, and 7.3% of 2200 descriptions in *Academic Documentation* respectively, clearly showing what the university requires the students to write.

4.6 Technologies

The second text analysis performed on the descriptions tries to determine whether the students are using relevant and modern technologies in their internship. As mentioned in Section 3.3.2, we ran the descriptions of the relevant categories of each cohort through the text analysis pipeline (see Figure 3), and subsequently checked for each technology found, how many distinct students mentioned that particular technology. The results of this analysis for the SN and SE cohorts are shown below are shown in Figure 12

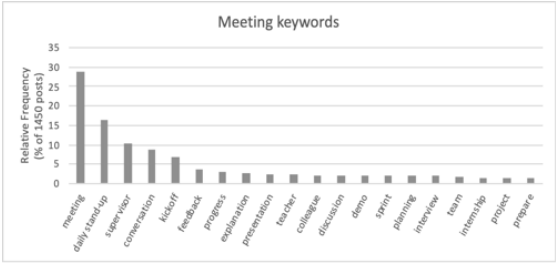


Figure 8. Frequency of keywords in the category *Meeting*.

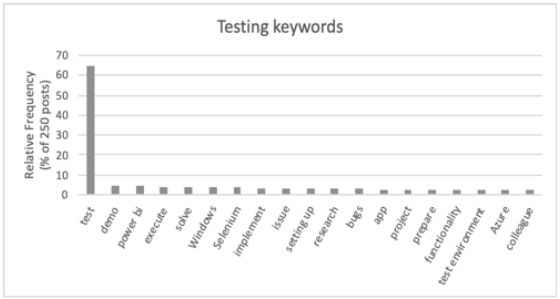


Figure 9. Frequency of keywords in the category *Testing*.

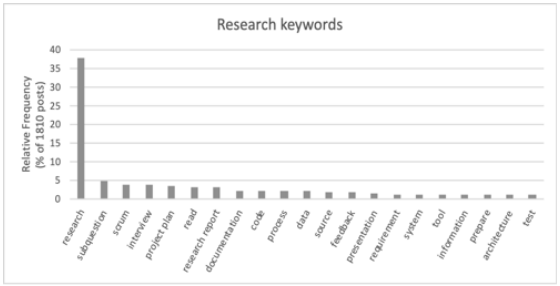


Figure 10. Frequency of keywords in the category *Research*.

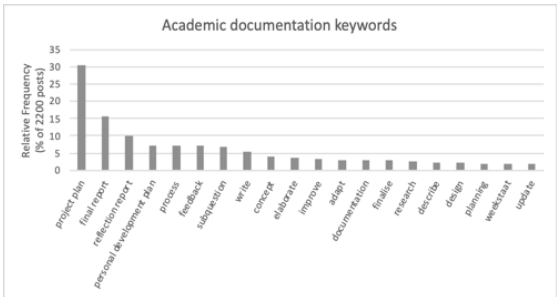


Figure 11. Frequency of keywords in the category *Academic Documentation*.

[2] Although the keyword 'realize' occurred in 15.1% of 2140 descriptions, we cannot definitively claim that students do *realize*, since the Dutch word from which it derives ('maken') can also be used as an auxiliary verb. The n-grams did not show any noteworthy patterns.

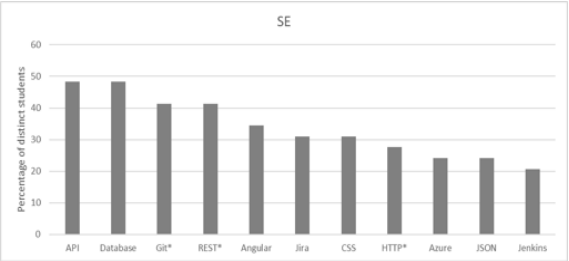


Figure 12. Technologies frequently mentioned by SE students.

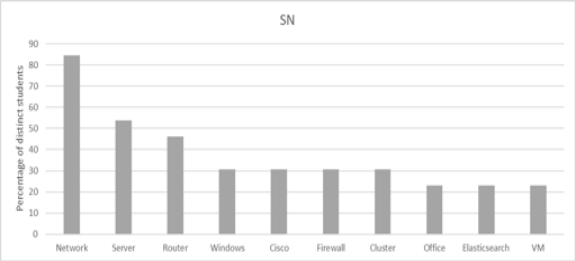


Figure 13. Technologies frequently mentioned by SN students.

and Figure 13, respectively. The analysis of the descriptions of the BI students did not lead to any conclusive technologies (top mentioned technology was 'Sharepoint' by four distinct students).

As shown by Figure 12, several software engineering technologies are mentioned frequently by different SE students; 'API' is mentioned by 48.3% of students, just as 'Database' 48.3%, 'Git*' (meaning anything starting with git: 'git', 'github', 'gitlab', etc.) 41.4%,'REST' and 'RESTful' by 41.4%, 'Angular' by 34.5%, 'Jira' by 31%, 'CSS' by 31%, and 'HTTP' and 'HTTPS' by 27.6%. Clearly, most of the SE students are indeed programming, even though our previous analysis was inconclusive on that matter. Moreover, most of them appear to be working on front-end (reactive webpages) and back-end (databases and api's) applications. The students mention the programming languages used less; 'JavaScript' is mentioned by 6 distinct students (20.7% of total); 'Java', 'Python' and 'C#' are mentioned each by two distinct students (6.9% of total).

Figure 13 clearly shows that the SN students are mentioning 'network' rather often (84.6% of total), again showing that they appear to be working on what is expected. Other frequently mentioned technologies are: 'server' by 53.8%, 'router' by 46.2%, and 'Windows', 'Cisco', 'Firewall', and 'Cluster' by 30.8% each.

5. CONCLUSIONS AND DISCUSSION

5.1 Conclusions

The goal of this study was to gain more cohort-based insight into which tasks computing students perform, and which technologies they use, while learning the workplace. To this aim, we gathered and analyzed workplace activity data of 54 students over the course of their third-year semester-long internships in the computing industry. We performed descriptive analyses to gain insight into which categories of activities students performed most, which were programming, documentation and research.

Students added their own categories to the application to label activities. Most notably, they created categories for orientation and introduction activities, getting acquainted with a company efficiently and setting up their technical working environment. In our university, we do not prepare our students for these activities. These new insights could help us refine and improve computing curricula, especially the curriculum parts before the internship. Additionally, we can use the results of this study for expectation management towards our students. When providing prospective interns with information and instruction, we could include data on which activities are performed most.

Subsequent text analysis gave us insight into students' perceptions of the categories used to label activities. The user-generated descriptions of activities in the categories testing, research, meetings, and academic documentation are congruous with their labels, while the analysis of the descriptions of implementation/ configuration, programming and IT documentation did not give conclusive results. Finally, we analyzed which technologies were used most by these students, which did not yield any conclusive insights for the BI cohort, but we did find technologies for the SE and SN cohorts which were in line with what we expected based on the learning outcomes of these programs. Repeating this analysis every year has the potential to reveal technology trends which could be used to update the curriculum accordingly.

The content analyis of the user-generated descriptions, combined with the analyses of the most occuring categories, implies that the common understandig which claims that interns are mainly busy getting coffee for others can be debunked, at least for this group of computing students.

5.2 Limitations

Based on the results, we conclude it is feasible to use user-generated data to get insights into workplace activities of computing interns. However, the quality of the data does hamper us from drawing certain conclusions, such as which activities are perceived as most difficult.

Registering time sheets at work is usually perceived as boring and bothersome. To some extent, we recognize 'resistance' to the workload involved in three main strategies students use to reduce the time needed to register their activities: 1) register work in very large portions (one or two activities per day), 2) creating custom 'container categories' to label activities, and 3) label activities as Easy to avoid a reflection prompt. All three strategies degrade the quality of the data and thus also the results obtained after analysis.

The text analysis we performed also has limitations. For example, in the category Programming, the variation of keywords used was very large. The text analysis pipeline we used did not allow to easily count and cluster different wordings for the same activities. Additionally, students do not always mention the technologies they used explicitly in the descriptions of their activities. Adjustments to the application (e.g., providing support questions in the input field for the descriptions or adding a field for technologies) could give higher quality data. A more advanced analysis of the user-generated texts can subsequently give more insightful results.

Finally, the students in this study opted in to participate, which can induce self-selection bias in the data. We analyzed data of merely 30% of all students performing an industry internship. We do not know whether their data was representative for the entire population. Furthermore, we only included students from one university in this study. All the above limitations have an influence the obtained results and the generalizability of our conclusions.

5.3 Future Work

We plan to extend this study in the near future by recruiting a larger set of students, also by cooperating with other higher educational

institutes. It would also be interesting to perform this type of study for other educational domains, such as economics, health or education. Furthermore, it would be interesting to include other, not user-generated, workplace data such as log files, etc. This was not feasible since the 54 students performed their internship at 54 different organizations. Finally, the text analysis of the descriptions of activities could be extended by comparing the results with the results of a similar analysis of school-based descriptions (syllabi), e.g. to compare which technologies are used and the relative weight of categories in internships compared to the relative weight in the curriculum in order to identify 'weak spots' in the curriculum that differ most from what industry requires from computing graduates.

REFERENCES

[1] 1st International Conference on Learning Analytics and Knowledge 2011: 2011. *https://tekri.athabascau.ca/analytics/*. Accessed: 2019-02-22.

[2] Andrews, J. and Higson, H. 2008. Graduate employability, "soft skills" versus "hard" business knowledge: A european study. *Higher Education in Europe*. 33, 4 (2008), 411–422. DOI:https://doi.org/10.1080/03797720802522627.

[3] Bosch, A. van den et al. 2007. An efficient memory-based morphosyntactic tagger and parser for Dutch. *Computational Linguistics in the Netherlands 2006: Selected papers from the seventeenth CLIN Meeting*. LOT. 191–206.

[4] Dawson, S. et al. 2019. Increasing the Impact of Learning Analytics. *Proceedings of the 9th International Conference on Learning Analytics & Knowledge - LAK19* (New York, New York, USA, 2019), 446–455.

[5] Dunbar, R.L. et al. 2014. Connecting Analytics and Curriculum Design: Process and Outcomes of Building a Tool to Browse Data Relevant to Course Designers. *Journal of Learning Analytics*. 1, 3 (Aug. 2014), 223–243. DOI:https://doi.org/10.18608/jla.2014.13.26.

[6] EC 2016. ICT for work: Digital skills in the workplace. European Commission.

[7] EPRS | European Parliamentary Research Service 2017. *Digital skills in the EU labour market*.

[8] van Gompel, M. and van den Bosch, A. 2016. Efficient n-gram, Skipgram and Flexgram Modelling with Colibri Core. *Journal of Open Research Software*. 4, 1 (2016). DOI:https://doi.org/10.5334/jors.105.

[9] Hernández-March, J. et al. 2009. Graduates' Skills and Higher Education: The employers' perspective. *Tertiary Education and Management*. 15, 1 (Mar. 2009), 1–16. DOI:https://doi.org/10.1080/13583880802699978.

[10] Hüsing, T. et al. 2015. *e-Skills in Europe*.

[11] Itani, M. and Srour, I. 2016. Engineering Students' Perceptions of Soft Skills, Industry Expectations, and Career Aspirations. *Journal of Professional Issues in Engineering Education and Practice*. 142, 1 (Jan. 2016), 04015005. DOI:https://doi.org/10.1061/(ASCE)EI.1943-5541.0000247.

[12] Kirkwood, A. and Price, L. 2014. Technology-enhanced learning and teaching in higher education: what is 'enhanced' and how do we know? A critical literature review. *Learning, Media and Technology*. 39, 1 (Jan. 2014), 6–36.

[13] Van der Kleij, F.M. et al. 2015. Integrating data-based decision making, Assessment for Learning and diagnostic testing in formative assessment. *Assessment in Education: Principles, Policy & Practice*. 22, 3 (Jul. 2015), 324–343.

[14] Ley, T. et al. 2016. Learning analytics for workplace and professional learning. *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge - LAK '16* (New York, New York, USA, 2016), 484–485.

[15] Méndez, G. et al. 2014. Techniques for data-driven curriculum analysis. *Proceedins of the Fourth International Conference on Learning Analytics And Knowledge - LAK '14* (New York, New York, USA, 2014), 148–157.

[16] Nijhof, W.J. et al. 2008. *The Learning Potential of the Workplace*. Sense Publishers.

[17] Ruiz-Calleja, A. et al. 2017. Learning Analytics for Professional and Workplace Learning: A Literature Review. *EC-TEL 2017, LNCS 10474* (2017), 164–178.

[18] Schildkamp, K. and Kuiper, W. 2010. Data-informed curriculum reform: Which data, what purposes, and promoting and hindering factors. *Teaching and Teacher Education*. 26, 3 (2010), 482–496.

[19] Schmidt, A. and Kunzmann, C. 2006. Towards a Human Resource Development Ontology for Combining Competence Management and Technology-Enhanced Workplace Learning. *OTM 2006, LNCS 4278* (Oct. 2006), 1078–1087.

[20] Siadaty, M. et al. 2012. Semantic web and linked learning to support workplace learning. *CEUR Workshop Proceedings* (2012).

[21] van der Stappen, E. 2018. Workplace Learning Analytics in Higher Engineering Education. *IEEE Global Engineering Education Conference, EDUCON* (Santa Cruz de Tenerife, Canary Islands, Spain, 2018), 15–20.

[22] van der Stappen, E. and Zitter, I. 2017. Design propositions for technology-enhanced workplace learning. *Proceedings of EAPRIL* (Hämeenlinna, Finland, 2017), 37–51.

[23] Technology enhanced learning | Higher Education Academy: 2018. *https://www.heacademy.ac.uk/individuals/strategic-priorities/technology-enhanced-learning*. Accessed: 2019-04-08.

[24] Toetenel, L. and Rienties, B. 2016. Analysing 157 learning designs using learning analytic approaches as a means to evaluate the impact of pedagogical decision making. *British Journal of Educational Technology*. 47, 5 (Sep. 2016), 981–992. DOI:https://doi.org/10.1111/bjet.12423.

[25] Tynjälä, P. et al. 2006. From university to working life: Graduates' workplace skills in practice. *Higher education and working life: Collaborations, confrontations and challenges*. January (2006), 73–88.

[26] Tynjälä, P. 2013. Toward a 3-P Model of Workplace Learning: A Literature Review. *Vocations and Learning*.

[27] Wickramasinghe, V. and Perera, L. 2010. Graduates',

university lecturers' and employers' perceptions towards employability skills. *Education + Training*. 52, 3 (Apr. 2010), 226–244. DOI:https://doi.org/10.1108/00400911011037355.

[28] Wiliam, D. 2011. What is assessment for learning? *Studies in Educational Evaluation*. 37, 1 (Mar. 2011), 3–14. DOI:https://doi.org/10.1016/J.STUEDUC.2011.03.001.

[29] Zitter, I. et al. 2016. A Design Perspective on the School-Work Boundary: A Hybrid Curriculum Model. *Vocations and Learning*. 9, 1 (Feb. 2016), 111–131. DOI:https://doi.org/10.1007/s12186-016-9150-y.

Is Deductive Program Verification Mature Enough to be Taught to Software Engineers?

Marc Schoolderman
Radboud University
Nijmegen, The Netherlands
m.schoolderman@cs.ru.nl

Sjaak Smetsers
Radboud University
Nijmegen, The Netherlands
s.smetsers@cs.ru.nl

Marko van Eekelen*
Open University of the Netherlands
Heerlen, The Netherlands
marko.vaneekelen@ou.nl

ABSTRACT

Software engineers working in industry seldom try to apply formal methods to solve problems. There are various reasons for this. Sometimes these reasons are understandable — the cost of using formal methods does not make economic sense in many contexts.

However, formal methods are also often greeted with scepticism. Formal methods are assumed to take too much time, require tools that are too academic, or to be too mathematical to be understood by practice-oriented software engineers.

We tested these assumptions by designing a small course around a framework for program verification, aimed at regular computer science students enrolled in a Master's programme. After four lectures and associated exercises, students were given a small verification task where they had to model and verify a real, non-trivial, C function in Why3.

A significant majority of students managed to prove a non-trivial functional specification of this C function in the time allotted, and many also pointed out inherent flaws of this function discovered during formalization. Participants reported no major difficulties or mental hurdles in learning Why3, and considered its approach to be appropriate for selected components of safety-critical software.

While formal verification tools such as Why3 still have lots of room for improvement, this experience shows that in a short amount of time, software engineers can be taught to use a program verification tool, and obtain usable results without being fully proficient in it. We further recommend that courses on formal methods should also let students explore these as techniques to be applied, instead of only focusing on the theory behind them, as we expect this to gradually lower the barrier to wider acceptance.

KEYWORDS

formal verification, teaching, Why3

*Also with Radboud University.

This work is part of the research programme 'Sovereign' with project number 14319 which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). .

CSERC '19, November 18–20, 2019, Larnaca, Cyprus

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Marc Schoolderman, Sjaak Smetsers, and Marko van Eekelen. 2019. Is Deductive Program Verification Mature Enough to be Taught to Software Engineers?. In *Proceedings of The 8th Computer Science Education Research Conference (CSERC '19) (CSERC '19).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Program verification is about as old as the field of computing science itself [17], and the focus of much academic research. Yet, it is seldom applied in industry. In the Sovereign project, we strive to find ways to apply formal methods — and deductive program verification in particular — to practical situations, in cooperation with partners from industry. This has made us acutely aware of the divide between industry on the hand, and academia on the other.

In our discussions, representatives of our industry partners expressed the sentiment that it was all well that *academics* understand the *academic* tools and can use them for selected problems, but that their company employs *software engineers* which, although academically trained, would not be able to learn and understand these techniques in such a way that they can use the effectively — and that formal verification can not hope to get a wider foothold in industry unless this problem was addressed.

This sparked our interest. Clearly, there is no argument that many programs are too large or too complex to be within reach of formal verification with the current tools. However, if the tools themselves are also perceived to be too complex or too theoretical to be understood, that is another problem altogether.

Therefore we were interested in testing the assumption that program verification, and associated computer-aided reasoning tools, would *intrinsically* be too much work to get accustomed with, for a practice-oriented software engineer.

One way to answer such a question is to design a crash course that teaches a formal method to software engineers, and see how they perform at a small verification task. However, since any *academically trained* software engineer will at some point have been a student, we can instead test this assumption by doing the same experiment with university students, which are readily available to us.

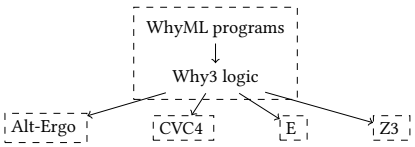
Therefore, we selected to teach the Why3 framework for program verification [11] in a short time span, as part of an existing course at Radboud University, and assess the performance of students at a small, but realistic verification task.

In Section 2, we will briefly introduce the Why3 verification framework. Section 3 provides an overview of the choices we made in teaching it to students. Sections 4 and 5 will present and discuss

our evaluation of the course. Section 6 provides an overview of related courses. Section 7 contains our conclusions.

2 OVERVIEW OF WHY3

Why3 [11] is a platform for deductive verification of programs. It takes programs written in a dedicated programming language (WhyML), which should be annotated with pre- and postconditions, assertions and loop invariants, and uses a weakest precondition calculus to generate verification conditions in a typed first order logic. These are then subsequently translated to the input formats of various automatic provers, in order to prove them, as visualized in the following diagram:



In Why3, as in many such systems, there is a difference between *programming constructs* and the *logical constructs*. A program is written in WhyML, and annotations are written in a logical language. There is a high degree of similarity between the two: they both share the same expression syntax and allow for the definition of functions. There are also subtle differences. Furthermore, Why3 supports *ghost code* [10]. This is any code or data that is not of any consequence for the outcome of a program, but *can* make its its verification easier. The Why3 type system will ensure that *ghost code* can not affect the outcome of a program, allowing the user to use purely logical constructs in computations that only have (*ghost*) side-effects. Furthermore, a user can 'promote' a programming construct to have an effect on the logical level. A function at the programming level can be used to either prove a logical lemma (a so called *lemma function* or `let lemma`), or provide an axiomatization of a function in a safe manner (Why3 calls this a `let function`). The similarities between the various levels of Why3 make it fairly easy to get novices started with Why3, at the same time, the subtle differences can also catch new users off-guard.

Most work in Why3 is done inside a graphical user interface where users can transform verification conditions (e.g., splitting a large conjunction into smaller ones), and send them to provers. Most actions are performed with a single mouse click. The user interface also uses colour highlighting to provide users with information about which programming constructs are involved in the current proof state.

3 APPROACH

Our treatment of Why3 was part of an existing course on 'Software Analysis', which is an elective course offered as part of the MSc programme in computing science at Radboud University. The scope of this course was intentionally broad, to treat varying topics in it relating to tools and techniques for analysing software. Formal program verification using Why3 was planned to take up the first half of this course, with the second half veering more towards static analysis. The entry requirement for this course is that students are in possession of a Bachelor's degree in computing science (or equivalent).

By using an existing course that was not advertised primarily as a course on formal methods, theoretical computer science, or theorem proving, we are confident that the students taking this course did not self-select, and form a unbiased representative sample of the computing science student population at Radboud University.

Secondly, to prevent students from deselecting after the start of the course, the guiding principle during teaching was to approach program verification using Why3 as a *tool* that might have a place in a software engineers toolbox, instead of treating formal verification as an intrinsically interesting theoretical topic, where Why3 is only used as a vehicle for demonstrating an application. So, for instance, students were not taught any core principles on which formal verification frameworks are based, such as Hoare logic or how to compute weakest precondition calculus.[1] Instead, this was explained only in so far as was necessary to give students an intuition into what Why3 is doing. Also, exercises focused mostly on familiarizing students with features of Why3, similar to a course on programming, and were chosen to enable quick positive feedback.

At the end of this part of the course, students had to write a short report about their work, in which they were also required to include a reflection on Why3. Furthermore, students were asked (but not required) to complete an anonymous survey intended to verify that they were indeed a representative sample of students enrolled in a Master's programme in computing science. In this survey, they are also asked about the time they needed for the course work, and their general disposition towards program verification after taking the course.

3.1 Course structure in detail

Teaching Why3 was split in two parts: in-class teaching with weekly lectures with associated homework exercises, and a small project where students would, in groups of two, tackle a verification challenge in the style of the VerifyThis[2] or SV-Comp[3] competitions. [4]

3.1.1 Teaching Why3 in four weeks. In-class teaching consisted of four weeks of lectures and exercises. The organization per week was as follows:

(1) Outlining a historical background motivating why program verification schemes should employ computer-aided reasoning in order to be feasible, and a first look at Why3 and how to write basic WhyML programs; as well as how to use the logical language of WhyML to write function contracts, invariants, and prove termination.
(2) How the Why3 type system works, and how to reason about mutable data such as arrays. At this point, students were subjected to an interactive demonstration where Kadane's algorithm [3] for finding the maximum subarray was verified, with the intent to set an example to students of how to write more complex logical specifications, discover invariants, and how to interpret the responses of the Why3 IDE.

[1] We do not mean to imply that taking that approach is not a valid didactic approach — however, it would not have been relevant for our research question.
[2] https://www.pm.inf.ethz.ch/verifythis.html
[3] https://sv-comp.sosy-lab.org/
[4] The course material, including exercises and project description is available online at https://cs.ru.nl/~M.Schoolderman/swan2019/

(3) Techniques in Why3 that can be used in more difficult situations where a proof is not solved automatically — such as cases where a proof by induction is needed. In particular, in this lecture the somewhat difficult concept of `ghost code` and `let lemma`'s [10] was introduced.
(4) Modelling a program in a different, more realistic, programming language in WhyML; e.g. how to handle integer overflow, or how to reason about a memory model that supports pointers (adapted from the lectures notes by Filliâtre [12]).

Weekly exercises were designed to fit with the level students were expected to have after each lecture. Students were required to use a Why3 installation on their own system instead of using a web interface for all of these, to have access to the full capabilities and multiple back-end provers. The objective of these exercises were, per week:

(1) Familiarizing with the WhyML syntax and Why3 interface. In particular, students had to finish a partial proof of a Russian peasant multiplication (adapted from an existing course by Bobot [5]). Students were also challenged to rewrite this verified program to perform exponentiation instead, with some hints.
(2) Writing a WhyML program and specification from scratch. Students had to pick a simple sorting algorithm and model it in WhyML, and prove it correct.
(3) Using *let lemma*'s to write inductive proofs, and a slightly harder partial proof (for the factorial function one taken from [17]) that needed to be finished.
(4) Modifying a WhyML program so that it can be used to extract C code.

We estimated that students would need not more than six hours for each exercise. Students were given formative feedback (including fixes to finish their proof efforts, whenever they were very close to a solution).

3.1.2 Verification challenge. After four weeks of in-class teaching, students project assignment. Students had to either choose a C functions from a small list of system library routines contained in the CloubLibc [24] library, or a test case that was designed to be similar to a routine for modular addition of 256-bit integers used in the cryptographic library TweetNaCL [4].

Given this C function, the assignment consisted of modelling it reasonably accurately in WhyML, providing a formal specification, and proving that the WhyML model adheres to this specification.

Most student teams chose to verify the `strlcat` routine from CloudLibc; probably because it looked the easiest due to its familiar operation (string concatenation) and relatively shorter size. Two teams chose the TweetNaCL-inspired test case.

3.1.3 Self-evaluation. Students were given four weeks to complete the verification challenge. Afterwards they had to write a report documenting their formalization, motivating their modelling choices and formal specification; students also had to reflect on what they considered to be the strengths and weaknesses of Why3, and were encouraged to do so from a software engineering viewpoint.

In order to learn more about the learning experience, we also sent a survey asking students about how they self-assessed their programming and mathematical maturity, how much time they spent on the Why3 exercises and the verification challenge, and what their disposition to formal verification was at the end of the course, as an extra check of (our interpretation of) the information contained in their reports.

4 RESULTS

In total 22 students participated seriously in the verification challenge, grouped into 11 teams. One other student made only a rudimentary attempt and would eventually drop out of the course after the part focusing on Why3 part was finished.

Out of the 11 teams, nine chose to verify the `strlcat` routine from the CloudLibc library, and two took on the modular addition routine, as mentioned in Section 3.1.2.

4.1 Verifying `strlcat`

Out of the nine teams that chose `strlcat`, seven teams produced a WhyML model with a logical specification that was fully verified in Why3. The remaining two teams delivered an incomplete proof. In both cases this seems to be due to teams choosing to build an axiomatization of the C memory model instead of a simpler approach. We discouraged students from doing this, because it risks introducing logical inconsistencies. In these two cases, however, the axiomatization was simply not powerful enough to support drawing the necessary conclusions.

The C function `strlcat` is intended to be a safer version of `strcat`, for concatenating null-terminated strings in a manner which is much less likely to cause buffer overruns, but also guaranteeing that the result is a proper null-terminated C string.

There are three major difficulties where the verification effort of this function is not straight-forward:

(1) `strlcat` is not required to (and in fact will not) perform its expected operation in case the two pointers it is passed point to overlapping regions of memory.
(2) In case `strlcat` is called with a `size` argument that is too small, there is a subtle safety mechanism that prevents it from accessing out-of-bounds memory addresses. However, in this case `strlcat` will not concatenate any strings or necessarily produce a result that is null-terminated.
(3) To reason about the length of strings at the specification level, the notion of the "terminating null character" needs to be expressed somehow.

All successful teams used the technique for modelling memory outlined in [12], where memory is modelled as an array of bytes and pointers are indices into this array. To tackle the first problem, students either used different arrays to model separate memory regions, or explicitly added a precondition that the two input strings should not overlap. One team was so precise in this that they proved the code works in some cases of overlap.

The second problem was handled by all teams either by adding the explicit precondition that the `size` argument is proper, or specifying a separate postcondition for the cases where it is improper. Some teams commented (rightly) that this made the formal specification of `strlcat` more intricate than would appear necessary, showing that they were able to draw conclusions about the subtleties of systems-level C code based on their formalization.

The third problem was one which was handled in different ways. Several teams took a hint that existential quantification can often be avoided by using *ghost code* [10], and simply added the location of the null character terminating a string as a *ghost argument* to `strlcat`. One team used direct existential quantification instead to find the null character. This resulted in a much harder, yet still successful proof effort. Several other teams added a logical construct (using a `let function`) which explicitly finds the position of the null character using a loop, but which can still be used in specifications. This essentially involves also proving the correctness of `strlen`, and is an interesting approach that these students discovered themselves.

4.2 Verifying modular addition

Only two teams attempted to verify the modular addition routine; both teams completed verification, where one team performed a thorough analysis, and the the other only proved a simple property.

Even though the modular addition code consisted of the least lines of code of all the options available to students, it was probably the most challenging to verify given that (unlike the CloudLibc routines) it did not have a clear (informal) specification, and has known problems that become apparent during verification.

During this challenge, both teams used Why3's machine integers to prove the absence of signed integer overflow under reasonable input conditions. Both teams were instructed to identify some property that the addition routine preserved. For example, whether if both inputs are already reduced modulo $2^{255} - 19$[5], it holds that the result will also be in a reduced form. Both teams correctly concluded that such an easy property did not exist. However, one team already reached this conclusion before formalizing the routine it. In the end, one team proved that the modular reduction step in the addition routine is never performed if the original inputs were in reduced form. The other team simply proved a bound on the output. Only one team tried to prove that the addition routine actually *adds* numbers; this was achieved with some supervision.

4.3 Self-evaluation

Students were asked to provide an evaluation of Why3 and to assess what role it could have in software engineering. The most often mentioned benefit of Why3 was that it makes performing formal proofs accessible and easier, and that proofs provide a higher degree of confidence in software than simple testing. On the other hand, students reported that the verification task involving only a small piece of code took many hours to complete. Students concluded that they considered Why3 inappropriate for regular software engineering due to this time investment, but well-suited for cases where safety or security of software is more important than economic arguments.

Other interesting general observations raised by students where the following:

- Students were generally positive about the graphical user interface of Why3, which can highlight assumptions and goals in the source code.
- Students criticized the error messages provided in case of a syntax or type error. This is an area where academic tools are often lacking, and this clearly hinders the learning process.
- Multiple students complained about the lack of an online community of Why3 users, on a platform such as StackExchange.

Many students also discussed how they experienced working with Why3 in the process of finding a correct proof; key points raised here were:

- Since Why3 uses automatic provers, proofs can fail for reasons that are not obvious to the users and hard to predict for novices.
- Why3 provides a mechanism to generate counter-examples. While students did report using this, the consensus seemed to be that they did not provide meaningful information except in simple cases.
- Similarly, some students were dismayed that an inconsistency in a lemma (or invariant) will allow any subsequent statement to be proven vacuously — giving the user the mistaken impression that the they only have one unproven goal remaining. On the other hand, other teams reported using the *smoke detector* of Why3 to catch these cases with success.
- To explore some program state in depth, several students reported missing the ability to have an interactive debugger in the Why3 IDE.
- Finding loop invariants is clearly the hardest part. Several students found it surprising that Why3 was unable to deduce simple loop invariants, or that a loop invariant also needs to be established if the associated loop is not executed.

Although students reported needing a lot of time to complete the verification challenge, most did not report an obstacle inherent to Why3 while working on the challenge. An obstacle that *was* reported by several students was that to model a C program in WhyML requires a deep understanding of C — a deeper understanding than these students professed to have. These students also pointed out that this translation of C to WhyML should be automated.

4.4 Survey

To learn more about the learning experience, a survey was sent to the 22 students that participated seriously in the verification challenge. One student was at this time no longer at Radboud University and could not be reached. Of the remaining students, 15 responded, for a response ratio of slightly above 70%, which we consider to be acceptably high for a student evaluation.

In general, students reported a much higher confidence in their programming ability than their mathematical ability. All participants reported that they felt at least somewhat skilled in programming (rating themselves at least a 5 on a 7-point Likert scale), whereas two thirds of the students reported their mathematical skill level to not that great (at most a 3 on a 7-point Likert scale). Only one student reported a higher skill level in mathematics than programming, but this student also reported having taken much more ECTS credit in maths and logic courses (120, instead of the average 18).

[5] A prime number that defines a finite field used in the Curve25519 elliptic curve.

When asked whether they had any experience in other formal method tools, only a handful students reported having some exposure to systems such as Coq or Uppaal (which are used to support teaching in some bachelor courses at Radboud University). On the other hand, many students reported having used ESC/Java2. This tool is used only briefly at another Master's level course at Radboud University for a simple weekly exercise, and so we do not consider this to have significantly impacted our teaching experiences or jeopardize the representativeness of our student sample. When asked in the survey how they compared learning Why3 to learning a new programming language, 12 students answered that learning Why3 was as hard or slightly harder, with only 3 indicating they found it considerably harder.

The survey results also indicated that the amount of time spent on the weekly exercises was within our estimate and fitting for the number of ECTS points that could be earned for the course. More interestingly, the amount of hours students spent on the verification challenge was reported to be slightly less than 20 hours on average, with the median being 18 hours. In advance, we had budgeted between 24 and 32 hours for the challenge. A histogram of the hours reported is shown in Figure 1.

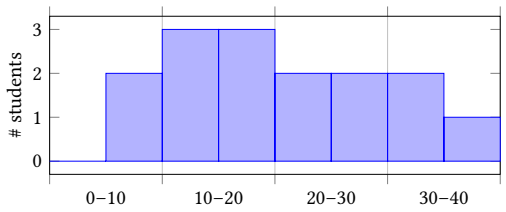


Figure 1: Hours spent on verification challenge by students

5 DISCUSSION

At the start of the verification challenge, we had expected about half of the students to succeed, and another half to deliver only a partial result which either proves only a single property, or where the final result is incomplete due to unproven goals. We also expected students to require around 24 to 32 hours of work. In total, 8 out of the 11 teams managed to complete the challenge with a successful verified result, which we found encouraging. If we assume that the three less successful teams (comprising 6 students) all make up the lower end of the graph of Figure 1, we can still conclude that students completed the challenge in less time than we anticipated.

Students, however, did report needing a lot of time for the verification task. We would like to put this in perspective. The code students were given was representative of systems-level code. Developing such code, including writing a good test suite, takes more time than students probably realize.[6] Compared to that, the amount of time students actually reported is rather low, especially if we take into account that they were all inexperienced users of Why3. To see how much faster they would be if we gave them another similar challenge would be interesting, but we could not justify this being part of the 'Software Analysis' course.

Ultimately, we think that our students were up to the challenge that we put to them, even though they had only received a crash course of four weeks.

5.1 Comparing students to software engineers

Our survey results indicate that our student sample was a typical set of students enrolled in the Master's programme at Radboud University, and did not have a significant prior bias or inclination towards formal methods. For example, in the survey, 60% of the students reported not being familiar with Hoare logic at the start of the course. In the other 40%, only one student reported that he or she used this familiarity while working with Why3. All were familiar with the concept of pre- and postconditions, but had little experience in reasoning about invariants.

Most of these students will, after graduation, apply for software engineering jobs, and so we believe our sample to be representative of a *highly educated* software engineer. Of course our results do not apply to all software engineers in possession of a Bachelor's degree in computing science, since our students did choose to enrol in a Master's programme at a university.

5.2 Modelling challenges

As mentioned in Section 4.3, some students reported finding it less satisfying to make a model of C code WhyML, due to the fact that they felt less confident that they could model C concepts accurately. At another stage in the 'Software Analysis' course, several students also expressed discomfort in reasoning about C programs. This was surprising to us. As one student team put it: "This decreases our confidence that when a WhyML model is proven correct the program in the target language will also be correct." In the anonymous survey, students would also indicate they found modelling the second hardest part in using Why3, after finding loop invariants.

Students identified that a remedy would be to have a tool that either directly verifies C code, or that automatically translates C code into WhyML, but that such a step would inevitably also make verification more difficult. In the survey, 80% of the students indicated that the design of modern programming languages should use formal verification to some degree, which we find consistent with the students' written remarks.

5.3 Using mature tools

In our course, we chose to not use the web interface of Why3[7], but required students to install it on their own systems. The advantage of this was that students could tackle more complicated proofs, since they had access to all the supported powerful automatic provers, and could benefit from all of the features of Why3, such as counterexample generation and the *smoke detector* feature of Why3. Even though there is much available to them in the full Why3 interface which they will not understand (at least at first), we do not find that this impedes the learning process.

The downside here was that installing Why3 is a rather involved process, due to the need to install it (and the *specific* versions of various automated provers that are supported by Why3) from source.

[6] In the case of `strlcat`, the function students ended up seeing is the end result of years of intermittent updates and tweaks

[7] http://why3.lri.fr/try/

To save time, and to ensure most students were on the same platform, we created a binary-only distribution of Why3 and a selection of automated provers for Linux. This worked well, but was a step in distributing Why3 to students that was cumbersome for us and that should not have been necessary.

5.4 Recommendations

Complex programming languages such as C++ are used daily by practitioners that will freely admit to not understanding all the intricate details. The same is also true for formal verification tools. However, in our experience, this is not how they are usually *perceived* in industry. In order to change this, we believe two things are necessary. First, verification tools should not forget to focus on user friendliness; this will lower the barrier to acceptance by engineers that are typically used to work with highly polished tools. Meaningful error messages will also ensure a positive feedback cycle that benefits the learning curve. Second, a university curriculum should not only present computer-aided reasoning tools as part of an (elective) theoretical course on program logics, as this will cement the notion that these tools are mostly an academic pursuit. Students should also be encouraged to explore the application of these tools to small but realistic problems.

6 RELATED WORK

Formal verification of software is a research area for which interest is still growing. It is therefore not surprising that attention is paid to it within higher education, also because of the great social importance of secure software.

We note that both fully automated proof tools and semi automated proof assistants are used within education. In a number of cases, this is done as support when teaching students about mathematics topics that are often experienced as difficult. For example, in [13], the design of the web server ProofWeb for Coq (developed to avoid installation difficulties with different versions of COQ) used to teach logic to undergraduate students is presented. Another project [6] uses the COQ proof assistant based on a two-step approach. When teaching students, the authors strictly stick to COQ in the first step. Thereafter, in the second step, they encourage students to gradually convert less formal ordinary textbook proofs into formal COQ proofs. The GeoGebra tool [15] is an automated reasoning tool for discovering theorems on constructed geometric figures, and proving these theorems automatically. The tool is intended to serve as a guiding stick fostering student activities while learning elementary geometry.

In the context of computer science teaching, formal verification is generally introduced at a more advanced level. A theorem prover is not a learning goal in itself but is rather considered as a framework for teaching other subjects. The idea is that using a formal language as a means for introducing new concepts helps student to get a deeper understanding of these. For example, [20] is a textbook on semantics entirely based on the proof assistant Isabelle[8], and the NASA PVS Library[9] contains a full formalization of Nielson and Nielson's textbook [19] on formal semantics. The main advantage of using a proof assistant in the teaching is that it allows students to experiment with their specifications, and to make proofs that are guided by the proof assistant which gives them immediate feedback. RISCAL [25] is a language for modelling algorithms and their properties. This language comes with a tool supporting model development and automatic verification. The tool has been used in two courses at the computer science department of the Johannes Kepler University Linz: (1) The course 'Formal Methods in Software Development' for master students, and (2) the 'Logic' course for undergraduate students. First experiences are promising: a small scale study indicated students seem to perform better if they can use the tool in its full potential. The PEST framework [7] is similar to RISCAL in the sense that is provides both a specification language and a tool (available as a plug-in for Eclipse) that facilitates automated reasoning. Classroom experiences (the framework was used in two undergraduate courses taught at the computer science department of the University of Buenos Aires) confirm the preliminary results of [25].

In [22] experiences are discussed with teaching formal program specification and verification using the specification language JML and the automated program verification tool ESC/Java2. The authors state that current program verification technology is sufficiently mature for students to use, even as part of courses which are not specifically about formal methods, such as standard programming or software engineering courses. However, the authors also indicate that the use of these tools is better limited to controlled experiments, where the students work with (relatively small) supplied programs, rather than code they develop themselves.

Other experiences are based on the approach in which students are supposed to develop loop invariants before actually writing their code, which is also known as *invariant based programming*. Invariant based programming was already introduced in the 1960s ([18],[14]), and developed further by e.g. [9]. In [1], the results are discussed of using this approach in two different courses. In both courses, the SOCOS [2] environment was used to develop invariant diagrams. The environment computes the verification conditions (VCs) automatically for all transitions in these diagrams, and sends them to either an automatic prover (SIMPLIFY [8]) or to an interactive prover (PVS [21]), similar to the technique that is employed by Why3. One of the courses was an advanced course for graduate students where they were asked to prove program correctness of the generated VCs using PVS. The second course was a beginners course, where the students could discharge the VCs to the SIMPLIFY SMT solver to perform automatic proving, and to PVS for those that could be automatically proved. In both cases, the authors observed that a suitable error reporting mechanism is clearly needed when using these tools in education. In particular, they commented on the difficulties of students when dealing with PVS. They also warn against the pitfall that the tools invite students to use a try-and-debug strategy instead of thinking beforehand about the constraints needed for the invariants.

Another project [23] presents a method to gain insight into the difficulties that students face while developing suitable loop invariants, and assist them in the process. The authors collected data in the background as students attempted to produce verified code with loop invariants. Analysis of this data indicated the kinds of information that can expected, and what kinds of feedback might be useful. In [16] the authors report on an experiment with invariant

[8] https://isabelle.in.tum.de/
[9] https://github.com/nasa/pvslib

based programming. They analysed a group of novice students and found that the main difficulty seemed to be lack of skills in formalizing expressions in general, rather than inventing specific invariants. Hence, to successfully use invariant based programming, appropriate training to develop these more general formalization skills is essential.

7 CONCLUSION

As the literature shows, tools for computer-aided reasoning have been used in the classroom successfully for many years, and we encourage this. It is clear that a computer can give students more instant feedback, and is less likely to make a mistake.[10] Also, in logic courses, it can alleviate tedium by offering automation.

The question that we sought to answer is whether computer-aided reasoning tools are also getting mature enough they can not only be used in the classroom for teaching, but that we can also train software engineers in industry to apply them to solve real — although perhaps small — problems in a short amount of time. Our results support the conclusion that this is the case.

Our survey indicates that our student sample is fairly representative for students who have completed a Masters degree in computing science, many of whom will eventually pursue a career as a software engineer. In fact all of our students, being in the possession of a Bachelor's degree, could just as well have been working in this field already.

A purist approach would only allow students to use automation offered by powerful tools only after the student demonstrates proper understanding of the actions that are being automated. When dealing with Why3, this is impossible — due to the fact that it relies on state-of-the-art SMT solvers to prove goals — and, we believe, not necessary. Students (and software engineers) with a sufficient level of higher education will have developed intuitions about reasoning about programs, and have had formal training in programming and logic. They can draw upon these experiences when learning Why3 (or we expect, similar tools) in an applied setting.

As a side effect, we also predict that trying out formal verification tools in a realistic setting on students will provide the developers of these tools with invaluable feedback. Having physical access to a novice user base trying to apply these tools will give insight into what makes them difficult to learn, or where they need to be more powerful. Ultimately, this will result in computer-aided reasoning that will be more usable and powerful for everybody.

ACKNOWLEDGMENTS

The authors thank Léon Gondelman for providing ideas in setting up the course on Why3.

REFERENCES

[1] Ralph-Johan Back. Invariant based programming: basic approach and teaching experiences. *Formal Aspects of Computing*, 21(3):227–244, May 2009.
[2] Ralph-Johan Back and Magnus Myreen. Tool support for invariant based programming. Technical Report 666, TUCS - Turku Centre for Computer Science, Turku, Finland, Feb 2005.
[3] Jon Bentley. Programming pearls: Algorithm design techniques. *Commun. ACM*, 27(9):865–873, September 1984.
[4] Daniel J. Bernstein, Bernard van Gastel, Wesley Janssen, Tanja Lange, Peter Schwabe, and Sjaak Smetsers. TweetNaCl: A crypto library in 100 tweets. In Diego Aranha and Alfred Menezes, editors, *Progress in Cryptology – LATINCRYPT 2014*, volume 8895 of *Lecture Notes in Computer Science*, pages 64–83. Springer-Verlag Berlin Heidelberg, 2015.
[5] Francois Bobot. MPRI lecture 2-36-1. http://francois.bobot.eu/mpri2018/, 2018.
[6] Sebastian Böhne and Christoph Kreitz. Learning how to prove: From the coq proof assistant to textbook style. In Pedro Quaresma and Walther Neuper, editors, *Proceedings 6th International Workshop on Theorem proving components for Educational software, ThEdu@CADE 2017, Gothenburg, Sweden, 6 Aug 2017.*, volume 267 of *EPTCS*, pages 1–18, 2017.
[7] Guido de Caso, Diego Garbervetsky, and Daniel Gorín. Integrated program verification tools in education. *Software: Practice and Experience*, 43(4):403–418, 2013.
[8] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. *J. ACM*, 52(3):365–473, May 2005.
[9] E. W. Dijkstra. A constructive approach to the problem of program correctness. *BIT Numerical Mathematics*, 8(3):174–186, Sep 1968.
[10] Jean-Christophe Filliâtre, Léon Gondelman, and Andrei Paskevich. The spirit of ghost code. *Formal Methods in System Design*, 48(3):152–174, Jun 2016.
[11] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 — where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *Proceedings of the 22nd European Symposium on Programming*, volume 7792 of *Lecture Notes in Computer Science*, pages 125–128. Springer, March 2013.
[12] Jean-Christophe Filliâtre. Deductive program verification with why3: A tutorial. UniGR Summer School on Verification Technology, Systems & Applications 2018, 2018.
[13] Maxim Hendriks, Cezary Kaliszyk, Femke van Raamsdonk, and Freek Wiedijk. Teaching logic using a state-of-the-art proof assistant. *Acta Didactica Napocensia*, 3:35–48, 2010.
[14] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, October 1969.
[15] Zoltán Kovács, Tomas Recio, and M Vélez. Using automated reasoning tools in geogebra in the teaching and learning of proving in geometry. *International Journal for Technology in Mathematics Education*, 25:33–50, 07 2018.
[16] Linda Mannila. Invariant based programming in education - an analysis of student difficulties. *Informatics in Education*, 9(1):115–132, 2010.
[17] F. L. Morris and C. B. Jones. An early program proof by alan turing. *Annals of the History of Computing*, 6(2):139–143, April 1984.
[18] Peter Naur. Proof of algorithms by general snapshots. *BIT Numerical Mathematics*, 6(4):310–316, Jul 1966.
[19] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: A Formal Introduction*. John Wiley & Sons, Inc., New York, NY, USA, 1992.
[20] Tobias Nipkow and Gerwin Klein. *Concrete Semantics – With Isabelle/HOL*. Springer International Publishing, 2014.
[21] S. Owre, J. M. Rushby, and N. Shankar. Pvs: A prototype verification system. In Deepak Kapur, editor, *Automated Deduction—CADE-11*, pages 748–752, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
[22] Erik Poll. Teaching program specification and verification using jml and esc/java2. In Jeremy Gibbons and José Nuno Oliveira, editors, *Teaching Formal Methods*, pages 92–104, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
[23] C. Priester, Y. Sun, and M. Sitaraman. Tool-assisted loop invariant development and analysis. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, pages 66–70, April 2016.
[24] Ed Schouten. CloudABI, cloud computing meets fine-grained capabilites. https://www.bsdcan.org/2015/schedule/track/Security/524.en.html, 2015.
[25] Wolfgang Schreiner. Theorem and algorithm checking for courses on logic and formal methods. In Pedro Quaresma and Walther Neuper, editors, *Proceedings 7th International Workshop on Theorem proving components for Educational software, THedu@FLoC 2018, Oxford, United Kingdom, 18 july 2018.*, volume 290 of *EPTCS*, pages 56–75, 2018.

[10] For instance, we would not have trusted ourselves to correctly check a pen-and-paper proof of the verification challenge described in Section 3.1.2.

Evaluation of a structured design methodology for concurrent programming

ABSTRACT

Learning how to design and implement a program is hard. Teaching methods and textbooks on Java programming often treat a new subject in terms of syntax and examples. Little attention is paid to systematically designing programs with these new concepts. Research has shown that such a complex task requires not only conceptual knowledge, but also explicit procedural support.

In this paper, we investigate the effect of combining conceptual and procedural guidance to teaching and learning concurrent programming. We build on earlier research in which we have introduced such a structured design methodology which divides the development of multi-threaded Java programs into a sequence of explicit, manageable steps: the Steps Plan.

We present our experiences with applying the Steps Plan in an introductory course on object-oriented programming, with multi-threading. The main questions addressed are: "What problems did the students encounter in direct relation to the Steps Plan?", and "What general problems surfaced?"

As to the first question, two important issues were that using a relatively far developed sequential solution as a stepping stone towards a multi-threaded solution wrong-footed some students, and that remedying race condition situations turned out to be supported at a too high level of abstraction.

As to the second question, two notable issues were that deciding on the right amount and type of concurrency by themselves is maybe too difficult for students at this level, and that the notion of (establishing) correctness or quality of a solution is, different from the sequential case, not intuitively clear to students.

For these issues, the paper recommends remedies and indicates directions for future work. We discuss the consequences for education as regards teaching materials and forms of teaching.

CCS CONCEPTS

• **Computing methodologies** → **Concurrent programming languages**.

KEYWORDS

Education, object-oriented programming, concurrency, Java, program design

CSERC '19, 18 – 20 November 2019, Larnaca, Cyprus

ACM ISBN 978-1-4503-6338-9/17/11…$15.00
https://doi.org/10.1145/3162087.3162088

1 INTRODUCTION

Many students have difficulties in learning to program. Programming is a very complex activity that requires effort and a special approach both in the way it is learned and taught. Traditional programming courses and standard textbooks on programming focus on conceptual knowledge rather than on procedural guidance of students. They mainly rely on the learn-by-example principle: After briefly introducing a new concept or language construct, and illustrating these new notions with examples, students are expected to construct their own programs 'by analogy'. The actual programming skills are then developed in supervised lab classes. This approach, while quite labour-intensive on the teaching side, provides the student with no methodical procedure for starting and systematically navigating the complex programming task. The lack of procedural guidance is often felt as ineffective. Kirschner, Sweller and Clark [11] argue that learning such a complex task not only requires conceptual knowledge, but also explicit procedural assistance. Essential is to understand which steps to take in order to solve the problem, as well as how to recognize a potential solution.

In this paper, we present our experiences with applying such a combined approach to teaching and learning concurrent programming. To support this complex task, we proposed a design methodology that provides scaffolding for the development of programs in the form of a sequence of explicit, manageable steps: the Steps Plan as presented at the CSERC 2017 conference [1]. This paper is part of an educational design research project [15] in which we improve our programming education through cycles of developing a concrete change in our teaching, applying it at several universities, collecting data on it and analyzing the effects.

To set the context, we briefly summarize the Steps Plan – for further detail we refer to our earlier paper [1]. The idea of the Steps Plan is to provide beginning students with a design method for simple concurrent programs in which the development of a program is divided into explicit, manageable steps. Each step consists of a procedural description of how to perform that step, and typically ends with one or more output artifacts that serve as input to the next step. This should scaffold the student's learning of concurrency concepts and their application. The steps proposed are essentially as follows, to be applied to a description of a programming problem that in a natural way involves concurrency: a simulation, a program with efficiency requirements or a program with responsiveness demands.

(1) *OO structuring of the problem domain* – Possibly a (sometimes given) sequential program using these classes that shows the essential behavior.
(2) *Adding the concurrency* – Analysis whether concurrency is needed, and what type of concurrency is involved. Decision of which activities should be performed concurrently, which determines the number and roles of threads.

(3) *Analysis on race conditions* – The program is analyzed on possible race conditions due to shared variables using an enhanced UML activity diagram. If needed, synchronization is applied to regulate all shared variable access.
(4) *Analysis on check-then-act race conditions* – Analysis of the program for check-then-act race conditions, again using the enhanced UML activity diagram, and preventing these using synchronized methods.
(5) *Reflection* – Assessing the quality of the resulting program. Reflection on analysis and design decisions made and going back to the appropriate step if remedy is required.

The artifacts here are the description of the problem, the class diagram, and the enhanced activity diagram. Furthermore, there is a more advanced part of the Steps Plan, concerning deadlocks, this is also not addressed in the present paper.

Our research questions are the following.

(1) What problems did the students encounter in direct relation to the Steps Plan, i.e., problems that can be related to direct deficiencies in steps or combinations of steps?
(2) What problems remain after providing the Steps Plan, i.e., what problems where encountered that are not so much deficiencies of the Steps Plan but more general issues with concurrency education?

Following up on this, we propose improvements, both in the Steps Plan and for the more general issues identified. Among the latter there also emerged matters impacting university education in a broad sense, such as the use of adaptive teaching materials and the balance between the number of subjects treated and the intensity with which they are practiced.

Our paper is organized as follows. Section 2 introduces the method used to investigate the application of the Steps Plan. Section 3 contains the analysis. We formulate the conclusions of the study and suggest improvements to our teaching method and plans for future research in Section 4.

2 METHOD

The study was conducted at three universities during a freshman course on object-oriented programming in Java. The research population was heterogeneous, with students from different academic levels and age-groups (varying from first-year bachelor students to students having a professional background in IT enrolled in a bridge program preparing for a master software engineering). The topic concurrency and the Steps Plan were taught and trained during two lectures of two hours and a tutorial of three hours. After that, ten students were observed during a practical work session while making an assignment on concurrency, on which they worked in pairs. The students were expected to complete this assignment in approximately 2 hours.

The assignment

The participants' activities were recorded while they were working on the following exercise:

> Imagine that a certain number of travelers (between 60 and 90) arrive at a train station. These people must continue to their final destination, a holiday resort, by taxi. Four taxis are available: two with a capacity of four and two with a capacity of seven people. The taxis ride back and forth as long as there are still people waiting at the station. Each taxi transports as many people as it can, or possibly less, depending on the actual number of people still waiting.

The goal of this assignment was to implement a concurrent simulation of the system that imitated reality as faithfully as possible. For this purpose, in order to reduce the task's size and complexity, the students were given a sequential solution that needed to be converted into a concurrent version. We expected that the students would need about two hours to complete this assignment.

session	duration	text	type
A	196 min	9137 words	Audio
B	51 min	842 words	Video+Screencasts
C	123 min	2188 words	Video+Screencasts
D	185 min	15413 words	Audio
E	32 min	3527 words	Audio

Table 1: Length of sessions

Data collection and analysis

The activities of five different pairs were recorded. This has happened in different ways. Only sound recordings were made of three groups. Video recordings were made from other two groups, combined with screencasts. The audio recordings were transcribed verbatim by the research team. The student activities in the screencasts and video recordings were described in text documents. First, all transcripts and descriptions were read in their entirety. Then, the first set of codes was generated through open pair coding. The main reason for pair coding is to obtain a consensus of two people for all decisions. Similar codes and quotes were clustered and labelled, and categories emerged from this process. Together, the research team organized the codes and clustered them into smaller thematic groups on a plenary team meeting. The discussions during this meeting led to a consensus on the division into themes. This axial coding process was performed using sticky notes on flip charts.

Results

The recorded sessions differed in duration; see Table 1. Table 2 summarizes the result of the coding process. The left column (named *category*) is the result of the axial coding (resulting in *themes* and corresponding *issues*) whereas the right column (named *observations*) shows some characteristic examples of codes that were placed in the same thematic group.

In the next section we will investigate these issues in depth.

3 ANALYSIS

By analyzing the observation results, we identified four shortcomings of our procedure. Two of these are shortcomings concerning an existing step, namely:

- We provided a sequential solution as a step towards a concurrent version. This confused some students, as they were

category	*observations*
Concepts	
Synchronization	Synchronizing the wrong code; synchronizing too much or too little code.
Shared resources	Identifying the wrong object as shared.
Race conditions	Misunderstanding the concept of race condition and/or of check-then-act.
Correctness	
Testing	Assuming that concurrent programs are deterministic and that tests are reproducible.
Input/Output analysis	Assuming the program is correct once it produces some output.
Procedural guidance	
Active Class design pattern	Misunderstanding the design pattern: failure to identify actor; referring to the design pattern at the wrong time.
Following the steps	Starting a next step before the preceding one has been completed; taking the steps in the wrong order.
Performing the steps	Confusing the active class with the class that creates the thread; being unable to perform the refactoring required by a correctly identified check-then-act situation; unclear how the domain model classes correspond to the thread model.
Implementation	
Emphasis on code	Inspecting code rather than design; ignoring the procedure and starting on the code right away.
Sequential simulation	Reproducing the limitations of the sequential simulation in the problem statement; trying to adapt the sequential simulation rather than designing afresh.
Other (unexpected) activities	
Anthropomorphisms	Ascribing human motives to threads and objects; detection of final state by observing prolonged inaction.
Unsuccessful approaches	Trial and error, (random) googling; having no plan at all; just responding to IDE error messages

Table 2: Code categories

unable to separate the program units to be reused in the concurrent version from those merely present for the purpose of sequential simulation.
- The steps are of too high a level, so that students, while aware what had to be done, were unable to perform the steps. Therefore the big steps have to be split into micro steps, and their execution needs to be practised.

The other two are points of interest missing in the procedure as a whole:

- The exercises do not specify the desired amount of concurrency.
- There is no explicit step requiring the students to check the output for correctness. This would also involve specifying precisely what correctness consists of.

Additionally, we make two observations about the use of anthropomorphism and the role of examples. These issues are discussed in more detail in the remainder of this section.

3.1 Taking a sequential solution as a step towards a concurrent version

We observed that students had difficulties identifying actors. Our procedure gives no concrete instructions on how to determine which activities should be performed concurrently. Moreover, in our exercise a sequential solution to the problem was provided, simulating concurrency through a *step* method. Students often persisted in maintaining parts of this 'step' function, leading to inadequate definitions of active classes, as illustrated by the following observation: one pair of students created a *Taxirunner* thread, without removing the *step* method, that was then started. They studied the output of the program and doubted the solution. However, they did not get the idea to abandon the use of the *step* method.

One explanation for this behavior is that novice programmers tend to be reluctant to modify code that they didn't write themselves, either because they don't understand it properly, or because they are afraid of breaking something that was already (close to) working. Another reason might be a deficiency in the student's conception of threads. Transforming a sequential program into a concurrent one, is by no means straight forward as an exercise for a first level course.

To remedy these issues, the assignments should indicate the desired behavior (e.g. the granularity of each task) more explicitly. Meanwhile, our procedure needs to more explicitly describe how students should decide which actions can be achieved concurrently. Therefore task definition (via the active class pattern) and task creation (using threads) have to be taken into account separately, avoiding any confusion of these activities.

3.2 Micro steps: the actual procedure is too high level

The suggested procedure is described at a high level of abstraction. In that sense, it is like a recipe that contains instructions like 'prepare chicken stock' without further explanation.

For instance, when a check-than-act situation is identified, the procedure states that 'the actions of checking and acting must occur in the same synchronized method'. In order to bring this about, refactoring steps will be necessary and it is assumed the student knows how to perform these. But it turns out that this is not always the case. If refactoring is done inadequately, this will lead to synchronized blocks that are far too large and take away most of the concurrency, resulting in programs that are correct but of low quality. This is not easily noticed by students [13]. The following fragment shows a example:

> **Student A:** *The problem is you can't put synchronized outside otherwise only one taxi will operate, that is not the intention of course. So you actually want to synchronize a block inside the while loop.*
>
> **Student B:** *If we synchronize this wrong, it still works, right? It should work, right? It is not just that efficient. You don't like one taxi to do all the things.*

A comparable problem occurs when the threads have to be created. One of the misunderstandings that occur here is that the active classes (those that implement *Runnable*) should have the responsibility for creating the threads. Another mistake we have seen is that far too many threads are created, namely a fresh one every time an object has finished some process step.

> **Student:** *The thread has to be created afresh every time. I just happen to know that. [...] There are four taxis and there will never be more. But each taxi is inserted into a thread as a task, and when it is finished its work it should go for a new ride. Then you should start a new thread, hence also create one.*

These are situations where students know which step of the procedure needs to be executed but lack sufficient competence to do so. There should be more detailed explanations available of what is to be done within each step of the procedure.

However, filling the teaching materials with detailed explanations of micro-steps will make them extremely boring for students that already have a good understanding of the required actions. In practice, some of the students entering the course will have had little exposure to programming, while others have several years' professional experience. It is impossible to provide a linear text that equally satisfying to both groups.

A possible solution may be found in the concept of adaptive hypertext [2]. This allows for details to be suppressed or provided according to the needs of the student. The decision may be taken explicitly by the student, by clicking some 'expansion' symbol, as has been already implemented by some online newspapers. Alternatively, the decision could be made automatically based on the results of the student's performance on formative tests.

3.3 Desired amount of concurrency

Once the decision has been taken that the program will have to execute some steps concurrently, the question naturally arises what will be allowed to take place concurrently and what not. For instance, a printer should be accessible to multiple threads containing print instructions, but characters produced by different threads should not be arbitrarily intermingled.

Initially, we provided exercises that did not explicitly specify the desired amount of concurrency: we assumed that this would be inferred from the intended use of the program in real life. However, this proved to be a mistake, as many students' submissions contained suboptimal choices in precisely this area. Hence, we now feel that either the exercise text itself must explicitly specify what is to be performed concurrently, or the students have to be trained in providing such specifications themselves.

The amount of concurrency has three aspects: the number of threads, the granularity of the protected code blocks, and the mutual synchronization of these blocks. The problem with the number of threads has already been addressed in Subsection 3.1. The granularity problem involves determining the size of a critical section: a code fragment that may refer to a shared resource that should not be accessed by different threads at the same time is called a *critical section* [3]. The size of the code fragment is important, because too large a critical section will result in threads being blocked unnecessarily long, thus reducing the advantages of concurrency. The mutual synchronization is important, because synchronizing blocks that do not interfere unnecessarily reduces concurrency, whereas not enough synchronization may lead to unsafe situations.

An extreme instance of a struggle with granularity choice that we observed was students protecting a block containing sleep statements.

> **Student A:** *Why don't you make the whole thing synchronized?*
>
> **Student B:** *Because the synchronized part should not be made too large.*
>
> **Student A:** *What's too large?*
>
> **Student B:** *You should not sleep within the synchronized block. Because there may be no people waiting at the station.*
>
> **Student A:** *Let's measure how long the sleep lasts.*

A distinct but related problem occurs if too many blocks are synchronized causing too many threads to be blocked, including ones that are not about to access the shared resource. The latter problem occurs when a single lock is used to delay all threads at the critical section, regardless of what resource is actually wanted in a thread's current context. To see an example of this, consider a system for reserving airplane seats: to prevent double bookings, a seat should

be blocked during such a transaction. Using a separate lock for each seat will solve this problem efficiently; using a single lock for all seats, although safe, will hamper progress quite unnecessarily.

The opposite effect, using too many different locks, will result in an unsafe solution. Two students observed that method *takePassengers* in class *Taxi* influenced the attribute *numberOfPassengersWaiting* in a different class, and reasoned as follows:

Student A: *Well, it's time to start synchronizing some methods. This method takePassengers will be called by different threads.*

Student B: *We solved that by making it synchronized.*

Student A: *Now that we've done that, I wonder whether we should also synchronize getNumberOfPassengersWaiting.*

Student B: *But that would synchronize all the code within that method.*

Student A: *Right, so we need only synchronize the top level.*

Student B: *It might still be the case that we synchronize too much code.*

Student A: *In principle we have only synchronized takePassengers.*

Here the students seem aware of the danger of blocking too much, but do not realize that by marking the method **synchronized** without explicitly mentioning a lock, they are implicitly using a separate lock for each (taxi) thread; thus different taxis are not prevented from accessing *numberOfPassengersWaiting* simultaneously.

3.4 Correctness

We found two issues concerning correctness, namely students often assume a program is correct once it produces some output, and students are often not aware that because of non-determinism different correct program outputs may result.

First, instead of comparing program output to the requirements in the exercise description, students often assume a program is correct once it produces some output. One example of this way of thinking is shown in the following fragment: students want to ask a question and, in the meantime, execute the program developed so far. They see that many taxis are created, but all these taxis do not transport any passenger. The teaching assistant appears. The students show the output and ask the assistant: 'Is this output correct?'.

In this fragment, the students are considering obviously incorrect program behavior as 'possibly correct', although as long as there are passengers to transport, taxis should not be empty.

Another example fragment is the following:

Student A: *While not station is closed, ... well, ...But, in that case he should close. The train will close the station ... Look at this!*

Student B: *All passengers have been transported.*

Student A: *I think it is ok so. We finished the job. We have to write our report.*

These students are assuming the program is correct because it produces some superficially correct output. But they did not inspect the output precisely, i.e., were all taxis always transporting the maximum number of passengers possible. Neither did they check on anomalous behavior such as halfway vanishing or materializing passengers as might occur due to race conditions.

These findings, i.e. that students are satisfied with a program that compiles and produces some output, correspond with Kolikant [12].

Second, students are not aware that, because of non-determinism in the scheduling of concurrent program parts, different correct outputs may result when running a program several times with the same input. The following fragment shows an example:

Student: *1 2 3 4 1 2. Hey! How is that possible? That is strange. Why didn't it do that a moment ago?*

Here, the student is surprised that the concurrent program produces an output different from the previous execution. This indicates a lack of understanding of concurrent execution and/or an inability to apply the conceptual knowledge concerning non-determinism into real situations. They are looking for *the* correct output, but several correct outputs are often possible.

Apparently, students do not have a notion of what correct behavior is in the multi-threaded application area, and when a program can be assumed to be correct. Observing that there is output is not enough. Instead, the output must be analyzed in relation to the informal specification of the program's behavior as (should be) described in the assignment. It should be noticed that proving a concurrent program is correct is beyond the scope of a freshman course on object-oriented programming. But analyzing and checking on some behavior properties is certainly possible.

In the think-aloud sessions we did not find any fragment where students talked in terms of correctness.

Looking again at our teaching material and exercises, we find that these do not discuss how to draw up informal specifications of program behavior and how to use these to reason about the correctness of a concurrent program. This corresponds to Goetz's observation [8]: '... we often don't write (adequate) specifications ...'. As a result, it is actually impossible to know whether a program is correct. Drawing up informal specifications of expected program behavior and using them in reasoning whether a concurrent program is correct should be clarified at the appropriate level for an introductory course.

This indicates the need for identifying properties that the behavior should have. For example, in case of our train-taxi-passenger exercise, one could require that the number of trains, taxis, and passengers should remain constant during program execution. Or, depending on the goal of the simulation, instead of simply counting passengers at the start and end of a program run, one could check on the passengers' IDs, because due to race conditions, one passenger could disappear while another is created.

In the case of simulation, a natural candidate for a condition on behavior would be an invariant property. Invariants and other conditions could be provided with the exercise, or, for more advanced students, be required to be constructed by the student from the exercise description.

The current Steps Plan has a step, Reflection, that concerns the correctness of the final multi-threaded program, but this step incites

only to reflect on the analysis and design decisions made, not how how this analysis should be performed. As a result, the Step Plan should be extended in two ways:
First, an activity analysing/drawing up correctness properties should be added in terms of, e.g., invariants. This can be done at the start of implementing the program, but properties can be further refined when steps towards the final version are being made. Note that formalization of the conditions is beyond the scope of an introductory course: the approach by Felleisen is an inspiring example of how to incorporate explicit condition at this level [7].
Second, at several points in the step plan, the student should be guided to reflect on how the program satisfies such properties. In particular the step Reflection should be extended with means how to use the properties to determine whether the final program behaves correctly.

Additionally, despite the explanation of the concept of non-determinism as part of the Steps Plan, it turns out that students are not aware of this concept in practical situations, i.e., that a correct concurrent program can produce several different outputs due to non-determinism in the scheduling of the concurrent program parts. This impacts the notion of correct behavior: students should be aware which differences in output over different runs are an indication of incorrect behavior and which are due to acceptable non-determinism induced by concurrency. Next to the explanation and the use of this concept in complete exercises, the application of this concept should be practiced in smaller exercises, so called part-tasks [18], to train this routine aspect up to a higher level of understanding and automation.

3.5 Anthropomorphism

Anthropomorphism is the attribution of human characteristics to things that are not human. Analyzing the think-aloud sessions, we observed, as a fortunate by-catch, that students often use an anthropomorphic style in talking about objects and their behavior. For example:

> **Student:** *Voilà, out of the box. O yes,* he *is going to create taxis first, then there arrives suddenly a train with 85 passengers. The taxis take all these passengers ...*

Here, *he* points to the part of the program responsible for creating taxi objects.

In literature, anthropomorphism is considered as an innate tendency of human psychology [14]. It is known that anthropomorphism can help in thinking and talking about difficult and unfamiliar topics [10]. Nevertheless, many scientists disapprove the use of anthropomorphism: It is considered as a symptom of professional immaturity, i.e. it disguises that one's knowledge is insufficient [4, 5, 10, 16].

In our recorded sessions, we found passages where the use of anthropomorphism is just a style of talking that does not seem to hinder understanding. The passage above is an example of this. Anthropomorphism may hinder understanding, however, when programming objects are attributed human characteristics that can not and will not be implemented. Then the student is constructing a programming model that does not comply with reality and will hamper his understanding and programming abilities.

The following fragment is an interesting example where this case may be applicable.

> **Student:** *Yes, that is wrong. There should be something ... indeed. [...] Can you say that after a number of taxi rides he simply stops? Or, that after a long time of waiting, in case he has waited ten times and there still aren't passengers there, he goes home?*

In this example, a final state of a taxi is observed by a prolonged inaction of that taxi. Where a human would be tired to wait, it is obviously not the reason for a thread to stop. How mistaken this anthropomorphism is, remains the question, though. Remember that we have a simulation here where the student's program is intended to simulate human behavior (human passengers, taxis driven by humans). From the exercise description, it is not so clear what exactly is simulated. Is it just a logistic procedure, implemented by the taxi drivers and not intended to be influenced by human considerations such as tiredness or are we simulating human behavior here?

In our opinion, anthropomorphic thinking can be helpful to explore a new subject or to get to grips with a new problem. It can, for instance, be used in an OO simulation game, resembling the well-known CRC modeling technique [9], performed by students to explore the way objects behave and interact with each other. However, if this is carried out too far, there is a risk that students will set themselves on the wrong track. In the case of simulations where humans are involved, extra care has to be taken to make clear to the students what behavior is to be simulated, e.g., behavior of an inanimate algorithm, possibly implemented by humans, or decisions made by humans. This may help the student to not confuse the innocent anthropomorphisms and the helpful anthropomorphisms of the phase of getting to grips with the problem and the possibly detrimental anthropomorphisms regarding, e.g., programming objects.

3.6 Unrealistic examples

Many instructors have found it difficult to construct high quality programming assignments. Several authors have looked at this issue and proposed different criteria which should be taken into account when developing programming assignments. For instance, Falkner and Palmer [6] state that a good programming assignment should be based on a real-world problem and allow the students to generate a realistic solution.

In our opinion, the three main reasons to introduce concurrency into a program, are improvement of efficiency and processor utilization, avoidance of nonresponsiveness, and simulation of inherently parallel processes in the real world.

It is relatively simple to give examples and assignments for the first two cases that are both practically relevant and relate to a realistic problem. If we look at our step-by-step plan, we find that provided procedural guidance mainly focuses on the third category of applications: the simulations. In practice, simulations are used to determine properties of processes by modeling these processes as a computer program and then analyzing the results of this program. However, often it is not necessary to use concurrency in such a model. On the contrary, in many cases a sequential implementation appears to work much better. This also seems to be the case

in the taxi assignment used in this study. The simulation in this assignment has no realistic goal. In fact, the taxi problem is only used as a metaphor to visualize the abstract concurrency concepts. The need to make explicit use of concurrency for this purpose is lacking, as a result of which students are often unable to discover the constructs intended by the lecturers themselves. This could explain why students stubbornly stick to the sequential solution that was provided as a starting point.

4 CONCLUSIONS AND FUTURE WORK

Using the Steps Plan in actual education has confirmed our initial intuition that something along these lines would be useful as a guide providing scaffolding for those students that have no experience how to attack concurrency exercises. The observation results do not show a reason to change the choice or the order of the macro-steps in the Steps Plan.

Students have to deal with amount of concurrency, correctness, race conditions and synchronization. Problems observed pose the question how to help the student make these things concrete and explicit, and how to provide procedural help. A general observation is, that this cannot be provided by the Step Plan alone: exercises need to be crafted that provide detailed explicit scaffolding at the beginning and gently evolve to posing more demands on the independent thinking of the student.

Analysis of the difficulties students ran into points the way to improvements.

- Providing a sequential solution to a problem for which a concurrent solution is sought is not helpful. If we want to assist students to get started, a framework of domain classes is preferable.
- Exercises should specify clearly which activities are supposed to proceed concurrently. If this decision is left to the (more advanced) students, they ought to provide explicit reasoning leading to their choice.
- Task definition (via the active class pattern) and task creation (using threads) have to be taken into account separately in the Steps Plan, avoiding any confusion of these activities.
- Detailed information in terms of micro-steps on how to perform the macro-steps in the Steps Plan should be available on demand: the role of adaptive hypertext needs to be investigated.
- The step Reflection of the Steps Plan should be extended with means of how to check the output for correctness. This requires an analysis to determine which correctness properties would be visible from the output, taking into account the non-deterministic character of concurrent programs.
- Examples and assignments should be both practically relevant and relate to a realistic problem. This is more often the case with problems addressing efficiency and responsiveness than with simulation of real-world parallel processes. Exercises should be adapted to this insight.
- Anthropomorphic thinking turned out to be helpful when thinking about a problem domain, but turned out to be dangerous when applied to, i.e., programming objects. We should make our students and teachers aware of this.

Two aspects of our research that have not yet been discussed are the following.

First, the detailed observation of students (through screencasts and audio recordings) and the comprehensive analysis of the generated data leads to valuable knowledge about the way students learn to program. This information remains invisible if we solely base conclusions about the learning behavior of students on the assessment of delivered products and disregard the way in which students proceeded. In our specific case of the Steps Plan, this has yielded a number of concrete points for improvement that would otherwise have gone unnoticed.

Second, while working on the assignment, students sometimes seemed not to make any progress at all. We observed that students often had difficulty with concepts that had already been introduced and that the teachers assumed students would master. Due to the cognitive load [17] that this creates, the entire development process seems to be stagnating. We think that for learning new concepts, students must have sufficient time and opportunity to practice. To create room for this, teachers should consider reducing the number of subjects in a course.

ACKNOWLEDGEMENT

This project is part of the research group Didactics of Informatics and is carried out in collaboration with E. Barendsen. Participating universities in this project are the Radboud University, Eindhoven University of Technology and Open University of the Netherlands.

REFERENCES

[1] A. Bijlsma, C. Huizing, R. Kuiper, H. J. M. Passier, H. J. Pootjes, and J. E. W. Smetsers. A structured design methodology for concurrent programming. In *Proceedings of the 6th Computer Science Education Research Conference*, CSERC '17, pages 1–9, New York, NY, USA, 2017. ACM.
[2] Paul De Bra. Adaptive educational hypermedia on the web. *Commun. ACM*, 45(5):60–61, May 2002.
[3] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 26(1):21–22, January 1983.
[4] E.W. Dijkstra. How do we tell truths that might hurt, 1975. EWD 498.
[5] E.W. Dijkstra. On anthropomorphism in science, 1985. EWD 936.
[6] Katrina Falkner and Edward Palmer. Developing authentic problem solving skills in introductory computing classes. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 4–8, New York, NY, USA, 2009. ACM.
[7] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to Design Programs: An Introduction to Programming and Computing*. MIT Press, Cambridge, MA, USA, 2001.
[8] Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, Doug Lea, and David Holmes. *Java Concurrency in Practice*. Addison-Wesley, Upper Saddle River, New Jersey, 2006.
[9] Lars Hvam, Jesper Riis, and Benjamin Loer Hansen. Crc cards for product modelling. *Computers in Industry*, 50(1):57 – 70, 2003.
[10] M. Kallery and D. Psillos. Anthropomorphism and animism in early years science: Why teachers use them, how they conceptualise them and what are their views on their use. *Research in Science Education*, 34(3):291–311, Sep 2004.
[11] 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.
[12] Yifat Ben-David Kolikant. Students' alternative standards for correctness. In *Proceedings of the First International Workshop on Computing Education Research*, ICER '05, pages 37–43, New York, NY, USA, 2005. ACM.
[13] Gary Lewandowski, Dennis J. Bouvier, Robert McCartney, Kate Sanders, and Beth Simon. Commonsense computing (episode 3): Concurrency and concert tickets. In *Proceedings of the Third International Workshop on Computing Education Research*, ICER '07, pages 133–144, New York, NY, USA, 2007. ACM.
[14] K.J. McCaffree. Is magical thinking good?. *Skeptic*, 19(1):59 – 61, 2014.
[15] T. Plomp. Educational design research: An introduction. In T. Plomp and N. Nieveen, editors, *Educational design research*, pages 10–51. Enschede: SLO,

2013.
[16] L. Vaillant. Anthropomorphism gone wrong: Poor motivating example for oop, 2015.
[17] Jeroen J. G. van Merriënboer and John Sweller. Cognitive load theory and complex learning: Recent developments and future directions. *Educational Psychology Review*, 17(2):147–177, Jun 2005.
[18] Jeroen J.G. van Merriënboer, Richard E. Clark, and Marcel B.M. De Croock. Blueprints for complex learning: The 4c/id-model. *Educational Technology Research and Development*, 50(2):39–61, 2002.

A class project to prepare software engineering students for their capstone projects

ABSTRACT

We discuss the design of a class project which we have introduced to improve our Software Engineering course presented on the third-year graduate level at our institution. For this project, the whole class collaborate to design and implement a single, reasonably large software system. We believe that the class project has the potential to provide an intensive learning experience for our students and may have several educational benefits.

We investigate the impact of the class project on student achievement and project success. We gauge the impact of the class project by analysing differences the academic performance of the students in the course, analysing the differences in assessment marks assigned to projects and by observing variations in the source code of the software systems delivered by the students through the application of popular software metrics.

Although the results are inconclusive, we feel the class project provides a unique opportunity for students to get hands-on experience in the development of real-world software for industry.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Software and its engineering** → *Software maintenance tools*; *Empirical software validation*; Programming teams.

KEYWORDS

software engineering, software metrics, capstone project class project, teaching software development

ACM Reference Format:

. 2019. A class project to prepare software engineering students for their capstone projects. In *CSERC '19: Computer Science Education Research Conference, November 18–20, 2019, Larneca, Cyprus.* ACM, New York, NY, USA, 12 pages. https://doi.org/xxxx

1 INTRODUCTION

In our software engineering (SE) module, we usually have between 80 and 100 students who are required to complete their capstone projects. For these projects, students are expected to design and implement a software system for an industry partner over a period of approximately five months. During this time, the students design and implement relatively large authentic systems to solve open-ended problems. To prepare our students for their capstone projects, we involve them in a class project during the first eight to ten weeks of the module. This class project is described in Section 2. We contend that the class project helps the students to be more successful when completing their capstone projects and ultimately to more in-depth learning of SE concepts and competencies.

In this paper, we describe our class project. We briefly state why it was introduced and report how it is conducted in its current state. We introduce software metrics we used in our investigation whether the class project has the assumed effect.

In order to enable us to observe the effect of the class project, we omitted the class project when we presented the SE module in 2018. In our investigation we observe the difference in achievement of the students who participated in the module in 2018 with those who participated in the module in the previous year when a successful class project was conducted. Firstly, we analyse the final marks that were achieved by the individual students in the module at the end of each of the two years. We compare the results of the years with one another to determine if the overall achievement of the students is better during the year in which the class project was included. Secondly, we analyse the marks that were assigned to these teams during continuous assessment of their progress during of each of the two years. We compare the results of the years with one another to determine if the students fared worse during the year in which the class project was omitted. Finally, we apply software metrics to assess the student code gathered from public git repositories that were used by student teams when they worked on their projects. We compare the outcome of the assessment of their code of the years with one another to determine if there is any decline in the quality of the code written by the students when they had not done the class project.

We conclude with a brief summary of the results of our investigations and discuss some implications thereof.

2 CLASS PROJECT

We observed that our students are likely to make many bad decisions while working on their capstone projects. One of the consequences of such bad decisions are that they learn more about how *not* to do things rather than gaining experience in how to do things right. We attribute some of the bad decisions to the inexperience and ignorance of the students. To address the problem, we applied a lesson learnt from Desai et al. [4]. They introduced a class project during their fifth semester to address shortcomings commonly found in capstone projects such as lack of teamwork, bad planning, weak requirements management, poor design and documentation, excessive code bulk, late integration, and weak testing. Likewise we inserted an intense class project before students commence with their capstone projects. The class project is an opportunity for the students to gain the experience in the application of SE concepts, practices, tools and the social skills required for the completion of their capstone project. The aim in doing so

CSERC '19, November 18–20, 2019, Larneca, Cyprus

ACM ISBN xxx-x-xxxx-xxxx-x/xx/xx…$15.00
https://doi.org/xxxx

is to reduce the problems and increase the value of a subsequent capstone project.

The class project is conducted before the students commence with their capstone projects. The main aim of the class project is to teach the software development life cycle in a hands-on manner and at the same time to ensure that students enter their capstone projects with more experience.

The project had evolved over the years. Since having introduced the class project in 2011 [11–13] we applied continuous action research to analyse and improve the process with each presentation of the course[10]. In its current state, this project involves the whole class to collaborate on the design and implementation of a single software system. This is done by conducting three consecutive micro-projects where lecturer assigned teams are shuffled for each micro-project. The tasks for the micro-projects are:

(1) Requirements specification.
(2) Design and deployment planning.
(3) Implementation and testing.

For the first two micro-projects the teams have the same task and should produce the required documentation while in the last micro-project the teams are each assigned a different module to implement. We require test driven development.

When implementing the system students are exposed to large scale development and the difficulties posed by the integration of the inter-dependant modules developed by different teams. They are guided to use dependency injection, unit testing with mock objects and integration testing. It is also expected that they use build tools which support dependency management, integrated testing and artefact distribution. The practical experience of applying these practices in software engineering, is required here to foster an appreciation of their benefits.

The system designed and implemented during the class project is chosen to be aligned with the latest technology trends in order to create a situation in which the students have to learn the application of contemporary technologies. The project should be suitable for its purpose. It should apply cutting edge technologies, yet should not be technically too challenging. The system should lend itself to modular design where each team can work independently on their assigned module and provide the exposure to the challenges associated with integration allowing for piece-wise integration.

We emphasise various aspects of documentation in order to guide the students to understand the need for proper documentation. Mitra [9] found that enforcing high level of precision UML modelling supporting direct mappings to code demonstrated the relevance of models to students and facilitated maintainability of developed systems. Similarly, we require technology-neutral component based modelling facilitating mapping of component contract specifications onto interfaces and unit tests and data structure specifications onto entities and activity diagrams onto method bodies.

We assign new teams for each micro-project in order to maximise the number of different people each student is required to work with during the class project. This creates an opportunity for the students to meet the people with whom they may team up to for their capstone projects. They are also exposed to a variety of personalities. During this time students are allowed to make mistakes without having to suffer the consequences in the longer term because the teams are short-lived.

After completing the task, we expect the students to participate in structured reflection on their newly acquired skills. We are aware that students are less inclined to judge each other openly and passing on critique to instructors is often felt as betrayal [1]. For this reason the feedback is confidential. To encourage students to be truthful, marks are awarded to individuals for the quality of their feedback. General guidelines to the individuals supports the identification of soft skills that could be improved in the next micro-project. It is assumed that the lessons learnt in each micro-project is acted upon in the next micro-project. The timeliness of processing the feedback is essential as the usefulness and impact thereof fades over time.

To maximise the actual learning that results from the opportunities created this way, we provide support. We inform them about the possible difficulties they may experience during their forthcoming micro-projects and give them guidelines on how to deal with expected difficult situations. We also encourage the students to seek help when needed.

Because students are regularly re-assigned to new teams during the class project, it allows students to acquaint themselves with most of the other students in the class. Furthermore, it creates situations where the students can learn teamwork skills [7, 14].

The class project provide a guided learning experience in which the students can master the software development life cycle and learn how to use software tools to support various aspects of the software development process. The project also supports the development of employability skills such as getting along with people of different genders, races, religions or political persuasions; defining one's role in a team; identifying the strengths of the other team members; and being able to lead a team effectively [6].

3 SOFTWARE METRICS

Quantitative measurements are essential in all sciences. Computer Science is no exception. There is a continuous effort by computer science practitioners to design quantitative measures to validate the quality of software. The goal is to obtain objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance, testing, software debugging and software performance optimisation.

The metrics that were chosen to assess the student code are Cyclomatic Complexity, Halstead Metrics and Maintainability Index [3]. We used freely available software tools to analyse the code.

3.1 McCabe's Cyclomatic Complexity

Developed by McCabe [8], this metric is used to indicate the complexity of a program, and measures the number of linearly independent paths through the source-code of a program.

Take for example the following section of code:

```
A = 10
IF (B > C) THEN
     A = B
ELSE
     A = C
ENDIF

Print A
Print B
Print C
```

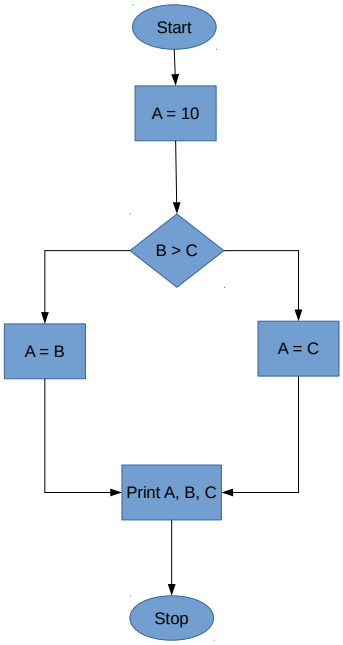


Figure 1: Control Flow Graph

The graph depicted in Figure 1 shows 7 shapes (nodes), 7 lines (edges) and 1 exit point. Mathematically, Cyclomatic Complexity is defined as:

$M = E - N + 2P$

- E = number of edges in the control flow graph
- N = number of nodes in the control flow graph
- P = number of nodes that have exit points

3.2 Halstead Metrics

Halstead [5] introduced software metrics as part of his treatise on establishing an empirical science of software development. He observed that language agnostic metrics should reflect the implementation of an algorithm in different languages, but be independent of their execution on a specific platform. His goal was to identify measurable properties of software and the relations between them. The following base measures can be collected:

- η_1 = number of distinct operators
- η_2 = number of distinct operands
- N_1 = total number of operators
- N_2 = total number of operands
- LOC = total lines of code (without comments)

In this paper, the following Halstead metrics are used:

3.2.1 Length. $N = N_1 + N_2$

This metric calculates the total number of operator occurrences and the total number of operand occurrences.

3.2.2 Vocabulary. $\eta = \eta_1 + \eta_2$

The total number of unique operator and unique operand occurrences.

3.2.3 Volume. $V = N \times log_2 \eta$

This metric calculates the relative program size.

3.2.4 Difficulty. $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$

As the volume of the implementation of a program increases, the program level decreases and the difficulty increases. Thus, programming practices such as redundant usage of operands, or the failure to use higher-level control constructs will tend to increase the volume as well as the difficulty.

3.2.5 Effort. $E = D \times V$

Measures the amount of mental activity needed to translate the existing algorithm into an implementation in the specified programming language.

3.3 Maintainability Index

$MI = 171 - 5.2 \times ln(V) - 0.23 \times M - 16.2 \times ln(LOC)$

Maintainability Index is a software metric which measures how maintainable (easy to support and change) the source code is. We use the improved four-metric model proposed by Welker et al. [15] to calculate the Maintainability Index of the source-code written by our students. This maintainability index is calculated as a factored formula consisting of Lines Of Code, Cyclomatic Complexity and Halstead volume.

4 SOURCE-CODE ANALYSIS TOOLS

The projects that were analysed were written in the Java, JavaScript or Python programming languages. In order to analyse the source-code of these projects, three open-source / free software tools (one

tool per programming language) were selected to perform the analysis. These tools were selected based on being freely available and their capabilities to determine the metrics discussed in Section 3. Table 4 shows the selected tools that were used to analyse the source-code of the student projects. The tools were installed and executed on a Linux Mint operating system.

4.1 Java and Source Meter

After Source Meter was installed, it was executed with the command shown in Figure 2 on a project with Java source-code. This command produces a number of result output files, among which is a comma-delimited file containing all the metrics mentioned in Section 3 (among others), calculated per method.

4.2 Python and Radon

After Radon was installed, it was executed to calculate the various metrics mentioned in 3 by executing the commands listed in Figure 3.

The command labeled a) is used to calculate Cyclomatic Complexity on a project with Python source code. The command calculates the Cyclomatic Complexity per function, method and class and pipes it to a text file.

The Maintainability Index is calculated with the command labeled b). The command calculates the Maintainability Index per file and pipes it to a text file.

The command labeled c) is executed to calculate the Halstead metrics. The command calculates the Halstead metrics per method and pipes it to a text file.

4.3 JavaScript and Plato

After Plato was installed, it was executed with the command shown in Figure 4 on a project with JavaScript source-code. This command calculates the metrics referred to in Section 3 per method and saves it in JSON-formatted result files in the specified folder.

5 IMPACT ON INDIVIDUAL PERFORMANCE

We assume that the students who have participated in the class project have a better understanding of the software engineering life cycle and have gained practical experience in SE technical skills as well as in employability skills. We further contend that those who enter their capstone project after having completed this preparatory project is more likely to succeed in their capstone projects. Ultimately these students have a wider frame of reference when learning during their capstone projects resulting in improved learning outcomes. The following hypothesis is formulated to assert this assumption:

> The overall achievement of students who had participated in a class project is higher than the overall achievement of students who had not participated in a class project.

To test this hypothesis we use the final marks that were achieved by the individual students in the module at the end of the year in 2017 and 2018. The students in the 2017 cohort had participated in a class project while the 2018 did not. We compare the results of the years with one another to determine if the performance of the 2017 cohort is better than the 2018 cohort.

In 2017 nine of the 103 students (8.7%) who started the module, failed or dropped out. In 2018 ten of the 81 students (12.3%) who started the module discontinued or failed. This increase in failure in the year where the class project was omitted, is an indication that the individual performance of the students is better when they participate in the class project as more of them successfully complete the project.

We further analyse the final marks of the students who passed the module. The students in the two cohorts are independent of one another, thus our data is unpaired. Our sample size is relatively small (93 in 2017 and 72 in 2018). Furthermore, the students in this course are not representative of the population, consequently their marks does not necessarily follow a Gaussian distribution. Based on these facts we decided to use a non-parametric test to test our hypothesis. Although we expected that the achievement of the 2017 cohort would be higher, we decided to performed the two-tailed t-test rather than performing a one-tailed test to provide for the probability that the class project may have had a negative effect on the performance of the students. Table 2 shows the group statistics of the final marks of individual students in the two groups.

The mean of the 2017 cohort is higher than the mean of the 2018 cohort. This is in line with our expectation. The p-value of Levene's Test for Equality of Variances is 0.345. This value is more than 0.05, therefore we know that the result has equal variance. Table 3 shows the result of the two-tailed t-test for equality of means assuming equal variances. The 95% Confidence interval of the difference is [3.257, 10.488]. Since the p value is practically 0, we reject the null hypothesis (of no difference) and conclude that there is a statistically significant difference of 5.108 between the mean marks of the 2017 cohort and the 2018 cohort at a 5% significance level.

We can conclude that the achievement of the students who participated in a class project before commencing their capstone project is statistically better than those who did not. This confirms that the class project had a positive effect on the achievement of the students in this case.

6 IMPACT ON TEAM SUCCESS

Apart from providing a broader learning experience for the students, culminating in better overall learning for individuals, we assume that the students who have participated in the class project are better prepared when they enter their capstone project. This should enhance the likelihood that they would succeed in their capstone projects.

The following hypothesis is formulated to assert this assumption:

> The projects done by teams consisting of members who had participated in a class project is more successful than the projects of teams who have members who had not participated in a class project.

Language	Tool Name	URL
Java	Source Meter	https://www.sourcemeter.com
Python	Radon	https://pypi.org/project/radon
JavaScript	Plato	https://github.com/es-analysis/plato

Table 1: Tools used in the analysis of the source-code

```
./SourceMeterJava -resultsDir=<dir_to_save_results> -projectName=<P1...Pn>
-runAndroidHunter=false -runVulnerabilityHunter=false -runFaultHunter=false
-runRTEHunter=false -runDCF=false -projectBaseDir=<root_of_project_to_analyze>
-runFB=false -runPMD=false
```

Figure 2: Command used to invoke Source Meter

```
a)  radon cc --show-complexity root_of_project_to_analyse > result_text_file
b)  radon mi --show root_of_project_to_analyse > result_text_file
c)  radon hal root_of_project_to_analyse > result_text_file
```

Figure 3: Commands used to invoke Radon

```
plato -r -d <dir_to_save_results> <root_of_project_to_analyse>
```

Figure 4: Command used to invoke Plato

year	N	Mean	Std Dev	Std.Err Mean
2017	93	72.25	8.744	0.907
2018	72	67.14	7.908	0.932

Table 2: Group Statistics - individual achievement

t	df	Sig (2-tailed)	Mean Difference	Std.Err Difference
3.879	163	0.000	5.108	1.317

Table 3: T-test for equality of means

We deem the marks assigned to project teams when assessing the projects at regular intervals as a good criterium for the success of the projects. Figure 5 shows the marks of the teams. The teams are numbered in order of the marks with the project with the highest mark is assigned the number 1, the number 2 is assigned to project with the second highest mark, etc. One can observe that the projects of 2018, when compared with the project in 2017 which is ranked in the same place, is consistently weaker in terms of these marks.

To test our hypothesis, we analyse the final marks that were achieved by the capstone teams in the module during the two years under investigation. These marks are compiled using the marks that where awarded to the teams at a variety of assessments during the course of the project. We evaluate the implementation of the projects in a series of demonstrations where the teaching assistants assessed their projects using rubrics that include the scope of the project, the professional conduct of the team, compliance with their development plan, as well as the quality of the code in terms of readability, re-usability and efficiency.

We compare the results of the years with one another to determine if the performance of the 2017 teams (who had a class project before commencing with their capstone projects) is better than the 2018 teams (who started with their capstone projects without participating in a class project).

The teams in the two years are independent of one another, thus our data is unpaired. We decided to use a nonparametric test to test our hypothesis because our sample size is very small (20 in 2017 and 16 in 2018). Although we expected that the achievement of the 2017 teams would be higher, we decided to performed the two-tailed t-test rather than performing a one-tailed test to provide for the probability that the class project may have had a negative effect on the performance of the teams. Table 4 shows the group statistics of the final marks of the teams in the two years.

year	N	Mean	Std Dev	Std.Err Mean
2017	20	71.76	11.278	2.522
2018	16	67.94	16.474	4.119

Table 4: Group Statistics - team achievement

In line with expectation it can be observed that the mean of the 2017 teams is higher than the mean of the 2018 teams. The p-value of Levene's Test for Equality of Variances is 0.391. This value is more than 0.05, therefore we know that the result has equal variance. Table 5 shows the result of the two-tailed t-test for equality of means of the team marks assuming equal variances. The 95% Confidence interval of the difference is [-5.703, 13.128]. Since the p value of

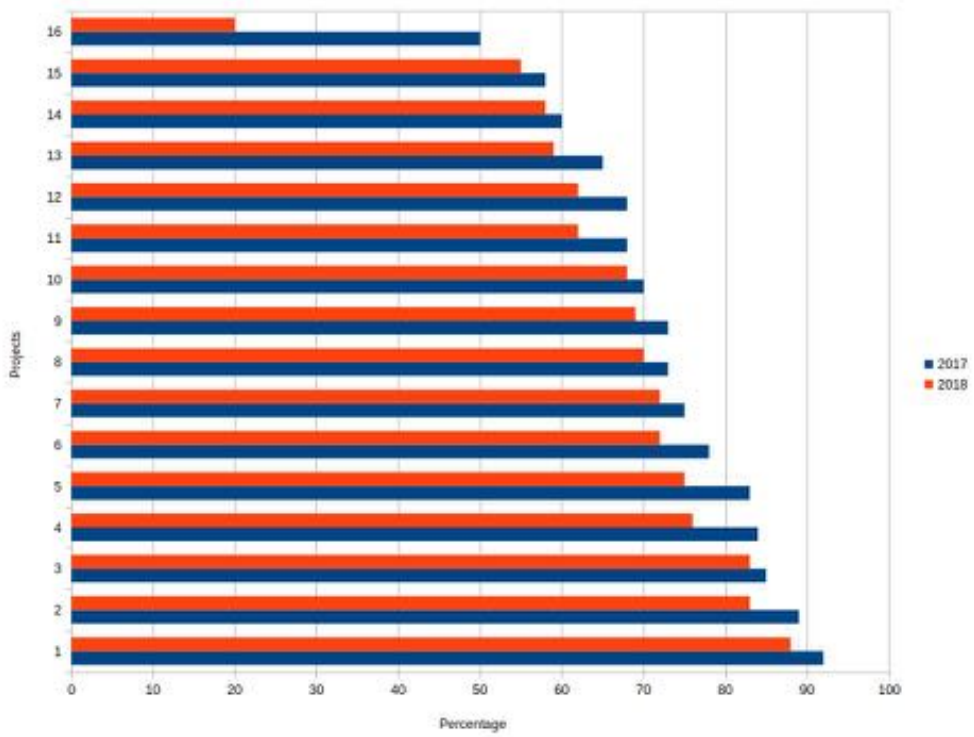


Figure 5: Final Project Marks

this t-test is 0.429 (>0.05), we cannot reject the null hypothesis. We conclude that the difference in marks assigned to the teams during assessment in these years does not provide evidence that the teams in 2018 were less successful than the teams in 2017 despite the visual impression created by the graph in Figure 5.

t	df	Sig (2-tailed)	Mean Difference	Std.Err Difference
0.801	34	0.429	3.713	4.633

Table 5: T-test - team achievement

We conclude that the achievement of the students who participated in a class project before commencing their capstone project is statistically better than those who did not.

Possible reasons for our failure to confirm our hypothesis include the following:

(1) The marks allocated to the teams cover a wide range of criteria. Certain aspects that were weighted high may have skewed the marks.
(2) The marks allocated to the teams were assigned at different points in time. There are teams who were very unsuccessful in the beginning, yet succeeded towards the end who ended with fairly low marks overall because of their initial failures. For some other teams the converse is true.
(3) Most of the marks were assigned through the application of subjective criteria

7 OBJECTIVE COMPARISON OF TEAM PROJECTS

In order to address some of the reasons why we could not confirm that the participation in the class project has a positive effect the work done by the capstone project teams, we decided to investigate the quality of the projects based on the objective measures discussed in Section 3.

Figures 6 to 12 show the source-code metrics that were calculated for the projects done in 2017 and 2018 per programming language. Where projects consist of multiple source-code files, the average metric value of all methods in all the files is taken. In most graphs the scale used to show the calculated values for source-code written in javascript is significantly larger than the scale used to show these values for source-code written in Java and Python.

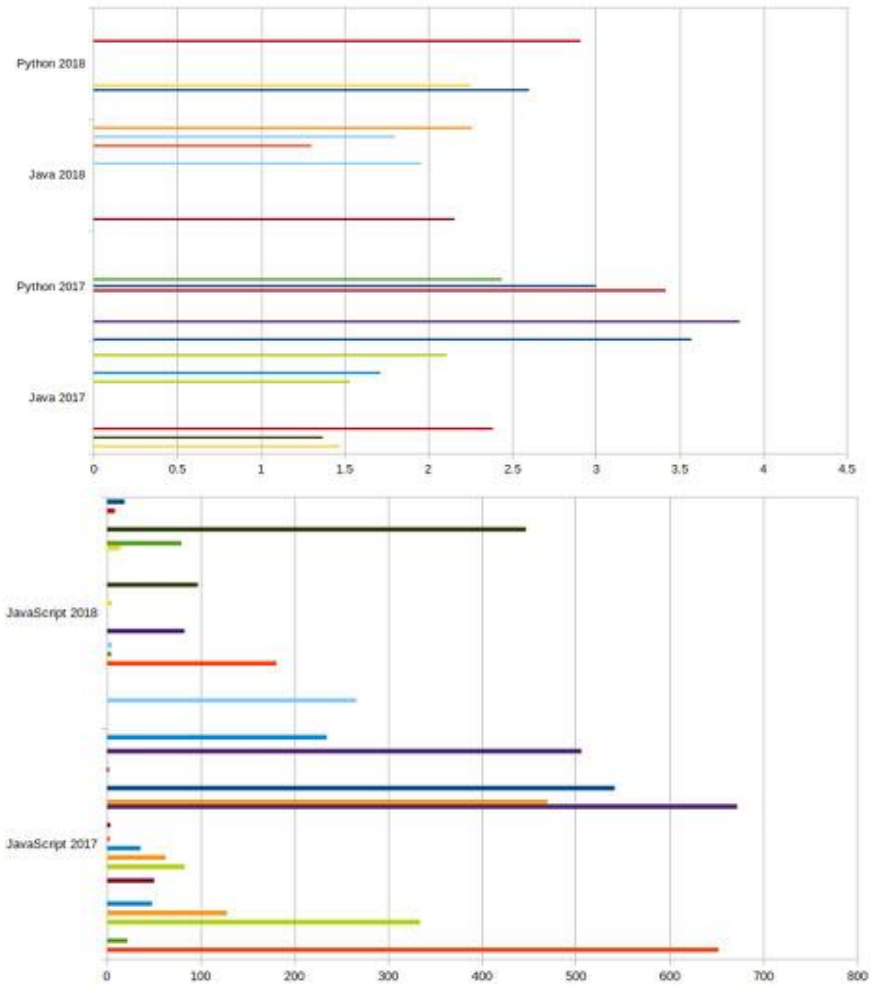


Figure 6: Average Cyclomatic Complexity

It can be observed in these figures that the use of javascript is prominent. In most projects the students prefer to use javascript. In some cases they used javascript in combination with another language.

Figure 6 shows that the projects delivered in 2017 has higher complexity. This is prominent by comparing the complexity of the projects between these years for the javascript source-code as the complexity of source-code using other languages are far less complex than the javascript source-code.

Figure 7 shows more projects of higher difficulty in 2017. This is true when comparing the difficulty of the projects between these years for the javascript source-code. This trend is true with one exception of a project including Python source-code in 2018 which has a far higher difficult value than the other Python implementations in our sample yet significantly less difficult than most javascript implementations in 2017. It thus seems if the students who participated in the class project, were generally able to deliver more difficult projects than those who did not participate in a class project. This observation is confirmed in Figure 8 which illustrates that the javascript source-code written in 2017 required more effort to write than the javascript source-code written in 2018. Figure 8 correlates with Figure 7. This is expected since higher difficulty implies higher effort to implement.

Figures 9 and 10 shows respectively that Halstead length and Halstead vocabulary of the source-code of projects that were written in 2017 is generally larger than those of the projects written in 2018 although there were some projects written in javascript in 2018

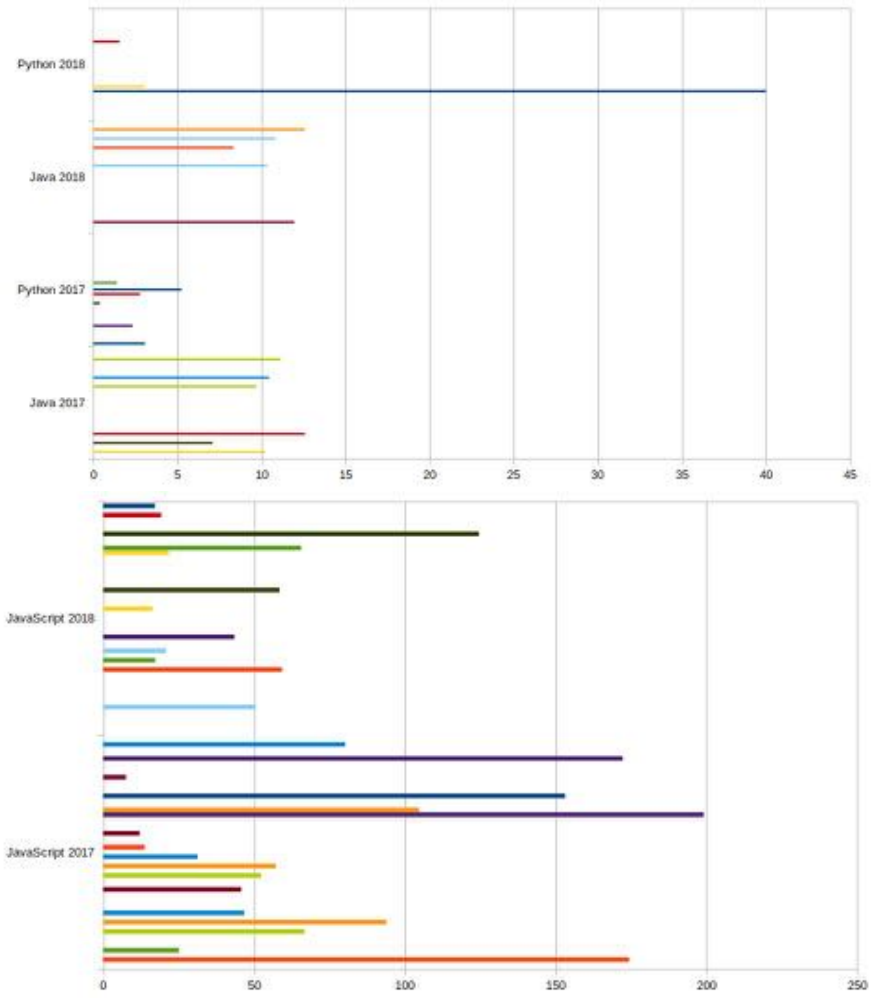


Figure 7: Average Halstead Difficulty

that had Halstead lengths and vocabularies exceeding the Halstead length and vocabulary of many of the projects written in 2017.

Figure 11 correlates with the previous figures since there are more unique operators and operands when programs are larger and more complex. The source-code written in Java and Python also have higher Halstead volume in 2017 when compared with those written in 2018. Figure 11 confirms larger relative program size in 2017.

It seems if the solutions delivered by the student teams who were exposed to the class project are more ambitious and comprehensive than the ones delivered by students who went straight into their capstone project. This is counter-intuitive since those who went straight into the capstone project effectively had more time to work on the project and yet delivered somewhat weaker solutions.

Figure 12 shows the average Maintainability Index (MI) of the source-code of the projects under investigation. Previous empirical testing of the MI model which we used in our analysis suggests two cut-off points. MI values below 65 indicates low maintainability, and values above 85 signifies high maintainability. The range of 65 to 85, inclusive, indicates moderate maintainability [2, 15].

When looking at the MI values for the javascript source-code written by our students it is evident that the maintainability of the projects across the years do not differ from one another. Most of the projects in both years fall in the moderate maintainability category while a few projects can be classified as low in maintainability.

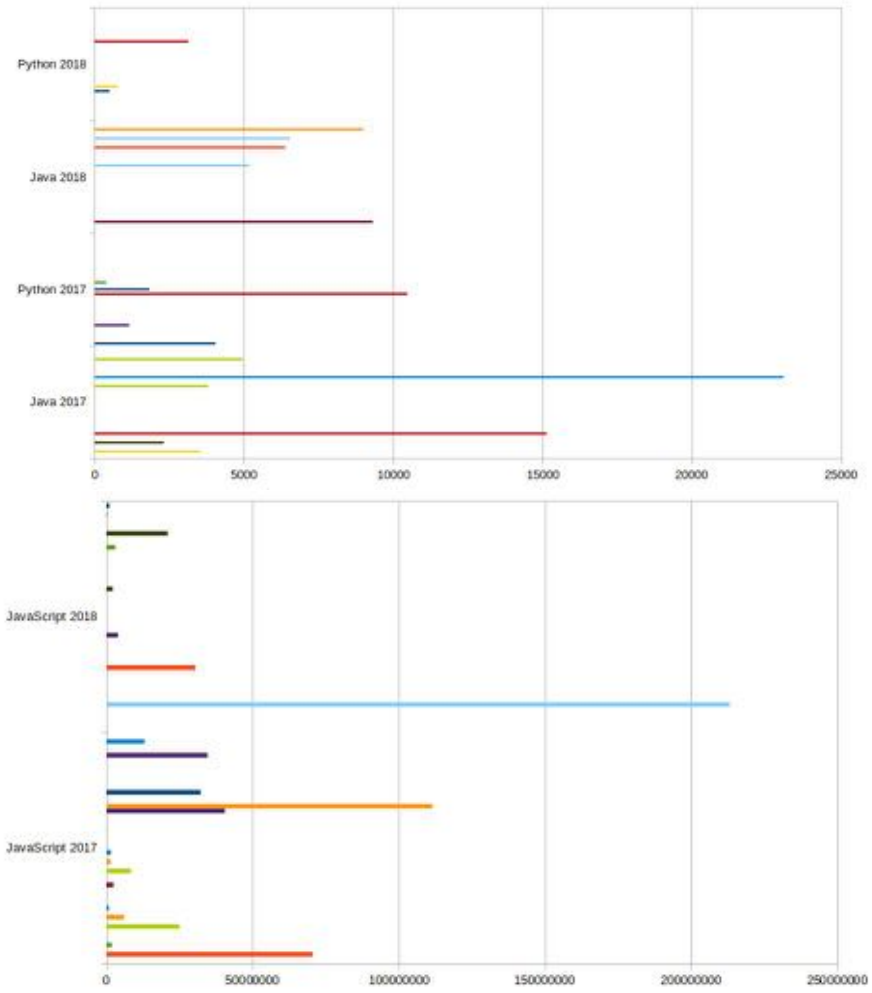


Figure 8: Average Halstead Effort

None of the projects are highly maintainable. This phenomenon is expected given the relatively low experience levels of our students.

The MI values for the Java source-code are all above 85 in both years indicating that our students are able to write highly maintainable Java code. Given that our advance programming modules use Java as the programming language for instruction, this confirms higher experience levels of our students when programming in Java. The maintainability of the Java projects seems to be similar across the years.

The only difference in maintainability across the years can be observed when looking at the projects using Python. All the source-code written in Python in 2018 is low in maintainability while those written in 2017 moderately maintainable.

These observations provide only weak support for the notion that the class project could have contributed to the ability of students to write more maintainable code. Students who have participated in the class project have written slightly more maintainable code only when considering source-code written in Python.

When further taking in consideration that relatively few projects used Python, poses a real a threat to the validity of this notion. We conclude that having participated in the class project seems to have no impact on the students' ability to write more maintainable code.

When considering the size of the projects measured using Halstead length, vocabulary and volume, it can be observed that students who had gone through the class project, had on average slightly larger projects than those who went into their capstone

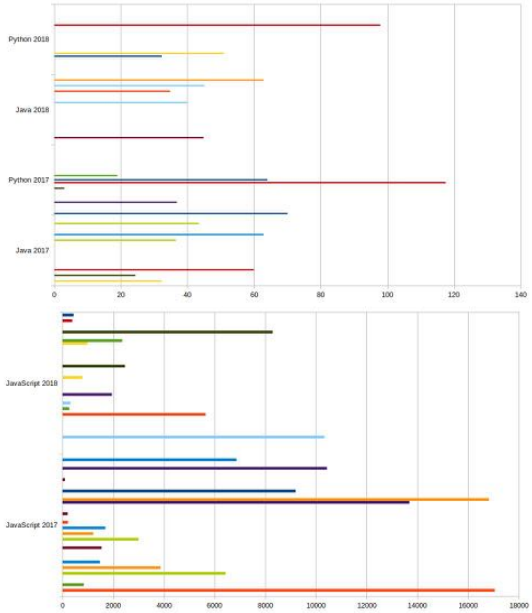


Figure 9: Average Halstead Length

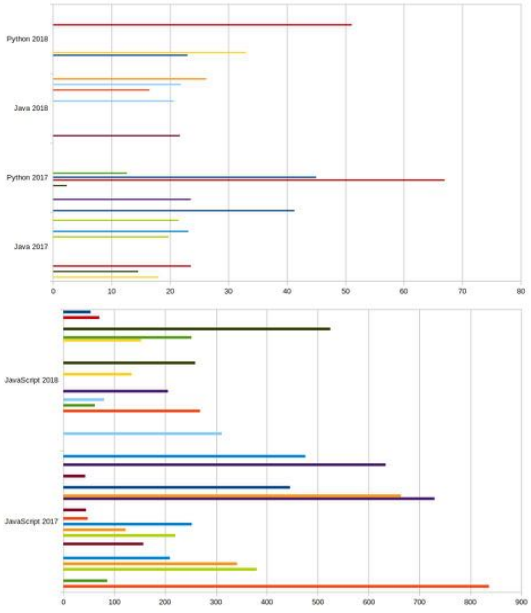


Figure 10: Average Halstead Vocabulary

projects without the preparation offered by the class project. Unfortunately, this is not a strong case to justify the class project. Only if the projects were significantly larger we could have argued that the students could solve more complex problems owing to the experience they had during the class project where they were exposed to working on an industry scale project.

We have, however, established that the source-code written by students who participated in a class project is likely to be higher in difficulty and complexity and was likely to require more effort when compared with the source-code written by students who did not participate in a class project. This is an indication that the students who had participated in the class project in general were able to produce more ambitious projects. This observation correlates with our general intuition that the projects of the year during which omitted the class project were less impressive than the projects done in the previous year.

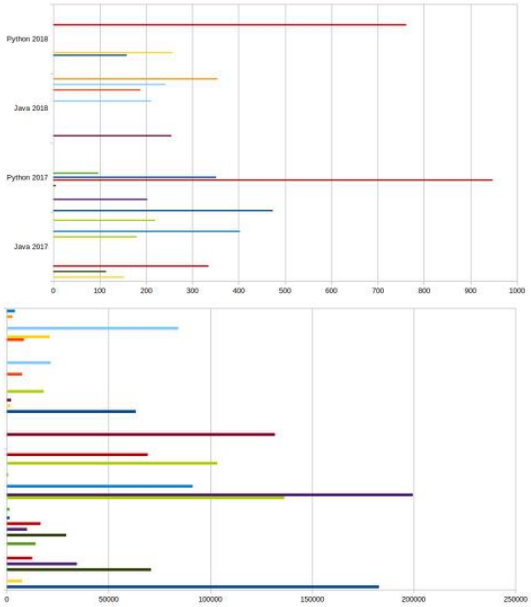


Figure 11: Average Halstead Volume

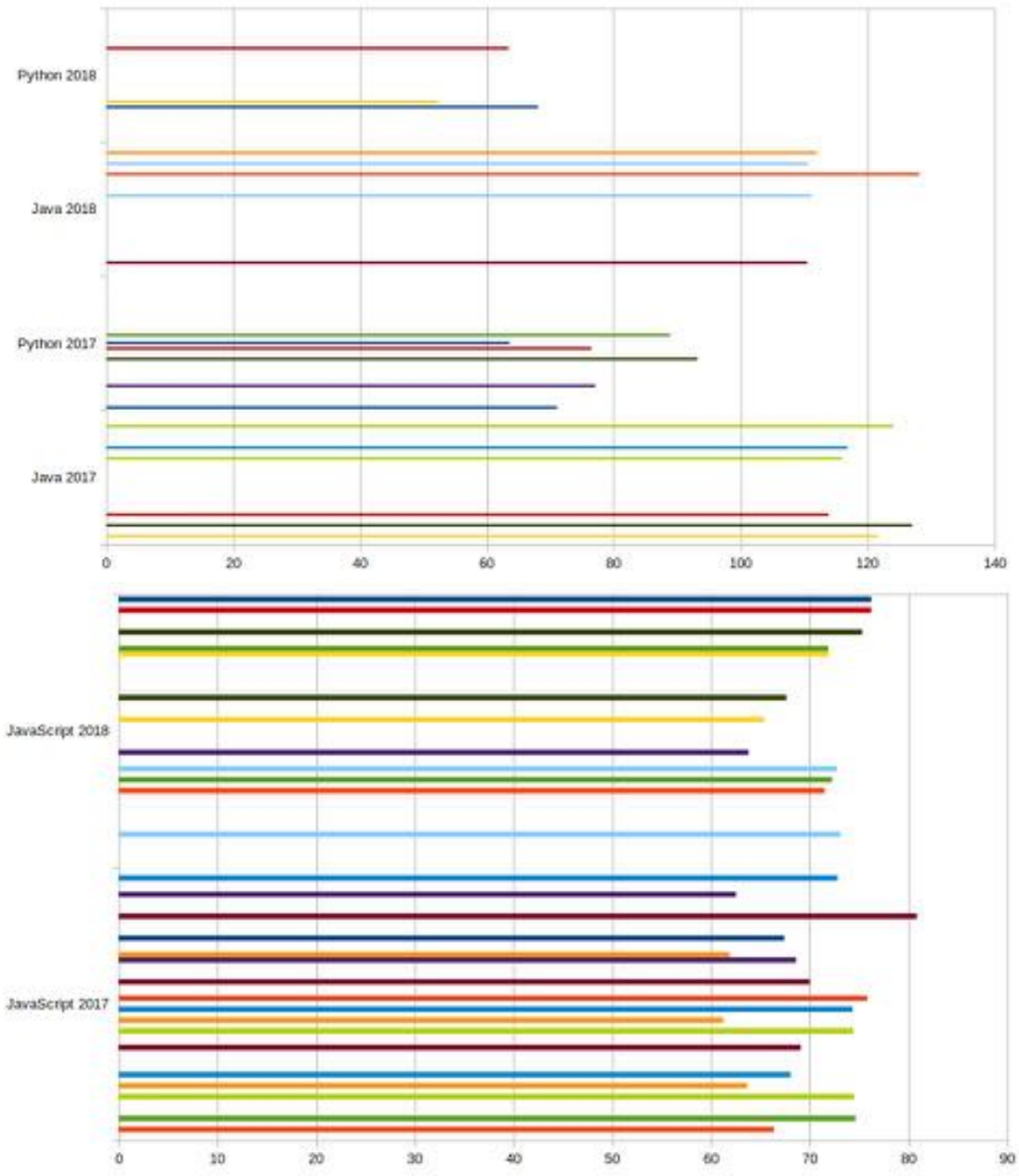


Figure 12: Average Maintainability Index

8 CONCLUSION

The SE module covers a range of Software Engineering concepts, tools, techniques and skills as well as team collaboration and competencies. Usually, the main goal of a software development process is the delivery of a high-quality, reliable, innovative and cost-efficient software product. The class project however, is an opportunity for the students to work on one fairly large software project as a small

group of 5 to 6 persons in a team. The main goal of the class project is to deliver a software product that may fail while, the students learn to develop a software product that is high-quality, reliable, innovative and cost-efficient software product. The capstone project requires the students to integrate the complete knowledge of their study in computer science and some elective modules to effectively carry out the capstone project. The capstone project exposes the students to a variety of methodologies for tackling different stages of the software life cycle, therefore providing a broader learning experience.

Based on the analysis of the individual student performance in the module for 2017 (i.e., when the class project is presented before the capstone project) and in 2018 (when there was no class project, instead the students started with capstone project). We conclude that the achievement of the students who took part in a class project before commencing their capstone project is statistically better. The findings further confirm the positive effect of the class project to the students' confidence and technical knowledge to carry-out the capstone.

REFERENCES

[1] Achilleas L. D. Buisman and Marko C. J. D. van Eekelen. 2014. Gamification in Educational Software Development. In *Proceedings of the Computer Science Education Research Conference (CSERC '14)*. ACM, New York, NY, USA, Article 1, 12 pages. https://doi.org/10.1145/2691352.2691353

[2] Don Coleman. 1992. Assessing maintainability. In *Proceedings of the Software Engineering Productivity Conference*. Hewlett-Packard, Palo Alto, CA, 525–532.

[3] Bill Curtis, Sylvia B. Sheppard, Phil Milliman, M. A. Borst, and Tom Love. 1979. Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics. *IEEE Transactions on Software Engineering* SE-5, 2 (March 1979), 96–104.

[4] P. Desai, G. H. Joshi, and M. Vijayalaskhmi. 2012. A novel approach to carrying out mini project in Computer Science & Engineering. In *Engineering Education: Innovative Practices and Future Trends (AICERA), 2012 IEEE International Conference on*. 1–4. https://doi.org/10.1109/AICERA.2012.6306699

[5] Maurice H. Halstead. 1977. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA.

[6] Carmel Marock. 2008. *Grappling with youth employability in South Africa*. Technical Report. Human Sciences Research Council, Pretoria.

[7] Linda Marshall, Vreda Pieterse, Lisa Thompson, and Dina M. Venter. 2016. Exploration of Participation in Student Software Engineering Teams. *ACM Transactions on Computing Education (TOCE)* 16, 2, Article 5 (Feb. 2016), 38 pages. https://doi.org/10.1145/2791396

[8] Thomas J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2, 4 (Dec 1976), 308–320.

[9] Sandeep Mitra. 2014. Using UML Modeling to Facilitate Three-Tier Architecture Projects in Software Engineering Courses. *Transactions on Computing Education (TOCE)* 14, 3, Article 17 (Oct. 2014), 31 pages. https://doi.org/10.1145/2635831

[10] Stacey Omeleze, Vreda Pieterse, and Fritz Solms. 2015. Teaching Modular Software Development and Integration. In *Proceedings of the 6^{th} Annual International Conference on Computer Science Education: Innovation & Technology (CSEIT)(Singapore, 5 - 6 October 2015)*. https://doi.org/10.5176/2251-2195_CSEIT15.25

[11] Vreda Pieterse, Fritz Solms, and Stacey Omeleze. 2017. Preparing software engineering students for industry. *Innovate* 12 (2017), 89 – 90.

[12] Vreda Pieterse, Lisa Thompson, Linda Marshall, and Dina M. Venter. 2012. An Intensive Software Engineering Learning Experience. In *Proceedings of Second Computer Science Education Research Conference (CSERC '12)*. ACM, New York, NY, USA, 47–54. https://doi.org/10.1145/2421277.2421283

[13] Vreda Pieterse, Lisa Thompson, Linda Marshall, and Dina M. Venter. 2012. Participation patterns in student teams. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE '12)*. ACM, New York, NY, USA, 265 – 270.

[14] Vreda Pieterse and Marko van Eekelen. 2016. Which Are Harder? Soft Skills or Hard Skills?. In *ICT Education: 45th Annual Conference of the Southern African Computer Lecturers' Association, SACLA 2016, Cullinan, South Africa, July 5-6, 2016, Revised Selected Papers*, Stefan Gruner (Ed.). Springer International Publishing, Cham, 160 – 167. https://doi.org/10.1007/978-3-319-47680-3_15

[15] Kurt D. Welker, Paul W. Oman, and Gerald G. Atkinson. 1997. Development and Application of an Automated Source Code Maintainability Index. *Journal of Software Maintenance: Research and Practice* 9, 3 (1997), 127–159.

E-Advise: An Adaptive Visual Toolset to Support Academic Advising

Hicham H. Hallal†
Computer Science and Engineering
American University of Sharjah
Sharjah, UAE
hhallal@aus.edu

Fadi Aloul
Computer Science and Engineering
American University of Sharjah
Sharjah, UAE
faloul@aus.edu

Sameer Alawneh
Computer Science and Engineering
American University of Sharjah
Sharjah, UAE
salawneh@aus.com

ABSTRACT

Academic advising, which aims at providing students with guidance to perform successful academic program planning, is a process that has undergone major transformations over the years. With the high level of connectivity witnessed in all aspects of life, the academic advising process is bound to benefit from automation. In this paper, we report on the development of a toolset to support academic advising and make the experience less tedious for both advisors and advisees. The E-Advise is an online application that interacts with an existing student information system (SIS) to read student information, presents the progress of students in their respective academic plans in a visual display, and allows students to select their courses for an upcoming term. The advisor, on the other hand, accesses the application to monitor student progress, confirm selected courses for registration, or to suggest modifications. All these features are comprised in a friendly graphical user interface, which allows the students and their advisors to exchange instant messages to discuss the selections or the progress. The E-Advise has been deployed and used in the Department of Computer Science and Engineering at the American University of Sharjah for two consecutive terms, where the feedback is all positive about the improved advising experience.

KEYWORDS

E-Advising, Academic Advising toolset, Tool Development, Curriculum planning

ACM Reference format:
Hicham H Hallal, Fadi Aloul and Sameer Alawneh. 2019. E-Advise: An Adaptive Visual Toolset to Support Academic Advising. In *Proceedings of CSERC '19 the 8th Computer Science Education Research Conference, Larnaca, Cyprus.*

1 Introduction

Academic advising has come a long way to support university students in their pursuit of completing their degrees in the optimal time and efforts. Efforts to optimize the advising process have been dedicated in many ways. With recent advancements in technologies, automating the process has become a viable option to make it both more flexible and efficient, which led to several tools implemented [1] to support this automation. The focus in the implemented tools is mainly on factors like:

1. Ease of use for both the student and the advisor.
2. Accessibility: the proposed tool should be accessible by users from anywhere and at any time.
3. Versatility: the different features offered by the proposed tool.

In this paper, we report on the development of an automated solution to support the academic advising activities in the department of Computer Science and Engineering (CSE) at the American University of Sharjah (AUS). The E-Advise tool is a mobile friendly web-based application that offers both students and their academic advisors several services that facilitate the advising process and make it more efficient. Worthy to note that prior to the use of E-Advise, academic advising was performed manually and by paper, where students meet their respective advisor who had to pull out courses from the system and then manually mark the advising sheets. This process used to be lengthy and prone to errors.

E-Advise interacts with an existing student information system to enable students to:

1. Access their advising records.
2. Visually display their progress in their academic plan.

3. Select their desired courses for an upcoming term.

On the other hand, the academic advisor accesses the E-Advise application to:

1. Monitor the student progress.
2. Confirm selected courses for registration or suggest modifications.
3. Communicate with the students via instant messaging to discuss decisions.

The E-Advise toolset has been deployed and used in the CSE Department at AUS for two consecutive academic years, where the feedback from students has been positive about their improved advising experience.

Section 2 provides a literature review of electronic advising systems. Section 3 discusses the design and implementation of the proposed design. An evaluation of the deployment of the proposed advising system is presented in Section 4. Finally, the paper concludes in Section 5.

2 Related Work

This work belongs to the category of attempts to automate the academic advising process for university students. Similar existing works include tools like

- My_eAdvisor [2], an automated tracking tool that provides students and advisors with immediate semester-by-semester feedback regarding the students' progression on their major study plans.

- E-Advisor [3], a multi-agent intelligent advising system designed for the Master of Science in Information Systems in Athabasca University in Canada [4]. E-Advisor allows students the ability to add preferences of specialization to their profile and then recommend courses based on these preferences.

- Adviseme [4], an intelligent web-based application, which provides a reliable, user-friendly interface for the handling of general advisory cases in special degree programs offered by the Faculty of Science and Technology (FST) at the University of the West Indies (UWI), St. Augustine campus.

- Online Advisor [5] is a decision system that supports both advisors and students in their use of than existing student information system. It integrates with the SIS to relieve clerical burdens and enable advisors to be student centered, allow academic advisors to aid students beyond the routine. The experiment was conducted in the school of business at the American University of Beirut, but was not extended to include other schools and departments.

While the main features of E-Advise toolset are common in existing solutions, the proposed solution provides a more interactive environment that follows a workflow between the advisor and the advisee. The student starts the process by accessing his/her study plan, selects the desired courses for the next term, and submits the selection to transfer the control to the advisor. The advisor can confirm, modify, or reset the selection and return it to the student. In both cases, a notification process is implemented through which both the student and the advisor receive email messages of the actions taken. Alternatively, the users can exchange instant messages within the main selection window to discuss modifications when suggested.

On the other hand, the E-Advise toolset is implemented independent of the student information system of the university. This allows for avoidance of interference between the two platforms and for more security in the SIS ecosystem.

3 E-Advise: Design and Implementation

Figure 1 shows the workflow of the E-Advise tool. The implemented tool includes two main components:

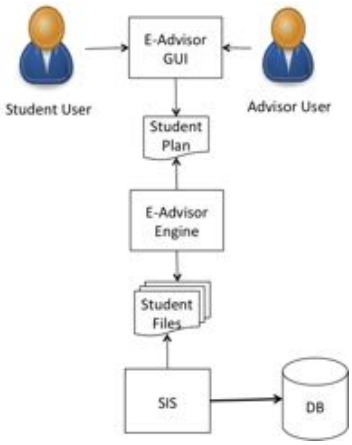


Figure 1: Workflow of the E-Advise toolset.

The GUI: E-Advise includes a friendly GUI that offers the following main features:

1. Visual display of the study plan of each student as shown in Figure 2.

2. Use of color-coding to distinguish between courses per category. One categorization of courses is following their status:
    a. Completed courses (pink)
    b. Currently registered courses (orange)
    c. Courses available for selection in the next term (yellow)
    d. Courses selected for next term (green)

    Another categorization is according to the type of the course:
    a. Required
    b. Major Elective
    c. General Education Requirement or Free Elective
3. Critical path identification: Through depicting the prerequisites of courses, the tool helps the user identify the critical path in the study plan by showing the courses that become available as a result of completing any specific course.
4. Identification of missing pre-requisites for any course that the student wants to take.
5. Manual insertion of courses in the schedule of the upcoming term, especially when the desired course is not in the displayed plan (Figure 2).
6. Display of the completed hours, currently registered hours, remaining hours, the major of the student, and the version of academic catalog followed to specify the study plan of the student.

Figure 2: Color-coded display of the study plan in E-Advise.

The Intelligent Engine: This component is responsible for the following functions:

a. Communication with the existing SIS to retrieve up to date student files and data related to their study plans.

b. Reorganization of the retrieved student records to become usable in the visual display of the study plan. This includes computation of values like:
   - Number of completed credits
   - Number of remaining credits

c. Communication between the student user and the advisor user to share comments on a specific study plan.

d. Analysis of the pre-requisite relationship between courses to indicate dependence and ordering.

3 System Evaluation

To evaluate the tool, we deployed a survey to students asking them about their impressions of the experience of using the automated advising approach and the electronic E-Advise application. In the survey, we asked students to answer two direct questions about their experience of using the E-Advise toolset and to compare the experience to the scenario before the deployment of the tool. The answers of the students showed a positive impression as indicated in the following tables.

The number of participating students was in the range 120 to 130 students out of almost 400 students in the CSE department.

Question 1: The new electronic advising system was useful

The new electronic advising system was useful?		
Answers	Results	
	Fall 2017	Spring 2018
Strongly Agree	71	76
Agree	29	37
Neutral	15	11
Disagree	2	2
Strongly Disagree	1	2
Score	4.42	**4.43**
Number of Participants	118	128

Question 2: I prefer the new electronic advising system over the traditional paper-based advising system

I prefer the new electronic advising system over the traditional paper-based advising system		
Answers	Results	
	Fall 2017	Spring 2018
Strongly Agree	80	94
Agree	19	21
Neutral	14	11
Disagree	3	0
Strongly Disagree	2	2
Score	4.46	**4.60**
Number of Participants	118	128

4 Conclusion and Future Work

We presented an automated approach to assist in the academic advising process. The approach involves the deployment of the web-based application E-Advise that interfaces users, students and advisors, to an engine capable of extracting student information from an existing SIS and present it in a visual display. Once displayed both the student and his/her advisor are able to modify it and exchange comments/feedback about the optimization of the advising process for a specific term and for the whole study plan as well. The feedback from users (mainly students) has been positive when compared with classical advising methods.

Further development of the toolset is already in process and more features have been added to it. In addition, a major improvement on the tool development is launch a mobile version of the application that can add to the flexibility of it use and to its efficiency.

ACKNOWLEDGMENTS

The authors of this work would like to recognize the contributions of the IT unit of the College of Engineering at AUS to the development and the expansion of the E-Advise toolset.

REFERENCES

[1] O. Iatrellis, A. Kameas, and P. Fitsilis, “Academic Advising Systems: A Systematic Literature Review of Empirical Evidence,” *in Education Sciences*, vol. 7, no. 4, pp. 90, 2017.

[2] L. Keston Henderson and W. Goodridge, "AdviseMe: An Intelligent Web-Based Application for Academic Advising" International Journal of Advanced Computer Science and Applications (IJACSA), 6(8), 2015. http://dx.doi.org/10.14569/IJACSA.2015.060831

[3] F. Lin, S. Leung, D. Wen, F. Zhang and M. Kinshuk, "E-Advisor: A Multi-agent System for Academic Advising," *in International Transactions on Systems Science and Applications*, vol. 4, no. 2, pp. 89-98, 20009.

[4] A. Noaman and F. Ahmed, "A New Framework for E Academic Advising," *in Procedia Computer Science*, vol. 65, pp. 358-367, 2015.

[5] T. Feghali, I. Zbib, and S. Hallal, "A web-based decision support tool for academic advising," in *Journal of Educational Technology & Society*, vol. 14, no. 1, pp. 82–94, 2011.

DI2-Co-Innovation Lab

Teaching software development in and for real business situations

Holger Günzel
Department Business Administration
Munich University of Applied Sciences
holger.guenzel@hm.edu

Lars Brehm
Department Business Administration
Munich University of Applied Sciences
lars.brehm@hm.edu

Hans-Jürgen Haak
Haak Consulting and Mediation
mail@haak-beratung.de

Mira Grönvall
Business Information Systems
Tampere University of Applied Sciences
mira.gronvall@tuni.fi

Anne-Mari Sainio
Business Information Systems
Tampere University of Applied Sciences
anne-mari.sainio@tuni.fi

ABSTRACT

Globalization and digitization affect not only the business world, but also significantly university teaching. Besides competencies in regular software development and software engineering, focusing on fixed, pre-defined, solvable tasks for single students it is required to teach flexible solution definition, development and testing in a complex, business-like context. The consideration of the topic interdisciplinary, international and spatially distributed teams deserves special mention.

The DI2-Co-Innovation Lab serves as a framework to work on joint projects and tasks with companies, students and lecturers in a spatial *d*istributed, *i*nternational and *i*nterdisciplinary setting (DI2) and at the same time to reduce the effort and risk of those involved. The Co-Innovation Lab brings a proven agile project management method with knowledge assets from previous projects to support the work on real business problems. Aspects of mediation mitigate the effects of the international, interdisciplinary and distributed setting. Tried and tested IT tools for acquisition, communication and knowledge management support the processing. Furthermore, branding and publications are used in the acquisition of companies, tasks and cooperating lecturers.

Our approach is based on earlier studies and field reports from practice and universities that try to learn and teach in similar scenarios. The added value of our Co-Innovation lab consists of an existing, flexible framework and appropriate processes to simplify and accelerate the set-up, execution and close of projects for all participants.

The DI2-Co-Innovation Lab will take place in the winter semester 2019/ 2020 for already the third time in a cooperation course between business students of the University of Applied Sciences Munich (Germany) and IT students at the University of Applied Sciences Tampere (Finland) and will be continuously developed further.

KEYWORDS

educational framework, interdisciplinary, international, experience paper

ACM Reference format:

Holger Günzel, Lars Brehm, Hans-Jürgen Haak, Mira Grönvall, Anne-Mari Sainio. 2019. DI2-Co-Innovation Lab: Teaching software development in and for real business situations. In *Proceedings of the 8th Computer Science Education Research Conference (CSERC '19)*

1 Introduction

Digitization is changing the business world as a whole [McKinsey2016]: New business models, changing processes and changing customer requirements are creating new challenges for all industries. Digital technologies in particular are changing the "rules of the game" in business life to date, and in this way information technology in turn.

In addition, the "globalization of brainwork" [BoesKämpf2009] is creating new challenges in the world of work, as for the first time highly qualified areas of work such as information technology are coming under pressure to change because of an international market. According to [BoesKämpf2009], employees will either show "the feeling of powerlessness" or as "manifestly employees" will have to consciously deal with the new situation in order to gain a self-confident self-image as employees and a new ability to act. It should not be forgotten that not all IT employees will be extroverted, but are often technology-fixed.

In contrast, the teaching of software development and software engineering often still focuses on "one-dimensional programming tasks" [Alkadi et al 2010] such as sorting algorithms, which have to be solved by the student alone and can often be copied/pasted from the Internet. According to [BusenbergTam1979], academic programming differs from real programming in the following ways

- often more complex and of a larger scale
- focus on both system software and application software

• developed by programming teams under constraints in both time and resources
• develop software requirements and detailed design specifications
• software documentation required.

For this reason, [Frezza et al 2018] describes in his competency-based framework for learning (CoLeaF) that the three "learning components: knowledge, dispositions and skills" must be covered. In the area of knowledge, for example, the theoretical foundation on programming, algorithms and data structures have to be lectured, in the area of disposition topics such as ethical awareness or never giving up on solving a problem have to be addressed, and in the area of skills communication, conflict and risk management, among other things, in order to discuss ideas or results with developers, users or customers. The highest priorities of his analysis are teamwork, communication and planning/time management. In the agile world, the changed responsibility of employees must be added, as management and control are replaced by self-management. Furthermore, decisions will be made by the employees themselves and under a clear uncertainty.

The article of [Hesmeralda et al 2018] on "Evaluation of the University Curriculum in the Formation of Competences for the Software Development Industry" resumes about the cooperation between universities and companies: "Looking back, the answer is always the same, the mission of the university must be the commitment to practical training, and the link of the graduate in the real workplace, the third mission, which society claims is what allows the articulation between its actors for its benefit". Here, too, the need for effective communication and conflict resolution is emphasized.

Therefore, the new didactic demands will be: opening the subject classification to situation dynamics and from instruction to self-directed learning [Schüßler 2008]. The fundamental change of perspective from a knowledge transfer didactic to didactics of self-directed appropriation of knowledge and competences is essential. Behind this is the insight that learning is most effective and efficient when the learner can independently acquire the knowledge, experience its sustainability and apply it in experiments [Schüßler 2008]. "Competence building and maturing learning is a self-motion through which the learning subject develops skills for self-organized and appropriate problem solving. It moves in a learning environment (which defines a competence profile and distribution channels), but at the same time realizes a learning world (self-learning and design)" [Arnold Erpenbeck 2014].

In the classic definition of self-directed learning, Knowles [Knowles 1975] describes the process of self-directed learning as a process in which individuals take the initiative - with or without the help of others - and analyze their learning needs, formulate learning objectives, identify human or material learning resources, select and implement suitable learning strategies and evaluate learning outcomes.

It should be particularly emphasized that a constant reflection and an improvement loop is built into the process of self-directed learning through the evaluation aspect. The teacher takes on different roles in the learning process. He is an expert for the learning content, an active listener and productive questioner, facilitator of a concentrated and trusting (learning) atmosphere, trainer who recommends exercises, and process facilitator in the sense of a "critical friend" ([Siebert 2009]).

Universities must ask themselves: How can a future-oriented teaching concept - a "learning world" [Arnold Erpenbeck 2014] - be implemented alongside theoretical lectures and exercises in order to develop the required "skills"? How can the competences for a new, international and interdisciplinary cooperation be developed? How can innovative tasks and experience gain be integrated into a module with little additional effort? Where do the real tasks come from? How can more cooperation be integrated with companies or organizations?

Chapter 2 deals with the requirements and existing literature for the implementation of the requirements. Then the concept of the Co-Innovation Lab and its extension will be presented (Chapter 3). This concept was improved in a multi-year cooperation course between German and Finnish students (Chapter 4), in addition to many years of experience in the Faculty of Business Administration. The article concludes with a summary and outlook.

2 Related work

In designing an appropriate learning environment, the professional and organizational requirements must be met by students, lecturers and participating companies as well. Students are interested in attractive (challenging but solvable) projects with a real issue, authentic customer contact, an usable, but flexible method with proven tools and infrastructure and an integration into existing courses. Furthermore, they require a complete overview over all responsibilities, expected results and grading for all participants in the beginning. Lecturers expect low or manageable risk, relatively small additional effort, rapid reuse of teaching concepts, integration into existing courses, easy contact with companies for acquisition of projects, visibility of cooperation results, and contact with other lecturers/disciplines. Interdisciplinarity and internationality must be implemented in the processes of all joining universities. Participating companies would like to achieve innovative results with an accurate generation of results with small cooperation effort and often the contact to potential employees. On the one hand, as much as possible must be covered by processes; on the other hand, flexibility must be maintained. The participants shouldn't get suck by a mass of different tools, processes and preferences.

From a variety of similar approaches, exemplary attempts are presented which partially build or support a learning world. [Gotel et al 2009] set up an infrastructure platform for student projects that also facilitates international cooperation. The focus is on the tools for cooperation and software development. In [Parker Holcombe 1999] article, project risks between universities and industry are reported, which are caused by the limited time and, above all, the new project organization to be set up. In the "protected learning space" [GroschelRoth-Dietrich 2018] approach, cooperation with industrial partners is supported by best practice for project management, process models and knowledge management. [Bernstein et al 2005] pursues a similar approach with the AlgorithmA Project, which sets up a "microcosm" of a software

company in which software engineering, but also teamwork and the holistic implementation of software projects can take place in 10 weeks. Knowledge for subsequent projects is also preserved. Analogously, [Way 2005] reports on his collaborative framework to simulate the real-world experience of working for a medium-sized software company to develop a new product under time pressure. The focus is always on the aspect of implementation: local and exclusively intra-disciplinary.

[Čavrak Bosnic 2018] discusses the aspect of cooperation with the resilience in project teams in distributed project teams in case of problems caused by missing employees and changes in requirements. The cohesion of the team is reflected in a stronger product (weak cohesion) or process focus (strong cohesion). A multi-year distributed software development study with three universities supports this theory.

In all examples, the focus is primarily on the student's point of view. However, a learning world must involve all participants - students, lecturers and companies in order to be successful in the long term.

3 DI2-Co-Innovation Lab

The DI2-Co-Innovation Lab serves as a framework to work on joint (spatial distributed, international and interdisciplinary) software development projects and tasks with companies, students and lecturers and at the same time to reduce the effort and risk of those involved.

It bases on the Co-Innovation Lab [Günzel Brehm 2018], initiated by Prof. Holger Günzel and Prof. Lars Brehm at the Munich University of Applied Sciences in the field of business management projects and supports with some extensions these projects. The Lab is an overarching concept for innovation projects of students with companies and serves at the same time as an organizational platform and interface between courses and companies. With the joint development environment between students and companies, temporary innovation partnerships are created - in the form of projects: from the idea through validation to the result in a maximum of ten weeks, in order to strive for a win-win situation for universities, students and companies. Companies receive innovative solutions from the point of view of an often unknown group of customers and they also meet the potential employees of tomorrow. The students build up the necessary competencies by moving from case studies to reality and intensifying the learning content.

The DI2-Co-Innovation Lab brings a proven agile project management method with knowledge assets from previous projects to support the work on real business problems. Aspects of mediation mitigate the effects of the international, interdisciplinary and distributed setting. Tried and tested IT solutions for acquisition, communication and knowledge management support the processing. Furthermore, branding and publications are used in the acquisition of companies, tasks and cooperating lecturers.

3.1 Concept

The four building blocks didactic, platform, stream and community shape the framework of the DI2-Co-Innovation Lab.

3.1.1 Didactic concept

The didactic concept base on small teams of four to seven interdisciplinary students (Business administration, computer science, business informatics, but as well design students or subject matter experts) who autonomously carry out real tasks in a self-directed learning mode. The lecturer acts as a coach. Analogous to the agile approach used at [Sutherland Schwaber2017], which consists of several iterations and retrospectives, the student teams set the requirements independently, prioritize, plan and implement them. The task is solved parallel to the theoretical units of courses (fig. 1). In most cases, it is not necessary to change the examination regulations. The theoretical units can be reduced to a minimum depending on the objectives of the course and the level of training. The lecturer uses a time contingent in the course to carry out the coordination and result presentations.

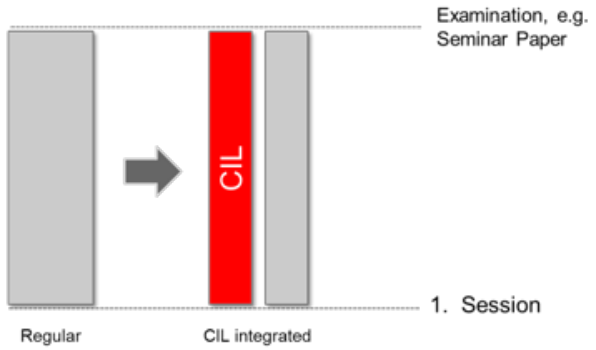


Figure 1: Integration of the DI2-Co-Innovation Lab in a course

3.1.2 Platform concept

In addition to providing the theoretical basis for the course, the most important task of the lecturer is the integration of innovation partner companies and project topics. The steps used are comparable to a consulting life cycle of a consulting company (Fig. 2).

- Acquisition phase (Acquire): The respective lecturers carry out the acquisition - they must decide whether the project topics are suitable for a course. Through personal contacts, previous projects or through contact with other colleagues, new project topics arise. In addition, a "pipeline" of project requests can be established. During the acquisition process, information about the company is collected and the problem outlined in order to prepare the students.
- Project implementation phase (Deliver): In the following, the scope, method, work products, organization and timing are described.

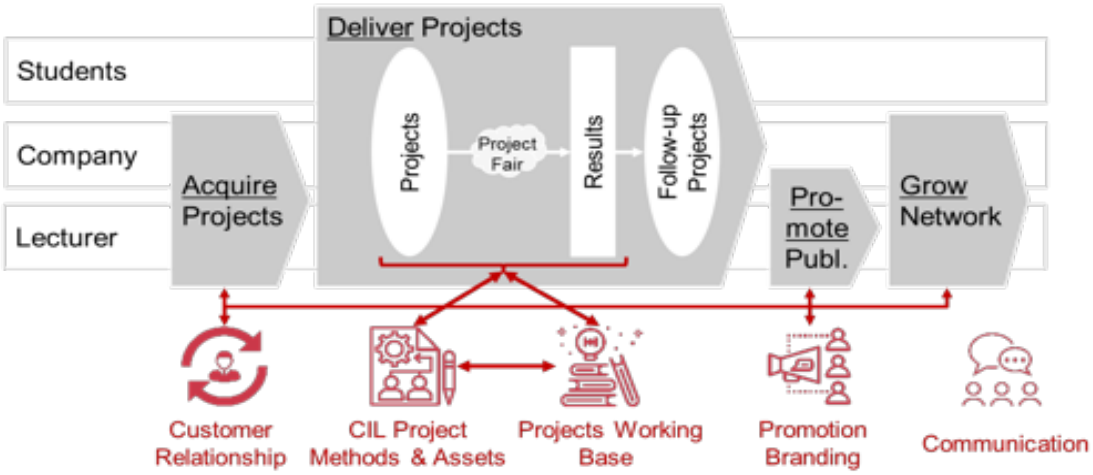


Fig. 2: Platform concept

Depending on the course and the level of knowledge, this task should be taken over by the students. Students interview the companies involved to find out the "real" problem (and not just the symptoms). Parallel to the initiation activities, formal contracts are concluded to define the legal framework. On the one hand, confidentiality clauses are required; on the other hand, the question of intellectual property in the results is clarified. The implementation is carried out according to the procedure model described in the project plan.

- During the follow-up (Close), the students deal with further internal questions. The students carry out a self-reflexion, the follow-up of the method for the internal knowledge management, the preparation of a case study for the education of other student groups and the request of a reference letter from the customer. Of particular relevance in the agile sense is the optimization of the knowledge database procedures for lecturers and students. In the final discussion with the students, the lecturer compares the expectations and experiences of the stakeholders.
- Marketing phase (Promote): In the marketing phase, the aim is to increase the visibility of the faculty, the professors and students involved through press articles on successful projects, but also through scientific publications. In addition to the short-term identification of topics and competence, the basis for the acquisition of the next projects is created here.
- Growth phase (Grow): The expansion of the network with companies and lecturers in turn supports the acquisition activity and the possibility of interdisciplinary and international projects.

3.1.3 Stream concept

Currently, several streams are identified in which the tasks of the companies can be classified. Projects with software development can be found in the category IT & Product Consulting.

- Strategic Impulses: The stream deals with the support of a real company in business models and strategies in the digital world. These concern the evaluation, further development or new development of goals, concepts and initiatives.
- Business Consulting: The aim of the stream is to apply the working practices and techniques of an innovation or digitization project under realistic conditions based on a concrete, complex project. Students acquire the ability to apply solution-oriented, adequate working practices, consulting and project management techniques to the implementation of theoretical knowledge in concrete projects.
- IT & Product Consulting: The stream focuses on the conception and prototypical implementation of IT as a mobile app, software application and extension of a website of real issues from industry. The students learn working techniques from the requirement gathering with customers up to the implementation. An emphasis lies on interdisciplinary cooperation and project management.
- Outsourced Services: This stream deals with the outsourcing of challenges of project partners. The focus is less on the conception than on the execution of operative tasks.

3.1.4 Community concept

The open, non-commercial community offers the possibility to profit from the didactic concept, the platform and the work results or the projects with one's own application. In addition, everyone is invited to add new results, procedures, publications or project partners in order to stimulate the expansion of the platform.

3.2 Project management

The DI2-Co-Innovation Lab uses a hybrid project management approach adapted by Scrum4Consulting [Kerscher Günzel 2019]. The team works in iterations and continuously documents the activities so that the customer and the lecturer can follow the progress. The project ends with the presentation and delivery of the results by the team. In this case, additional persons from the partner organization are often invited, which is an additional challenge for the students, but also motivation.

3.2.1 Agile Values and Principles

Through the agile values [Agile Manifesto 2001] and the focus on interactions, working products, cooperation with the client, acceptance of change, commitment, focus, courage, respect and openness become essential components of the approach. These are required from both, the students and the client. The second must actively engage in the cooperation, otherwise, the project cannot be carried out successfully. Therefore, the principles (derived from the agile manifesto) such as continuous delivery of increments, positive response to change, a self-organized team, daily cooperation, personal dialogue, professional as well as technical excellence and reduction to the essentials are essential fundamentals of the concept.

3.2.2 Roles

The roles are based on the roles of traditional Scrum. However, some necessary ones extend these roles and tasks are adapted in order to reflect the temporary cooperation between companies and students.

Roles	Tasks
Scrum Master	The ScrumMaster (SM) is responsible for ensuring that the process is followed. For this, he protects his team and eliminates disturbances and obstacles. This position is often covered by a student. Particular attention is paid to the work on the agile mindset of the participants, since his focus is on increasing the productivity of the team.
Product Owner	The Product Owner (PO) is responsible for the value of the project and prioritizes user stories by value. The PO delivers the requirements that the student team needs for the project. He is either a trained employee of the client, who can also be supported by a students, or a student.
Team	The student team consists of students and client's employees. Together they are responsible for a constant and qualitative delivery of results (self-organized). Together with the product owner, the strategic added value and the direction towards product development are permanently worked on.
Client	The client has requested the task and is the first contact person for contractual and organizational matters. He mainly works together with the PO. He receives and accepts the product increments in each review meeting and coordinates them continuously with the PO.
Mana-gement	The management is defined in this concept as the manager one level above of the client in the advised company. He is responsible to ensuring that all essential elements for product development are available to the team. In cooperation with the scrum master, he revises structures and conditions in the client's company.
User	The user is the customer of the product or service. The user is always actively asked for feedback and can be invited to review meetings. Depending on the type of project, the user can be an external customer or an internal employee.

3.2.3 Phases and Procedure

The Scrum4Consulting process is divided into an initiation phase, an implementation phase and a final phase (figure 3), with adjustments being made in the first two phases. In comparison to a waterfall procedure model with a conventional project management approach, client's requirements are fixed in the beginning and completely worked towards the entire project, the initiation phase is also used to understand the project scope and client's requirements, but not to analyse and document them comprehensively. The focus is on breaking down the requirements into redundancy-free "stories" for the iterative implementation phase, which, in addition to fast partial results, provides intensive cooperation and optimisation of the process.

Initiation Phase: The initiation phase (according to [Gloger 2016] also called strategic phase) is used in Scrum4Consulting to carry out the elements to be planned, such as the detailing of the idea, generation of a common vision, creation of a backlog with user stories, its prioritization and initial estimation of the Scrum4Consulting Team. This phase is characterised by the fact that work is not done directly for the result, but the requirements of the client are fixed [Gloger 2016] [Heikkilä et al. 2015].

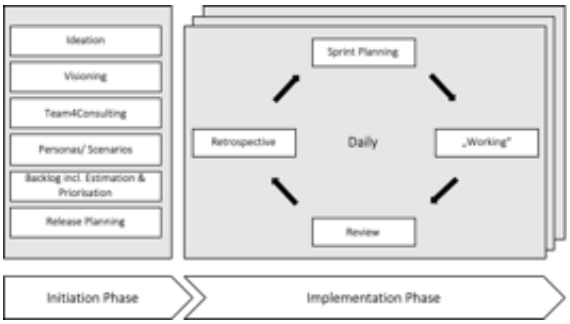


Fig. 3: Initiation and Implementation Phases with Work Products

Depending on the type of project, a new project is triggered by either a client's idea or need or their management. For example, business process optimization serves to improve quality or process execution speed. The vision creates a motivating target picture for the team. The challenge in creating a vision is to create a strong product vision that simultaneously triggers emotions [Gloger, 2016] [Gottesdiener Gorman 2011]. The creation of this vision often already takes place within the assignment. It is the responsibility of the Product Owner to create the vision. In "Team4Consulting", in addition to the selection of the client's employees for the team and team building, work is done with the company on the activation of the agile mind-set - which is currently not available in all companies. The mind-set is an essential success factor or, in a negative case, a show stopper for agile projects [Gloger 2016]. Furthermore, the common "rules of the game" in the team such as working hours, exchange of work results, sprint length, definition of done, etc. are defined. An important extension is the procedure described in chapter 3.3, which starts at this point. The Scrum Master supports the team during the whole project. Personas serve as a description of a potential target group that uses the result. In addition to the description of the person, goals of the person, life circumstances and context of the product use are recorded. The scenarios show the use and influence of the product on the target group. The backlog consists of the scrum backlog with the user stories as functional requirements for the result and additional requirements (called side products) of the client. In some situation makes it necessary to introduce client stories like the creation of a business case.

Implementation Phase: In the Scrum4Consulting concept, the implementation phase describes all activities to be performed by the implementation team. These activities are carried out in time-limited ("timeboxed") iterations. The sprint length for student projects is set to two weeks. The software development is run through during the sprint for the individual story. The final rollout only takes place when the solution has proven to be suitable through the test of the implementation phase. The implementation phase includes the sprint planning meeting, the review meeting, the retrospective and the daily [Wirdemann Mainusch 2017]. Analogue to Scrum, sprint planning is separated into two different meetings in order to carry out planning (sprint planning 2) only after understanding and creating a team commitment in sprint planning 1 [Gloger 2016] [Maximini 2018]. For each selected backlog item, the requirements must be clarified, acceptance criteria (behaviour) and constraints (framework conditions) fixed and a test generated [Gloger 2016][Maximini 2018].

3.3 DI2-extension: Coaching Concept for International and Interdisciplinary Projects

Based on the specific situation (distributed, international and interdisciplinary) an associated coaching for the scrum master and the team is introduced. The coaching of the scrum master puts its focus firstly on sensitizing that besides the common challenges as missing skills or motivation there could be more crucial and hard-to-handle challenges as cultural differences, deviating mother languages and missing face-to-face contact. Secondly the scrum master gets an introduction into mediation methods to better fulfill his/ her role. During the project phases the coach supports the team members to articulate the impediments frankly and to find and decide proper solutions. After an initialization, the regular retrospectives are supplemented by coaching sessions. In particular, interpersonal and communicative obstacles are addressed, in the team, between the project roles and with other persons involved. In addition, the scrum master is supported with a toolset that specifically contains challenges and possible solutions. Furthermore, the teams and the team members had the opportunity to be coached individually in telephone calls or Skype sessions.

3.3.1 Initialization

The scrum master supports and is responsible for the solution process within project. The participants try to reach a common agreement that corresponds to their interests. Analogous to the basic principles of agile project management, the responsibility for a solution of a conflict remains in the team.

The toolset, handed over to the scrum master is based on a simple finding: Having success is a very strong motivator - for all team members. The scrum master helps the team to achieve the targets in time and budget. Aspects of mediation will help: It is about people, individual interests and needs and communication skills. It provides methods to respectfully work together, to avoid critical situations, to cope with conflicts and may be adopted through all phases of a project. The focus is on the work process to increase team cohesion.

3.3.2 During the Iterations

Agile methods rely on a regular improvement process in relation to the product and the process. Retrospectives are team meetings to learn from the past [Andriyani et al. 2017}. In the best-case scenario, the team addresses problems of the last iteration and jointly finds improvement actions. The retrospective is a protected space for the team and the product owner in which problems can be addressed openly and solutions found together. Although there is a multitude of different methods, they all contain the same basic elements: introduction, data collection, insight, actions, effectiveness testing, and closure.

In previous projects, it became obvious that team members often had intrinsic problems to articulate impediments because of inadequate co-operation or even missing support and input from their colleagues. The fear to blame someone could be observed frequently. So special attention of the scrum master should be given to the way of communication of the team members. Be aware that communication between people follows specific rules. When a party argues his/ her position, he/ she follows (probably non-transparent) interests which are based on (mainly emotional) needs. Basic skills in detecting those cases and therefore in excellent communication are e.g. listening respectfully, asking the questions needed for your better understanding or telling your opinions, views, needs & abilities and explaining them.

3.3.3 Communication Disorders and Conflicts

In addition to the regular retrospective, the support of the scrum master can also become necessary in case of conflicts. Strong signs for a critical situation or a potential conflict are among other things emotional comments (accusing, shouting, less esteem), less attention (not listening, interrupting, talking simultaneously), generalization (using "always", "never", arguing with rules/ regulations), endless discussions (repeating the same arguments, insisting) or nonverbal signals (turning away, folding one's arms). In critical situations: Identify whether all parties involved judge the situation as a conflict. If so, bring them together, i.e. don't act bilaterally.

3.4 Utilized Toolset to Support Framework

The following technical infrastructure and tools are utilized to support the participants in the overall project lifecycle.

- Acquire: A cloud CRM platform is used from a startup MyTaskey (www.mytaskey.de). The requirements are that the customer data and project ideas are collected and used quickly and easily. A rights concept ensures appropriate access to the data. The contact persons for the project partners establish a contact with other lecturers and topics.
- Deliver:
    - The project management method is supported either by MyTaskey or by Trello (www.trello.com). MyTaskey offers more possibilities in terms of protocols, scheduling and document management. Trello provides a visual advantage.
    - As an exchange platform, the university's own cloud platform ("Sync&Share") is used, which functions analogously to commercial tools.
    - Nuclino (www.nuclino.com) is used for the joint processing of documents.
    - Gitlab/ Github and documents in Markdown format are used as a knowledge base for lecturers and students.
    - Slack (www.slack.com) (asynchronous) and Skype (synchronous) are used in the student teams as communication tools for spatially distributed processing.
    - The development tools are not explicitly fixed as they depend on tasks and customers.
- Promote: Project articles are published via the university or faculty website and Facebook. More and more, articles are the basis for new customer publications in journals - the aim is to expand the community and fill the pipeline with new tasks.
- Grow: Slack and Skype serve as a communication platform for lecturers to exchange project ideas and initiate joint projects.

4 Case Study

Both the Tampere University of Applied Sciences and the Munich University of Applied Sciences have many years of experience in project-based learning and want to take project learning to the next level in an international context with their cooperation. The increase in complexity, interdisciplinary, intercultural with spatial separation, was brought about as a cooperation project between the Master of Business Administration students from the Munich in Germany and Bachelor of Business Information Systems students from the Tampere in Finland and one or more clients.

The Co-Innovation Lab has existed for almost 5 years in the Faculty of Business Administration at Munich University of Applied Sciences. A large number of students experienced the cooperation with industrial companies and real tasks. Various processes and tools support lecturers and students.

The students have to support the above-mentioned phases as well as to fulfill the customer requirements. In previous cooperation, for example, projects such as an app for a smart meter for an energy company or a gamification approach for logistics processes were designed and implemented. Furthermore, the students use the knowledge database and contribute to the extension of this platform. In addition to a reference, a project description and a press article, value is placed on the extension of the knowledge database with templates, process models and literature.

The first run was characterized by a sequential execution with a customer in Munich with a conception in the summer semester by business students and the implementation by business informatics students in the winter semester. The general goal that students work realistically and interdisciplinary was achieved. Despite extensive documentation, the typical problems of a waterfall procedure have arisen, i.e. the requirements were either not detailed enough or have changed in the meantime.

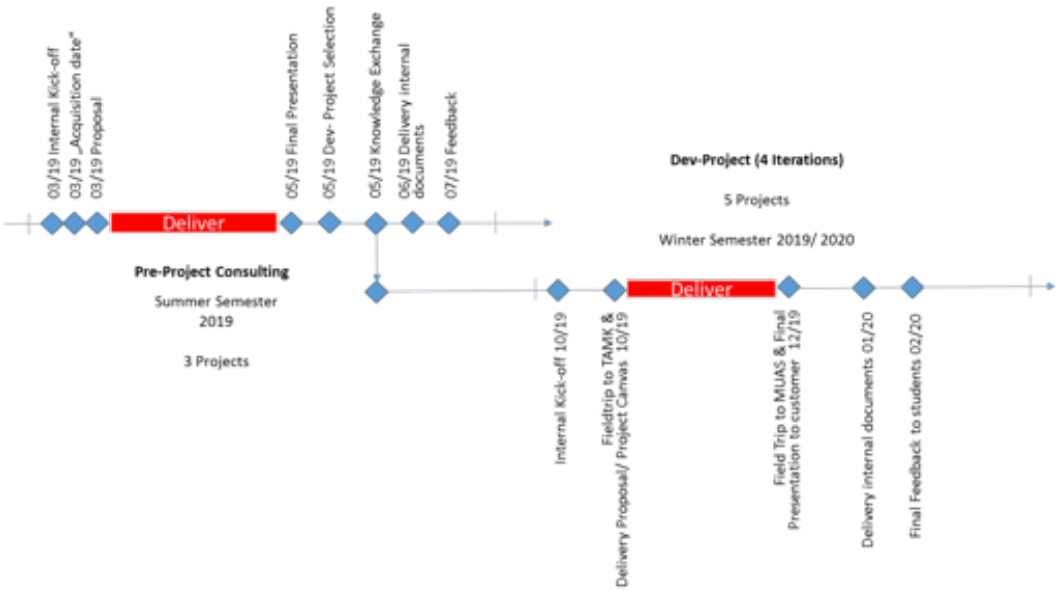


Fig. 4: Schedule in Winter Semester 2019/ 2020

In the second year, the setting was fundamentally changed. Mixed teams started together in the winter semester. The given roles as product owner, scrum master and functional team (both with business and IT expertise) were filled and the project was planned with four sprints. One lecturer assumed the role of team coaching; other lecturers provided technical and organizational support. The students of the participating universities were invited to an intra-university kick-off at the beginning of the semester to get to know the rough project requirements and customer profiles. The project was divided into four teams with participants from each university.

On the day of arrival at Tampere for a 3-day field trip (with an effective 48h on site), the students from Tampere and Munich got to know each other. On the following day, the customer presented his business area and an assignment in a two-hour presentation and discussion. The rest of the day and the following day were characterised by the planning of the procedure and the generation of initial concept ideas. After another three days, the students were asked to submit a project proposal, which was subsequently commented on and revised by the lecturers. In the following ten weeks, an agile process approach with 2-week sprints was carried out. The review meetings took place virtually with the student teams, the company and the lecturers. A trained coach (lecturer) supported the retrospectives. A few days before the final presentation, a dry run was carried out within the university to ensure quality. With the final presentation, the cooperation with the company was concluded, and by the end of the semester, further internal university results had to be finalised (i.e. assets for knowledge management). After the presentation, four final retrospectives were held with each team and a final discussion was held at the end of the semester. All teams have succeeded with the assignments; the cooperation was more time-consuming due to the increased communication effort, but more customer-specific in terms of the result.

In the current third collaboration round, the general procedure will be maintained (Fig. 4). A preliminary phase in the summer semester with other teams with strategy projects allows the students to get to know the customer and his area of business. A handover in the current semester with the students of the winter semester enables early mental involvement with the topic. The implementation takes place again in the winter semester. Additional interdisciplinarity is achieved through the involvement of design students. Further skills in usability and design are expected. Coaching in the area of retrospectives will be extended. The use of an enlarged team canvas around the aspects of communication and conflict resolution should make the aspects of team building more visible. The students receive technical literature on the different discipline's processes and work products even before the project in order to create a project canvas faster and speak the same language.

5 Summery

The existing approaches in project-based learning at both universities will be extended to address the international context. The participating lecturers and students see the Co-Innovation Lab as an excellent opportunity to come into contact with practice-relevant topics at the university. However, the greater added value lies in the expansion of the competences required by the digital transformation and globalisation. The participants move from hard

facts to soft skills and can deal with the future world of work. In the DI2 setting, real interdisciplinary and international experiences can be made.

The following topics have been repeatedly shown: The topic should be based on a real need in the company, so that the seriousness for a consultation and a sufficient time commitment of the company is given. The topics must come close to the topics of the study programme so that they can be solved in a short time. An autonomous planning of the project with regard to content and effort of the students and the comparison at the end lead to an increased interest in the project and a significant increase of the learning success of the students. The commitment of the students is constantly high due to their practical experience and their personal responsibility. This has a high learning effect: The students experience "up close" the effects of incorrectly calculated costs, inaccurately assumed changes and framework conditions or incorrect planning in the project plan.

Despite the very positive results in the existing rounds, the cooperation and approach is constantly being further developed. Nevertheless, it should not be forgotten that not only the formal framework for achieving success is in place, but also the positive spirit of cooperation for teaching and research of the persons involved.

REFERENCES

Agile Manifesto: Manifesto for Agile Software Development, online: https://agilemanifesto.org/ (last access: 19.05.2019), 2001

Alkadi, G.; Beaubouef, T.; Schroeder, R.: The Sometimes Harsh Reality of Real World Computer Science Projects, ACM Inroads V1, Nr.4, 2010, pp. 59-62

Andriyani, Y.; Hoda, R.; Amor, R.: Reflection in Agile Retrospectives, In: XP 2017: Agile Processes in Software Engineering and Extreme Programming, 2017, pp 3-19

Arnold, R.; Erpenbeck, J.: Wissen ist keine Kompetenz. Dialoge zur Kompetenzreifung, 2014

Bernstein, M.; FitzGerald, K. M.; Macdonell, J. P.; Concepcion, A. I.: AlgorithmA Project: The Ten-week Mock Software Company, Volume 37, number 1, Feb 2005, pp. 142 – 146

Boes, A.; Kämpf, T.: Offshoring und die neuen Unsicherheiten einer globalisierten Arbeitswelt. In. Hochseilakt - Leben und Arbeiten in der IT-Branche, 2009, pp. 23-41

Busenberg, S. N.; Tam, W. C.: An Academic Program Providing Realistic Training in Software Engineering, Communications of the ACM, Volume 22 Issue 6, June 1979, pp. 341-345

Čavrak, I.; Bosnic, I.: Team Resilience in Distributed Student Projects Team Resilience in Distributed Student Projects Proceedings of the 13th International Conference on Global Software Engineering, ICGSE '18, 2018, pp. 112-120

Frezza, S.; Daniels, M.; Pears, A.; Cajander, Asa; Kann, V.; Kapoor, A.; McDermott, R.; Peters, A.; Sabin, M.; Wallace, C.: Modelling Competencies for Computing Education Beyond 2020: A Research Based Approach to Defining Competencies in the Computing Disciplines, In: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, 2018, pp. 148-174

Gloger, B.: Scrum: Produkte zuverlässig und schnell entwickeln. Hanser, 2016

Gotel, O; Kulkarni, V.; Phal, D.; Say, M.; Scharff, C.; Sunetnanta, T.: Evolving an Infrastructure for Student Global Software Development Projects: Lessons for Industry, ISEC '09 Proceedings of the 2nd India software engineering conference, Pages, 2009, pp. 117-126

Gottesdiener, E.; Gorman, M.: It's the Goal, Not the Role: The Value of Business Analysis in Scrum, Online: https://www.agileconnection.com/article/its-goal-not-role-value-business-analysis-scrum (last access: 19.05.2019), 2011

Groschel, M.; Roth-Dietrich, G.: Acquisition of Practical Skills in the Protected Learning Space of a Scientific Community, ECSEE'18 Proceedings of the 3rd European Conference of Software Engineering Education, 2018, pp. 63 – 71

Günzel, H.; Brehm, L.: Co-Innovation Lab – a Platform for Learning the Competences of the Future, The Future of Education Conference 2018, Florence, 2018

Heikkilä, V.T.; Paasivaara, M.; Rautiainena, K.; Lasseniusa, C.; Toivola, T.; Järvinen, J.: Operational release planning in large-scale Scrum with multiple stakeholders: A longitudinal case study at F-Secure Corporation. In: Information and Software Technology, 57, 2015, pp. 116–140

Hesmeralda R. E.; Marleny P. A.; Ronald R. A.; Willie Á. C.: Evaluation of the University Curriculum in the Formation of Competences for the Software Development Industry, ICBIM '18: Proceedings of the 2nd International Conference on Business and Information Management, 2018

Kerscher, S.; Günzel, H.: Scrum4Consulting – Agile Project Management for Consulting Projects, 8th International Scientific Conference on Project Management in the Baltic Countries, April 25-26, 2019, Riga, pp. 242 – 253

Knowles, M. „Self-directed learning: A guide for learners and teachers", New York, Association Press, 1975

Maximini, D.: Scrum-Einführung in der Unternehmenspraxis: Von starren Strukturen zu agilen Kulturen. Gabler, 2018

McKinsey Quarterly (ed): "Digital strategy: The economics of disruption", Number 2, 2016, online: https://www.mckinsey.com/quarterly/the-magazine/2016-issue-2-mckinsey-quarterly

Parker, H.; Holcombe, M.: Campus-based Industrial Software Projects: Risks and Rewards, In SIGCSE Bull, Volume 31, Number 3, 1999, p. 189

Schüßler, I.: Reflexives Lernen in der Erwachsenenbildung – zwischen Irritation und Kohärenz, Bildungsforschung, 5 (2), 2008, https://uhh.de/k9dmq

Siebert, H.: Selbstgesteuertes Lernen und Lernberatung. Konstruktivistische Perspektiven, ZIEL Verlag: Augsburg, 2009

Sutherland, J., Schwaber, K., The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game, 2017

Way, Thomas P. A Company-based Framework for a Software Engineering Course, SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education, SIGCSE Bull.Volume 37, Number 1, Feb 2005, pp. 132 - 136

Wirdemann, R.; Mainusch, J.: Scrum mit User Stories. Hanser, 2017

A Flipped Classroom Experiment

The implementation of Semi-Synchronous Learning

Hani Alers
The Hague University of Applied Sciences
Zoetermeer, Netherlands
HAL@HHS.NL

Marcella Veldthuis
The Hague University of Applied Sciences
Zoetermeer, Netherlands
M.Veldthuis@HHS.NL

Aleksandra Malinowska
University of California, Santa Barbara
California, USA
Amalinowska@ucsb.edu

Tim Cocx
The Hague University of Applied Sciences
Zoetermeer, Netherlands
T.Cocx@HHS.NL

ABSTRACT

With semi-synchronous learning, students are provided with online learning material, and allowed a window of time within which they learn the material. This allows the students to employ mastery-based-progression and just-in-time learning approaches. Furthermore, this also frees lecturers' time, allowing them to focus on helping students lagging behind.

This article describes the process of converting a ten-week university course to a semi-synchronous teaching format. The course is followed yearly by an average of 200 students. All theoretical course lectures were replaced by short online videos and a few interactive workshops that encourage student progression. Other aspects of the course were left unchanged. Feedback about the course was collected from lecturers and students. Student grades and video viewing statistics were also used to evaluate the new approach.

Results show that some students had an initial resistance to the new format and leaned towards traditional teaching methods. However, these complaints quickly subsided. Video viewing statistics show that students remained engaged in learning the theory throughout the course duration. The new approach did not result in a significant difference in student grades. Additionally, involved lecturers who were less familiar with the theory simply viewed the videos to quickly prepare for their role. Moreover, lecturers were also able to save time by referring students with basic questions to the correspondent videos

KEYWORDS

Semi-synchronous learning, flipped classroom, blended learning

1 Introduction

Active learning in the classroom has taken the forefront as a teaching method. Eric Mazur, a Harvard physicist and educator who has campaigned against the lecture format for years argued that ’Lecturing is outmoded, outdated, and inefficient,’ (2014). His statement is based on a study that shows that test scores significantly improve when lectures are replaced by other methods of active learning. Students should not be passive listeners, but use their time in the classroom to actively participate and engage (Freeman, Eddy, & McDonough, 2014).

In an era where information is freely available to everyone, employees are expected to be able to independently acquire knowledge and skills. The traditional lecture does not prepare students for the context of their future professional practice. A different kind of education is needed, a semi-synchronous teaching format that allows students to employ just-in-time-learning and mastery-based-progression.

‘Online education has clear pedagogical advantages over traditional education’ (Caplan, 2018). With semi-synchronous learning, students are provided with online learning materials (such as videos) and are give a timeframe to prepare for seminars, where they discuss what they have learned. This teaching format has certain advantages. Through video lectures Students have access to top teachers on a specific topic. Furthermore, students can consume the content at a time that is convenient, and they can rewatch the video until they have mastered the content.

The use of videos combined with seminars also helps students develop planning skills. Studies indicate that the adolescent brain does not fully develop until the the age of 25. Some adolescents may find regulating their behavior difficult based on long-term abstract goals, struggling with skills such as planning, anticipating, prioritizing and focus (Nelis & Van Sark, 2009; Slot & Van Aken, 2013). Within a specific timeframe, students choose to view content specific videos, preparing them for seminars. The videos are deliberately kept short, to ensure ease of focus and to invite students to rewatch the videos in order to fully master the content.

This article will focus on the effects of semi-synchronous learning at the The Hague University of Applied Sciences (HHS). The research is conducted in a ten-week research course. The research course is a project based course given to second year information and communication technology (ICT) students. The course aims to teach students how to conduct research in the context of their future professional practice. Research groups consisting of four people choose a (semi structured) assignment from a client. The course provides students with a full set of research skills, starting with a first interview with a possible client and finalizing with the group presenting their results in a scientific research paper.

Certain challenges have been noted in the construction of the course. Due to a large number of students enrolled, a low interest in research, and a shortage of qualified staff, (also see chapter 3) the number of students who pass the course is relatively low compared to other courses students attended. Also, a discrepancy exists between the education the students receive and the tools needed for future employment. According to Wolbers (2003) the right tools are not being utilized to provide students with the knowledge and skillset they need to succeed. Therefore, a more effective program is needed.
This article describes the process of converting the ten-week research course to a semi-synchronous teaching format. It provides an answer to the question if semi-synchronous learning can be used in a university of applied sciences. With this setup, students are allowed to learn just in time and employ mastery-based-progression. We hypothesize that there will be an increase in success rate, skill level, and student involvement.

2 Old design of the Research-Course

For the experiment, a course was chosen which proved challenging for both students and instructors. This Research-Course taught students applied research methods. The course was taught in the traditional classroom format for the last five years. The course is given simultaneously to students from different specialization programs in order to encourage interfaculty collaboration.. The Research-Course consists of three components as explained below.

2.1 Research-Theory

The first part of the course consisted of five lectures explaining research theory. The lectures were based on the book “Research: This Is It!” (Ben Baarda, 2010). Students were encouraged to use the book but it was not obligatory for the subject. The end assessment was an individual written exam based on the book.

2.2 Research-Plan

Students were allowed to choose a research challenge to work on which was coupled to one of the research groups within the university. The students worked in groups using the theory the students learned in ``Research-Theory`' to analyze the challenge and break it down into a specific research question. Students then crafted a research plan describing the topic, research questions, and data gathering methods. Four interactive workshops were used to discuss the progress and provide feedback from research coaches and fellow students. The students were assessed based on their final research proposal.

2.3 Research-Project

Once the students completed the research proposal, they used it to start a research project where they carried out that plan. During this period, students had regular meetings with their research coach to check whether their work is still relevant to the original research challenge and whether they were conducting the research appropriately. Additional, three lectures were given describing how to perform data analysis and write a scientific report. These lectures were not covered by the written exam mentioned in section 2.1 and were solely aimed to help students with the research project. The students completed the project by writing a research report detailing methodology, results, and conclusions. Grades were awarded per group based on the quality of the research and the report.

3 Challenges facing the research course

As mentioned before this course has always proved challenging for both students and instructors. Below are the main challenges faced by the course

3.1 Low interest in research

Students in the university of applied sciences are more interested in applied fields than theoretical ones. They prefer working in project based environments. They are therefore less motivated in learning research methodology, as to them this seems like a course more fitting for a technical university. For example, the final 3 lectures of the course had extremely low student attendance. Even though these lectures were vital for completing the Research-Project, they were not covered by the exam of the Research-Theory. The students did not care enough about the subject to attend these last few lectures.

3.2 Large number of students

Students within two different specializations and located in two different campuses take the course. Students from Information Security Management (ISM) are all located in Zoetermeer while Software Engineering (SE) students can take the course either in Zoetermeer or in Den Haag. Combined, there are approximately 200 students in total.

3.3 Shortage of qualified-staff

Since the course is given in a university of applied sciences, There is a shortage of qualified staff specializing in research methods. Previously, additional staff with limited experience and training were asked to assist in the course. This is exacerbated by the previous issue of having a large number of students over multiple locations. Although qualified staff gave the main course lectures and were available for questions, there was not enough time to monitor the progress of each individual student. This task fell on the additional staff, who simply lacked the hands on experience to provide ideal support.

3.4 Low passing percentage

As a direct result of the issues mentioned above, about half the students pass the theory exam on their first attempt. A large number of students show lack of knowledge of the basic theoretical concepts.

4 New flipped classroom format

To implement self synchronous learning, a flipped classroom was used. I a flipped classroom, the students learn the theory of the course at home and they come to the university to work on

excursuses to practice the theory they learned. The research proposal and research project parts of the course are largely unaltered. The changes are mainly in the way the that research theory is taught to the students. The changes are listed below.

4.1 Online videos replace classroom lectures

In the new flipped classroom approach, all seven classic classroom lectures are cancelled. The theory of the course is instead explained in 19 videos. The first 14 videos are tested in the written exam, corresponding to the first five lectures. Videos 15-19 correspond to the last three lectures which are aimed to help students with completing their research projects. The videos are kept short to help maintain student attention. Durations ranged between 5-10 minutes. To achieve this, all repetition of information from the lectures was eliminated in the videos. Instead, videos refer to relevant topics in other videos and provide links so students can easily jump to the relevant topic if needed. Illustrations and animations are also added to the videos to help sustain student attention. The illustrations were also intended to serve as mnemonics to help students remember information. These illustrations also help the students find the relevant spot in a specific video when they quickly scroll through it.

The videos were recorded in a studio environment to ensure high production quality. Efforts were made to ensure that sound, video, and color rendering were as natural as possible. The videos were uploaded in 1080p resolution to YouTube and made publicly available.

4.2 Interactive workshops

To ensure minimum student progression, three new interactive workshops were planned during the course. Each workshop covers a subgroup of the videos and is used to discuss questions about the material explained in these videos. Students have to complete homework exercises as preparation for each workshop. These homework assignments are discussed at the beginning of each workshop, then new questions are presented to the students which they can work on trying to answer in groups.

If at any point during the workshop it became clear that a student has not watched and understood the videos before hand, the student is asked to leave the workshop. They can only join the workshop again once they have watched an understood the needed videos. Although the workshops are optional, they are presented to the students as important practice for the written exam. They are therefore motivated to attend these workshops, and as a result are pressured to progress through the course theory.

The workshops use a Kahoot setup. Kahoot is an online tool which allows lecturers to present questions to the students and let the students individually try to answer the questions using their mobile phones. By immediately seeing how many students are able to choose the correct answer, lecturers can immediately determine how much difficulty the students are facing with the question and decide how much time to spend on discussing it.

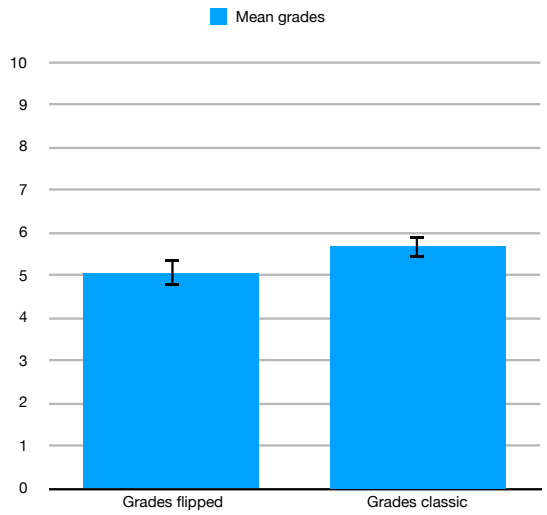


Figure 1: **Mean grades scored by students for the Research-Theory part of the course using the flipped classroom approach (left) and lecture based approach (right). Error bars show the mean standard error.**

4.3 Templates with links to videos

The research proposal and research project parts of the course are graded based on a report (one for each) provided per group. Links to the videos were added in the templates for each report. The video links are targeted so that in each section of the report the student can see a link to a video explaining that specific topic.

5 Results

Looking at the grades the students scored in the exam for research theory (Figure 1), it is possible to see that the students performed at a similar level with the new flipped format. This indicates that the flipped classroom is equivalent to the classroom approach when it comes to the overall level of the students' understanding of the theory.

Student feedback (Frigure 2) shows that a large number of students found the format enjoyable. Students explained that the videos had surprisingly high production quality. They were short and therefore easy to watch repeatedly when needed. Figure 3 shows that leading to the exams, students have repeatedly rewatched the videos. Therefore it is clear that the students heavily relied on the videos to understand the theory and prepare for the exams.

Besides understanding the course theory, students reported that the videos also helped them while conducting the practical part of the research course (research proposal and research project) as can be seen in figures 4 and 5. They helped the students plan the research, and also propelled them further when they had trouble progressing. We interpreted this to mean that despite the short format of the videos, they were still a useful guide for conducting

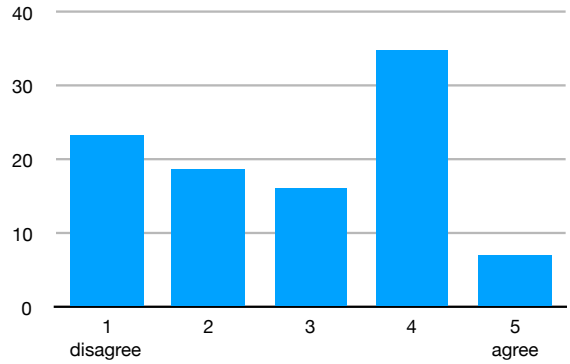


Figure 2: **Student feedback "I found the EWR course format (videos + interactive lectures) to be enjoyable"**

applied research. With the students leaning on the videos for help, this freed time for the instructors from answering simple questions. Students who came with questions that indicated they had not put in effort to learn the theory, were referred to a video that answered their question. This allowed instructors to focus their efforts on students who were lagging behind their peers, or advanced students who wanted to know more about the topic than what the videos provided.

With the help of the videos, it was easy to provide all students following the course a comparative level of instruction, thereby bypassing the problem of lacking enough qualified staff. However, the interactive workshops differed in quality depending on the instructor. By offloading some of the work to the online videos, qualified staff had more time to play a greater role in coaching a larger number of students than would have been possible otherwise.

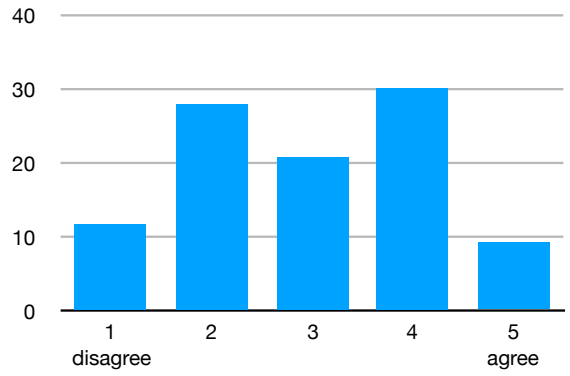


Figure 4: **Student feedback "The videos helped me in planing how to do" the research project"**

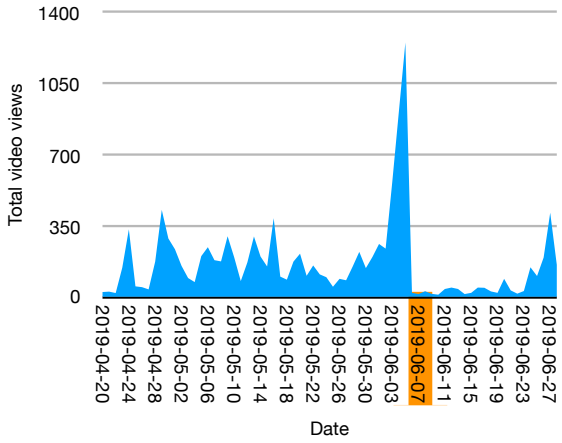


Figure 3: **Total views of the video recordings. Day of the exam is highlighted in orange**

Less than half the students participated in the interactive workshops. Students were only allowed to participate if they watched the required videos associated with the lecture and completed the required homework. Those who did not adequately prepare for a workshop were sent away. Those who did participate showed good understanding of the theory. Lecturers noted that the discussions they had with these students were more advanced than in previous years, and that they found these interactions "more enjoyable" than discussions during traditional lectures.

On the other hand, as Figure 6 indicates, the majority of the students preferred the old classroom based lectures to the flipped classroom. When sent away from the interactive lectures for not having watched videos in preparation, students reacted negatively claiming that they were "treated like children". It is possible that,

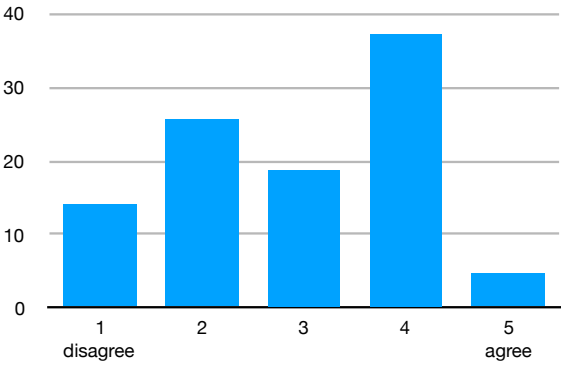


Figure 5: **Student feedback "When I got stuck with the research the videos helped me figure out how to proceed (videos + interactive lectures) to be enjoyable"**

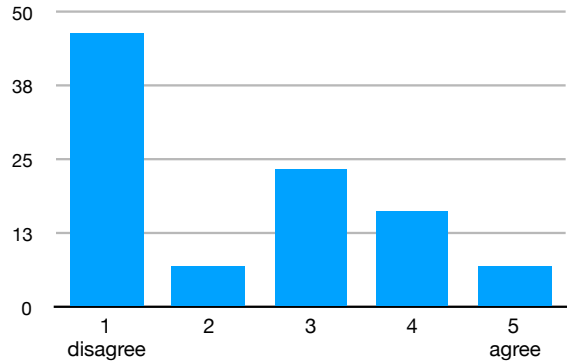


Figure 6: **Student feedback "I prefer this flipped classroom format to traditional lecture based format"**

this being the first course to use the flipped format, the students were simply not used to the novel methods. Instructors commented that this format also required the students to "take more responsibility in planning their progress", which the students "did not seem too eager to take".

6 Conclusions and recommendations

The flipped classroom was an effective way to teach the course to a large number of students. Their performance was equivalent to that using lecture based instructions for the same course the previous year. The flipped format helped free resources allowing instructors to focus their efforts where they were really needed. Although students enjoyed having video lectures, they still preferred a lectures based approach.

It is clear that students had problems keeping up with the expected minimum student progression expected for the course. They were not able to plan their progress and keep up with the pace the course required. One possible solution to enforce minimum progression is to have weekly assessments in the interactive workshops as part of the final course grade.

REFERENCES

Bajak, A. (2014, May 12). Lectures aren't just boring, they're ineffective, too, study finds. Retrieved from https://www.sciencemag.org/news/2014/05/lectures-arent-just-boring-theyre-ineffective-too-study-finds

Caplan, B. (2018). The Case against Education: Why the Education System is a Waste of Time and Money. Princeton, NJ: Princeton University Press.

Freeman, S., Eddy, S.L., McDonough, M., Smith, M.K., Okoroafor, N., Jordt, H. & Wenderoth, M.P. (2014). 'Active learning increases student performance in science, engineering and mathematics.' Proceedings of the National Academy of Sciences, 111(23), 8410-8415. Retrieved from https://www.pnas.org/content/111/23/8410

Nelis, H. & Van Sark, Y. (2010) Puberbrein binnenstebuiten. Wat beweegt jongeren van 10 tot 25 jaar? (7e, herziene en uitgebreide druk). Utrecht/Antwerpen, Kosmos Uitgevers B.V.

Slot, W., & Van Aken, M. (2013). Psychologie van de adolescentie. Basisboek. Amersfoort: ThiemeMeulenhoff.

Baarda, B. (2010). This is It!: Guidelines for Setting Up, Doing and Evaluating Quantitative and Qualitative Research. Noordhoff.

Wolbers, M.H. (2003). Job mismatches and their labour-market effects among school-leavers in Europe. *European Sociological Review, 19*(3), 239-266.

REDUCING TEAMWORK FAILURES BY TYING ETHICS TO TEAMWORK TRAINING

Alan P. Sprague Raquel Diaz-Sprague
Dept. of Electrical and Computing Engineering
University of Alabama at Birmingham
Birmingham, AL, USA
sprague@uab.edu diazspra@uab.edu

Abstract. Most engineering curricula require ethics teaching. Competence in teamwork is considered an essential skill that students need to acquire during their undergraduate education. Both ethics and teamwork are course objectives mandated by the Accreditation Board for Engineering & Technology (ABET). However, those subjects - ethics and teamwork - are often difficult to teach and evaluate. We present our experience offering a short refresher on ethics principles to Electrical & Computing Engineering juniors prior to offering teamwork training. We use an andragogical approach – guided discussions, not lectures, and student-led demonstrations. We report preliminary results and observations.

Keywords—teamwork, ethics, engineering education.

I. INTRODUCTION

Most engineering curricula require ethics teaching. Competence in teamwork is considered an essential skill that students need to acquire during their undergraduate education. Both ethics and teamwork are course objectives mandated by the Accreditation Board for Engineering & Technology (ABET). However, those subjects - ethics and teamwork - are often difficult to teach and evaluate. Lingard [5] suggests that one of the reasons that teamwork is difficult to teach is the fact that most engineering faculty have not themselves had much teamwork training.

The literature reports that teamwork is difficult for students [5]. Training in teamwork for undergraduate students commonly takes the form of having students take a course/s requiring them to produce a product as a member of a team. Lingard and Barkataki [6] consider this to be a poor method – they state "it is unlikely that without adequate faculty guidance students can pick up (effective teamwork) skills through ad-hoc group project experience".

The realiy of being better trained in technical matters than in interpersonal communication and other people-related skills also extends to industry: Crawford (2) reports that managers of engineering projects in industry are better prepared and trained in management of projects than management of people.

II. TEAMWORK AMONG STUDENTS CAN BE FRAUGHT WITH FAILURE

Failure in team projects is not uncommon. One mode of failure in teamwork that we have observed/experienced is contrasting work styles by team members -- one team member wanting to get started immediately while teammate/s prefer to procrastinate. Another more surprising mode of failure that we observed starts with a team leader who is an overachiever, who grabs a central part of the project and keeps the results under wraps, and then when the semester is coming to a close these results cannot be successfully integrated with the portions constructed by the other team members.

There are differences between the consequences of teamwork failures in certain courses requiring teamwork: In some classes, especially Computer Science classes, each team is to produce a piece of code – a program that accomplishes some goal. The goal is sufficiently complex that a team of 3 to 6 students is expected to spend a semester accomplishing it. In other classes, like the Electric & Computing Engineering class where we presented our minimodule, each team was expected to come up with a design for a new product, including some business/financial aspects of the product. Computer code to accomplish some task is a very touchy thing: very small errors can result in catastrophic failure of the program. Design seems to be much less fragile.

III. Our Minimodule

In fall 2018 we dedicated two class periods to review moral reasoning theories, based on short video lectures. This is followed by guided student group discussions of engineering practice scenarios presenting moral dilemmas. The rationale for this approach is to provide reasonably realistic scenarios that young professionals may encounter. Also, within a group even a single person can serve as a powerful force to choose to do the right thing over expedience and/or apathy, thus contributing to foster a moral climate

We use an andragogical approach – guided discussions, not lectures, and student-led demonstrations.

Description of our mini-module:
The 35 undergraduate students in our class had already been assigned to 7 teams of 5 students each. The 3 graduate students in the class formed the eighth team.

We started the first class period with watching the video "Justice: What's the right thing to do?", an ethics class taught by Professor Michael Sandel (Harvard, a regular class with several hundred students, presented in a large auditorium). He presents a scenario which contains a dilemma, asks some specific questions and asks for answers from volunteers, and very occasionally takes a poll of the students. After about 30 minutes watching Sandel's lecture, we handed each of our teams a page containing a few questions based on the Sandel lecture and told our students to discuss these questions within the teams and write answers to one question on the page. Each team was to then report discussion results orally to the class as a whole.

The second class period started with watching a video of Professor Frans de Waal of Emory University, on morality among animals (Capuchin monkeys, elephants, and others), emphasizing fairness and reciprocity as pillars of morality. The class period was followed by guided student discussion in the same way as the first.

For the third and fourth class periods of our minimodule occurred 4 weeks later. Students were to demonstrate their understanding of teamwork. There are at least two ways for students to present the concepts of teamwork to the class. A team can make a powerpoint presentation to the class. Alternately, a team can direct the remaining teams in playing a game. Many internet sites describe games that are intended to foster teamwork among the players [9,10].

Each group had been assigned (during class period 2) to present either an activity/game that promoted teamwork among the members of the team, or to present a 10-minute lecture on teamwork to the class. Numerous web sites promote games that encourage good teamwork among members of a team (the setting for this often is the employees of a company). Most teams chose a game to lead the other teams in playing, and just two teams presented powerpoint lectures. Feedback from the students showed that the students liked the games better than the powerpoint presentations.

IV. Discussion

Our goal was to help students understand moral reasoning and how moral principles are related to ethical teamwork behaviors. Based on experience on teaching the UAB Computer Science Capstone course, we believe that lectures are not an effective method for presenting ethical material. We decided to use Harvard edX's online ethics videos followed by in-class discussion.

Following our minimodule tying ethical principles – fairness and reciprocity – to teamwork training, no teamwork failures were observed in EE485, a UAB Electrical & Computing Engineering class of 38 students in the fall of 2018. This compares favorably with three (3) documented teamwork failures among 37 students in CS499 in the fall of 2016. CS499, the Computer Science capstone course, requiring group work, was taught by these coauthors at the UAB Computer Science Department in 2016-7. No ethics-teamwork minimodule was offered.

Given that prior to the fall 2018, the UAB EE485 class was plagued by teamwork failures, we preliminarily conclude that the inclusion of a 4-class minimodule tying ethics to teamwork was influential or perhaps instrumental in improving a sense of fairness and accountability in group work. We observed that the guided discussions were highly animated. At semester's end, several students expressed appreciation to these co-authors as instructors of the ethics-teamwork minimodule.

V. Conclusions

Providing a brief refresher on moral principles, emphasizing fairness and reciprocity prior to assigning students to do group work in Electrical and Computer Engineering can significantly reduce teamwork failures and enhance the chances of successful teamwork.

It may also make a critical difference in enhancing the attitudes and behaviors of Electrical and Computer Engineering students, which may have a broad and lasting impact on their professional practice, employability and leadership, beyond graduation.

.

References

[1] S. Ciston. Berkeley Center for Teaching and Learning. https://teaching.berkeley.edu/news/building-teamwork-process-skills-students
[2] M. Crawford. Teaching Teamwork to Engineers. ASME Magazine, October 2012.

[3] F. de Waal, https://www.youtube.com/watch?v=GcJxRqTs5nk

[4] W. Frey. Ethics of Team Work. U Puerto Rico. (https://www.saylor.org/site/wp-content/uploads/2011/07/psych304-7.3.1.pdf)

[5] R. Lingard. Teaching and Assessing Teamwork Skills in Engineering and Computer Science. Systemics, Cybernetics, and Informatics, vol 8, issue 1, 2010, p. 34-37. (www.iiisci.org/journal/CV$/sci/pdfs/GQ816EX.pdf)

[6] R. Lingard and S. Barkataki. Teaching Teamwork in Engineering and Computer Science. Frontiers in Education Conference, IEEE, 2011.

[7] M. Sandel, justiceharvard.org

[8] R. Diaz-Sprague., A. Sprague, Software for the Soul: Our Experience Teaching Ethics to Computer Science Seniors. Presentation at the 26th APPE Annual International Conference, Dallas, TX, February 25, 2017.

[9] wheniwork.com/blog/team-building-games

[10] www.teambonding.com/five-fun-team-building-activities

Project TOMO: immediate feedback enabling service in teaching programming

Matija Lokar
Matija.Lokar@fmf.uni-lj.si
University of Ljubljana
Ljubljana, Slovenia

ABSTRACT

Teaching programming is demanding since it is not enough to explain the ideas and terms. Programming is a skill and mastering a skill requires a lot of practice and thus inevitably a lot of mistakes. Most beginner mistakes are easy to correct if the students are properly directed. If the teacher's assistance is not available immediately, the students get stuck and their progress slows down considerably. Good and immediate feedback is a key component that ensures fast success. Therefore a tool enabling quick feedback is very useful in the teaching process. It is no wonder that services providing automated programme assessment (SAAP) have become a very popular choice in programming courses.

The teacher at our Faculty agreed that using such a service could improve our teaching. Unfortunately most such services already available had considerable shortcomings. The biggest was that the feedback was more or less limited to the information whether or not the result matched the expected outcome. Therefore we developed a new web service called Project Tomo https://www.project-tomo.si/. It is completely open and available to all. There are more than 4000 programming exercises that can be adapted and re-used in new courses. At the moment the service is used by over 30 educational establishments, most of which are high schools.

The article describes the options available in the service and presents its use. It is shown how the analysis of the solution attempts can pinpoint the students' false ideas and therefore serves as the basis for improving feedback. User opinions are given as well.

CCS CONCEPTS

• **Social and professional topics** → **Student assessment**; • **Applied computing** → **Computer-assisted instruction**; • **Software and its engineering** → *Software notations and tools.*

KEYWORDS

programming, beginners, services providing automated programme assessment, web service

CSERC '19, November 18–20, 2019, Larnaca, Cyprus

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Matija Lokar. 2019. Project TOMO: immediate feedback enabling service in teaching programming. In *Proceedings of CSERC '19: The 8th Computer Science Education Research Conference (CSERC '19).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As stated in [15] "Reading all the textbooks and listening to all the best lectures cannot teach you to program a computer. Learning to program is only possible if you sit down at the computer and start writing programmes." It has to be considered that ([9]) code writing in itself is not the only important thing. It is equally important to get feedback on how the programme is written, whether it fulfils the demands of the specified problem, and if the appropriate techniques were used. Constant monitoring and assessing the students' work during the learning process ensures that the students' write a sufficient number of programmes and that they get feedback about the quality of their solutions. That is actually the primary role of the teacher. The large numbers of students in classrooms make exactly that very difficult as several studies show, including [5].

Most beginner mistakes are easy to correct if the students are properly directed. Good and immediate feedback is a key component that ensures fast success. If the teacher's assistance is not available immediately, the students get stuck and their progress slows down considerably. Therefore, a tool enabling quick feedback is very useful in the teaching process. Discovering mistakes in the syntax is not too difficult as contemporary tools provide reasonably good assistance. There are many more problems regarding semantic mistakes. The teacher cannot usually be available for all the beginners in need of assistance. Therefore, it is often the case that students are not even aware that their solution is wrong or incomplete precisely because of lack of teacher's assistance. Let us provide a simple example. Experience shows that an exercise asking the students to *Write a programme that will read a whole number and provide its reciprocal.* will often result in solutions such as the one shown in Fig. 1

The students tested their solution, got accurate results for their examples and believe that the exercise has been solved successfully. Not many of them will test their programme with 0. Here the teacher is the one who should provide the appropriate explanation. Because programming courses are usually held in larger groups, immediate intervention is not always possible and is often not sensible as there are usually several students who thought of the boundary example on their own. In such situation the real value of services providing automated programme assessment (SAAP) becomes apparent, as the students are immediately provided with feedback and reminded of the forgotten example. Of course the

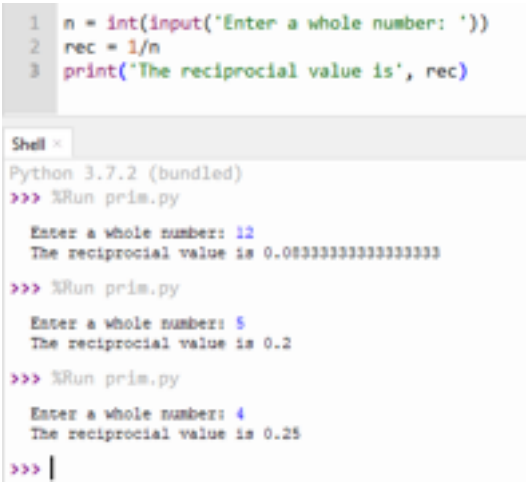


Figure 1: The example of an incomplete solution

teacher is still the key component in the process: if the prepared testing examples are not good (the teacher can just as easily forget the boundary example of 0) the service is not of much assistance.

There is a number of studies (e.g. [8, 14, 16]) describing the utility of such services. As Pieterse and Liebenberg state in [13] the SAAPs are very appropriate in formative assessment used in lab exercises.

The programming courses at our institution could be improved with the use of SAAP as well, mainly in subjects where lab exercises involve writing a programme. Thus the first version of Project Tomo web service was developed in 2010 ([19]). As it turned out it was a very practical teaching tool. The experience led to the development of the next version in 2015. In it some drawbacks of the previous version were addressed, for example the teacher can more easily follow the students' progress and prepare study materials.

It is fully open and accessible to everyone. At the moment the service is used by over 30 educational establishments, most of which are high schools. So far more than 3000 students submitted almost 900.000 attempted solutions to over 180.000 exercises, and the numbers keep growing. The bank of exercises keep growing, too, and it includes well over 4.000 different programming exercises that can be adapted and reused in different sets of exercises.

Since it has been suggested in Slovenia as well ([17]) that a basic knowledge of a programming language is part of contemporary elementary literacy, a further increase in the use of such services can be expected.

2 A SHORT PRESENTATION OF THE PROJECT TOMO SERVICE

After the decision to develop an SAAP tool was reached we checked a number of services that belong in the category. The primary focus was on the services that are used for assessment in computer competitions, because they are also most often used when teaching programming ([6]). Unfortunately most of the systems had shortcomings. The biggest issue proved to be the feedback, which usually centred on whether or not the solution matched the expected programme exactly, which excludes certain types of feedback and often requires different wording of the exercises. For example, if the assessed programmes are to provide a large number of possible results, the exercise had to ask for the results to be ordered in a certain sequence even if the order of the results was not of any considerable importance.

Because of this and several other shortcomings like the quick insight into class- and individual students' performance, the decision to develop our own service was reached. The development of the service itself and the decisions that were taken in the process as well as the reasons behind them are described in more detail in [12, 15].

The service is available at https://www.project-tomo.si/. The service requires login as teachers should track their students. This can be done with an existing account for Facebook, Google or ArnesAAI ([3]). The possibility to use the ArnesAAI account is very useful for Slovene teachers and students, as most schools in Slovenia already use the system. This means they can access the school services and Project Tomo with the same username and password which simplifies the administrative process. The teachers agree on this point: *Not having to create a new account for TOMO is great. All our students are in the e-identities management system (mdm.arnes.si) and have AAI access that they used to register for TOMO. With rare exceptions everyone could register with no difficulties. The few students who experienced difficulties in the registration process, used their Google accounts.*

At registration the user sees the first page that lists all the available subjects. The top part of the page shows the subjects that the user is already registered in, and below is the list of subjects they can register for. These are listed according to the educational establishments and projects. The materials within a subject are divided into units. Each unit consists of several exercises and each exercise has one or more sub-exercises.

The service was initially used for teaching those subjects only where Python was the chosen language, but it soon became apparent that quick automated feedback is very useful in all subjects that include code writing in the lab exercises. Therefore the service enables each exercise to be in its own programming language. At the moment three more languages besides Python 3 are supported: Octave and MATLAB that we use in numerical mathematics, and R that is used in financial mathematics and statistics.

It is not particularly hard to add a new programming language. At the moment the support for CSharp and Java is also planned.

The chosen exercise is transferred as a text file by the student. The file contains everything that is necessary for solving the exercise: the text of the exercise and the space for the student to enter the solution.

The file can be opened in any programming environment for the given language (e.g. Python IDLE, Thonny, PyCharm âĂę) and the student can immediately start working. The service has purposely been formed in such a way that it does not require learning a new programming environment in order to minimize extraneous cognitive load ([4]), a student is exposed to when having to solve a certain problem through code writing. This ensures the possibility of increasing the effects of intrinsic load ([20]).

Below the space for entering solutions is the code that is launched whenever the file is launched. This means that nothing else needs to be done to test the programme but to launch the code in the file. Then basically two things happen.

Solution test code is launched. It can be as complex as desired. A few test examples are usually used to show the student whether or not the expected results can be obtained with the student's solution. If the results are not suitable, feedback is instantly provided in the console (Fig. 2). It is important to know that this step is possible even when the student is offline. The test examples are namely launched locally, on the student's computer. This is important as it also ensures that the Project Tomo's web server can not be overloaded. That in turn eliminates the problems arising with the number of users or the attempted solutions (e. g. solutions that might start endless loops or harm system files)

If the web server is available, the student's solution is stored on it. It stores the entire history of attempted solutions for each student. This history is also used to make work easier for the students: if they had done the exercise before, the last attempted solution is automatically included in the transfer file. The students can then resume working at home without having to worry about code transfer from the school computer to the one at home. The main purpose of keeping history is that we are planning to develop tools that will enable the analysis of the history. Therefore, the teacher will be able to determine typical mistakes, possible misunderstood concepts, and show a graph representing individual exercises and groups of students. At the moment the analysis is possible, but must be done manually.

So when the exercise is downloaded, the student solves it and enters the solutions in the appropriate space. When the programme is launched, the student is notified of the successfulness of the attempt.

```
>>> %Run grupa_tnt.py
Saving solutions to the server ... Solutions saved.
1. subtask has no valid solution.
  - Expression zrna(3, 7) returns 0.42857142857142855 instead of (0, 3).
    Expression zrna(3, 7) returns a float instead of a tuple.
  - Expression zrna(32, 7) returns 4.571428571428571 instead of (4, 8).
    Have you used the appropriate operators?
  - Expression zrna(35, 7) returns 5.0 instead of (5, 5).
  Too many errors ... Further tests suppressed.
2. subtask has no solution.
3. subtask has a valid solution.
```

Figure 2: The example of an incomplete solution

In Fig. 2 we see that the student's attempt was wrong for the first sub-exercise, there has been no attempt at solving sub-exercise 2, and sub-exercise 3 was solved correctly. At the same time the markings on the web page are synchronised to reflect the student's success.

In our experience, Project Tomo's greatest strength as a teaching tool lies in the ease of the preparation of study materials, students' work analysis and following the students' progress.

3 PREPARING STUDY MATERIALS

Most subjects available on TOMO are meant to be teaching materials at a certain school (classes, extra-curricular activities ...). However, certain subjects are meant mostly as an exercise source for the teachers preparing exercise collections. In the forthcoming version of TOMO we plan to appropriately visually distinguish between those two types of subjects.

The first step towards preparing a study material (a collection of exercises) is to register as a teacher in a certain subject. The teacher's status can be assigned by one of the current teachers of the subject, or the website administrators can be asked to open a new subject. When are user has a role of a teacher, he can create a collection in which the exercises will be added. The exercises can be transferred from other subjects and used in original or adapted form or can be created from scratch.

3.1 Transferring materials from other subjects

When we are assigned teacher status in a subject, this allows us to transfer all but the hidden exercises from all other subjects. The service itself also offers several exercise banks that can be used. For example NAPOJ project ([2]) incorporated all exercises from the Informatics e-textbook ([1]) in an exercise bank, adding also a collection of exercises appropriate for extra practice. Extensive exercise banks were created also in the course of two student projects as well: in project ProNAL ([11]) and in the project PiR ([10]). Two subjects within those projects deserve to be specially mentioned: Python - a course and R - a course, where most sets are video enhanced. The videos present the students' opinions on what is essential within a certain set (e. g. on functions, loops âĂę) for the specific programming language. For the participating teachers it was a refreshing (and sometimes surprising) view on what in a certain topic is considered important in the students' eyes.

The ProNAL project is about the exercises that were included in the first group of the ACM competition in computer and information science ([18]). The exercises are divided into several subjects. All subjects use the same problems but differ according to the method of solving. This is an attempt towards enabling the teachers to lessen the cognitive load through different approaches [20]. For example, an entire subject consists of exercises that offer the student the correct solution in the wrong order. This is the so called Parson type problem approach for which studies show (e. g. ([7]) that is very useful in teaching programming languages.

Anyone who has been assigned teacher status can therefore transfer exercises from all subjects (their own and other teachers') into their own subjects. That enables the teachers to quickly and efficiently prepare exercises on a given topic even when there is little time available.

The possibility to copy exercises has been well received by teachers. However, they expressed the regret that there was no option of simply copying the link to the exercise. If it turns out that an exercise needs to be corrected, that needs to be done in every single subject where the exercise is included. If there was the option to simply copy the link to the exercise, it would only have to be corrected once. On the other hand, that means that if the author later decides to change the original exercise, all the copies are changed as well.

In one of the future versions of Project Tomo the option to copy the exercise and keeping it independently of the original exercise will be complemented with the option to using the exercise as a direct link to the original one.

3.2 Adaptation of existing exercises

If an existing exercise is not to the teacher's satisfaction, the teacher can modify it. First an editing file is downloaded. Just like the student opens the exercise, the teacher opens the editing file in the preferred word processing programme and fixes the text and then launches the file (Fig. 3). Upon launching, the file is transferred to the web server and any changes are immediately visible on the web page. The text of the exercise is written in Markdown format that allows some simple text editing.

```
# ---------------------------------------------------------------------@021309~
# Write a function minmax(a, b, c, d, e) that returns min value $m$ and max
# value $M$ of the function $ax^2 + bx + c$ an a closed interval ${d, e}$.
# Values must be returned as a tuple $(m, M)$.
#
# Example:
#
#     >>> minmax(1, 0, 1, -1, 1)
#     (1, 2)
#     >>> minmax(1, 0, 1, 1, 2)
#     (2, 5)
# ---------------------------------------------------------------------------
def minmax(a, b, c, d, e):
    candidates_x = [d, e] # Store the x-coordinates of the candidates in the array
    vertex_x = -b/(2*a)
    if d <= vertex_x and vertex_x <= e:
        candidates_x.append(vertex_x)
    values = [a*x**2 + b*x + c for x in candidates_x]  # Compute function values
    return (min(values), max(values))

Check.part()
(m, M) = minmax(1, 0, 1, -1, 1)
if M == 1:
    Check.error(("For values {0} the max value is not 1. "
                 "Did you consider the endpoints of the interval?"),
                (1, 0,1,-1,1), )
```

Figure 3: Editing an exercise

This enables quick modifications and corrections during the exercises if the teacher notices a mistake in the exercise or simply wants to formulate the exercise in a better way. When the students refresh the page, the edited version of the exercise is already available.

When an exercise is made, the official solution must be formulated. At first sight this means more work for the teacher. According to our experience, however, this approach has many advantages. It is very often the case that only when we attempt to solve an exercise we become aware of the problems in its formulation and come up with proper testing examples for the particular exercise. Besides, the official solution is visible to the students after they had entered their own correct solution (this option can be blocked, e. g. for homework assignments). Therefore, it makes sense to adapt the available official solution when we copy an exercise, in order to adapt it to our particular programming style and code formulation.

The official solution is followed by testing examples. Those are parts of code that check if the solutions (the student's as well as the official one) provide the expected results. Tests can also include checking for particular formats (e. g. the student's code must use the for loop, not the while loop). When the teachers are writing test cases, they can call the student's methods and access the student's original code. Since test writing takes a long time, a Check class is offered, which contains some handy methods of easy testing. It is possible to use simple comparisons such as whether a recall of a particular expression provides expected results. As the testing part allows programming as well, the tests can be as complex as desired. For example, the student's solution can be analysed with the use of abstract syntax trees and checks that the student's programme never recalls the method with the name det.

4 MONITORING THE STUDENTS' PROGRESS

The new version of Project Tomo a lot of work was done on the user interface. We wanted it to be simple and transparent, but nevertheless contain all the important information. The teachers should have access to the success rate of the entire class in the entire subject as well as in separate sets and each individual exercise, and for each individual student.

In order to monitor the progress, the service uses numerical data (in percent or absolute value) and colour coding. Red colour shows the number of exercises in a set the students never attempted to solve, yellow shows the number of exercises that were incorrectly solved, and green show the number of accepted solutions.

The coding is used in visual aids such as banners and pie charts. Banners are used to monitor the students' progress in sets. As teachers we see the banners above the sets on the first page. The banners are colour coded (red, yellow, green) as described above. This allows us to see as soon as we log in how successful the students were in the most current (last three added) sets. When a subject is chosen, the teachers can see the same information for all sets, and the list of students' names participating in the subject can be seen on the right hand side.

Figure 4: Success rate - group and individual

Next to the name of each student a diagram shows how successful that student is when doing the exercises in the subject. If the teachers are interested in seeing a student's success rates more closely, a click on the student's name will open the section with detailed information.

This approach has proven to be useful for determining the rate of difficulty for the exercises and for determining which type of exercise is more difficult for the students. It is often only at this stage that it becomes apparent which parts need to be revised again. If we want to see the names of the students who had succeeded in solving an exercise correctly, we click the pie chart diagram. That opens a page where we can see a list listing the names of all students that shows which of the students had solved which of the exercises.

4.1 Accessing a student's solutions

In order to monitor a student's progress successfully, the teacher must have appropriate access to the student's solutions. Project Tomo has several tools allowing that. The basic tool is the insight into the current solution of an exercise. If the appropriate circle is clicked, this shows the student's solution as well as the official solution.

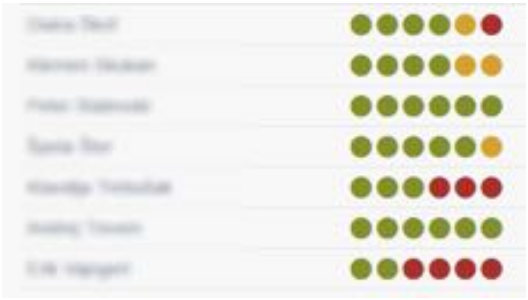


Figure 5: Success rates for individual students

A very useful option allows us to view all the student's attempts at solving an exercise. At the moment Project Tomo does not yet allow a direct option for this. Some typing is required which then takes us to the archive that contains the information of the correctness of all submitted solutions of all sub-exercises, and files arranged according to exercises that contain the last submitted solution for each student. Those files are likewise marked with correctness checking code so that the teacher can check what is wrong in each particular solution; perhaps only a simple correction of punctuation is enough. In this case, the solutions are of course not stored on the server.

The history directory contains all the student's submission in the set, arranged according to the time of the submission and containing the feedback received by the service. The teacher can see the changes in the code were made by the student between two submissions. The time of the submissions also tells us that the changes were more or less unpremeditated attempts as the time difference between both submissions is only two minutes.

5 USING THE SERVICE IN PRACTICE

The Project Tomo service has been used in different courses for quite a while at the Faculty of Mathematics and Physics, at Ljubljana University. Besides programming courses and courses in data structures and algorithms, the service is also used in courses in numerical mathematics and financial mathematics. It turned out to be an effective tool for lab exercises. Most allow the students to do exercises on their own and at their own speed. This allows the teaching assistant to deal with the "real" problems and directs the students, explains certain concepts when necessary, and does not need to waste the time for solving minor issues.

The option that allows quick fixes and adaptations of an exercise or test has proven to be extremely useful. It has often been the case that an exercise was found to be poorly formulated or lacking a test case during the course of the lesson itself. The design of the service allows fast changes and adaptations of exercises and the changed exercise is immediately available to the students.

Another one of the qualities of the system is that it takes over the role of the cranky teacher who insists that the solutions must correspond precisely to the exercise. This can be clearly seen from the letter we had received from a professor at one of Slovene grammar schools: "What bothered them was that they got an orange or even a red mark for the solutions that gave correct results but did not use the path specified in the exercise or even if they simply named the variables with different names than specified. Their progress was then incorrectly marked."

Of course that seemed odd, as the whole idea behind the service is that all solutions are accepted if they withstand the testing procedure. Therefore, the professor was asked to specify the exercise where that happened. It was one of the exercises in the new e-textbook for subject Informatics in grammar schools [1]. Comparison of the provided solution and the student's solution shows that the solutions differ in a single capital letter. The student wrote "enter" without the capital, but the exercise asked for the capitalization of the sentence. Of course we could debate whether or not the exercise is overly exact, but let us assume that it is not.

This is yet another instance that proves that Project Tomo is just a teacher's tool. The teacher is the one who determines what is important and what is not. The exercise above offers several different options. Let us mention four of them.

(1) The exercise can be left as is. The purpose is to make the students read the instructions carefully and stick to them precisely. Project Tomo can help us with that immensely, as we do not have to explain to each individual student that the solution is invalid because of a single capital letter. The service simply does not accept the solution as correct and the exercise is marked as incomplete.
(2) The test can be changed so that the student is warned of the mistake in the capitalization in the feedback provided by the system, which tells the student that a common mistake was made but does not "punish" the student for it.
(3) The test can be adapted to accept mistakes in the capitalization. This makes sense if the stress in the exercise lies elsewhere and the displayed text is less important.
(4) The test can be changed to accept any form of imperative the student uses.

Most important is the teacher's ability to react to any situation that arises in the teaching process. That is exactly the reason behind the development of Project Tomo: to lessen the teacher's load and remove the menial tasks, thus providing additional time for the teacher âĂŞ student communication during lab exercises. We firmly believe that this enables in-depth debate between the teacher and the students as the teacher does not have to devote so much time to predictable mistakes.

5.1 Teachers' opinions on Project Tomo

Project Tomo is used by many high school teachers in their programming courses. In order to improve the service even further, they were asked to share their opinion of the service. Some opinions have already been presented in the article, some more interesting answers follow. The authors are all professors at Slovene grammar schools.

- *"The tool is especially useful when teaching Matura students as their knowledge and aptitude regarding programming differ to such extent that attempting to teach them in a conventional manner could only be considered a hopeless task."*
- *"I used to work along the lines of exercise - hint - solution. Tomo, however, is something else. Now I can finally devote my*

time to a struggling student individually while the others can continue to work at their own pace"

- *"I am impressed with the system's insistence on the exact formulations, which is often difficult for the students and teaches them to read the instructions carefully and to fulfil the demands exactly."*
- *"Both my students and I have extremely positive experience with the system. The students like that they can do the exercises at their own pace or even at home. In the meantime, I can devote more time to the beginners."*
- *"A tool such as Project Tomo is very welcome in lessons. In the first place it simplifies the work immensely. Perhaps a banal thing, but it is handy that exercises do not have to be individually searched for and then projected on the board, as the exercises are already well written and include all the necessary instructions."*
- *"The fact that they could see their progress was good motivation, and for me it meant that I could quickly see what someone has done and what has not been done yet."*
- *"Tomo has proved a very efficient tool in programming lessons. The initial time input is quite extensive, but that means a lot less work and a much better overview of the students' work in the end. It is also much easier to work with students who come to the course at very different levels of knowledge."*
- *"The TOMO web service enables me to assign more work to those students that wish to work more, and to devote more time to the students who need more practice and additional explanations. It is TOMO that provides the basic feedback."*
- *"Another important thing to consider when using Tomo is the large amount of available exercises. The existing exercises are also easily adapted to your own needs without actually changing the original exercise in the system."*

6 CONCLUSION

Project Tomo makes work easier for the teachers in several ways described above. Thus the teachers can devote more of their time to the preparation of the study materials and the teaching itself. It aids the students, too. They are given instant feedback regarding the correctness of their solutions which enables faster progress.

Tools such as Project Tomo will probably become widely useful, as even the public opinion regarding the knowledge of how to program a computer is slowly changing towards the notion that at least a basic familiarity with programming is becoming necessary, especially when it comes to the way of thinking. As the experience in other countries shows, the main problem for the teaching of programming seems to be the lack of good teachers. Of course, no such service can replace teachers. However, it can help the teachers to teach large groups of students and to focus on the struggling students who need help.

The aforementioned experiences with using the Project Tomo service in Slovene grammar schools show us that the service eases the teachers load. Thus the teachers can devote more time to preparing study materials and the teaching itself. It is of especially great help when the groups consist of students with different levels of existing knowledge. The teacher can more easily focus on individual students, as beginners have very different needs from the more proficient programmers.

REFERENCES

[1] Anželj, G., Brank, J., Brodnik, A., Bulic, P., Ciglarič, M., Dukic, M., Sterle, P. (2018). Računalništvo in informatika v2.12; E-učbenik za informatiko v gimnaziji. https://lusy.fri.uni-lj.si/ucbenik/book/index.html

[2] Anželj, G., Brodnik, A., Lokar, M. (2018). NAPOJ - proti aktivni skupnosti učiteljev računalniških predmetov. Vzgoja in izobraževanje v informacijski družbi - VIVID 2017: zbornik referatov. Ljubljana: VIVID.

[3] Arnes AAI. (2017). https://aai.arnes.si/

[4] Chandler, P., Sweller, J. (1996). Cognitive Load While Learning to Use a Computer Program. Cognitive Psychology.

[5] Corbett, A. T., Anderson, J. R. (2001). Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes. Proc. of the SIGCHI Conf. on Human Factors in Computing Systems (CHI '01) (pg. 245–252). New York: ACM.

[6] Duch, P., Jaworski, T. (2018). Dante - Automated Assessments Tool for Students' Programming Assignments. 11th International Conference on Human System Interaction (HSI).

[7] Ericson, B. J., Margulieux, L. E., Rick, J. (2017). Solving parsons problems versus fixing and writing code. Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling '17).

[8] Higgins, C. A., Gray, G., Symeonidis, P., Tsintsifas, A. (2005). Automated assessment and experiences of teaching programming. J. Educ. Resour. Comput.

[9] Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10). ACM.

[10] Javni sklad Republike Slovenije za razvoj kadrov in štipendije. (2018). Po kreativni poti do znanja (PKP). http://www.sklad-kadri.si/si/razvoj-kadrov/po-kreativni-poti-do-znanja-pkp/

[11] Javni sklad Republike Slovenije za razvoj kadrov in štipendije. (2018). študentski inovativni projecti za družbeno korist (šIPK). http://www.sklad-kadri.si/si/razvoj-kadrov/studentski-inovativni-projecti-za-druzbeno-korist-sipk/

[12] Jerše, G., Lokar, M. (2018). Uporaba sistema za avtomatsko preverjanje nalog Project Tomo pri učenju programiranja. Vzgoja in izobraževanje v informacijski družbi - VIVID 2017 : zbornik referatov. Ljubljana.

[13] Pieterse, V., Liebenberg, J. (2017). Automatic vs manual assessment of programming tasks. Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling '17) (pg. 193–194). ACM.

[14] Poon, C. K., Wong, T.-L., Yu, Y.-T., Lee, V. C., Tang, C. M. (2016). Toward More Robust Automatic Analysis of Student Program Outputs for Assessment and Learning. IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), (pg. 780–785).

[15] Pretnar, M., Lokar, M. (2015). A Low Overhead Automated Service for teaching Programming. Proceedings of the 15th Koli Calling International Conference on Computing Education Research. Koli, Finland: Proceedings of the 15th Koli Calling Conference on Computing Education Research. doi:https://doi.org/10.1145/2828959.2828964

[16] Rajala, T., Kaila, E., Lindèn, R., Kurvinen, E., Lokkila, E., Laakso, M.-J., Salakoski, T. (2016). Automatically assessed electronic exams in programming courses. Proceedings of the Australasian Computer Science Week Multiconference (ACSW '16). ACM.

[17] RINOS. (2018). Snovalci digitalne prihodnosti ali le uporabniki? https://fri.uni-lj.si/sl/novice/novica/uporabniki-ali-snovalci-digitalne-prihodnosti

[18] Tekmovanje ACM iz računalništva in informatike. (2018). http://rtk.ijs.si/

[19] UL FMF. (2010). Project Tomo. https://www.project-tomo.si

[20] Wilson (ed.), G. (2018). Teaching Tech Together - Cognitive Load. (Lulu.com) http://teachtogether.tech/en/load/

Static Detection of Design Patterns in Class Diagrams

ED VAN DOORN, The Hague University of Applied Sciences
SYLVIA STUURMAN, Open University of the Netherlands
MARKO VAN EEKELEN, Open University of the Netherlands and Radboud University

Teaching Object-Oriented design on the class diagram level is often a cumbersome effort. Requiring the use of specific design patterns helps the students in structuring their design properly. However, checking whether students used the right design pattern can be a very time-intensive task due to the variety of possibilities of creating structure using design patterns on the high-level of class diagrams. For the same reason, it is hard for students to check for themselves whether their solution fulfills the basic requirements that are required by the teacher with respect to the use of design patterns. Efficiency and the quality of design pattern education can be improved by automatic detection of design patterns in UML class diagrams. We introduce a new method to detect design patterns in class diagrams, together with a prototype of a tool which uses this new method. Using this tool, a teacher needs less effort to review solutions of design exercises since the tool can check the basic class requirements automatically. Consequently, a teacher can focus on the more high-level requirements that were set in the exercise and students can easier check for themselves whether their design satisfies the basic required properties on the pattern level.
The method offers *static decidability* for those design patterns, that are identified by structural properties i.c. the names of the classes and their associations. It is *non-duplicating*, i.e. a specific occurrence of a design pattern is not reported multiple times. The method not only detects all 17 *static* Gang of Four design patterns without false positives or false negatives, but also it can detect redundant relations. The prototype, our tool can contribute to the quality and efficiency of design pattern education, both for students and for teachers.

Additional Key Words and Phrases: efficiency of learning and education, design pattern detection

ACM Reference Format:
Ed van Doorn, Sylvia Stuurman, and Marko van Eekelen. 2019. Static Detection of Design Patterns in Class Diagrams. 1, 1 (November 2019), 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Ed van Doorn, The Hague University of Applied Sciences, The Hague; Sylvia Stuurman, Open University of the Netherlands, Heerlen; Marko van Eekelen, Open University of the Netherlands, Heerlen, Radboud University, Nijmegen.

XXXX-XXXX/2019/11-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In many educational situations, automatically detecting design patterns in a class diagram would be of value. For instance, teachers could be supported by their evaluation of designs that students send in [SF04]. This support will save time for teachers. It also guarantees equal assessment and reduces the amount of monotonous work. It would be also useful as an aid in a class diagram editor for students, to check whether they have represented an intended design pattern. Feedback on simple mistakes would also be welcome.

In an educational environment, a tool that uses UML class diagram as input and can detect design patterns is wanted. Automatically detecting design patterns based on structural properties can be generalized to automatically detecting *any pattern* that consists of classes and their relations. For that purpose, obligatory larger *static* parts may be defined. This paper describes the contributions:

- The new theoretical concepts: static, and non-static design patterns, static decidability, generally complete, non-duplicating.
- A new subdivision of design patterns based on these concepts.
- A new method for detecting design pattern based on these concepts.
- A prototype of a tool which uses this method. This tool can detect all 17 Gang of Four static design patterns and is non-duplicating.
- Detecting design patterns which partially exist.
- Generating limited feedback.

The tool [1] is publicly available. Our paper is structured as follows. We define a new concept in section 2: *static* design patterns, that defines a set of design patterns that are completely defined by their class names and their relationships. We also define the concept of *static decidability* for algorithms that can detect *static* design patterns. The concept of *non-duplicating* is also defined in section 2. For implementation of an algorithm that can detect all *static* design patterns in a UML class diagram, several requirements are wanted. First of all, it should make no errors and so, only detect true *static* design patterns. Templates for *static* design patterns should be easily definable. It should also detect permutations of design patterns and multiple occurrences of one design pattern only ones, i.c. the algorithm should be *non-duplicating*.

In section 3, we describe earlier approaches to detect design patterns in UML class diagrams. Here, we show that none of these approaches meet our requirements. Some of these approaches can detect all *static* design patterns, but these approaches have shortcomings, e.g. they are not *non-duplicating*.

In section 4, we describe our approach in detail. We explain how the above requirements are fulfilled. Our prototype uses 3-tuples for representing class diagrams.

[1] http://members.chello.nl/e.doorn1/DesignPatterns/static_decidability

We describe the results of our prototype in section 5. We show that the implementation of our method results in a prototype which works in practice. The prototype can detect all 17 *static* Gang of Four [GoF] design patterns [GHJV95], because they are fully defined by their class/interface-names and their relations. The number of false positives and negatives, the speed of detection and the improvements of the detection method are discussed. It can read the XMI representation of a class diagram by ArgoUML, the drawing tool for class diagrams and reads templates for design patterns in XML-format. The results are compared to the result of other tools.

In section 6, we refer to articles about four subjects. Experiences about education at university level related to design patterns is clearly an important issue. Several approaches for detecting design patterns in source code are mentioned. Also, an approach to detect anti-patterns is referenced. Research on the relation of occurrences between design patterns and code smells is referenced.

In section 7, we describe our conclusion. The theoretical and practical advances are recalled. The results are summarized as well as the limited feedback.

In section 8, we show the limitations of our approach and give suggestions for future research. In particular, we mention *generally complete* detection algorithm, improved detection for the Abstract factory, improved feedback and measuring the quality of UML class diagrams.

2 STATIC / STATIC DECIDABILITY / GENERALLY COMPLETE / NON-DUPLICATING

In this section, we introduce a new subdivision of the set of all design patterns into two new complementary subsets: *static* and *non-static*. Each subset is related to a type of detection algorithm, that offers: *static decidability* or *generally complete*. The approaches to detect design patterns in section 3 are labeled by these new concepts.

2.1 New categorization for Design patterns and detection methods

Traditionally the set of all design patterns is subdivided by their use. Creational patterns are used to create complex objects. Behavior patterns are used to divide and assign responsibilities to classes and to describe the communication between objects. Structural patterns are used to associate classes to bigger structures [GHJV95].
A second subdivision is based on analyzing source code. Detection algorithms can use static analysis, during which the code is not running and during dynamic analysis when the code is running. The first subset of all design patterns is *static structural patterns*, which are detectable based on their classes and relationships. *Dynamic behavior patterns*, which are detectable based on the interaction between objects. This subset of design patterns can be detected by a combination of static and dynamic analysis. *Program-Specific patterns*, which are detectable based on specific keywords and code styles. This last subset of design patterns can also be detected by a combination of static and dynamic analysis [LYL08].
A third subdivision resulted in PINOT, an automated detection tool, which uses the source code as input[SO06]. The subdivision consists of five subsets. The first subset is based on the programming language. Java provides the Observer and Iterator pattern. The second subset contains structure driven patterns, that are fully defined by the relationships between classes. The third subset are behavior driven patterns. These patterns model behavior aspects. Examples are Singleton, Strategy, State, Factory method and Decorator. The fourth subset is the domain specific patterns. The Interpreter and Command combined with the composite and visitor design pattern are used for specific languages. The fifth subset is used for general concepts. The Builder and Memento design pattern constitute this subset. Their structure is detectable, but their behavior aspects are hard to detect. PINOT can detect structure and behavior driven patterns. PINOT could easily be extended to detect patterns, which are provided by the language. It would only be necessary to detect some keywords.
Another example of the third subdivision is described by Bernardi [BCDL14]. He uses a domain specific language to define patterns and source code. A graph matching search method is used to detect design patterns. The tool can distinguish between the State and Strategy patterns which are structurally identical. The Singleton pattern is detectable and variants of design patterns are also detectable, but the recall and precision of the tool is not 100%.
The first subdivision is not helpful to create detection methods, because it does not give any information about easily identifiable characteristics of design patterns. The second and third subdivision are based source code analysis, but inspired us to give a subdivision for UML diagrams. When we first look at a class diagram of a design pattern, we see a number of classes and relationships, which concepts are easily identifiable. This leads to the following definitions.

Definition 2.1. A design pattern is *static* if it is completely defined by the names of their participating classes and their relationships.

The Adapter pattern is a simple example because the pattern consists of a few classes with one association and one inheritance relation. See Figure 7. A Singleton pattern is not a *static* design pattern, as explained after the next definition.

Definition 2.2. A design pattern is *non-static*, if it needs more characteristics than names of their participating classes and relationships to be defined.

An example is the Singleton pattern. The Singleton class needs a static operation which returns the unique object and a static attribute, which contains the unique object or a null-value.

From an educational point of view, one could state that modeling design patterns which are *non-static* needs more attention and effort than modeling design patterns which are *static*.

Definition 2.3. A detection algorithm offers *static decidability*, if it can detect all *static* design patterns.

Such an algorithm can detect e.g. the Prototype pattern, see Figure 6 and the Adapter pattern.

Definition 2.4. A detection algorithm is *generally complete*, if it can detect all design patterns.

This type of algorithm offers more than those that offer *static decidability*. It is not only able to detect e.g. the Adapter and the Prototype pattern, but also e.g. the Singleton pattern.

The relations between these definitions is depicted in Figure 1.
The purposes of the classical subdivision and our subdivision differ. The classical subdivision, creational, behavioral, and structural patterns is directed to *using* design patterns. Our subdivision is directed to *detecting* design patterns. The second subdivision is similar to our subdivision, but they also differ. Both subdivisions use static characteristics. However, the second subdivision is based on source code, which have to run to detect dynamic behavior. Our subdivision is based on design characteristics.

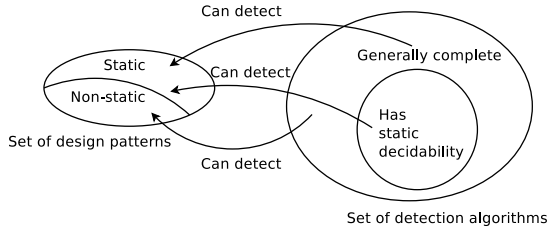


Fig. 1. Relation between definitions

The *static* patterns can be shown in a UML structure diagram e.g. a class diagram. In contrast to a class diagram, names of attributes and signatures of operations are not involved in the definition of *static decidability*. An interface can be regarded as a class without the implementation of methods, and a class may be abstract. When to decide whether a design pattern is *static* or not, the differences between the concepts: class, abstract class, and interface are irrelevant. Relationships are (directed) association, aggregation, composite, inheritance, realization and dependency. Potential multiplicities of relationships are irrelevant, because they are not used by Gamma et al. [GHJV95].

Table 1 denotes the subdivisions: the *static* and *non-static* Gang of Four design patterns. Design patterns to which an *asterisk* is added, such as Adapter, contain attributes and/or operations in their definition giving by Gamma et al. [GHJV95]. Their static structure is unique and their given attributes and/or operations have minor influence on the intention of the design pattern. So they are considered to be *static*. However, false positive detection of a design pattern remains possible. For instance, the Factory Method pattern has an uniq static structure. So, a method which offers *static decidabilty* will detect an occurrence of a Factory Method pattern. But, an occurrence of a Factory Method needs to create an object. Without creating an object, a false negative will be generated.
There are 23 GoF design patterns, of which 17 are *static* and so an algorithm that offers *static decidability* is sufficient to detect them. For the remaining 23 - 17 = 6 design patterns, we give arguments as to why algorithms that offer *static decidability* are not capable to detect them.

- Façade: Any pattern where one class has connections with at least two other classes, would be a façade pattern. So, a false positive detection would be likely. More information is needed to decide whether this pattern is a façade pattern.
- Prototype: The operation `clone` is essential for detection. So, class/interface-names and their relations do not contain sufficient information to detect this pattern.
- Singleton: For detection, a static operation returning the value of a static attribute, is necessary. So, class/interface-names and their relations do not contain sufficient information to detect this pattern.
- State pattern: This pattern is structurally identical to the Strategy pattern. So, more information is needed to distinguish these patterns.
- Template Method: The Template Method can only be detected by taking operations into consideration.
- Visitor: The number of operations of the interface visitor should be equal to the number of classes that implements the interface element.

Static	**non-static**
Creational Patterns	
Abstract Factory	Prototype
Builder∗	Singleton
Factory Method∗	
Structural Patterns	
Adapter*	Façade
Bridge*	
Composite*	
Decorator*	
Flyweight*	
Proxy*	
Behavior Patterns	
Chain of reponsibility*	State/Strategy
Command*	Template Method
Interpreter	Visitor
Iterator*	
Mediator	
Memento*	
Observer*	

Table 1. Subdivisions of Gang of Four design patterns

In anticipation of the problems in section 4.4, which describes the multiple detection of one occurrence of a design pattern, we introduce a definition of a type of detection algorithms with a higher quality.

Definition 2.5. A detection algorithm is *non-duplicating*, if it detects every occurrence of a design pattern only once.

3 INTRODUCTION TO DETECTION APPROACHES

This subsection gives an overview of detection methods based on representations of UML class diagrams. It is indicated whether they are generally complete or offer *static decidability* or not. Whether they are *non-duplicating* or not, is also denoted. Approaches are

explained by an example consisting or the Prototype design pattern, which is searched for in a UML class diagram of an example system. See Figure 2, which represents the essential classes and relationships of the Prototype design pattern. Figure 3 represents the example system, which contains the Prototype design pattern.

3.1 Matrices

One technique is to represent design patterns as a matrix [DSZ08, TCSH06]. For every type of relationship in a class diagram, e.g. association, inheritance, and dependency, a matrix denotes the relationship. If a relationship (for instance, an association) exists between two classes, the corresponding matrix element is one; if not, it is zero.

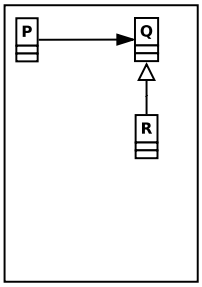


Fig. 2. The Prototype pattern

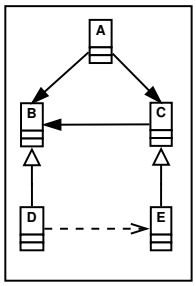


Fig. 3. Example of a system

As an example, Table 2 shows for figure 3 the corresponding matrices for the associations, inheritances, and dependencies. Class A is associated with class B, is represented by the number 1 in row A and column B in the association matrix. Likewise, class D inherits from class B, is represented by the number 1 in row D and column B in the inheritance matrix, and class D depends on class E, is represented by the number 1 in row D and column E in the dependency matrix.

Association matrix						Inheritance matrix						Dependency matrix					
	A	B	C	D	E		A	B	C	D	E		A	B	C	D	E
A	0	1	1	0	0	A	0	0	0	0	0	A	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0	0	0	B	0	0	0	0	0
C	0	1	0	0	0	C	0	0	0	0	0	C	0	0	0	0	0
D	0	0	0	0	0	D	0	1	0	0	0	D	0	0	0	0	1
E	0	0	0	0	0	E	0	0	1	0	0	E	0	0	0	0	0

Table 2. matrices representing Figure 3

Similarly, abstract classes and interfaces can be represented by matrices. If a class is abstract, its corresponding diagonal element of the abstract matrix is set to one. These matrices can be combined, resulting in one overall matrix [DSZ08]. The overall matrix is constructed as follows. The association matrix is as associated with the number 2, inheritance matrix is associated with 3, and the dependency matrix is associated with 5. The value of the elements of the overall matrix in Table 4 is defined by

$$overallMatrix_{i,j} = 2^{associationmatrix_{i,j}} * 3^{inheritancematrix_{i,j}} * 5^{dependencymatrix_{i,j}}$$

In Table 4 the value 2 in row A and column B is calculated by:

$$overallMatrix_{1,2} = 2^{associationmatrix_{1,2}} * 3^{inheritancematrix_{1,2}} * 5^{dependencymatrix_{1,2}} = 2^1 * 3^0 * 5^0 = 2$$

The association matrix shows in row A and column B the value 1.

An overall matrix is likewise constructed for a design pattern. For Figure 2 the overall matrix is Table 3.

Overall matrix

	P	Q	R
P	1	2	1
Q	1	1	1
R	1	3	1

Table 3. Overall matrix for Figure 2

Overall matrix

	A	B	C	D	E
A	1	2	2	1	1
B	1	1	1	1	1
C	1	2	1	1	1
D	1	3	1	1	5
E	1	1	3	1	1

Table 4. Overall matrix for Table 2 and so Figure 3

To determine whether a design pattern is present in a class diagram, one has to compare the overall matrix of the design pattern to the overall matrix of the class diagram. In example, compare Table 3 with Table 4. This can be done by cross-validation [DSZ08]. Alternatively, this can be done by using Blondel's or Zager's algorithm [TCSH06, BGH+04, ZV08]. This will result in the solution as shown in Table 5.

	Solution		
	1	2	3
P →	A	C	A
Q →	B	B	C
R →	D	D	E

Table 5. Solutions of comparing Table 3 and 4

By representing design patterns by a matrix, one can detect at least 10 GoF design patterns. When one searches for one of the patterns that can be detected, there are, with some exceptions, no false negatives. These exceptions involve permutations and interpretations of the Factory Method and the State pattern. So, this approach is focused on *static* design patterns and offers *static decidability*. The approach is not *non-duplicating* because permutations of one occurrence are not interpreted as one.
The authors did not give any details about the time to detect design patterns.

3.2 Decision trees

This subsection describes the use of decision trees as an approach to find design patterns. This approach also uses matrix representations of design patterns. Instead of comparing overall matrices as in section 3.1, a direct search that compares individual matrices one by one is used. This approach results in a decision tree [TCHS05]. The approach starts with constructing 7 matrices and one vector for representing 20 design patterns. The matrices are used for representing associations, aggregations, generalizations, instantiations of objects, method parameter references, similar method invocations, and abstract method invocations. The vector is used to indicate

whether a class is abstract or is an interface.
The design of the system under consideration is also described by these 7 matrices and one vector.
Some patterns contain another pattern. For instance, the Abstract Factory contains the Factory Method. So the search for these patterns can be combined. If a Factory Method is detected then the search can be continued to detect an Abstract Factory.
The search for Interpreter, Proxy, Composite and Decorator patterns can be combined [TCHS05], because their structures resemble partly.
These examples demonstrate that the detection of a design pattern can be based on decisions about continuing a search or differences between patterns. These decisions form a decision tree.
The authors claim that their approach can detect 20 out of 23 GoF design patterns. This approach does not offer *static decidability* because instantiations of objects, method parameter references, similar method invocations, and abstract method invocations are taken into account. It is unknown whether this approach is *non-duplicating*, because the authors did not pay attention to the possibility of multiple detections of one occurrence of a design pattern.

3.3 Prolog clauses

UML class diagrams may also be represented by Prolog clauses and rules. In that case, classes and relationships are represented by clauses, while design patterns are represented by rules.

As an example, the Prototype pattern (see Figure 2) can be represented by rules, as follows.

```
prototype(P, Q, R):-
class(concrete, P),
class(concrete, Q),
class(concrete, R),
association(P, Q),
inheritance(Q, R).
```

Here, P, Q and R are the names of classes or interfaces, and the rules specify whether a class should be concrete (or abstract or an interface), and which relationship should exist between which classes.

Prechet et al. demonstrate this approach for the patterns Adapter, Bridge, Component, Decorator, and Proxy. The estimation is that there are no false negatives, but there are false positives [PK98]. This approach has also been used to detect the following patterns: Factory Method, Prototype, Abstract Factory, Composite, Decorator, Adapter, Bridge, Proxy, Observer and Iterator [BP00].

An advantage of this approach is that it is possible to generate critique on the detected patterns, involving the names of classes, attributes, operations, scope of operations missing operations, operations that may prevent reusability, and suggestions for the implementation of the pattern. The critiques can be shown in ArgoUML [BP00].
The authors did not give any information about the speed, and the number of false negatives and false positives. It is not clear, whether this approach offers *static decidability*. The description of the representation of a class and relationships suggests a *static decidability* approach, but the authors claim the detection of the Prototype pattern, which indicates an approach that offers more. It is unknown whether this approach is *non-duplicating* or not, because the authors did not pay attention to the possibility of multiple detections of one occurrence of a design pattern.
A test has shown, that 70 design patterns were detected in 2 seconds on a PC with a Pentium P133 under Windows 95. The precision was ± 14%.

3.4 Four Tuples

Another approach to represent a class diagram is to use 4-tuples [BBQ14]. A 4-tuple (A, B, T, S) represents the relationship between classes or interfaces A and B. T stands for the type of relationship. T is an integer and may stand for direct association, dependency or generalization. S is a boolean, to indicate the existence of a self-loop (a class which has an association with itself).
However, our research shows that for none of the 17 *static* GoF design patterns is it necessary to use the boolean self-loop element of a 4-tuple to define or detect a *static* design pattern.

Table 6 shows the possibilities for T.

type of relations	
1	direct association
2	dependency
3	generalization

Table 6. Representation of types of relations

This representation has not been implemented, but a straight forward depth-first search algorithm is expected to match the four tuples of a design pattern with a subset of four tuples representing the system under consideration. This search will detect all design patterns in a UML class diagram, that are fully defined by their class names and their relationships. So, this approach offers *static decidability* and is unknown whether this approach is *non-duplicating*.

Summarizing, we found five ways representing structural elements in a UML class diagram and approaches to detect design patterns: matrix-based combined with crosshypvalidation, matrix-based with a decision tree, Prolog clauses, sum of products and 4-tuples.
Implementations exist for matrix-based combined with cross-validation and Prolog clauses, but is not certain whether they of *static decidability*. For the other three approaches, the four tuple approach has several advantages (see section 1): it will not result in false negatives or positives because an exact search is used, design patterns can easily be defined, a detection algorithm seems to be implementable. It is not clear whether one of the three approaches can detect permutations and multiple occurrences of design patterns.

4 OUR APPROACH

Our approach for detecting *static* design patterns is based on using 3-tuples [Doo16]. To make this approach useful in an educational and professional environment, several features are implemented. We show the representation of templates of design patterns and class diagrams. Templates of design patterns are described by XML. The drawing tool ArgoUML transforms class diagrams to XMI. The XML- and XMI-files are used as input of our prototype, see Figure 6. The problem of multiple detections of one occurrence of a design pattern is explained. We also explain the solution to this problem.

3-tuples DP	(P, **Q**, 1, 0)	(R, **Q**, 3, 0)
SYS 1	(A, **B**, 1, 0)	(D, **B**, 3, 0)
SYS 2	(A, **C**, 1, 0)	(E, **C**, 3, 0)
SYS 3	(C, **B**, 1, 0)	(D, **B**, 3, 0)

Table 8. Corresponding relationships between DP and SYS

Detection of illegal relationships within a detected design pattern are described. Finally, we pay attention to detecting design patterns that partially present in a class diagram

4.1 3-tuples

A *static* design pattern is completely defined by the names of their participating classes and their relationships. Representing a design pattern we need the two names of the classes/interfaces, which are associated and the type of relationship. Possible types of relationship are directed association, inheritance, aggregate and dependency.

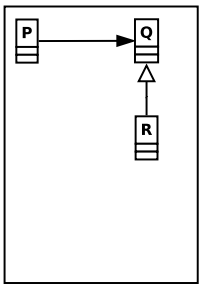


Fig. 4. The Prototype pattern

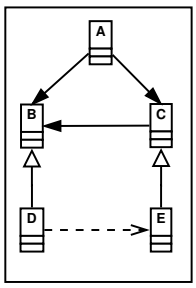


Fig. 5. Example of a system

As an example, we repeat Figures 2 and 3 by Figures 4 and 5. We show the 3-tuples for Figure 4 and Figure 5 in Table 7. The table shows two sets of 3-tuples: SYS and DP, representing the system under consideration and the prototype pattern, respectively.

SYS =	{ (A, B, 1), (C, B, 1), (A, C, 1), (D, B, 3), (E, C, 3), (D, E, 2) }
DP =	{ (P, Q, 1), (R, Q, 3) }

Table 7. 3-tuples for Figures 2 and 3

The match between the 3-tuples of SYS and DP is shown in Table 8. The first row shows the two 3-tuples of DP. Each of the subsequent rows show occurrences of corresponding 3-tuples in SYS. An occurrence means that there is a combination of three classes or interfaces of SYS that are can be mapped on both 3-tuples of DP.

We use a recursive depth-first search algorithm to detect the occurrences of DP in SYS. The resulting table is Table 8.

The first row shows the two 3-tuples of DP. Each of the subsequent rows show occurrences of corresponding 3-tuples in SYS. An occurrence means that there is a combination of three classes or interfaces of SYS that are can be mapped on both 3-tuples of DP. We show, by making the names of a class bold, which classes should be the same in the two 3-tuples in a row. E.g. SYS 1 is one of the three occurrences of the Prototype pattern. The map of the classes of DP into SYS is: P → A, Q → B and R → D. See also Figures 2 and 3.
A recursive search algorithm tries to match all the tuples of DP one by one. The recursive search starts with a randomly chosen tuple of DP. If a match of this tuple in SYS can be found then the second recursive call tries to match another tuple and so on.

In general, DP contains several tuples. For performance reasons, every time a tuple of DP is chosen, it should connect to already chosen tuples of DP. Therefore, the chosen tuples of DP always constitute a connected graph. The number of tried matched would be enormous when the chosen tuples would not form a connected graph. For example, in the first row of Table 7 the classes P and Q are matched with the classes A and B. In the next step only one of the tuples (C, B, 1), (A, C, 1), (D, B, 3) can be chosen, because the A or B matches.

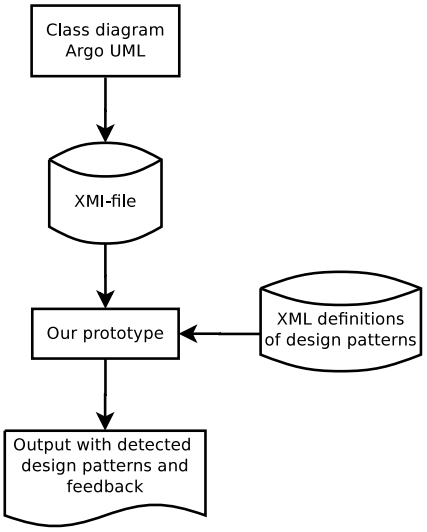


Fig. 6. Prototype structure

4.2 The prototype tool with ArgoUML as input

The structure of our prototype is depicted in Figure 6.
We used ArgoUML [2] to create UML class diagrams and to generate an XMI-file. A generated XMI-file contains all information of a UML class diagram, which is needed by an algorithm that offers *static*

[2]http://argouml.tigris.org/

decidability. We used XML templates to represent the definitions of design patterns. The structure of XML templates is described in paragraph 4.3.

4.3 Design pattern definions by XML

To detect different design patterns during one run of the prototype, we defined an XML-structure.

The XML-template defines the Adapter and Memento patterns, which are shown in the Figures 7 and 8.

```
<xml version="1.0" encoding="UTF-8"?>
<templates>
<!--
<edge ... />
is a threetuple
node1 and node2 are a (abstract)class or een interface
type describe the type of association  between node1 and node2
-->

<template name ="Adapter">
<edge node1="Client"    node2="Target"
type="ASSOCIATION_DIRECTED"/>
<edge node1="Adapter"   node2="Target"
type="INHERITANCE"/>
<edge node1="Adapter"   node2="Adaptee"
type="ASSOCIATION_DIRECTED"/>
</template>

<template name ="Memento">
<edge node1="Memento"    node2="Caretaker"
type="AGGREGATE"/>
<edge node1="Originator" node2="Caretaker"
type="DEPENDENCY"/>
</template>
</emplates>
```

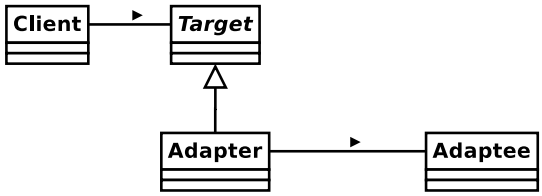


Fig. 7. Adapter pattern

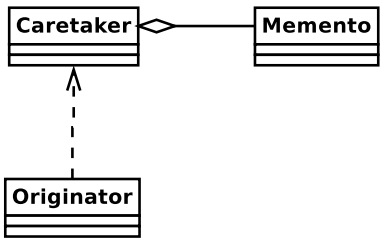


Fig. 8. Memento pattern

This overall structure of a template of design patterns can be easily parsed by using the standard SAX-parser of Java. So, every pattern can one after the other be fed to the method `findMatch`.

4.4 The problem of false multiple occurrences

A naive approach of our algorithm could detect a single occurrence of a design pattern several times. We will exemplify two different reasons, one for the Abstract Factory and one for the Bridge pattern. We will also describe how these problems are solved in general and therefore we developed a *non-duplicating* algorithm that offers *static decidability*.

In Figure 9, we see one Abstract Factory.

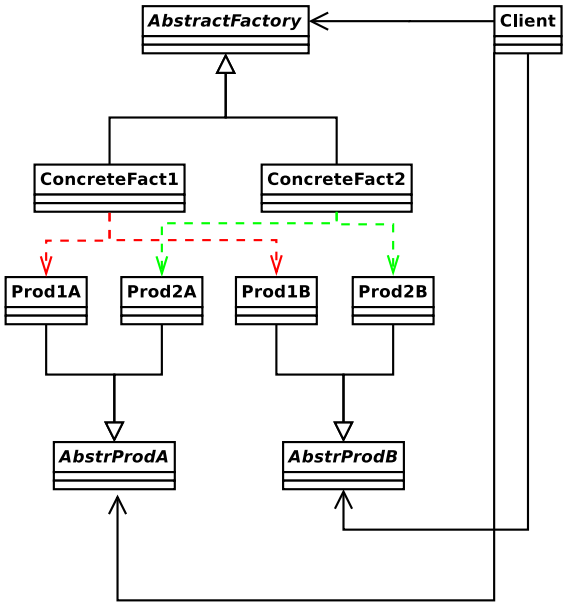


Fig. 9. Abstract Factory pattern

Our first approach detected the Abstract Factory pattern four times in this system, because *Prod1A* and *Prod2A* can be interchanged, and *Prod1B* and *Prod2B* can be interchanged. Finding four instances of the pattern is obviously wrong because the four matches are permutations of one match.

Therefore, we improved the algorithm by adding a call to `isUniqueSolution`. These permutations of matches can easily detected. Permutations have two relevant properties. They consist of the same classes, which is easy to check. Second, the classes are identical connected to the classes of the design pattern. Considering that candidate permutations are found in the same recursive search for a particular design pattern, these classes constitute the same design pattern. A similar problem arises when one matches Figure 10 with the Bridge pattern (Figure 11).

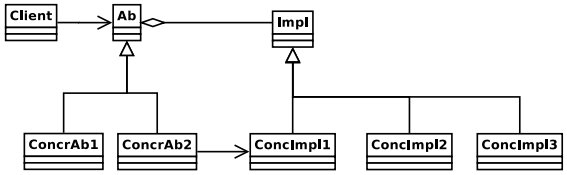


Fig. 10. System with Bridge pattern

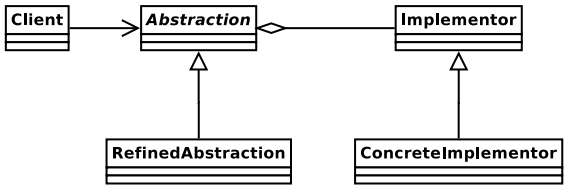


Fig. 11. Bridge pattern

Our first approach detected the Bridge pattern 6 times in Figure 10 because the classes *ConcrAb1* and *ConcrAb2* can be interchanged and second, two of the classes *concImpl1*, *concImpl2*, *concImpl3* have to be chosen, which can be done in three ways.

Although the Abstract Factory and the Bridge patterns seems to occur multiple times, the reasons for the multiple occurrences differ. The Abstract Factory may have *two or more* abstract products and *two or more* concrete factories. The Bridge pattern has *one* abstract class *Ab*, which may have multiple realizations and *one* abstract class *Impl* which may have multiple realizations. It is hard to detect the complete Abstract Factory in the system under consideration, if there are more than two products or concrete factories. The problem of detecting a Bridge pattern in the system under consideration is solvable.

This problem is solved by defining a special inheritance association, which may occur in the design pattern. This association indicates that the inheritance association may occur multiple times in the system under consideration and the specializations do not have other associations in the design pattern. The specializations in the system under consideration may however, have associations. The classes in Figure 10 *Ab* and *Impl* have different numbers of specializations and the specializations *ConcrAb2* and *ConcImpl1* are associated. If the design pattern contains a normal inheritance association then numbers of specializations in the system under consideration and the design pattern should be equal, as shown in Figures 2 and 3.

4.5 Feedback on illegal relationships

Figure 10 shows another interesting phenomenon. The association between *ConcrAb2* and *ConcImpl1* is an association between classes, that are part of a match with the Bridge pattern. However, the association does not belong to the Bridge pattern and therefore it may be a designer's mistake. This means that here, feedback is wanted. We implemented a method `showSolution` that generates this type of feedback. During the search of a design pattern, all matched classes and associations are marked. `showSolution` checks whether not marked associations between marked classes exist. If so, then this association is redundant.

4.6 Detecting partial matches

The detection of partial existence of a design pattern is useful in an educational and professional environment. It helps to check whether the answer to an exercise is partly realized with the obligatory pattern, or in professional practice to check whether a design pattern is fully modeled. The search algorithm tries to match all 3-tuples. If no match is found, the algorithm tries to match all but one 3-tuples of `DP`. The number of unmatchable 3-tuples is a parameter of the search algorithm. However, finding a partially matched design pattern is not always useful, see Figure 12.

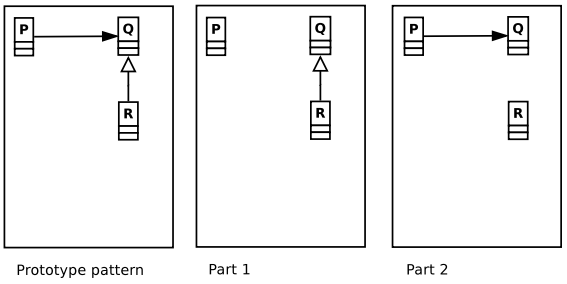


Fig. 12. Prototype pattern and two parts

If one association may be missing, the partial design pattern will be detected many times because a single class combined with one inheritance (part 1 in Figure 12) or one association (part 2 in Figure 12) occurs frequently in class diagrams.

Lacking associations may result in false positives. E.g. two Factory Methods which do not constitute as an Abstract Factory, may be detected an Abstract Factory, when several associations may be missing.

Detecting design patterns, which lack at least one relationship is useful if the remaining part of the design pattern can be represented by a connected graph.

5 VALIDATION IN PRACTICE

Here, we show that the implementation of our method works in practice. Our prototype can detect 17 of the 23 GoF patterns based on structural properties. We searched for a perfect match with templates of design patterns, which are based on literature [GHJV95] So, no false positives and false negatives were reported.
Permutations of design patterns can be detected and reported once. This is shown in the example using the Abstract Factory and the Bridge pattern. There are two key differences between these patterns. The first difference is the number of repeating inheritance structures. The number can be variable, for the Abstract Factory or constant for the Bridge pattern. The second difference is the number of relationships that specializations may have. The specialization in an Abstract Factory has relationships, whereby the specializations

in a Bridge pattern do not have relationships.
Redundant associations in a design pattern are reported as feedback. The tests also showed that the speed of the prototype is good. A class diagram represented by an XMI-file with 33 classes, 49 relationships and 17 partially overlapping design patterns was processed within 0.8 seconds. These tests were executed on an AMD Athlon(tm) 64 X2 Dual Core Processor 5000+ [Doo16].
All the necessary software is publicly available[3].

6 RELATED WORK

In an educational environment, much time is spent on how to apply design patterns since this increases the maintainability of the software. Experiences with teaching design patterns at university level are described by S. Stuurman [Stu15].
The research on detection of design patterns was at first focused on analyzing source code. A modern example is APRT, Another Pattern Recognition Tool. This tool can detect design patterns by parsing Java Sources. The parser is generated by ANTLR (https://www.antlr.org/). The design patterns are language independent specified. So, by changing the specification of the source language i.c. Java. Other object-oriented languages can be used as well [BR17].
Source code may besides by parse trees, also be represented by graphs. Structural and behavioral characteristics of design patterns may be represented by graphs. Bahareh Bafandeh Mayvan et al. showed by analysis of JHotDraw5.1, JRefactory 2.6.24 and JUnit 3.7 that ten design patterns could be detected with 100% precision and 100% recall. In some cases, the precision or recall was not available, because the division was by zero. Their approach can not to distinguish Strategy and State patterns. [BMR17].
De Lucia et al. analyze class diagrams, which are generated from sources. The behavior of candidate design patterns is analyzed based on generated sequence diagrams and runtime analyzes of bytecodes within jar files based on generated testdata. They focused on creational and behavioral design patterns [LDGR18].
Prechelt, as described in section 3.3 was able to detect 70 occurrences of patterns of 4 design patterns with a precision of ± 14% [PK98]
Matrix representation of class diagrams for detecting design patterns is described by the approach of Tsantalis [TCHS05]. He claims that his approach could detect 20 out of 23 GoF patterns, but there is no empirical evidence.

The approach of Oruc et al. is based on finding subgraphs, representing design patterns, in a graph by a standard algorithm. Their approach succeeded in detecting all 23 Gang of Four design patterns with a precision of 80% and recall 88% on average [OAS16].
Pelzus et al. use a rule-based approach, together with a technique to detect multiple design flaws and detection of code smells to detect anti-patterns such as *The Blob, God class, and Swiss Army Knife* [PKLS16].
Quality of software is positively influenced by using design patterns and detecting and removing code smells. Walter et al. discovered negative relations between some design patterns and some code smells [WA16].

[3]urlhttp://members.chello.nl/e.doorn1/DesignPatterns/static_decidability

7 CONCLUSION

We have showed the relevance of automatic detection of design pattern for an educational environment. We have introduced a new subdivision of the set of Gang of Four design patterns: *static* and *non-static* and a new classification of detection algorithms: *static decidability* and *generally complete*. We have made several contributions to the progress of research to automatically detecting design patterns in UML class diagrams. Addressing the problem of multiple detections of one occurrence of a design pattern is new. We introduced the concept of *non-duplicating* for higher quality detection algorithms. With the presented prototype, 17 out of 23 Gang of Four Design patterns are detectable, within an acceptable time. 17 partially overlapping design patterns were detected within 0.8 seconds in a class diagram containing 33 classes and 49 relationships. Permutations and so multiple occurrences of these design patterns are also detectable. We, therefore developed a *non-duplicating* algorithm that offers *static decidability*. The remaining six design patterns are not detectable by our approach, because they are not fully specified by structural properties.

Feedback can be given but is restrained to the notification of superfluous relationships. Detection of a design pattern, that lacks one or more relationships is useful, if the remaining graph of the design pattern is connected, otherwise false positives may result.
Our method and prototype for detection and the introduction of a new subdivision of design patterns offers perspectives educational application.

8 FUTURE WORK

Our next research will focus on the benefits of the use of the prototype in an educational environment in which *static* design patterns are introduced. Obviously, teachers and students would be helped if all design patterns would be detectable. So, a *generally complete* algorithm is an important goal for future work. This requires figuring out what information is necessary by such an algorithm and how the information should be structured and supplied to a tool.

If Figure 9 would contain three abstract products, the current prototype would detect Abstract Factory three times while it is actually one occurrence. There is currently no simple way disclosed to solve this problem.

Automatically giving feedback would, of course, be a large improvement. This will require a significant extension of the algorithm.

Can the algorithm be extended to analyze Sequence and State diagrams in order to detect non-static patterns?

Finally, the scope of our research could be extended to include other characteristics of class diagrams which may give information of the quality of design, both in education and in development.

REFERENCES

[BBQ14] Afnan Salem Ba-Brahem and M. Rizwan Jameel Qureshi. The proposal of improved inexact isomorphic graph algorithm to detect design patterns. *CoRR*, abs/1408.6147, 2014.

[BCDL14] Mario Luca Bernardi, Marta Cimitile, and Giuseppe Di Lucca. Design pattern detection using a dsl-driven graph matching approach. *J. Softw. Evol. Process*, 26(12):1233–1266, dec 2014.

[BGH+04] Vincent D. Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Rev.*, 46(4):647–666, apr 2004.

[BMR17] Bahareh Bafandeh Mayvan and Abbas Rasoolzadegan. Design pattern detection based on the graph theory. *Know.-Based Syst.*, 120(C):211–225, mar 2017.

[BP00] F. Bergenti and A. Poggi. Idea: A design assistant based on automatic design pattern detection. In *Proceedings of 12th International Conference on Software Engineering and Knowledge Engineering (SEKE 2000)*, pages 336–343, 2000.

[BR17] Chris Bates and Ashley Robinson. Aprt – another pattern recognition tool. *GSTF Journal on Computing (JoC)*, 5(2), 2017.

[Doo16] Ed van Doorn. Supporting design process by automatically detecting design patterns and giving some feedback. Master's thesis, Open University, Heerlen, The Netherlands, 8 2016.

[DSZ08] Jing Dong, Yongtao Sun, and Yajing Zhao. Design pattern detection by template matching. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 765–769, New York, NY, USA, 2008. ACM.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[LDGR18] Andrea De Lucia, Vincenzo Deufemia, Carmine Gravino, and Michele Risi. Detecting the behavior of design patterns through model checking and dynamic analysis. *ACM Trans. Softw. Eng. Methodol.*, 26(4):13:1–13:41, feb 2018.

[LYL08] Hakjin Lee, Hyunsang Youn, and Eunseok Lee. A design pattern detection technique that aids reverse engineering. *International Journal of Security and its Applications*, 2, 01 2008.

[OAS16] M. Oruc, F. Akal, and H. Sever. Detecting design patterns in object-oriented design models by using a graph mining approach. In *2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT)*, volume 00, pages 115–121, April 2016.

[PK98] Lutz Prechelt and Christian Kramer. Functionality versus practicality: Employing existing tools for recovering structural design patterns. *j-jucs*, 4(12):866–882, dec 1998.

[PKLS16] Sven Peldszus, Géza Kulcsár, Malte Lochau, and Sandro Schulze. Continuous detection of design flaws in evolving object-oriented programs using incremental multi-pattern matching. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, pages 578–589, New York, NY, USA, 2016. ACM.

[SF04] Sylvia Stuurman and Gert Florijn. Experiences with teaching design patterns. *SIGCSE Bull.*, 36(3):151–155, jun 2004.

[SO06] N. Shi and R. A. Olsson. Reverse engineering of design patterns from java source code. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pages 123–134, Sept 2006.

[Stu15] Sylvia Stuurman. *Design for Change*. PhD thesis, Open University, Department of Computer Science, 2015.

[TCHS05] Nikolaos Tsantalis, Alexander Chatzigeorgiou, Spyros T. Halkidis, and George Stephanides. A novel approach to automated design pattern detection. In Panayiotis Bozanis and Elias N. Houstis, editors, *10th Panhellenic Conference on Informatics, PCI 2005, Volos, Greece - November 11 - 13, 2005*, volume 3746 of *Lecture Notes in Computer Science*. Springer, 2005.

[TCSH06] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S.T. Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, Nov 2006.

[WA16] Bartosz Walter and Tarek Alkhaeir. The relationship between design patterns and code smells. *Inf. Softw. Technol.*, 74(C):127–142, jun 2016.

[ZV08] Laura A. Zager and George C. Verghese. Graph similarity scoring and matching. *Applied Mathematics Letters*, 21(1):86 – 94, 2008.

Design decisions under object-oriented approach: A thematic analysis from the abstraction point of view

ABSTRACT

Many authors consider abstraction as one of the key principles in objected-oriented software engineering, but also very difficult to achieve. Specifically, during the software design stage, abstraction allows decrease the complexity and achieve a more efficient decomposition in a software architecture. However, despite its importance and difficulty, there is a lack of theoretical or empirical research that explores how to enhance such ability. In this paper, we report the results of a research that was undertaken in order to address this gap in the body of knowledge. Particularly, we conducted a qualitative study through a thematic analysis to explore how students apply abstraction during the object-oriented software design. Our results reveal that during the modeling of the problem domain in Unified Modeling Language (UML), students express a deficiency of abstraction, being the possible causes: strict copy of reality to software, influence of structured approach, tendency to simplification, and lack of understanding of the concepts of object-oriented approach.

CCS CONCEPTS

•Software and its engineering → Software design engineering; •Social and professional topics → Software engineering education; *Computational thinking;*

KEYWORDS

Abstraction, object-oriented approach, qualitative research

ACM Reference format:
. 1997. Design decisions under object-oriented approach: A thematic analysis from the abstraction point of view. In *Proceedings of ACM conference, , 2019,* 8 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Computer Science and Software Engineering are defined as disciplines that bring together different thinkers who are characterized by their algorithmic thinking that changes rapidly from levels of abstraction [18]. Several studies agree that one of the key fundamentals of Computer Science and Software Engineering is abstraction and the ability to abstract [1, 4, 13, 28]. This concept is also closely related to the algorithmic formation [41], the automation of notations and models [43], the decomposition of complex tasks [42], and the generalization through the definition of patterns [41].

ACM conference,

DOI: 10.475/123_4

In software design and programming, particularly in the object-oriented approach, abstraction also plays an important role as a tool for managing complexity [20, 25, 27, 38, 39] being referred as a threshold concept, together with the inheritance, modularization, encapsulation and polymorphism [4, 28, 29, 36].

Several authors suggest that within the skills required for computational thinking, there is the reasoning about abstract objects [3, 8, 17, 34]. This is evidenced in the report of the European Commission *Developing Computational Thinking in Compulsory Education*, where the main articles of the literature review, place abstraction as a core concept [7]. According to Booch [16], *"An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer".*

In terms of objects, abstraction focuses on the outside view of an object, and so serves to separate an objects essential behavior from its implementation [16]. Efforts are made to construct abstract entities, because these go together with the domain of the problem. If we talk about characterizing the behavior of an object, it must be considered the services that it provides to other objects, as well as the operations that it may perform upon other objects. Meyer [23], define the *contract model* of programming: "the outside view of each object defines a contract upon which other objects may depend, and which in turn must be carried out by the inside view of the object itself". Wirfs [44] says in other words "this contract encompasses the responsibilities of an object, namely, the behavior for which it is held accountable". Therefore, a way of expressing abstraction is to grant all the behavior of an object, designating operations we can meaningfully perform upon an object and how that object reacts [16]. In this study we will be guided by this definition of abstraction.

Even though, abstraction is an important concept for Computer Science and Software Engineering; in particular for software development and programming with the object-oriented approach [4, 5]; there are very few studies that empower this mechanism in students. In spite of that, there are studies that show the difficulty of abstracting [26]. Under this perspective, this work focuses on studying the abstraction, analyzing the students' assessments that face the resolution of software design exercises under the object-oriented approach. The results were obtained through a qualitative case study applied to university students in Computer Science career.

The rest of the article is structured as follows. Section II shows several related works found in the literature and the contribution that our work has regarding them. Section III presents the research approach. Section IV shows the development of the thematic analysis. Section V presents the findings of the investigation. Next, in Section VI presents details the trustworthiness of the research. Finally Section VII details the conclusions and future work obtained.

2 RELATED WORK

In the literature, we found works that addressed the concepts of software design under the object-oriented approach, some of them focused on case studies carried out with students and developers. The first one, corresponds to the study of Ragonis and Ben-Ari [32], which focuses on the understanding of learning concepts of object-oriented programming in first-year students through a qualitative longitudinal study. The findings were divided into four primary categories: class vs. object, instantiation and constructors, simple vs. composed classes, and program flow. In these four categories, conceptions and difficulties are identified, however, within the findings it was not possible to demonstrate the relationship with the student's ability and abstraction.

Another work is the one presented by Sanders et al. [37], where the authors carry out a qualitative study about the software programs written by a group of students. The study analyze the misconceptions in turn, which were grouped in basic mechanics, instance/class conflation, problems with linking and interaction, class/collection conflation, problems with hierarchies and abstraction and failures in modelling. The difficulties related to abstraction focused mainly on the difficulty of the students to be able to conceive a structure of inheritance when it is required to homogenize classes or when using interfaces. Almost 10 years later a work led by Sanders et al. [36], allows to identify through the literature, that Abstraction is considered a threshold concept in computing.

There is also the work of Rachel Or-Bach et al. [29], who present a qualitative study based on the analysis of the solutions of an exercise sent to students as homework. This is one of the few studies that presents results related to the design of object-oriented software with special attention to abstraction, which can be showed through inheritance and the inclusion of relevant attributes and methods in a class.

Moström et al. [25], from empirical evidence, reported in their research the abstraction is manifested in students as transformative experiences, not by general abstraction, used to conceptualize at a higher level, but by particular, practical concepts that relate to abstraction. In [17], uses the dual process theory to research the understanding of concepts and principles of the object-oriented paradigm of software development practitioners. Within the studied software principles, abstraction is considered, where a low level of abstraction was evidenced when objects were created close to reality, difficulties with inheritance and understanding of abstract classes, and the information hiding principle.

The last work presented was developed by the same authors of this research [2] and was carried out through a qualitative case study aimed at understanding the Information Hiding Principle. In this study students' perceptions are collected, being one of the findings the relationship that students make between the abstraction and the polymorphic property of the inheritance. Finally, in [5], an evaluation based on model object-first is carried out in a computer science subject where the results conclude that abstraction in general has a positive impact on programming, nevertheless, it was not possible to demonstrate a correlation between cognitive development and the grades obtained by students in programming.

The contribution of this study, lies in studying the abstraction in software design under the object-oriented approach. This study focuses on discovering the perceptions of students when designing software and how they manifest abstraction through it. This study will be carried out with exercises that allow to analyze the students' perceptions, where the results allow the reflection of those who study abstraction and its influence on the learning of object-oriented software design.

3 RESEARCH METHODOLOGY

This section shows the research questions, the chosen methodology and the details of the selected case study.

3.1 Research Questions

This research have been conducted by two research questions:

- What are the design decisions of students from the point of view of abstraction?
- What are the possible causes of student design decisions?

3.2 Research Method

The research questions offer a questioning that addresses unknown aspects about how students perceive abstraction through the object-oriented approach. A qualitative research has as a key aspect the understanding of the phenomenon of interest from the perspective of the participants [22].

In particular, the thematic analysis approach is one of the most common forms of analysis within qualitative research. Thematic analysis is a method for identifying, analysing and reporting patterns (themes) within data. It minimally organizes and describes your data set in (rich) detail. [10].

This approach interconnects the themes which emerge from the study. Interconnecting themes means that the researcher connects the themes to display a chronology or sequence of events. [11]. Under this perspective, a thematic analysis is adjusted to the raised problem, since our study investigates a through the some events the decisions made by the students around object-oriented software design applying abstraction take into in account events sequential of the students.

3.3 Setting

The case study was conducted with a group of students from the University R in the Faculty of Computer Systems. The details are shown below:

3.3.1 Background of the subject. The course corresponds to the subject of Applications in Propietary Environments, mandatory subject of the fifth semester in Computer Systems. The subject is taught during 4 hours a week during 16 weeks. The students that take this subject must have studied previously the subjects of Database Management, Programming I and Programming II; and have as a co-requisite the subject of Software Engineering I.

The content of Programming II focuses on the teaching of object-oriented programming languages, that is, students already have prior knowledge about the concepts of this approach.

3.3.2 Background of the participants. The group of students of this subject consists of 26 students, out of which 23 became participants in this research. Three students were discarded for the study since this is the second time they take the subject.

4 THEMATIC ANALYSIS

The research process was based on the model proposed by Seidel [40] and has the following stages:

- Data collection
- Coding
- Refinement
- Grouping

4.1 Data collection

In this stage the collection techniques are defined and the data is collected. In this study, qualitative data were collected from the diagnostic test and the interview, as shown in Figure 1; which represents the sequence of the data collect process used for this study.

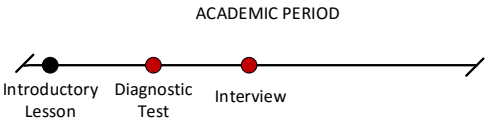


Figure 1: Sequence of data collect

4.1.1 Introductory lesson.

Before the data collection, the students received a two-hour class corresponding to concepts of Unified Modeling Language (UML), in order to have a standard language at the modeling of the problem. The class also covered the notation of the behavior diagrams (sequence diagrams), static diagrams (class diagrams, dependency relationships) and a special attention was given to the semantic of structure of the inheritance. Finally, we also refresh concepts related to the object-oriented approach like object, class and message. It is important to emphasize that the students have already received subjects where they have previously programmed in object-oriented languages, it means that the concepts taught in class were a reinforcement to their knowledge about the concepts of object orientation.

4.1.2 Diagnostic Test.

Documents are a singularly useful source of information, although they are often ignored [21]. In this case study, the documents collected refer to 5 design exercises that were part of a ***diagnostic test***. The test was taken to the 26 students at the beginning of the academic period. The test consisted of 5 exercises, where the design of each statement was requested through class or sequence diagrams. The general directions presented in the test were the following:

- In each proposed exercise, it is asked to design a software system that meets the stated requirements. The design should be modeled through the required diagram. In case of classes diagram, the student should write the specification of each class, attributes, methods and the relationships between classes.

The statements of each exercise are shown below:

Exercise 1: Betting System

It is required to make an application that is in charge of the sports betting service, where a user must register in the system to have an account for manage bets. Bets can be received by transfer or by card. The system supports different types of bets, for example: simple bet (which team wins), special bet (which minute marks the first goal) and others.

Exercise 2: Furniture transfer between rooms

This application is responsible for moving a piece of furniture from one room (contiguous or not) to another. Between adjoining rooms, the furniture can only pass through the door, while in separate rooms the furniture can pass through the walls (See Figure 2).

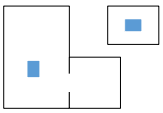


Figure 2: Graphic representation of exercise 2)

Exercise 3: Library

- Definitions
  - Client: active teacher or student enrolled in the school in the current course.
  - Library: the software system that must be designed. The physical library has at least one copy of each book.
- Client Requirements:
  - Request the position of a copy of a book on the shelves.
  - Borrow a book from the library. It may or may not be available. If not available, the library informs the client when it will be available.
  - Returns a borrowed book to a person in the library who confirms the return.
  - Buy a book from the library, which varies - Borrow a book that does not exist in the library, in order to the Library to request it from another library.
  - Buy a book in a bookstore where the book is available and add it to its book collection.
- Library Requirements:
  - Notifies the client the date of return of the book when this date is approaching.

- Punishes the client prohibiting loans for a given time, if the date of return is passed.
- Invoices and charges the customer for the purchase of a book. Libraries are added and deleted.
- Bookstores are added and deleted.

Exercise 4: Draw circles

This application consists of drawing a small circle inside a larger one. The smallest circle can move inside the big circle, without getting out.

Exercise 5: Booking rooms in a hotel

This application is responsible for booking rooms in a hotel. The dates of the reservations and the verification of the availability of the room must be considered.

4.1.3 Interview.

Dexter [14], defines an interview as a conversation with a purpose. In this study, the interview was focused on explaining the design decisions made by the students when modeling the domain problem. The interview consisted of certain questions for each exercise; it was conducted after the diagnostic test and was recorded in audio.

Exercise 1: Betting System

- In what part of your diagram do you think the bettor has money to bet?
- In what part of your diagram is the type of bets handled?
- What class is responsible for making the bet?
- Is the sporting event managed somewhere?
- Where is the type of payment handled?

Exercise 2: Furniture transfer between rooms

- How do you handle the differentiation between the contiguous room and the non-contiguous one?
- How would you generalize your design in the case of having more rooms?
- In what part is the collision with the walls handled?

Exercise 3: Library

- How do you control the expiration notice of the book loan?
- How do you handle the location of the book?
- How do you add and delete libraries?

Exercise 4: Draw circles

- In what part is the small circle considered to be within the large circle?
- In case the student has considered more than two classes for a circle. How would you solve the problem with a single circle class?

Exercise 5: Booking rooms in a hotel

- How do you check the availability of the room?
- Could I book several rooms?
- In what part of you diagram do you make the reservation?

4.2 Coding

In this stage, codes were assigned on the collected documents and audios. "*A code in a qualitative inquiry is defined most often as a word or short phrase that symbolically assigns a summative, salient, essence-capturing, and/or evocative attribute for a portion of language-based or visual data*" [35, p. 23]. The processing of the data was done with the ATLAS.ti software [15].

4.3 Refinement

In this stage, the preliminary codes obtained in the previous stage were analyzed, matching or detailing similar codes. This stage was performed iteratively until the codes remained relatively stable.

4.4 Grouping

In this stage, the codes generated in the Refinement stage were grouped. The grouping was carried out following the criteria of the research questions. In the next section we can see in detail the results obtained from this stage.

5 RESEARCH FINDINGS

This section answer both research questions which conducted by this study, this results are shown below.

5.1 Research Question 1: results

The results obtained from this stage answer the first research question "What are the design decisions of students from the point of view of abstraction?" and these are presented below:

1. Tendency to create an "intermediate" class to register the details of the related classes as in the entity-relationship model, the creation of the entity `Invoice-Detail`.

- Example 1: Class `Loan-Detail` between the classes `Loan` and `Refund`. Explicit quotation from the student E19: *"Confusion with data structures and databases"*.

- Example 2: The question was: Where would you put the value of the bet?, the response was *"I would go between class* `Bet` *and* `Person` *with a class* `Bet-Detail` *which includes the value of the bet and the name of the person"*.

- Example 3: The class `Library-Client` between classes `Client` and `Library`. The Figure 3 below, shows some examples.

2. Assignment of behavior to classes that represent reality in a strict way.

- Example: The class `Administrator` is responsible for booking the room. Student answer to the question: Which class is responsible for booking the room? *"I should have created a class* `Administrator`*, that will have the method related to the booking, because when I go to a hotel the person who is in charge of administration or reception do the reservation"*.

3. Lack of creation of classes for relevant aspects of the application.

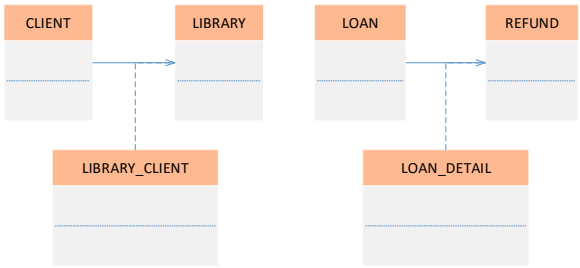


Figure 3: Concepts from the entity-relationship model

- Example: The fact that the students have not considered that the bettor must have money to make the bet.

4. Design classes with different names, but with the same structure.

- Example: If the exercise consisted to draw two rooms, the students create two classes: `Room1` and `Room2`, with same methods and attributes.

5. Design classes without any behavior. Special interest in defining classes only through their attributes.

- Example: The example is shown in Figure 4. A particular case of this design decision is when the students put classes just with `get()` or `set()` methods.

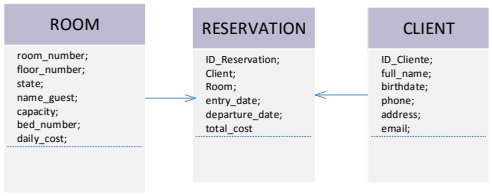


Figure 4: Classes without methods

6. Place responsibilities on classes that should not be responsible for that behavior.

- Example 1: For example, the `Hotel` class is the one who makes the reservation of the room or the registration of the client or the `Library` class is the one who is in charge of the expiration date of the book.
- Example 2: The `Account` class has a bet method. In Figure 5 there are the examples. A consequence of this decision, the design is getting overloaded classes or with an alien behavior.

7. Differentiation between the subclasses and the superclass only by their attributes.

- Example: The type of payment is made with an inheritance with a superclass `Bet` and two subclasses `Simple Bet` and `Special Bet`, which have as an attribute `winningteam` and `goalminute` respectively. In Figure 6 below see the example.

Figure 5: Methods not related to the class

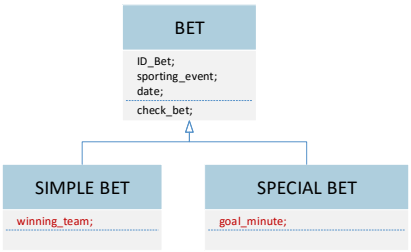


Figure 6: Specialization of subclasses through their attributes

8. Assignment of complex behaviors as attributes.

- Example: The type of payment as an attribute of the `Bet` class.

9. Belief that the students can perform an action on the same class because there is an ID or numbering attribute, a one-to-many relationship or because it can have multiple instances.

- Example: Some students believe that they can book several rooms, due to the fact that each class `Room` can instantiate several objects of the class `Room`. When asked: Is it possible to book several rooms?
  - Student E03: *"Yes, because an attribute of the class* `Room` *is the number of rooms, and so you could reserve several rooms"*.
  - Student E23: *"Yes, because there is a relationship between the classes* `Client` *and* `Reservation` *that allows you to book several rooms"*.
  - Student E17: *"Yes, because the one-to-many relationship between the classes* `Hotel` *and* `Room` *was established"*.

10. Definition of classes that are not concepts.

- Example: Classes `Transferable` or `Movement`.

5.2 Research Question 2: results

The results previously analyzed in Section 5.1 have been grouped into four possible causes, this classification answer to the second research question: "What are the possible causes of student design decisions?".

5.2.1 *Strict copy of reality.*

Within this cause, two behaviors can be identified:

- Absence of a concept because it is not concrete in reality

Some results of the analysis of the exercises suggest that some students do not dare to create objects that are not familiar to them in reality. For example, in the Betting exercise, the difficulty of conceiving a concept "payment" as a class that can be responsible for making the payment of the bet. Another example is the absence of objects that represent the repositories, that is, they do not imagine a class that lists the elements instantiated by a class. Nor do they create collections of objects, as observed in the Booking Room exercise, where the classes corresponding to a `List/Collection` of rooms were absent, the same that could be defined as an Abstract Data Type (ADT).

- Transfer of a concept that exists in reality

On the other hand, there are other students who forced the creation of a concept, for the fact that in reality it is a concrete concept, as is the case of the person who manages a hotel, called "Administrator". This concept was transferred by the students to a class called `Administrator` who was responsible for booking a room; which means that he/she personified reality in the software. Rosson et al. [33], supports the idea of taking the real-world aspects as a reference, and proposes the so-called "metaphorical extension of the problem", which in essence is to modify or extend the entities of the problem that may be useful. Nevertheless, he says explicit that the concept could be extended or modified in the software, otherwise it would be limited to what that concept does or solves in real life.

According to [6], the process of abstraction starts from placing a name to a class and granting it what essentially belongs to that concept under that name, however, in the students the transfer of reality prevailed before the abstraction. Both behaviors of students, obviate a concept or transfer another from real life, directly influenced on how they decompose and the way in which students give meaning to each part of the division, in other words how they abstract.

Other authors [5, 19, 30, 31], mention that one of the properties of the object-oriented approach is the so-called *naturalness*, understood as the ease of objects to provide abstract versions of reality. This property is still discussed by other authors [13] and criticized for being a limitation to enrich the object-oriented software design [33].

5.2.2 *Structured approach.*

The structured approach made its appearance in student resolutions, especially with the entity-relationship model, which is a diagram of the data model [12]. Associated with the entity-relationship model, there is also the data table, which is responsible for the specification of the data, and where the first field in the table is the `ID`. The entity-relationship model and the use of an `ID`, were some of the symptoms of the adoption of the structured approach in the object-oriented design. Another evidence of the interference of the structured approach, is to visualize classes that do not have behavior, and that are required by other classes that process their attributes, corresponding this to the definition of structured approach by Tom DeMarco [12], where it was established three main parts: the description of data objects, the specification of the process, and the specification of the control.

The influence of prior knowledge has been studied in the work of Morris et al. [24], where he researches the previous experience as a learning difficulty for the acquisition of new knowledge. Also in [9], the author presented a research whose objective was to describe the experience of migrating from structured analysis to object-oriented modeling.

5.2.3 *Simplistic Overview.*

Simplistic understood as treating complex issues and problems as if they were much simpler than they really are. For example, what was reflected in the results, was the inclusion of attributes that could have complex behavior in the same class. This behavior could respond to a simplistic view on the part of the student, that is, the student does not consider the complexity of a concept and places it as an attribute. A concrete example of this is the Betting exercise, where the students created a class `Bet` that was part of an inheritance as a superclass, having as subclasses the type of bet, for example `Simple Bet` and `Special Bet`, however, both the superclass and the subclasses lacked additional behavior to the `get()` and `set()`, and differed only by their attributes.

Many authors [8, 34, 44], resume the abstraction as the process of reducing concepts to their essence in such a way that only the necessary elements of that concept are represented. Taking these words as a reference, abstraction allows us to define an object in its completeness, with its name, attributes and behavior. A basic level required of abstraction could be shown with an appropriate class diagram that put attention in defining each class with its relevant attributes and methods [29], in this sense, the students demonstrate a lower abstraction level due to the difficulty of reducing an object to its essence and granting it only the responsibilities that corresponds. The students at the first instance see the object as a simpler concept whose complexity can be handled through attributes, resulting in overloaded and complex classes with high dependency between them.

5.2.4 *Lack of understanding of object orientation concepts.*

It was also possible to see confusion with some object orientation concepts, as well as its representation in UML. In particular, problems with abstract classes also were reported in [29], and its relation with the understanding of inheritance. Hadar [17], also studied the differences between how the inheritance is conceived in object oriented and the way we naturally interpret inheritance. The misunderstanding of inheritance has a direct consequence with the understanding of polymorphism, the main benefit of inheritance. We also observed problems distinguishing the concrete classes and instances of those classes, for example when the Room class could be instanced instead of creating another Room1 class.

On the other hand, we observed some misunderstandings related with the semantic in sequence diagrams. This was the aspect that could be reflected in a lesser degree in the students.

From the exercises analyzed in the diagnostic test, it was also possible to appreciate that exercises 1, 2 and 4 led students to leave aside actions that were not expressed explicitly in the statements of the exercises, actions such as: check if the bettor had money, the hit of the furniture with the walls of the room or the strike between the circles. While, in exercises 3 and 5, students were closer to the

identification of the objects that intervened in the domain of the problem, since the statement explicitly described the scope of the exercise.

6 RESEARCH TRUSTWORTHINESS

The study satisfies the quality criteria defined by Lincoln and Guba [21]. This study presents explicitly the questions asked in the interviews, as well as the statements of the exercises that were part of the diagnostic test.

In addition, this research provides details about the environment and the participants; also the information obtained from the students is triangulated by taking two sources of data such as the diagnostic test and the interview, thereby complying with the indicators of *credibility*.

With respect to *transferability*, and under the guidelines of Lincoln and Guba, the term transferability differs from the term generalization from quantitative approaches, and rather speaks of applicability that is to try to establish that the results can be applied under certain circumstances; that is, that they are potentially applicable.

To ensure the *reliability*, the researchers detail and make explicit the trace from the origin of the data and its manipulation until the declaration of results, thus facilitating traceability.

Additionally, in order to avoid conflicts of ethical nature with respect to the manipulation of the data collected from the students, the informed consent forms were made where the anonymity and confidentiality of the data obtained from the students is guaranteed. This report was read and signed by the students prior to the investigation.

7 CONCLUSIONS AND FUTURE WORK

The research questions that led this study, *What are the design decisions of students from the point of view of abstraction?* and *What are the causes of student design decisions?*, have been answered. Based on our analysis, we conclude that the students design decisions reveal a basic level of abstraction, based on the definition of abstraction related to grant all the behavior of an object, designating operations we can meaningfully perform upon an object and how that object reacts. It was evidenced when the students didn't model a concept or transfer a concept that exist in reality in their diagrams. We observed the lack of classes with a entire behavior, such as the existence of objects without any behavior (without methods), overloaded (many methods) or with alien behavior (methods that do not correspond to it). Actions that are a voice of alarm when evidencing student's gaps by not being able to represent a concept for its essential characteristics. Also, we noticing that the students design decisions where also influenced by transfer of concepts of the structured approach, such as the entity-relationship model, despite the fact that students have already received subjects such as object-oriented programming. The design decisions shown by students were grouped around four causes: strict copy of reality to software, influence of structured approach, tendency to simplification, and lack of understanding of the concepts of object-oriented approach.

The results presented in this study are useful for several reasons. For a professor: a) it is a way of learning about difficulties that the student have when implementing the abstraction. b) Analysis the kind of exercise that allows to increase the ability of abstraction; it helps in the construction of software design exercises for educational purposes. c) Put special attention in the origins of the design decisions. On the other hand, the student could reflect around other solutions of the same problem where the level of abstraction is higher.

This study also raises challenges related to how to overcome the difficulties evidenced by students, both at the design and pedagogical level, such as: how do students stop transferring the knowledge learned in the structured approach when they design with the object-oriented approach? or what is the best pedagogical strategy to learn an object-oriented approach? This study will be extended analysing the state of the students at the end of the academic period. It will allow us to know if the first design decisions were kept or it changed.

REFERENCES

[1] Alfred V. Aho and Jeffrey D. Ullman. 1992. *Foundations of Computer Science.* Computer Science Press, Inc., New York, NY, USA.

[2] Anonymous.

[3] Michal Armoni. 2010. COMPUTING IN SCHOOLS On teaching topics in computer science theory. *ACM Inroads* 1, 1 (2010), 21–22.

[4] Deborah J Armstrong. 2006. The quarks of object-oriented development. *Commun. ACM* 49, 2 (2006), 123–128.

[5] Jens Bennedsen and Michael E. Caspersen. 2006. Abstraction Ability As an Indicator of Success for Learning Object-oriented Programming? *SIGCSE Bull.* 38, 2 (June 2006), 39–43. DOI: http://dx.doi.org/10.1145/1138403.1138430

[6] Alan F Blackwell, Luke Church, and Thomas RG Green. 2008. The Abstract is an Enemy: Alternative Perspectives to Computational Thinking.. In *PPIG*. 5.

[7] Stefania Bocconi, Augusto Chioccariello, Giuliana Dettori, Anusca Ferrari, Katja Engelhardt, P Kampylis, and Y Punie. 2016. Developing computational thinking in compulsory education. *European Commission, JRC Science for Policy Report* (2016).

[8] Grady Booch. 2006. *Object oriented analysis & design with application.* Pearson Education India.

[9] Roger Box and Michael Whitelaw. 2000. Experiences when migrating from structured analysis to object-oriented modelling. In *Proceedings of the Australasian conference on Computing education.* ACM, 12–18.

[10] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. DOI: http://dx.doi.org/10.1191/1478088706qp063oa

[11] John W Creswell. 2012. *Educational research: Planning, conducting, and evaluating quantitative.* Prentice Hall Upper Saddle River, NJ.

[12] Tom DeMarco. 2002. Structured analysis and system specification. In *Software pioneers.* Springer, 529–560.

[13] Françoise Détienne. 1990. Difficulties in designing with an object-oriented language: An empirical study. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction.* North-Holland Publishing Co., 971–976.

[14] Lewis Anthony Dexter. 2006. *Elite and specialized interviewing.* Ecpr Press.

[15] Susanne Friese. 2018. ATLAS.ti 8 Mac User Manual. (Aug. 2018). https://downloads.atlasti.com/docs/manual/manual_a8_mac_en.pdf?_ga=2.243651149.915135695.1535427970-250496249.1520305147

[16] Booch Grady. 1994. *Object-oriented analysis and design with Applicactions.* The Benjamin/Cummings Publishing Company, Inc.

[17] Irit Hadar. 2013. When intuition and logic clash: The case of the object-oriented paradigm. *Science of Computer Programming* 78, 9 (2013), 1407 – 1426. DOI: http://dx.doi.org/10.1016/j.scico.2012.10.006

[18] Juris Hartmanis. 1994. Turing Award lecture on computational complexity and the nature of computer science. *Commun. ACM* 37, 10 (1994), 37–43.

[19] Orit Hazzan. 2003. How students attempt to reduce abstraction in the learning of mathematics and in the learning of computer science. *Computer Science Education* 13, 2 (2003), 95–122.

[20] Jeff Kramer. 2007. Is abstraction the key to computing? *Commun. ACM* 50, 4 (2007), 36–42.

[21] Yvonna S Lincoln and Egon G Guba. 1985. *Naturalistic inquiry.* Vol. 75. Sage.

[22] Sharan B Merriam. 1998. *Qualitative Research and Case Study Applications in Education. Revised and Expanded from" Case Study Research in Education.".* ERIC.

[23] Bertrand Meyer. 1992. Applying'design by contract'. *Computer* 25, 10 (1992), 40–51.

[24] Michael Morris, Cheri Speier, and Jeffrey Hoffer. 1999. An Examination of Procedural and Object-oriented Systems Analysis Methods: Does Prior Experience Help or Hinder Performance?*. *Decision Sciences* 30, 1 (1999), 107–136.

[25] Jan Erik Moström, Jonas Boustedt, Anna Eckerdal, Robert McCartney, Kate Sanders, Lynda Thomas, and Carol Zander. 2008. Concrete Examples of Abstraction As Manifested in Students' Transformative Experiences. In *Proceedings of the Fourth International Workshop on Computing Education Research (ICER '08)*. ACM, New York, NY, USA, 125–136. DOI: http://dx.doi.org/10.1145/1404520.1404533

[26] Dung Nguyen and Stephen Wong. 2001. OOP in introductory CS: Better students through abstraction. In *Proceedings of the Fifth Workshop on Pedagogies and Tools for Assimilating Object-Oriented Concepts*.

[27] Jaime Niño and Frederick A Hosch. 2008. *Introduction to programming and object-oriented design using Java*. Wiley Publishing.

[28] Rachel Or-Bach and Ilana Lavy. 2004. Cognitive activities of abstraction in object orientation: an empirical study. *ACM SIGCSE Bulletin* 36, 2 (2004), 82–86.

[29] Rachel Or-Bach and Ilana Lavy. 2004. Cognitive Activities of Abstraction in Object Orientation: An Empirical Study. *SIGCSE Bull.* 36, 2 (June 2004), 82–86. DOI: http://dx.doi.org/10.1145/1024338.1024378

[30] Jacob Perrenet and Eric Kaasenbrood. 2006. Levels of Abstraction in Students' Understanding of the Concept of Algorithm: The Qualitative Perspective. *SIGCSE Bull.* 38, 3 (June 2006), 270–274. DOI: http://dx.doi.org/10.1145/1140123.1140196

[31] Jacob C. Perrenet. 2010. Levels of thinking in computer science: Development in bachelor students' conceptualization of algorithm. *Education and Information Technologies* 15, 2 (01 Jun 2010), 87–107. DOI: http://dx.doi.org/10.1007/s10639-009-9098-8

[32] Noa. Ragonis and Mordechai Ben-Ari. 2005. A long-term investigation of the comprehension of OOP concepts by novices. (2005).

[33] Mary Beth Rosson and Sherman Alpert. 1990. The Cognitive Consequences of Object-oriented Design. *Hum.-Comput. Interact.* 5, 4 (Dec 1990), 345–379. DOI: http://dx.doi.org/10.1207/s15327051hci0504_1

[34] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William E Lorensen, and others. 1991. *Object-oriented modeling and design*. Vol. 199. Prentice-hall Englewood Cliffs, NJ.

[35] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*.

[36] Kate Sanders and Robert McCartney. 2016. Threshold Concepts in Computing: Past, Present, and Future. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research (Koli Calling '16)*. ACM, New York, NY, USA, 91–100. DOI: http://dx.doi.org/10.1145/2999541.2999546

[37] Kate Sanders and Lynda Thomas. 2007. Checklists for grading object-oriented CS1 programs: concepts and misconceptions. In *ACM SIGCSE Bulletin*, Vol. 39. ACM, 166–170.

[38] Kathryn E Sanders. 2006. *Object-Oriented Programming in Java: a graphical approach*. Pearson/Addison Wesley.

[39] Robert W Sebesta and Soumen Mukherjee. 1999. *Concepts of programming languages*. Vol. 8. Addison-Wesley Reading, Massachusetts.

[40] John V Seidel. 1998. Qualitative data analysis. The Ethnograph v5. 0: A Users Guide, Appendix E. Colorado Springs, Colorado: Qualis Research. (1998).

[41] J Wing. 2011. Research notebook: Computational thinking—What and why? The Link Magazine, Spring. (2011).

[42] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.

[43] Jeannette M Wing. 2008. Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences* 366, 1881 (2008), 3717–3725.

[44] Rebecca Wirfs-Brock and Brian Wilkerson. 1989. Object-oriented design: a responsibility-driven approach. In *ACM SIGPLAN Notices*, Vol. 24. ACM, 71–75.

Teaching Data Structures through Group Based Collaborative Peer Interactions

Sajid Nazir
sajid.nazir@gcu.ac.uk
Glasgow Caledonian University
Glasgow, UK

Stephen Naicken
s.naicken@alueducation.com
African Leadership University
Pamplemousses, Mauritius

James H. Paterson
james.paterson@gcu.ac.uk
Glasgow Caledonian University
Glasgow, UK

ABSTRACT

Data structures and algorithms is an important subject in Computer Science curriculum and builds upon the programming concepts learned by the students in their earlier courses. However, the abstract nature of the concepts can often be difficult for students to grasp. This problem becomes aggravated in an international setting with students from diverse academic backgrounds, resulting in some students losing interest and failing to follow along.

This paper describes our novel approach to teach data structures for Computing undergraduates from 30 African countries at a college in Mauritius in partnership with a UK university. The blended learning program uses as a student led "flipped classroom" approach, requiring students to view lecture and supporting material online prior to engaging in on-campus seminar session with the tutor.

Peer instruction is a key component of the flipped approach. In seminars, students worked on group based problem-solving activities in data structures supported by the tutor. The students devised their solutions on white boards taking ownership of the problem, became motivated to discuss their ideas freely, and to select a group solution. The group solutions were then shared with the other groups and peer reviewed, led by the tutor. This collaborative learning environment was observed to facilitate healthy discussions, and students' contributions and performance in later assessments offered evidence of understanding of core subject concepts.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

data structures, group learning, active learning, blended learning

ACM Reference Format:
Sajid Nazir, Stephen Naicken, and James H. Paterson. 2019. Teaching Data Structures through Group Based Collaborative Peer Interactions. In *CSERC '19: The 8th Computer Science Education Research Conference, November 18–20, 2019, Larnaca, Cyprus.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

CSERC '19, November 18–20, 2018, Larnaca, Cyprus

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Data structures and algorithms is an important subject in Computer Science curriculum [13] in universities throughout the world. Typically, the data structures course follows on from the introductory programming courses, and is considered useful for other advanced courses such as computer architecture, and networks etc. [4, 7]. Thus, it progresses students' knowledge required for understanding advanced programming concepts in their further courses [10]. The subject is also helpful for the students in their professional careers as software engineers, programmers, analysts, and information technology specialists.

Data structures concepts are abstract and the problems of its teaching stem from students failing to grasp these abstract concepts [13]. Low motivation of students is the first difficulty encountered by the tutors [12, 17]. Some students who seem to understand the concepts in class were unable to write programs [17]. Difficulties faced by the students in understanding basic data structures are identified in [19]. A common reason for students failing the data structures course is that they do not complete their programming tasks [9]. Thus, it is very important that the subject material and learning activities are engaging so that the students do not lose interest.

Visualizations and animations have found to be helpful in helping the students to understand the abstract concepts [7, 13]. Most students prefer a hands-on approach and enjoy the participation in the learning process [3]. It is imperative that the students are engaged in activities that motivate them to complete their programming assignments. The curriculum must also stress good programming style, which helps the students to write correct and efficient programs [10].

This paper describes the experience of teaching data structures and algorithms during 2017/18 in a bachelor's degree programme delivered at a newly established college in Mauritius. The mission of the college to bring together and train the future leaders of African countries, and the students were from a wide range different countries all across the African continent with diverse academic backgrounds. In keeping with this mission, the students are encouraged throughout the programme to be active learners through encouragement and participation in peer activities. The final degree is awarded by a UK university, and delivery is done as a collaboration between the college and the university, with online learning material initially developed in the UK and in-person teaching designed by tutors locally in Mauritius.

The in-person class sessions were key to the teaching approach. These involved peer discussion by students of problems set for them which led them to arrive at and articulate suitable solutions. The focus was on supporting the online learning content and activities

through simple yet effective teaching innovations that made the subject interesting to the students. The experience showed that the students enjoyed the process, had developed better understanding of the core subject concepts. Healthy discussion ensued where the students could discuss and understand the relative merits of their group solution in relation to those developed by the other students. The students became active learners and the results were remarkable.

The rest of the paper is structured as follows: Related work is described in Section 2. Methods and materials are covered in Section 3. Section 4 describes the relevant activities of group based peer interactions. The experiences and evaluations of the proposed method is provided in Section 5. Section 6 concludes the paper.

2 RELATED WORK

The use of competitive programming to teach data structures is reported in [4]. The students participated in a competition to improve their games project code against instructor-defined code and code of other students. Evaluation of code against other students was found to encourage students to put more efforts into their code. The project code was to be shared on a web server and the site ranked the students based on their code and other students were challenged to improve the ranking. The students were then evaluated based on code working properly.

The methodology of teaching data structures to be mathematical, theoretical or hands-on was considered in [8]. The three approaches were considered at different institutions by the panelists. The approach, where students were given problems and had to develop and analyze different possible solutions, helped produce a course with broad coverage preparing students to understand similar trade-offs in choice of data structures on their own. The students appreciated more hands-on approach during lab sessions. The students were assigned a major report writing exercise but then the danger is that the student would only be better at the data structure assigned. Thus, only a seminar/presentation style would be inadequate.

The use of different tools for teaching data structures and algorithms were explored in [3]. By actively involving the students, raised the students interest and helped the students to understand the various algorithms. The students' learning was improved with use of games and real-life examples. The teaching tools were used to supplement programming, and other assessments. Pieces of wood of varying lengths were used to teach about sorting algorithms with the students moving the wooden pieces according to the algorithm.

A technique termed as Visual Kinesthetic Psuedocode (VKP) was developed [13] with an aim to help the students code without coding, and providing necessary support in data structure implementation with actual code. VKP was designed to help the students to do actual coding by first helping the students to visualize the data structure, followed by textual pseudo-code that is, engaging the students in active learning.

The importance and use of online multimedia course content compared to face-to-face classroom teaching was explored in [7]. In order to increase the understanding of students they must be involved through active learning. Visualization through multimedia improves the learning experience of students.

Two tools for supporting the teaching were proposed in [1], visualizers provided visualizations of user data structures, whereas Testers then checked the implementation of visualizers. All the visualizers and testers were developed in Java. However, some students found it difficult to think of visualizers and testers as complementary tools.

Different techniques were used in [2] to make it easier to explain the data structure concepts to students. For example, simulations with software tools were used for teaching sorting, and Travelling Salesperson problem with play activity for students to visit all places to keep distance minimum.

For better learning of students, abstract concepts can be imparted using visualizations [14]. Thus animations can be developed to aid understanding of complex concepts such as Dijkstra algorithm. These animations can help the classroom teaching. Two training modes were used, with one in classroom to aid understanding, and the other to aid in programming.

Teaching of data structures in a creative way, Creative Lab with Active Participation (CLAP), is described in [12]. This provided an open problem solving approach, where solutions were articulated using an algorithm animation environment . The teaching is student-centric and the students had to take initiative. The CLAP course was conducted in student pairs.

An active learning approach for teaching data structures to postgraduate students is described in [15]. The aim was to increase the programming concepts learning in the lab. All exercises were open ended and the students could use their own ways and means to solve it. This encouraged students to come up with different applications from other courses being studied.

Interactive learning through visual environments for automatic feedback to students was used in [11]. Students also provided feedback to each other by peer reviewing and it was observed to produce good learning results. Use of graphics to teach algorithms and data structures was explored in [5]. Image processing and rendering projects were used to teach memory allocation and matrix manipulation. Computer graphics provided a mechanism to teach problem based learning.

Much of the above work is based on tools and visualizations to support learning. In contrast, our approach is focused on the active learning that takes place through group based peer interactions and collaborations in a problem-solving context.

3 METHODS AND MATERIALS

3.1 Learning environment

The data structures course was taught to the first cohort of students in July 2018. The course was delivered using a blended learning approach. To support the approach, online material was developed to present the theory of a range of data structures and examples of their implementation and use. The course materials were made available through a Virtual Learning Environment (VLE) and students were expected to study the online material before coming to the class. In the active learning class session, the instructor presented a real-world problem that could be solved using data structures which had been introduced in the online material. In class, concepts were reinforced and further explored through peer interactions both within and between groups, and finally the tutor

Trimester	Module	Language
Year 1	Programming 1 Programming 2	Java Java
Year 2	Data Structures and Algorithms	Java
Year 3	Big Data	Python

Table 1: Programming related courses on the programme

summarized the discussion by providing feedback on the strengths and weaknesses of the various group approaches.

The students were from a wide range of different countries and have experienced widely differing teaching methodologies and first programming languages in their previous education. However, they have on their current programme become accustomed to a blended approach and highly interactive peer learning activities in class. The challenge in this particular course was to apply this approach and exploit the students' collaborative skills in motivating and facilitating learning abstract concepts which often do not engage the interest of students.

3.2 Choice of language and data structure

The concepts taught during a typical computing science data structures curriculum are quite advanced. The programming related courses at the African college are shown in Table 1.

Students are exposed to Java during year 1. Java is a simple and elegant programming language for teaching data structures compared to C/C++. The use of frameworks such as Standard template Library (STL) is described in [16] which is important to teach the integration of user code with existing libraries. Teaching of data structures with Java language is described in [18], highlighting the improvements that the language has over C++.

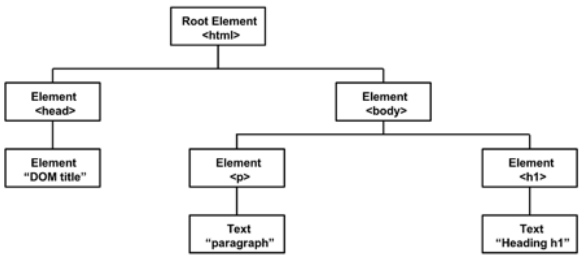
The advantage of using Java for the students was that they were already familiar with the core language concepts from the earlier programming modules. Hence they can concentrate on understanding and implementing the data structure constructs without being distracted or hindered by the language elements.

Data Structures curriculum was taught by reinforcing the theoretical concepts with innovative practical activities. The initial discussion about a data structure was used to relate the abstract concepts to real world usage of those concepts known to be familiar and of interest to the students. For example, for introducing the students to trees, students were given examples of Document Object Model (DOM) with which they were familiar by studying HTML and XML document structure from the web platform development module.

Figure 1 shows the DOM representation as a tree. As can be seen the document has nodes that correspond to the tree structure.

In order to highlight our strategy for improving the student experience by engaging them in group peer interactions, we have chosen the DOM based problem that was used to teach the tree data structure. The problem assigned to the students was to develop a pseudocode solution and Java code for populating a tree data structure by reading data from a given HTML document that uses a simple subset of HTML.

Figure 1: Document object model displayed as a tree.



3.3 Open plan environment

It is important for free peer interactions that the classroom activities are flexibly designed to encourage open discussions between the group members. We used the traditional white board approach for sharing the group's work in progress by assigning each group to one of the whiteboards that were installed along the walls all around the classroom. This physcial environment does not rely on any expensive technology, but it is a simple and effectively designed space.

This allowed for peer collaboration within a group without interfering with the discussions or disturbing the other groups. Also, it made it easier for the tutor to coordinate and interact with the group activities. The setting enabled the students in later stage of inter group discussion to look at the reasoning, approach and development of solutions by other groups.

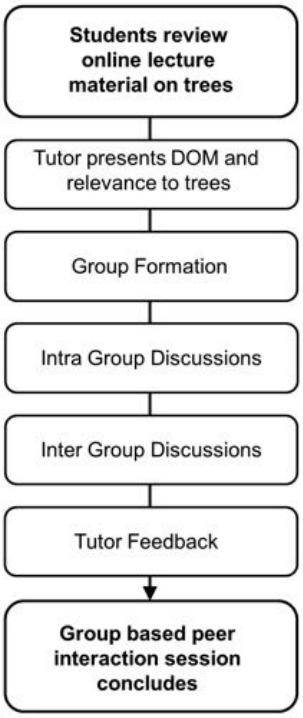
3.4 Group formation

The idea of group is central to our approach as the peer interactions take place both within and between groups. It was therefore critical to make a different group for the next problem. The groups were formed by the tutor by assigning students randomly to different groups. This helped students not only to work with different peers and hence with students of varying backgrounds but also helped them learn the group dynamics and peer support. Peer interactions helped to reinforce the data structures and algorithms concepts being studied providing an active learning environment for the students.

3.5 Facilitated learning

The sequence of actions with the learning activity is summarized in Figure 2. The problem to be solved was introduced to the students before they were divided into groups. This sets a baseline for all the groups, that is, for the DOM Tree scenario they were introduced to the construction of the DOM as a real-world problem that motivates the use of a data structure. The groups were required to fully develop a solution in a top-down manner, initially within the group, and are encouraged by the tutor to also capture their solution through diagrams, flow charts and pseudo code on the white-board. Students do not use their laptops when working on the problem, so that the focus is not on code and implementation, but the design of the algorithm and appropriate use of the relevant data structures,

Figure 2: Sequence of actions to communicate full understanding of a data structure.



so that students can develop their computational thinking. The tools used in the session aid and enhance the student learning and experience [10]. This also makes it possible for the tutor to see the progress and the thought process of the group. This is helpful to the tutor to later reinforce the correct approaches taken by different groups highlighting the differences and merits in each case.

4 GROUP PEER INTERACTIONS

In this section, we describe the group activities and show how these contributed to improve the students' understanding through peer interactions.

The interactions were closely watched by the tutor and students were prompted and encouraged to analyze their solutions as they were being developed. The interactions took place within a group to arrive at a mutually agreed solution to the given problem.

This also helped the tutor to encourage and direct the students in arriving at the class wide best solution and see for themselves the other possible solutions together with their strengths and weaknesses.

4.1 Student intra-group interactions

The design of the solution by the students required three distinct stages, (i) an initial informal discussion in the group with charting the program elements on the white board, (ii) iterative refinement of the solution with peer interaction which is also observed by the tutor, (iii) development of a fully refined solution. Figure 3 shows the free-form scribbling on the board for one of the groups to design and develop a group solution.

Figure 3: Output from group discussions on whiteboards

The initial discussions in the group sometimes resulted in emergence of one or more group leaders due to group dynamics where each member was trying to increase their influence. This was beneficial for the group because in peer learning they can quickly follow a leader to start developing their solution through distribution of work and coordination of effort.

The students were encouraged to use flowcharts and pseudocode to help outline, develop and discuss their solutions. This approach of a free form solution design had benefits similar to VTK [13] by helping students to understand the core tree data structure before implementing it in Java code.

The solutions were developed on the whiteboard and students were encouraged to graphically depict the solution as shown in Figure 3. This also made it easier for the tutor to move around between groups and discuss their progress and provide feedback to put them on a desired course of action. The whiteboards as an aid helped significantly as the tutor could easily scan the progress and direction of each group from a central position. The individual participation and leadership of each group also became apparent to the tutor who can then encourage those students who were not fully engaged with the group activities. The students in general seemed to be very keen to learn through peer interactions and took ownership of the learning process.

The algorithm efficiency was also considered during the solution development process as ultimately it decided how useful a solution really was in terms of time and space complexity. The emphasis was on developing efficient solutions [10]. It was critical for students to complete their assignments up to a good standard [9].

The development of a working solution was a good starting point and a motivator for the students to work hard to improve

their solution. After that the tutor encouraged the groups to have a deep look at their solutions and see how those could be made better. The in-group discussions and attempts to improve the group solution ensured that every group and student got to understand the weaknesses and strengths of various possible approaches to solving the given problem.

4.2 Student inter-group interactions

During inter group deliberations, the students took turns in discussing their group solution with other groups. The tutor assumed the role more like a moderator to keep the discussion on course. The objective was to make it easier for the students to arrive at the best solution to a given problem and how their group solution compares with other solutions. The students were found to be very excited and keenly took part to logically give arguments to support their approach and solution. The student understanding of the topic was helped by the fact that they were excited to ask questions and provide answers, displaying a scientific approach, positive criticism and feedback.

The process comprised of the following steps:

4.2.1 Presentation of solution to other groups. After the students in all the groups have had intra-group discussions and converged to a solution, then the groups were asked to present their solution and approach to the other groups. Other groups asked questions with an aim to understand the approach taken and to uncover any perceived weaknesses in the solution.

4.2.2 Peer Evaluation. Peer evaluation made other groups familiar with the data structure under consideration and thus students could clearly see the advantages and limitations of various approaches together with their time and space complexities.

4.3 Round-up Discussion

After all groups had presented their solution, the tutor described the relative merits of each approach and concluded by highlighting the best approach. Students had the opportunity to ask questions to the facilitator and any group members to further understand the proposed solutions or the facilitator's prefered approach.

At the end of the session, the tutor requests that students complete the implementation of the problem prior to the next session. Students share their code on Github Gist[1] where the facilitator and other students may leave feedback using the comments system. Students are encouraged to leave peer feedback and to iteratively improve their implementation with respect to feedback.

4.4 Benefits

The students perceived both the intra and inter group interactions to be of great benefit to their understanding of data structure concepts. The group interactions provided a means for the students to learn in a fun way. The student involvement in many practical exercises helps to reinforce the concepts and develops the skill set of the students [6].

The active learning environment in the group setting made the students more confident to voice and share their ideas. In some groups it was noticed that one or two peer teachers emerged that were leading the discussion. However it was found to be of benefit to all students.

The group interactions and discussion around various implementations of the same problem is important as this comparison and evaluation of the different solutions fosters better understanding of the critical concepts which are required by the student in future career as programmer, analyst etc. [10].

[1] http://gist.github.com

5 EVALUATION

The feedback received for the proposed technique was very positive and some students described Data Structures and Algorithms module as the best part of the programme and that it helped them understand the difficult concepts. The technique of collaborative peer learning was tested on a class of 37 students with varying programming backgrounds and skills. The interest of students in the module content was found to have increased considerably and the students rated their learning environment of data structures to be the best compared to other modules that were using only a flipped classroom approach.

5.0.1 Student Feedback. The feedback received for the proposed technique was very positive and some students described it as the best part of the study programme and that it helped them demystify and understand the difficult core module concepts.

5.0.2 Inspiration. Many students reported being really inspired as a result of taking ownership of the learning process through group based peer interactions. They reported that their initial barriers and fears of learning a difficult topic such as trees were adequately overcome.

5.0.3 Student experience. The availability of the online material and the expectation that they had to study it before coming to the classroom improved the student experience. The tutor led classroom activities reinforced the core concepts with the group interactions enabling the students to learn actively with their peers. The students who would otherwise not participate in classroom activities taking the whole classroom as a group, also feel empowered and contribute to the group discussion thus enhancing their learning in the process. The ultimate result is an enjoyable learning experience for the students.

5.0.4 Improvement in results. The module assessment comprised of both theoretical knowledge and concepts, and practical programming of data structures. The pass rates for the module were better than other modules where the techniques of group based peer interactions were not applied.

6 CONCLUSION AND FUTURE WORK

This paper has described the trialing of an innovative technique of creating an easy to access discussion medium to peer learn and teach an important and complex tree data structure concept in a classroom environment. The peer interactions made it possible to freely discuss their ideas and approaches to solve the given problem. The students could easily articulate their solution on the whiteboards readily available to the instructor and peers to see, comment and improve upon.

The objective was to make it easier for the students to interact and understand the complex and abstract data structure concepts through active learning. The interactions were facilitated by dividing the class into random equal-sized groups and encouraging them to devise a group solution to the given problem by making use of white board to freely share and comment ideas. The group solutions were then shared across groups and invited comments in terms of completeness and efficiency from other groups. It was observed that the students acquired thorough understanding of the data structure under study and felt empowered and confident to tackle complex problems in data structure curriculum.

In our future work we aim to improve the benefits of this study further by announcing the group problem before the actual class so that the students get more time and a chance to devise their individual solutions and before coming to the class. We would also consider to record the understanding of students of a particular data structure both before and after group interactions in order to quantify the benefits achieved through peer learning.

Empowering the students with the right training of the data structures prepares them well for the diverse and complicated data structures that they are most likely to encounter in real-world settings. [10].

REFERENCES

[1] Ryan S Baker, Michael Boilen, Michael T Goodrich, Roberto Tamassia, and B Aaron Stibel. 1999. Testers and visualizers for teaching data structures. *ACM SIGCSE Bulletin* 31, 1 (1999), 261–265.

[2] Suhas Bhagate and Uday Nuli. 2016. Innovative Methods for Teaching Data Structures and Algorithms. *Journal of Engineering Education Transformations* (2016).

[3] Martin J Biernat. 1993. Teaching tools for data structures and algorithms. *ACM SIGCSE Bulletin* 25, 4 (1993), 9–12.

[4] Donald Chinn, Phil Prins, and Josh Tenenberg. 2003. The role of the data structures course in the computing curriculum. *Journal of Computing Sciences in Colleges* 19, 2 (2003), 91–93.

[5] Andrew T Duchowski and Timothy A Davis. 2007. Teaching algorithms and data structures through graphics. In *Proc. of Eurographics*.

[6] Alan Fekete. 2002. Teaching data structures with multiple collection class libraries. In *ACM SIGCSE Bulletin*, Vol. 34. ACM, 396–400.

[7] Sahalu Junaidu. 2008. Effectiveness of multimedia in learning and teaching data structures online. *Turkish Online Journal of Distance Education* 9, 4 (2008), 97–107.

[8] Danny Kopec, Richard Close, and Jim Aman. 1999. How should data structures and algorithms be taught. In *ACM SIGCSE Bulletin*, Vol. 31. ACM, 175–176.

[9] Ramon Lawrence. 2004. Teaching data structures using competitive games. *IEEE Transactions on Education* 47, 4 (2004), 459–466.

[10] Karen Mackey and Howard Fosdick. 1979. An applied computer science/systems programming approach to teaching data structures. In *ACM SIGCSE Bulletin*, Vol. 11. ACM, 76–78.

[11] Lauri Malmi and Ari Korhonen. [n. d.]. A pedagogical approach for teaching data structures and algorithms.

[12] Veijo Meisalo, Erkki Sutinen, and Jorma Tarhio. 1997. CLAP: teaching data structures in a creative way. In *ACM SIGCSE Bulletin*, Vol. 29. ACM, 117–119.

[13] Ogen Odisho, Mark Aziz, and Nasser Giacaman. 2016. Teaching and learning data structure concepts via Visual Kinesthetic Pseudocode with the aid of a constructively aligned app. *Computer Applications in Engineering Education* 24, 6 (2016), 926–933.

[14] Deng Rui, John T Thompson, Yang Hong, Zhou Xing-sheng, Liu Ke-jing, and Neil Alexander Macintyre. 2008. Imagery training in the teaching of the data structure curriculum. *ACM SIGCSE Bulletin* 40, 4 (2008), 92–94.

[15] C Sujatha, GN Jayalaxmi, and GK Suvarna. 2012. An innovative approach carried out in data structures and algorithms lab. In *2012 IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA)*. IEEE, 1–4.

[16] Josh Tenenberg. 2003. A framework approach to teaching data structures. In *ACM SIGCSE Bulletin*, Vol. 35. ACM, 210–214.

[17] Zhao Wang. 2012. The Research on Teaching Ideas ofâĂİ Data Structure and AlgorithmâĂİ in Non-computer Major. In *Advances in Computer Science and Education*. Springer, 249–254.

[18] Mark Allen Weiss. 1997. Experiences teaching data structures with Java. *ACM SIGCSE Bulletin* 29, 1 (1997), 164–168.

[19] Daniel Zingaro, Cynthia Taylor, Leo Porter, Michael Clancy, Cynthia Lee, Soohyun Nam Liao, and Kevin C Webb. 2018. Identifying Student Difficulties with Basic Data Structures. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ACM, 169–177.

DaST: An Online Platform for Automated Exercise Generation and Solving in the Data Science Domain

ABSTRACT

Over the last few years data science has emerged both as a new research field and as an educational domain that attracted a large number of researchers and data practitioners. Although data science research is developing at a high pace, the educational process in the field has been left behind in terms of educational tools and practices, despite the high number of data science courses offered and the number of involved stakeholders (professors, tutors, and students). The present work aims to cover the gap of educational data science tools by proposing a novel platform for online exercise solving in the data science domain; the platform, coined Data Science Tutor (DaST), is a free online tool that offers automated step-by-step exercise solving in a variety of data science algorithms and techniques aiming at giving insight to the particularities of each algorithm. The solutions of the exercises are accompanied with in-context explanations that refer to the operation of the respective algorithm/technique, and are compatible with the terminology and the methodology in popular textbooks. Through the proposed platform students in the data science field or in related courses (e.g., machine learning, information retrieval) may get solutions for different types of exercises and focus on the details of each algorithm, while tutors (lecturers, lab assistants) may easily produce a wide variety of exercises with the accompanying solutions and use them in classroom or as an auxiliary tool for test correction. To the best of our knowledge, the proposed platform is the first educational tool that aims at the data science field, and in its one year of operation has been warmly accepted by departments worldwide.

CCS CONCEPTS

• **Information systems** → *Data management systems*; • **Applied computing** → **Education**; **Computer-assisted instruction**; **Interactive learning environments**; **Collaborative learning**.

KEYWORDS

data science, online platform, exercise solving, tertiary education

ACM Reference Format:

. 2019. DaST: An Online Platform for Automated Exercise Generation and Solving in the Data Science Domain. In *Proceedings of CSERC 2019: 8th Computer Science Education Research Conference (CSERC 2019).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

CSERC 2019, November 18–20, 2019, Cyprus

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Lately, the explosion of available data and the need to extract insight out of this data triggered the creation of a new interdisciplinary field that involved areas ranging from applied mathematics to statistics and from artificial intelligence to machine learning. This field, known as data science [5, 6, 15, 25], stimulated changes both in research and education in universities around the world. Towards this new direction, basic and applied research has focused on ways of managing, analysing, extracting, and visualising the information that resides behind the data, and utilising it in decision-making.

Naturally, data science requires skilled and appropriately trained data scientists, and this need led several departments to incorporate data science related courses, e.g., big data management, in their (undergraduate or postgraduate) studies program. These courses were offered alongside traditional information/data management courses such as machine learning or information retrieval, thus creating a large educational "market" involving a significant number of people (tutors, lab assistants, students) that teach or study methods, algorithms, and techniques in the broad area of data science [4, 11]. Despite this explosion in interest for data science related studies, there is a severe lack of educational tools that could assist the process of teaching and learning in such a broad and rapidly evolving scientific field [8, 11, 22, 26].

This paper presents a *novel online platform*, coined Data Science Tutor (DaST), that is used for automated problem solving in the data science domain. Users may automatically design and solve exercises for data science algorithms by resorting to a completely free and easy-to-use tool. The solutions of the exercises are presented in a step-by-step manner, followed by explanations that refer to the operation of the respective algorithm or technique, and are compatible with the terminology and the methodology in popular textbooks, like [1, 3, 17, 19]. The DaST platform acts as a computer-assisted tutor for a wide range of data management techniques and algorithms related to the broader field of data science, where trainers and trainees may create their own exercises, resort to the ready-to-use example ones, or solve exercises submitted by other users. The proposed platform targets two main categories of users, revealing the different objectives for each category. Undergraduate or postgraduate students in the field of data science (i) have the potential to learn the particularities of each algorithm and identify cases that are not covered by the in-class teaching material and (ii) study in an interactively assisted way the reasoning behind problem solving and the exercise solution process for different types of exercises. Similarly, through the proposed platform, the tutors of a course (lecturers or lab staff) may easily produce a wide variety of exercises, with the accompanying solutions, and use them in the classroom or as an auxiliary tool for producing and correcting student tests. The DaST platform is in use for about one year and is continuously enriched with new functionality and algorithms. To

the best of our knowledge, the DaST platform is the *only online educational tool* available in the data science domain and has already been warmly accepted by faculties worldwide.

The rest of the paper is organised as follows. In Section 2, we present the available approaches related to our platform. Section 3 describes the architecture, the capabilities and the provided functionality of the DaST platform, while in Section 4, we present and discuss the user acceptance of the proposed platform. Finally, Section 5 gives directions for future research and concludes the paper.

2 RELATED WORK

Our research is broadly related to approaches that deal with the organization and management of the study, including educational portals and online learning environments. In [20] an educational portal based on personalized information retrieval is presented; the proposed solution aims to act as an aid at the learning process by retrieving information pertaining to the context of the problems studied by the students. In a similar spirit, [27] visualizes a user model to achieve adaptive information retrieval in a learning environment, while [12] addresses effective content-delivery with the help of a university intranet, and collates the teacher-moderated online forum with the associated intranet portal. Online educational tools for guiding learners through the programming process have also been lately proposed; [14] focuses on the programming process by providing a tool that helps students learn the necessary steps towards designing a computer program, while [21] concentrates on JavaScript. Finally, [7] presents the educational goals and objectives when dealing with a broad teaching domain and aims at student collaboration for analyzing and tackling difficult problems.

Works that are most closely connected to our platform include [9, 10, 16]. In [9] an educational system that assists students in learning and tutors in teaching is presented; however, this system deals only with search algorithms within the artificial intelligence (AI) domain. Towards a different direction, [10] proposes a computer-supported learning environment using an information retrieval (IR) game, aiming to provide a realistic environment for demonstrating the performance of queries in different types of search situations. The experimental evaluation of this system revealed that students found different characteristics of the IR game both enhancing and inhibiting learning. Similarly, [16] presents an AI game platform, which includes state-of-the-art algorithms and aims to bring a rich and fun learning experience in the AI domain. Finally, problem meta-heuristic solver [13] is an educational software tool that aims at the generic study of the concepts related to the optimization field, by covering the main stages when solving optimization problems; the results obtained by its usage suggest that the tool improves the understanding of the theoretical concepts and reduces the workload of the students when implementing optimization methods.

Educational tools prove important not only to the computer science field, but also to other fields like mathematics or art. Wolfram Alpha is a platform similar to DaST, that targets exercise solving in the field of mathematics [2, 18, 23]. This platform is a computational knowledge engine that is able to answer factual mathematical queries, providing (with a subscription fee) a step-by-step solution for a wide variety of mathematical problems. Maestoso [24] is an intelligent computer-assisted educational application that allows people to learn the theory of music through sketching, and enables them to progress through provided lessons of important fundamentals in music theory.

Although all the above approaches concern different aspects of on-line student aids, none of them focuses on computer-assisted learning on the data science field, despite the fact that research [22] has indicated the need for such tools in a rapidly evolving domain that involves a broad spectrum of techniques and algorithms. This need for learning environments related to data science inspired us to design and develop the DaST platform as an *on-line, free-of-charge tool* that will enhance the students' learning process and assist tutors in their teaching tasks.

3 THE DAST PLATFORM

In this section, we describe the architecture, the capabilities and the quality of the provided functionality for the DaST platform.

The DaST platform has been implemented using widespread programming languages and tools, while its architecture is stratified with the goal of creating an easy way to add new techniques and algorithms. It is entirely based on php and javascript, fully compatible with the latest HTML and CSS standards, while for displaying widgets (e.g., charts or graphs) it uses the Google Chart API. At this stage, the platform is hosted on a typical Ubuntu Linux (version 16.04.3) server with 64bit / Dual Core 2GHz / 4GB RAM.

3.1 Features and functionality

Access to the platform is possible via any modern browser and is provided for free, does not require any sort of registration, and does not collect any type of personal information. Thus, by just typing the web address of the DaST platform, [1] the user is transferred to the graphical interface of the proposed application (Figure 1) and may benefit from all the functionality provided by the DaST platform. There, a variety of algorithms and techniques related to the data science domain are presented, each alongside three different choices meant for creating, and subsequently automatically solving, an exercise. The user may choose the algorithm/technique he is interested in practicing or for producing auxiliary material, and may also select to create the exercise from scratch, get a randomly generated one, or by resorting to an already submitted exercise by another user. In its current version, the DaST platform provides the opportunity to produce, practice with, and solve several types of exercises for fourteen (14) different algorithms alongside their variants. The DaST platform provides the users with the choice to print or export to a PDF file the exercise with the provided step-by-step solution. The whole interaction with the platform, as well as the provided step-by-step solutions and comments of the exercises are in English, and all the provided information on the algorithms references the relevant literature.

All algorithms and techniques available in the DaST platform are accompanied with a concise presentation of the related theory as well as assistance for the type of entry required to create an exercise for each algorithm or technique. It is worth mentioning that the DaST platform addresses the tertiary education and focuses on a broad and demanding field of science; the algorithms and techniques related to the data science domain require specialized knowledge,

[1] Address is disclosed for facilitating the double-blind review process.

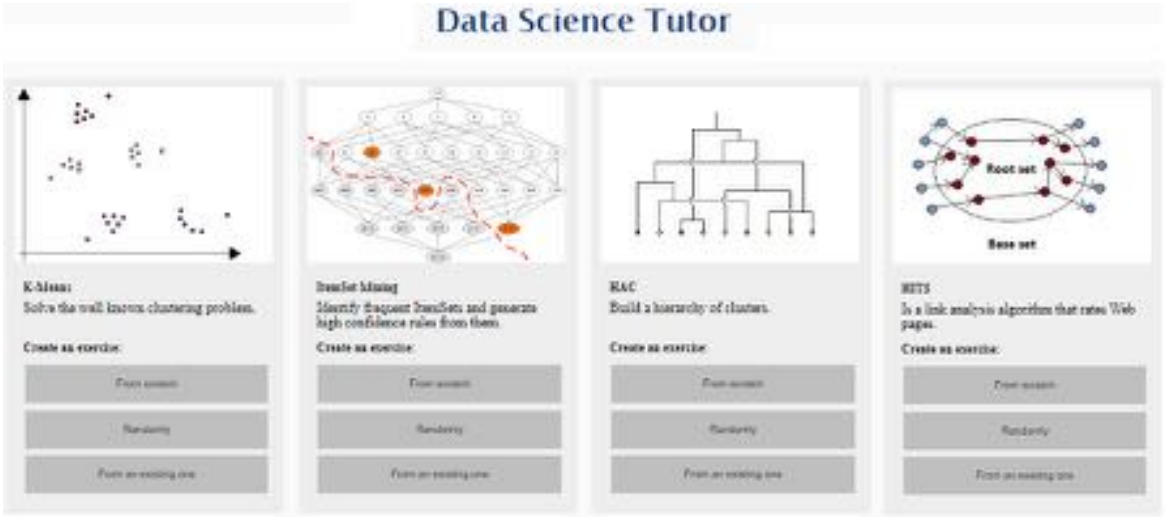


Figure 1: Part of the home screen of the DaST platform presenting (some of) the available algorithms

even for the data entry needed to create an exercise. For this reason, the provision of a guide for each algorithm/technique was deemed necessary after the requirements analysis and personal interviews conducted to users of the platform. The guide provided by the DaST platform for each algorithm or technique contains all the necessary explanations that refer to its operation (and to the step-by-step solutions of the corresponding exercises), is fully compatible with the terminology and the methodology met in popular textbooks, and is used within a wizard-style environment to facilitate the exercise creation process.

In what follows, we present the three different ways the user may utilise to create an exercise, after having chosen an algorithm or technique provided by the DaST platform.

- **From scratch.** In this case, the user may create an exercise by providing/uploading at the DaST platform his own data. The process for entering the necessary input data for each algorithm is done using a simple, user-friendly wizard that guides the user in filling in the required fields. User data input is supported in three ways: (i) by using the online graphical environment, (ii) by uploading an appropriately formatted text file containing the input data (parameters still have to be inserted through the graphical interface), or (iii) by uploading an appropriately formatted XML file containing all the necessary parameter setups and data input that will allow the creation of the exercise (and its subsequent solution). The different options for providing input address the needs of both beginner/occasional users and also more expert ones who often require an easy, fast, and semi-automated way to create exercises. For all file-based input the system provides example (.txt and .xml) files for downloading, while to support interoperability with other applications the XML schema is also available through the platform.
- **Radomly.** In this case, the chosen algorithm or technique is fed with randomly generated data and parameters. Choosing this way to create an exercise, the interested user has the opportunity to understand the functionality of an algorithm by studying the step-by-step solution of an exercise, without being puzzled with the data entry to create the exercise. This choice mostly targets students or beginners that are looking for a simple and straightforward way to familiarise with data science concepts and methods through exercise solving. For repeatability reasons the random exercise generation process asks the user to input a seed that may be used again later on to recreate the exercise. If none is provided, the system generates one seed by resorting to a time variable, and reports it to the user for future reference.
- **By using an existing exercise.** In this case, the user may select to use and solve one of the exercises that were previously submitted by another user. Please notice that user anonymity is fully protected as users are not required to register to the platform and the application does not relate in any way which user created which exercise. The users may change the data and/or the parameters of submitted exercises and study the way the algorithm and the provided solution are affected. The selection between the existing exercises is performed either randomly or based on a specific feature (e.g., difficulty of the exercise, type of user –tutor or student– that created it). Using exercises provided by others, while protecting the anonymity of the users, provides the DaST platform with an interesting social dimension. This feature is currently under alpha testing.

3.2 Supported algorithms

The full list of algorithms that are available in the current edition of the DaST platform are presented below. The algorithms are classified here in research fields related to the data science domain.

3.2.1 Unsupervised clustering. Unsupervised clustering identifies previously unknown patterns in data sets without pre-existing labels. In essence, unsupervised clustering involves a set of simple-yet-powerful tools for identifying groups of objects, and typically serves as an introduction for more complicated techniques that involve also (semi-)supervised counterparts.

K-Means is the most well-known algorithm for unsupervised clustering of objects; objects in DaST are represented as vectors with up to 10 dimensions – remember that the aim is exercise solving, not real-world operation. Users input the vectors and the algorithm setup. The number of dimensions is automatically inferred (vectors with less dimensions are padded with zeros), while the user may tune the number of clusters, the number of iterations, the initial cluster seeds, the distance method (Euclidean or Manhattan), and the algorithm variant (sequential of on-line).

K-Medoids is another well-known unsupervised clustering algorithm that is robust to noise and outliers. The options provided for the algorithm are the same as those described above for algorithm K-Means.

HAC (Hierarchical Agglomerative Clustering) is a popular bottom-up clustering algorithm that merges objects to create a dendrogram, in contrast to the flat clustering algorithms presented above. The DaST platform supports the four most popular variants of HAC (single-link, complete-link, centroid-link, average-link), and the user may also tune the number of iterations and the distance method (Euclidean or Manhattan).

3.2.2 Classification. Classification identifies to which among a set of categories a new observation belongs, based on training with data that contain known data memberships.

Naive Bayes is the simplest family of probabilistic classifiers that operate with the assumption of independence between the features. The DaST platform supports document classification, and users are required to input only the number of documents and the document terms.

3.2.3 Information Retrieval Models. These models typically refer to a representation of documents and queries that is suitable to facilitate the retrieval of information relevant to a user need.

Boolean Model is a standard textbook representation for documents/queries that assumes a binary vector representation. Users input the query, the documents, and the scoring method (Euclidean or Manhattan).

VSM (Vector Space Model) is the most popular representation for documents/queries that assumes a real-valued vector representation. Users input the query and documents, the model variant (12 variants are supported), and the distance method (Euclidean or Manhattan).

3.2.4 Link Analysis. Link analysis refers to graph-analysis techniques used to evaluate relationships (connections/edges) between nodes (vertices) in a graph.

PageRank was introduced in the Google search engine as a way to measure the importance of web pages. The user inputs the graph (by means of edges between vertices), the number of iterations, and the Pagerank variant (with/without dampening and Google matrix usage).

HITS (Hyperlink-Induced Topic Search) is a well-known textbook algorithm that rates web pages based on the concepts of hubs and authorities; user input is the same as that of Pagerank provided above.

3.2.5 Decentralised object location. Inspired from work in P2P networks, decentralised object location and routing currently form the backbone for most big data filesystems including GFS, HDFS, Cassandra and others.

Chord is the most famous object location protocol that is currently taught as the prominent representative of distributed hash tables (DHTs). Its key-value pair philosophy blends well with many big data system concepts. The user input involves the size of the Chord ring, the active Chord nodes, and other distributed search-specific data.

3.2.6 Association rule mining. Association rule mining is a rule-based machine learning method for discovering interesting relations between transactions in large databases; it discovers interesting rules in databases using different measures of interestingness.

Apriori uses a breadth-first search strategy to count the support of itemsets and constitutes the most popular textbook algorithm for association rule mining. To create an exercise the user inputs the transaction database (up to 20 transactions – remember that the aim is exercise solving, not real-world operation), and algorithm parameters (i.e., support and confidence).

The DaST platform is regularly enriched with new algorithms and methods from the data science domain; additions are simple, due to the architectural design of the platform that allows developers to deploy new algorithms independently. In the future we plan to open-source the platform code and create a community of developers that will help the expansion of the platform and the verification of the submitted code.

3.3 Solution visualisation & step-by-step guide

As mentioned above, when selecting an algorithm/technique, the user is presented with a wizard-styled web interface to input the appropriate data and parameters for exercise creation. After the exercise creation, the user is presented with the solution of the exercise in a step-by-step manner, emphasizing the details of the algorithm. An example of such a solution is shown (in part) in Figure 2, where an exercise on the Apriori algorithm for itemset mining is solved. Please notice the stepwise execution, the color-coding and graphical elements marking the pruned and non-pruned itemsets, and the concise writeup that is meant to improve the readability and understanding of the solution.

If applicable, an appropriate visualisation of various exercise elements is also included in the provided solution; for example in link analysis the user is provided with a visualisation of the input graph, while in itemset mining a color-coded latice representation of the frequent itemsets is given as a visual aid of pruned and candidate itemsets (see top right corner of Figure 2 for an example). These graphical representations of the input data and the solution space help users to get a better insight of the functionality of the examined

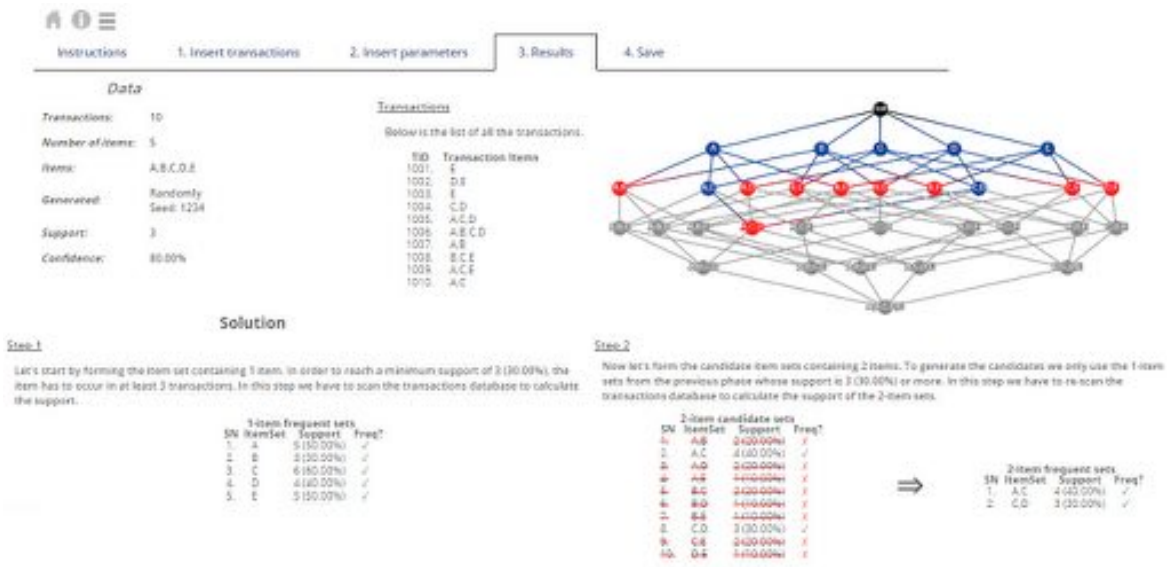


Figure 2: Solution of an itemset mining exercise (algorithm Apriori), and a graphical representation of the (in)frequent itemsets

algorithm/technique and help tutors provide better explanations to students.

For the interested reader we have created an anonymised Google Drive folder that contains more screenshots of the DaST platform; the folder is accessible at http://tinyurl.com/y4h7u3za.

4 PRELIMINARY EVALUATION

The DaST platform is online and fully functional for about one year, and we had the opportunity to make a first assessment of its acceptance by the community of tertiary education. According to user feedback we received, the platform has so far been used in a variety of courses related to the data science domain worldwide and is utilised in the context of both undergraduate and postgraduate curricula. Our assessment was carried out over three different axis: (a) qualitative and quantitative assessment from standarised anonymous questionnaires handed out from the quality assurance unit of our university, (b) student performance measurement by using a control and a test group, (c) a questionnaire survey of our own focusing on the platform specifics. The results of our preliminary assessment are reported below; we are currently in the process of performing a large scale assessment in cooperation with departments that have used our platform.

4.1 Evaluation from the quality assurance unit

Given the short life span of the platform, the time to conduct a full-scale evaluation of the learning outcomes that derive from it was limited. However, the acceptance of the DaST platform by the students of our department was encouraging. More specifically, our students evaluated the DaST platform by responding to the standarised questionnaires that are distributed by the quality assurance unit of our university and target the evaluation of the department's courses. The students' comments about the DaST platform were very positive and mostly addressed the usefulness of the platform with respect to (i) helping them "prepare for the final exams" in data science related lessons, (ii) providing them with "insights on the operation" of the respective algorithms, (iii) assisting them get non-trivial "peculiarities of the algorithms that are typically not even addressed in class", and (iv) "assessing the results and marking of tests and homeworks". One of the questions included in the questionnaires disseminated by the quality assurance unit referred specifically to the online educational material and its contribution to the better understanding of the Information Retrieval course; the students replied that the educational material was helpful and provided an overall score of 4,33 out of 5.

4.2 Students performance report

The second axis of our assessment involved at looking into student performance metrics, such as the achieved student grades at the final written examinations. We separated the students in one of the related courses into two different groups of ten students each; Group A had the opportunity to use the DaST platform during the semester, while Group B was not aware of the platform and did not have access to it. The chart shown in Figure 3(a) reveals the impact of the platform on student grades as marked by the corresponding instructor; the students having used the DaST platform achieved on average 36% better score (mark range is between 0-100) in the final written examinations when compared to the students that did not use the platform. Although our sample is not big enough for extracting accurate statistical results, the particular assessment indicates that the DaST platform was beneficial to the student performance and the learning process itself.

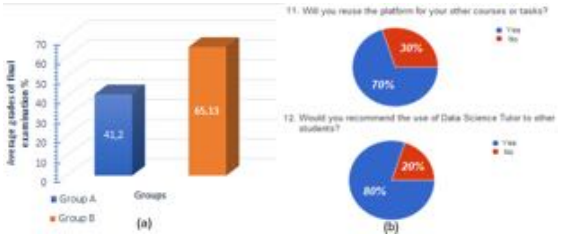


Figure 3: (a) Average score in final exams, (b) aggregate answers to (some of) the questions

4.3 Questionnaire survey

Finally, the third assessment axis involved the design of a specialised questionnaire survey to gather information specifically concerning the DaST platform and its different dimensions (e.g., usefulness, usability, user experience). Our survey (shown in Figure 4) was performed by directly forwarding the questionnaire to users that provided email feedback to us and asking them to fill it in anonymously. The positive feedback we received, shown in part in Figure 3(b), is representative of the overall positive stance of the users' response to our questionnaire. Therefore, although an

Figure 4: Part of the questionnaire

extensive user study of the DaST platform has not been performed so far, user acceptance and preliminary user feedback are positive and encouraging.

5 CONCLUSIONS AND FUTURE RESEARCH

The proposed DaST platform is a novel online tool that facilitates the learning and teaching of data science related algorithms/methods through *automated, step-by-step problem solving* of *user-generated* or *automatically created* exercises. The proposed platform is continuously expanded with new algorithms and functionality, and has so far been warmly accepted by the data science community in tertiary education. Future research directions and extensions include (i) extended user studies to extract macroscopic findings concerning the learning outcomes and related targets, (ii) introducing different levels of difficulty in exercise generation, (iii) creating an online community around DaST to help development and result verification, and (iv) integration with popular learning management systems such as Open eClass and Moodle.

REFERENCES

[1] C.C. Aggarwal. 2015. *Data Mining: The Textbook.* Springer.
[2] D. Arnau, M. Arevalillo-Herráez, L. Puig, and J.A. González-Calero. 2013. Fundamentals of the design and the operation of an intelligent tutoring system for the learning of the arithmetical and algebraic way of solving word problems. *Computers & Education* 63 (2013).
[3] J. Berman. 2013. *Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information.* Morgan Kaufmann.
[4] R.J. Brunner and E.J. Kim. 2016. Teaching Data Science. *Procedia Computer Science* 80, C (2016).
[5] W.S. Cleveland. 2001. Data science: an action plan for expanding the technical areas of the field of statistics. In *International Statistical Review.*
[6] V. Dhar. 2013. Data Science and Prediction. *Commun. ACM* 56, 12 (2013).
[7] E.N. Efthimiadis, J.M. Fernandez-Luna, J.F. Huete, and A. MacFarlane. 2011. *Teaching and Learning in Information Retrieval.* Vol. 31. Springer.
[8] G.Press. 2013. Data Science: What's The Half-Life Of A Buzzword? Forbes. https://www.forbes.com/sites/gilpress/2013/08/19/data-science-whats-the-half-life-of-a-buzzword/#47952ffa7bfd
[9] F. Grivokostopoulou, I. Perikos, and I. Hatzilygeroudis. 2016. An Educational System for Learning Search Algorithms and Automatically Assessing Student Performance. *International Journal of Artificial Intelligence in Education* 27 (2016).
[10] K. Halttunen and E. Sormunen. 2000. Learning Information Retrieval through an Educational Game. Is Gaming sufficient for learning? *Education for Information* 18, 4 (2000).
[11] S.C. Hicks and R.A. Irizarry. 2018. A Guide to Teaching Data Science. *The American Statistician* 72 (2018).
[12] K. Viswanathan Iyer. 2017. A dynamic intranet-based online-portal support for Computer Science teaching. *Education and Information Technologies* 22, 3 (2017).
[13] C.E. Izquierdo, I. López-Plata, and J.M. Moreno-Vega. 2015. Problem MetaHeuristic Solver: An educational tool aimed at studying heuristic optimization methods. *Computer Applications in Engineering Education* 23, 6 (2015).
[14] H. Keuning, B. Heeren, and J. Jeuring. 2014. Strategy-based Feedback in a Programming Tutor. In *Proceedings of the International CSERC.*
[15] J. Leskovec, A. Rajaraman, and J.D. Ullman. 2011. *Mining of Massive Datasets.* Cambridge University Press.
[16] W. Li, H. Zhou, C. Wang, H. Zhang, X. Hong, Y. Zhou, and Q. Zhang. 2019. Teaching AI Algorithms with Games Including Mahjong and FightTheLandlord on the Botzone Online Platform. In *Proceedings of the ACM Conference on CompEd.*
[17] J. Lin and C. Dyer. 2010. *Data-Intensive Text Processing with MapReduce.* Morgan Claypool.
[18] M. Lvov, I. Chernenko, L. Shishko, and E. Kozlovsky. 2018. Mathematical Models of Supporting the Solution of the Algebra Tasks in Systems of Computer Mathematics for Educational Purposes. In *Proceedings of the 14th ICTERI. Integration, Harmonization and Knowledge Transfer Workshops.*
[19] C.D. Manning, P. Raghavan, and H. Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA.
[20] L. Nakayama, V.N. de Almeida, and R. Vicari. 2004. A Personalized Information Retrieval Service for an Educational Environment. In *Proceedings of the International Conference on ITS.* Springer.
[21] H. Passier, S. Stuurman, and H. Pootjes. 2014. Beautiful JavaScript: How to Guide Students to Create Good and Elegant Code. In *Proceedings of the International CSERC.*
[22] V. Putnam and C. Conati. 2019. Exploring the Need for Explainable Artificial Intelligence (XAI) in Intelligent Tutoring Systems (ITS). In *Joint Proceedings of the 24th ACM IUI Workshops.*
[23] S. Shirai, T. Fukui, K. Yoshitomi, M. Kawazoe, T. Nakahara, Y. Nakamura, K. Kato, and T. Taniguchi. 2018. Intelligent Editor for Authoring Educational Materials in Mathematics e-Learning Systems. In *Proceedings of the 6th ICMS.*
[24] P. Taele, L. Barreto, and T.A. Hammond. 2015. Maestoso: An Intelligent Educational Sketching Tool for Learning Music Theory. In *Proceedings of the 29th AAAI Conference on AI.*
[25] S. Tansley and K.M. Tolle. 2009. *The Fourth Paradigm: Data-intensive Scientific Discovery.* Microsoft Research.
[26] P. Warden. 2011. Why the term "data science" is flawed but useful. O'Reilly Radar. http://radar.oreilly.com/2011/05/data-science-terminology.html
[27] S. Willms. 2003. Visualizing a User Model for Educational Adaptive Information Retrieval. In *Proceedings of the International Conference on UM.* Springer.

DigitalJS: a visual Verilog simulator for teaching

Marek Materzok
University of Wrocław
Institute of Computer Science
Wrocław, Poland
marek.materzok@cs.uni.wroc.pl

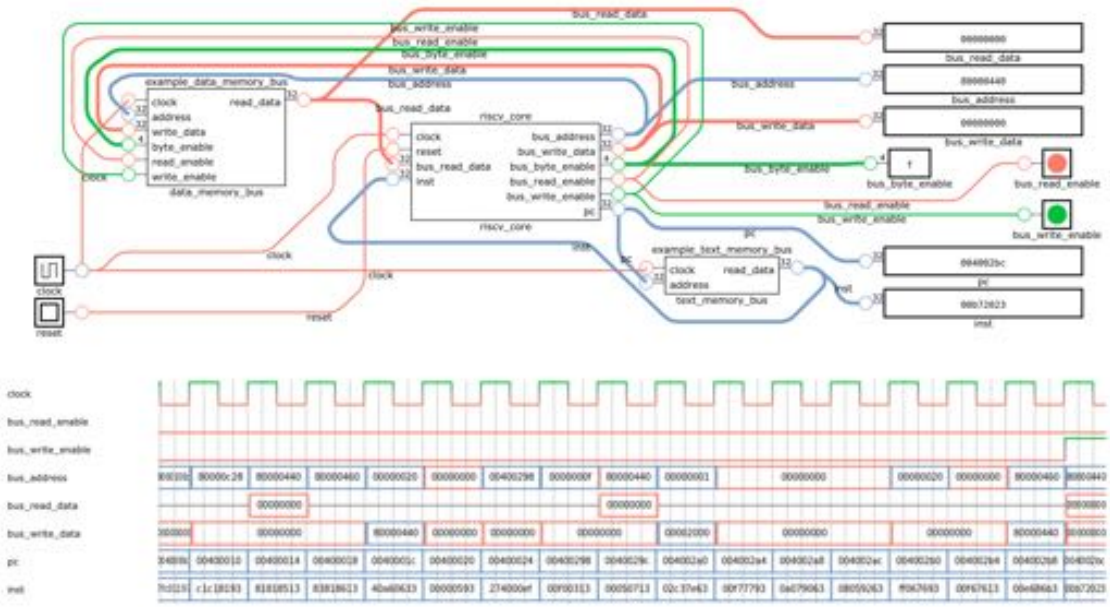


Figure 1: A RISC V core simulated in DigitalJS

ABSTRACT

This paper describes a visual circuit simulator tool designed for teaching students digital circuit design. The tool runs in the browser and is simple to use. It allows to visualize the synthesized circuit generated from Verilog/SystemVerilog code and interact with it.

CCS CONCEPTS

• **Applied computing** → **Education**; • **Hardware** → *Hardware description languages and compilation*; **Software tools for EDA**; • **Social and professional topics** → **Computing education programs**;

KEYWORDS

Verilog, Yosys, HDL, logic synthesis, logic simulation, teaching

ACM Reference Format:
Marek Materzok. 2019. DigitalJS: a visual Verilog simulator for teaching. In *Proceedings of 8th Computer Science Education Research Conference (CSERC 2019).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

CSERC 2019, November 2019, Larnaca, Cyprus

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Thanks to FPGAs (Field Programmable Gate Arrays), creating practical digital circuits no longer requires expensive silicon production, and their reconfigurability means that digital circuits can be designed similarly to software, with easy experimentation and fast prototyping. The availability of FPGA SoC chips with traditional CPUs integrated, and of PCI Express boards with FPGAs, means that FPGAs can be used as domain-specific accelerators, like GPUs are. This makes hardware design an useful skill for a software developer or computer scientist to have, and therefore an useful skill to teach computer science students.

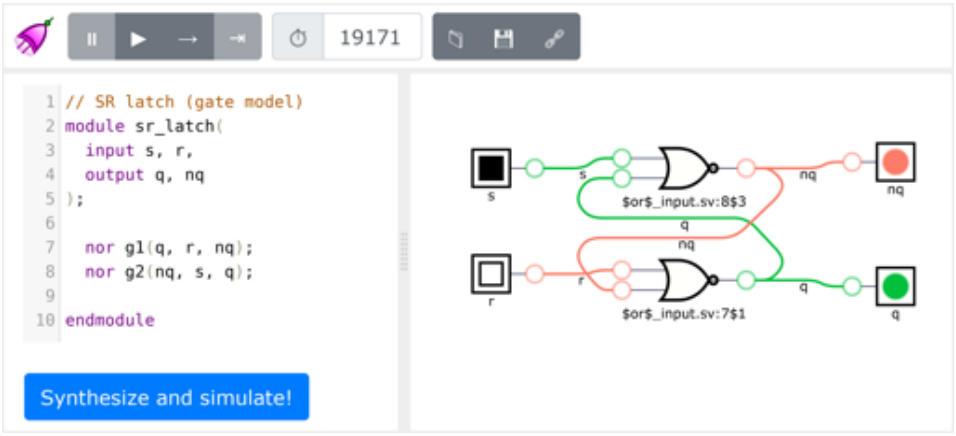


Figure 2: Screenshot of the DigitalJS online demonstration app, simulating an SR latch

But how do you do it? Thanks to hardware design languages (HDLs), such as Verilog [9], SystemVerilog [10] and VHDL [8], a computer science student familiar with C can quickly get started. Unfortunately, using them for this purpose has certain pitfalls. They are modeling languages, and it is possible to write code which does not map well (or at all) to hardware. Computer science students in particular might try to use them, by force of habit, like a programming language for CPUs, which would lead to code which runs just fine in simulation, but synthesizes to non-working or badly working hardware.

The author believes that it would be a great help for students if they are provided with a tool which reminds them that, even though they are writing code, they are actually designing hardware. Such a tool could help them visualize the structure and workings of the circuit they are designing, allow to inspect it, interact with it, and quickly see the effects of their changes in HDL code. This paper describes DigitalJS, a tool in proof-of-concept stage developed for this purpose.

The tool is currently being used for the first time in an introductory course on digital circuit design. It was positively received by the students.

2 DESIGN GOALS

DigitalJS was designed specifically for teaching digital design, simultaneously with elements of Verilog/SystemVerilog HDL, to undergraduate computer science students. With this in mind, the following design goals were set:

Focus on synthesis. It is well known that it's possible to write Verilog code which gives different behavior in the synthesized circuit than in the simulation [14]. This is a consequence of the fact that Verilog is at heart a language for programming concurrent processes, communicating via wires and events, and some processes or process interactions do not have natural representations as circuits. Experience with teaching students, my own and other educators' [5], suggests to adapt a synthesis first approach, where the students are not taught simulation-only constructs until they have mastered circuit synthesis. Therefore, the tool will only accept the synthesizable subset of Verilog, and simulate the circuit using the synthesized, gate-level schematic.

Support for hierarchical designs. Hierarchical modules are at the heart of digital design. Transistors combine to form gates, gates combine to form basic combinatorial and sequential circuits like multiplexers, encoders and flip-flops, which combine to form more complex circuits like finite state machines, memories and controllers, and so on. To capture this, support for modular designs must be simple and natural. Verilog modules need to correspond to subcircuit blocks.

Readability. The simulated circuit's construction must be readable. This is very important for students, because most of them do not have previous experience with digital design, and therefore even minor readability problems can be distracting or confusing. To achieve this goal, the tool needs to use standard textbook symbols (e.g. for gates and other parts), markings (clock lines, bus sizes, etc.) and conventions (e.g. inputs on the left, outputs on the right). Also, to make clear the correspondence between the Verilog code and the resulting circuit, parts and nets must be labelled.

Easy inspection. The current state of the circuit must be easily visible, even for complex parts, like buses, memories and subcircuits. The importance of this goal is that seeing the evolving state of the circuit helps understand its workings or fix design problems. A good example of a simulator that does this well (albeit for analog circuits) is Paul Falstad's circuit simulator [7]. To this end, the state of wires, buses, inputs and outputs need to be presented using colors, the exact value on buses must be easily displayable (e.g. by mouse hover), and the subcircuit's state must be possible to see simultaneously with the main circuit.

Usability and portability. As the tool is to be used by undergraduate students with different degree of previous experience with complex software, installing and using the tool must be as simple as possible. It should also be platform independent, so that students can run the tool on their own computers. Currently the best way to achieve this goal is to use a solution based on Web technologies: JavaScript, HTML, CSS and SVG.

Possibility of batch operation. During a digital circuit design course, the student will design a number of circuits for assignments. It would be beneficient for both the students and the instructor if the students' designs were automatically tested for correctness. A model-view-controller (MVC) architecture allows to separate the tool's simulation engine from the graphical user interface, which enables fully automatic testing.

Acceptable performance. It would be beneficial if the tool could be used to simulate not only simple examples (like adders, latches, finite state machines, etc.), but also larger, realistic circuits. Students could then continue to use the tool as they tackle more advanced topics. If simple processors (CPUs) can be simulated, the tool could be used for teaching computer architecture.

3 RELATED WORK

There are numerous tools which simulate Verilog designs directly. One of the most known is ModelSim by Mentor Graphics. It is a fully-featured, industrial quality simulator, included with Intel Quartus and Xilinx Vivado design tools. It is a great tool for HDL developers, but using it for teaching newcomers to HDL and digital design has several downsides. First, its user interface is quirky and requires some practice to get used to it. Second, it uses the simulation semantics of Verilog, which, as said before, makes it easy for students to write non-synthesizable or ambiguous Verilog constructs.

Other such tool is EDA Playground by Doulos [4]. It is a Web application, which allows to quickly prototype HDL code, simulate it using one of the available backends (which include Synopsys VCS, Cadence Incisive, Aldec Riviera Pro and the free Icarus Verilog), and see the synthesized circuit diagram. It's user friendly and an useful prototyping tool, but it does not offer much help to the beginner to visualize the connection between the HDL code he is writing and the working of the resulting circuit.

On the other hand, there exist tools which simulate digital circuits drawn as diagrams. One of them, specifically created as an educational tool, is Logisim [2]. It is used for teaching students in many universities around the world [11]. One of its advantages for teaching is that it presents the workings of the simulated circuit graphically, helping the student understand it. It also allows to export the circuit to Verilog code. Unfortunately, it does not work with circuits synthesized from HDL code.

4 USING DIGITALJS

The easiest way to get started with DigitalJS is via the online demonstration app (Figure 2). It presents the user with a screen split into three parts. On the left is a code editor, with syntax highlighting for SystemVerilog, and a button which instructs the app to synthesize a circuit from the code. A set of simple example source codes is also provided to help beginners get started quickly. On the right is a visual, interactive circuit representation. The top panel is a toolbar, which in the current version allows to control simulation time, save and load the circuit schematic, and get a link for sharing.

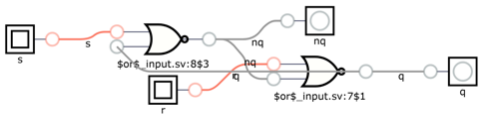


Figure 3: A SR latch directly after synthesis

After clicking the "Synthesize and simulate" button, if the code is valid, synthesizable SystemVerilog, the synthesized circuit is laid out automatically and presented on the right panel. Single bit inputs automatically receive a button widget driving them, similarly, single bit outputs are connected to a LED-like display. For buses, numeric input/output widgets are generated; these can be set to display their value in binary, octal, decimal and hexadecimal. You can immediately interact with the circuit; for example, for the SR latch circuit in Figure 2, you can click the S and R buttons and see the circuit react.

The simulation uses three-valued logic, and the circuit is initialized in an undefined state. Figure 3 presents the SR latch circuit as it looks like directly after synthesis. You can see that the latch state is initially undefined, and only after assigning a high value to one of the inputs the latch stabilizes into a defined state. Colors are used to display signal values: green for logical one, red for logical zero, gray for undefined. For buses, the same colors are used if every bit in the bus is high, low or undefined; otherwise they are colored blue. Bus wires are drawn with wider lines than single bit signals.

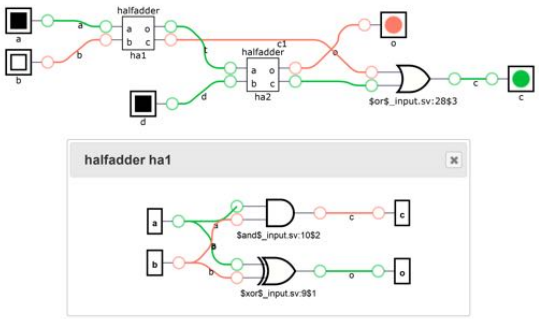
The displayed circuit can be easily modified. The layout can be changed by dragging the circuit parts, the paths the wires take can also be modified by adding and removing control points. The actual connections also can be changed; wires can be added or removed, and the simulation reacts in real time. If an input becomes undriven, it automatically assumes undefined state.

If the SystemVerilog code entered contains multiple modules, a module which does not reference other modules is selected automatically as the top level circuit. The other modules are treated as subcircuits, which are displayed as rectangular blocks. The contents of a subcircuit can be displayed by double-clicking a subcircuit block or hovering the mouse over it and then clicking a loupe icon which appears over it. The subcircuit is displayed in a window, and is as interactive as the top level circuit is. Figure 4 presents a full adder circuit, composed of two half adders.

Amongst the basic elements included in the simulator, other than logic gates, are: arithmetic operations, bit shifts, multiplexers, D latches and flip-flops, single and multi-port memories. Their use is inferred in the synthesis process as usual. Bus grouping and ungrouping appears explicitly as circuit elements; the tool forbids connecting inputs and outputs of differing bit width. Figure 5 shows

```
module halfadder(
  input a, b,
  output o, c
);
  assign o = a ^ b;
  assign c = a & b;
endmodule
module fulladder(
  input a, b, d,
  output o, c
);
  logic t, c1, c2;
  halfadder ha1(a, b, t, c1);
  halfadder ha2(t, d, o, c2);
  assign c = c1 | c2;
endmodule
```

(a) SystemVerilog source code



(b) Synthesized circuit graph

Figure 4: Full adder composed of half adders

an accumulating adder circuit, which uses some of the mentioned parts.

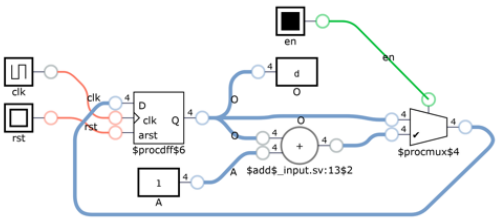
Combinatorial logic delays are simulated by default. This allows to simulate latches and flip-flops built from gates, like the one in Figure 2, and to visualize glitches occuring in combinatorial circuits due to the delays. The simulation time controls present in the top bar can be used to see how signals propagate through a circuit.

Signal waveforms can be displayed by hovering the mouse pointer over a wire and clicking a loupe icon. The waveforms are updated in real time, like in an oscilloscope. Old signal values can be displayed by dragging the waveforms, and the time scale can be changed using a mouse scroll wheel. The waveforms are colored in the same way as signal wires are to improve readability. Figure 5c presents signal waveforms for an accumulating adder circuit.

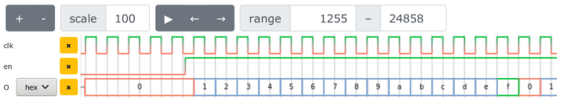
If the design to be simulated is composed of multiple files, they can be uploaded by using a file selection dialog window. Additional files, e.g. containing initial memory state for the circuit, can also be uploaded. Figure 1 presents a simple, but fully functional RISC V core being simulated.

```
module accumadder
#(parameter WIDTH = 4)(
  input clk, rst, en,
  input [WIDTH-1:0] A,
  output [WIDTH-1:0] O
);
  always_ff @(posedge clk or posedge rst)
    if (rst) O <= 0;
    else if (en) O <= O + A;
endmodule
```

(a) SystemVerilog source code



(b) Synthesized circuit graph



(c) Waveforms generated by the circuit

Figure 5: Accumulating adder circuit

5 IMPLEMENTATION

DigitalJS is implemented using the JavaScript programming language. This choice was made for two reasons. First, it allows to use the browser rendering engine for the user interface, which, in the author's opinion, is the best option in terms of usability, portability and flexibility. Second, the JavaScript community offers a large choice of useful libraries.

One of them is JointJS [3], which is used in DigitalJS as the graph rendering engine. Using this library allowed the author to avoid writing a lot of user interface code, and to focus on the key parts of the project. The JointJS library is designed well, with clear separation between the model and view layers. This separation enabled easy implementation of subcircuits, and also added a possibility for the simulator to run headless, which is useful for automatic testing of designs submitted by students, but also for regression testing of the tool itself.

The circuit diagrams are drawn using the Scalable Vector Graphics format (SVG), which is universally supported in modern Web browsers. This allows to easily accomplish the clean and readable look neccessary in educational software.

The open source project Yosys [18] was used in the backend for synthesizing circuit schematics from Verilog and SystemVerilog code. The circuit netlist can be extracted from Yosys in JSON format,

which is easily processed in Javascript. Because the Yosys JSON format is rather complex, the author decided not to use it directly as the input format for DigitalJS, but to have a simpler intermediate JSON format. In this way, processing Yosys output can be decoupled from the simulator itself. This also opens the way to interfacing DigitalJS with other software.

6 CLASSROOM EXPERIENCE

The tool is currently being used for an introductory course on digital circuit design in the Computer Science Institute, University of Wrocław. Thirty four students are enrolled, which includes 14 first year students, 12 second year students and 8 third year students. The curriculum is based on two textbooks [1, 12], both of which present digital design topics simultaneously with Verilog circuit descriptions. The classes include lectures, pen-and-paper exercises and lab assignments.

DigitalJS is used during lectures and for lab assignments. During lectures, it helps to illustrate concepts of digital circuit design with interactive examples. Having circuit state change in response to input, and having signal waveforms drawn in real-time, in author's opinion gives students better learning experience than just drawings and tables on a blackboard.

For lab assignments, student use DigitalJS on the department's computers or on their own. Thanks to the tool, they can instantly see what circuit their SystemVerilog code synthesizes to, and change their solution if the result is not as expected. The simulation allows to interactively test and debug the synthesized circuit.

The assignments are submitted online using the Web-CAT [6] automatic grading system. On the grader, DigitalJS is used for testing the correctness of the submitted designs. When the complexity of the assignment is small, solutions are tested exhaustively; if not, the checker uses randomly generated test cases. The solutions are also checked for meeting various constraints of the assignment; for example, names and types of inputs and outputs, circuit abstractions used (e.g. multiplexers, registers, memories...), critical path length. The student receives feedback within seconds of the submission, which includes, for example, which test cases have failed. The score received by the student is a sum of the correctness score and coding style score given by the instructor.

The students were given a voluntary, anonymous survey, where they could express their opinion on DigitalJS. Ten students (about 30 percent of students enrolled) responded. All ten of them have declared that their experience with the tool was positive. The specific responses to the questions in the survey, translated to English, are presented in Figure 6. If a similar response was given by two different students, only one of them is presented.

7 USER EXPERIENCE AND FEEDBACK

Other than in the classroom, the tool has potential to be used for fast prototyping of simple circuits for FPGAs, by both hobbyists and professionals. To assess its usability in this area, I have collected some feedback from FPGA users.

Cezary Siwek, who is one of my MSc students and who recently published a paper on using FPGAs for general game playing [15], said the following:

What is the most useful functionality of DigitalJS?

- Interactive representation of the circuit
- Waveform display for selected signals
- Signal visualization, in particular the possibility to display oscillograms
- A lot of interactivity
- It works :) Different number system display is useful
- Possibility of observing state changes step by step, displaying waveforms
- Clear, readable circuit visualization

What is missing in DigitalJS?

- Changing scale and possibly rotation of the circuit, so that it fits in the window
- Testing framework, for checking expected input/output pairs
- Better error messages
- Better presentation of error messages
- Clock signal configuration
- A document or tutorial describing its features

What bothers you the most in DigitalJS?

- Text formatting when copying/pasting from the in-browser editor
- Laconic error messages
- Working with inputs and outputs is troublesome in larger circuits
- Little control on the final layout of the circuit, in particular the positions of inputs and outputs

Figure 6: Student survey results

> Really impressive tool! It's the first time when I can quickly see how sequential logic synthesizes. It seems to be more efficient for quick prototyping of small modules than writing testbenches and looking on resulting waveforms.

An anonymous user identifying himself as *captain_wiggles_* on Reddit wrote in a comment to my announcement post on `/r/fpga` [13]:

> I really like how you can move stuff in the schematic around, including the nets. That's an awesome feature. [...] I like how the symbols for the components are descriptive too, and not just rectangles. [...] I'm really interested in automatically drawing schematics from the code, and then the ability to move stuff around and make it look neater manually. There's a lot of tools that auto generate the schematic, but I know of none that let me edit it like yours do.

He also pointed out some missing features, like the ability to undo changes in the schematic.

Al Williams, a writer for Hackaday, wrote in his article [17]:

> You don't usually think of simulating Verilog code – usually for an FPGA – as a visual process. You write a test script colloquially known as a test bench and run your simulation. You might get some printed information or you might get a graphical result by dumping a waveform, but you don't usually see the circuit. [...] This isn't a bad way to learn Verilog since you can

> quickly see what the code is doing. It isn't flexible enough to be a workhorse simulator [...] this is fun, though.

One of the commenters wrote: "This would be super cool integrated into IceStudio, the block based IDE for IceStorm." IceStudio [16] is a simple IDE for the Lattice Semiconductor iCE40 series FPGA chips. As IceStudio is also written in JavaScript, integrating DigitalJS and IceStudio is in the realm of possibility.

8 CONCLUSION AND FUTURE WORK

DigitalJS is currently in a proof-of-concept stage. There are many features missing, and their inclusion would improve the tool greatly. I believe the most important are:

- Defining triggers which pause the simulation under specific conditions. This feature is essential for debugging circuits.
- More functional schematic editor. Current editor does not allow to add new circuit elements manually, and there is no capability to undo changes. This would allow quick and dirty experiments with generated circuits.
- Better layout algorithm, especially for wires. Currently, DigitalJS uses an algorithm built into the JointJS library, which puts input/output blocks in unpredictable positions, often places wires on one another and creates unreadable layouts for complex circuits.
- Ability to simulate finite state machines as a circuit element, with states and transitions visualized. This would help students who struggle with understanding FSMs in digital circuits.

The tool is open source, published under a BSD license, and contributions are welcome.

REFERENCES

[1] Stephen Brown and Zvonko Vranesic. 2014. *Fundamentals of Digital Logic with Verilog Design* (3rd ed.). McGraw-Hill, New York.

[2] Carl Burch. 2002. Logisim: a graphical system for logic circuit design and simulation. *Journal on Educational Resources in Computing (JERIC)* 2 (March 2002), 5–16. Issue 1.

[3] client IO s.r.o. 2009. JointJS. Retrieved May 16, 2018 from https://www.jointjs.com/

[4] Doulos. 2013. EDA Playground. Retrieved May 16, 2019 from https://www.edaplayground.com/

[5] R James Duckworth. 2005. Embedded system design with FPGA using HDL (lessons learned and pitfalls to be avoided). In *Proceedings of the 2005 IEEE International Conference on Microelectronic Systems Education (MSE'05)*. IEEE, Anaheim, CA, USA.

[6] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. *SIGCSE Bull.* 40, 3 (June 2008), 328–328.

[7] Paul Falstad. 2005. Circuit Simulator Applet. Retrieved May 16, 2018 from https://www.falstad.com/circuit/

[8] IEEE 1076:2009 2009. *IEEE Standard VHDL Language Reference Manual.* Standard. Institute of Electrical and Electronics Engineers.

[9] IEEE 1364:2005 2005. *IEEE Standard for Verilog Hardware Description Language.* Standard. Institute of Electrical and Electronics Engineers.

[10] IEEE 1800:2017 2017. *IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language.* Standard. Institute of Electrical and Electronics Engineers.

[11] Vanja Lukovic, Radojka Krneta, Ana Vulovic, Christos Dimopoulos, Konstantinos Katzis, and Maria Meletiou-Mavrotheris. 2016. Using Logisim Educational Software in Learning Digital Circuits Design. In *Proceedings of the 3rd International Conference on Electrical, Electronic and Computing Engineering (IcETRAN 2016)*. ETRAN Society, Zlatibor, Serbia.

[12] Morris Mano and Michael Ciletti. 2017. *Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog* (6th ed.). Pearson, New Jersey.

[13] Marek Materzok. 2018. Tool for learning Verilog – feedback welcome! Retrieved May 16, 2018 from https://www.reddit.com/r/FPGA/comments/9c1f1n/tool_for_learning_verilog_feedback_welcome/

[14] Don Mills and Clifford E. Cummings. 1999. RTL Coding Styles That Yield Simulation and Synthesis Mismatches. In *Synopsys User Group 1999 Proceedings (SNUG 1999)*. Boston, MA, USA.

[15] Cezary Siwek, Jakub Kowalski, Chiara F. Sironi, and Mark H. M. Winands. 2018. Implementing Propositional Networks on FPGA. In *Proceedings of the 31st Australasian Joint Conference on Artificial Intelligence*. Springer, Wellington, New Zealand, 133–145.

[16] Jesús Arroyo Torrens. 2016. IceStudio Github page. Retrieved May 16, 2018 from https://github.com/FPGAwars/icestudio/

[17] Al Williams. 2018. Visualizing Verilog Simulation. Retrieved May 16, 2018 from https://hackaday.com/2018/09/03/visualizing-verilog-simulation/

[18] Clifford Wolf and Johann Glaser. 2013. Yosys – A Free Verilog Synthesis Suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip 2013)*. Linz, Austria.