Towards Stakeholder Specific Visualization of Learning Paths in Software Engineering Teaching

Henri Bomström
University of Oulu
Oulu, Finland
ORCID 0000-0002-7028-9044
Outi Sievi-Korte
Tampere University
Tampere, Finland
ORCID 0000-0002-4956-8989
Terhi Kilamo
Tampere University
Tampere, Finland
ORCID 0000-0002-9561-1116
Kari Systä
Tampere University
Tampere, Finland
ORCID 0000-0001-7371-0773
Elina Annanperä
University of Oulu
Oulu, Finland
ORCID 0000-0002-2615-9576
Markus Kelanti
University of Oulu
Oulu, Finland
ORCID 0000-0003-1886-8521
Kari Liukkunen
University of Oulu
Oulu, Finland
ORCID 0000-0002-0719-4712

ABSTRACT

In software engineering and other technology related teaching educators increasingly integrate de-facto online tools into coursework. However, the impact of using these tools is not clearly understood. To this end, this research project will provide a visual dashboard with extensive and stakeholder-specific visualizations to serve the diverse needs of different stakeholders, e.g., teachers, teaching assistants, administrative personnel and students.

This paper reports the results of our initial analysis of what kind of views teachers want to take to their courses and what kind of information teachers see as valuable visualizations on learners' progress. We conducted 17 semi-structured interviews in two universities. The interviews were thematically analysed, giving as results three key themes. The results give a good starting point to create a visual course dashboard. Our study takes a step towards supporting various stakeholders in learning environments through visual means. While the input data, metrics and visualizations are based on the tools used in software engineering courses, we see that several results can be applied to other contexts.

Conference Key areas: Essential elements for the online learning success.

Keywords: visualization, software engineering, software engineering education, online learning

1 INTRODUCTION

Online teaching tools have been utilized in teaching for decades and their potential is widely recognized [1]. Massive Open Online Courses (MOOC) have risen as a method to provide non-formal and informal learning to many [2]. At the same time, the importance of software engineering education has increased because of the constant growth in the need for software engineering professionals. In addition, professionals in many other fields today need basic programming skills. When the COVID-19 pandemic hit, formal teaching was also forced to take a major leap toward online teaching and learning.

The number of courses and the number of participants per course in software engineering higher education today are big. This leads to the teaching staff needing more tools to 1) support a growing number of students and 2) deal with students with heterogeneous backgrounds and motivation. Learning management systems (LMS) such as Moodle, TIM [3] and A+ LMS [4] are used to reduce the teachers' workload by the means of online course material and automation in grading [4, 5] and the students' learning is student-driven and independent of time and place. In addition, the content is categorized so that students who specialize in Information Technology can take more difficult exercises than students from other fields [6].

Already, LMSs combined with professional software engineering tools such as Continuous Integration (CI)[1] and version management tools give the teacher information of the outcome of the students work, but little about the progression of work before final submissions or the students' general feeling and motivation towards learning. Hence, there is a need for more information and several tools - teaching and programming - that can provide that information.

The goal of *VISDOM* project's teaching use case is to provide a visual dashboard that can meet the diverse needs of different stakeholders, e.g. teachers, teaching assistants, admin personnel and students - mainly in the context of online and hybrid learning since there the students are assumed to work more independently. To study educator needs to monitor learners' course progress and learning, we interviewed 17 programming and software engineering teachers in two universities. Our research question (RQ): What kind of visualizations software engineering teachers would like to have? is answered with thematic analysis of the interviews giving three key themes.

The rest of the paper is structures as follows: Section 2 gives the background of the work. Section 3 describes the planning and execution of the interviews and Section 4 gives the results of the initial analysis. Section 5 gives the discussion of the results and Section 6 concludes the paper.

2 Software Engineering Education

The general guidelines for software engineering undergraduate [7] and graduate [8] curriculum are published by Assosiation of Computing Machinery (ACM). These curriculum guidelines specify software engineering education content in terms of elements of skill and knowledge the students should learn, including the software development process. Similarly, the software engineering Body of Knowledge [9] is used when building university level software engineering curricula. While these plan out the thematic content, the teaching methods should support the students' learning as well.

In their systematic literature review Santos et al. [10] identified four approaches to set up innovative approaches in teaching programming, one being project-based teaching practices. They found that students are better motivated when collaborative learning and continuous monitoring are used. However, the study does not indicate any particular use of software development tools to support learning or how information about the learning process and student progress should be shown to teachers or students.

[1] a practice commonplace in software development where members in a software team integrate their work frequently. https://www.martinfowler.com/articles/continuousIntegration.html

A gap between industry and teaching [11] has been identified to exist, mainly because of the nature of the software engineering profession; software industry expands to new areas faster than the academia. This leads to problems in teaching, as student projects tend to lack realism, and courses on different topics may be isolated and the connections between courses are not visible enough to students. Software engineering education is also not considered to teach enough soft skills (collaboration and teamwork). Further gaps related to the required skills in industry have been identified in [12]. A large number of students makes it difficult for a teacher to focus on individual issues and provide needed tutoring. To improve learning results and increase the real-world competencies of the students, more comprehensive education setups have been created and used at universities around the world [12]. Such setups pursue to offer the students an opportunity to practice their software development skills in environments and tasks that are as close to typical industrial contexts as possible [13].

A survey on program visualizations in education has been conducted by Sorva et al. [14]. There is a variety of approaches and tools to visualize programming structures and the behavior of software, such as Jeliot [15], Javavis [16] and AnyviewC [17], that are used in programming courses. Visualizations have also been used in teaching project management. There the most common visualizations used are Gantt charts showing the project activities such as those presented by Vanhoucke et al. [18]. Other works have used Gantt charts alongside other visualizations. Deblaere et al. [19] have create RESCON – an educational tool for illustrating scheduling and project management concepts, where they have included project duration curves and resource profiles in addition to Gantt charts. Salas-Morera et al. [20], who have created PpcProject to teach project management, also include visualizations on resource allocations in addition to Gantt charts. Scheduling and resource allocation are also key elements in a simulator developed by Collofello [21], where schedule pressure is visualized with a speedometer.

Matthies et al. [22] used ScrumLint, which enables comparing teams and there progress, to check for process violations on a software project course using Scrum. They only used the tool post-hoc as a way to supplement their survey and tutor-based evaluations of students' performance, but discuss how the tool could be helpful already during the course in showing teams how they are performing. Mäkiaho et al. [23] have developed the MMT tool for teachers to monitor progress in students' capstone projects through a visual dashboard based on manually entered data.

To the best of our knowledge, there are no studies where visualizations would have been used in the context of teaching software engineering processes or project management using real life data from various tools and combining that data with informative visualizations. Furthermore, existing studies have mainly used visualizations to illustrate one particular aspect, such as the duration of the projects or the overall progress. In this research we aim at visualizations that are created from real software development work of students, using multiple data sources and providing useful information to several stakeholders.

3 RESEARCH METHODOLOGY

This work aims to identify, what kind of visualizations teachers would like to have from online and hybrid course work. The study was conducted as a single case study [24] to gain insight on educator needs in higher education. The method was selected as case study research [24, 25] is suitable to study a topic tied to and not clearly separable from its practical context. This is true for higher education learning where teachers have practical experience on the everyday learning in online and hybrid learning.

The case study was executed by interviewing teachers from two universities. The interview questions were iteratively designed by the authors in late May, early June 2019. The authors have extensive experience in software engineering higher education. The goals of the research project as well as knowledge on the state of the art of software development tools was used in designing the interview protocol (interview protocol is online[2]). It's worth to note that the protocol was designed before

[2]The interview protocol:
https://docs.google.com/document/d/1LtRR4GBa_12ZfMDjcv0JotYQrnRSaYM30jfHaYyJXp0/edit?usp=sharing

the COVID-19 pandemic. The interviewees were identified from the universities based on availability, experience and the topic of the courses taught. The courses ranged both undergraduate and graduate level teaching.

3.1 Data Collection

Data was collected through 17 semi-structured interviews of teachers in two universities. The interviews followed the interview protocol and lasted approximately one hour. Two researchers, one conducting the interview and the other taking notes, were present in each of the interviews. Open ended questions were used to ask further questions based on interviewee's answers. The interviews were recorded with consent and the tapes transcribed with professional transcription services. However, one interview is included in the analysis only based on the notes as the interviewee states not wishing the interview to be recorded. From a total of 17 face-to-face interviews, 11 were conducted in October 2019, whereas 5 interviews were conducted between June 2019 and March 2020. The last interview was conducted over Zoom[3] in March 2021 as a complementary interview due to the changed needs for the global pandemic. The interviewee demographics can be found online[4].

3.2 Data Analysis

Thematic analysis [26, 27] was used to analyze the collected data. Data-driven coding [28] was utilized instead of any predefined codes to ensure the interviewee's experiences were represented accurately. The initial thematic analysis was developed by one researcher who went through each interview and coded meaningful segments. A second researcher participated in discussions regarding the codebook's structure to achieve consensus on coding practices. The codebook's validity was evaluated in three rounds by having two other researchers fit a selection of citations to the correct codes. In the first iteration the codebook and 26 citations, two randomly selected for each theme, were assigned to evaluators. The initial codebook was restructured after the first evaluation round based on evalutors performance and feedback. The second evaluation round was conducted with the same logic by providing the restructured codebook with two randomly selected quotes for each theme, overall 18 citations. The second evaluation round resulted in an interrater reliability (IRR) of 0.43, signifying a moderate agreement [29], which was calculated using Fleiss' Kappa [30, 31] as it allows IRR to be calculated for three or more raters. After the second round, the coder and evaluators had a session where coding related disagreements were discussed and resolved. A third round of evaluation was performed with 12 citations, again two randomly selected citations per theme, once the themes were agreed upon by the three researchers. In the third round, the coder and two evaluators discussed their disagreements and the round resulted in an IRR of 0.66, signifying a substantial agreement. The final themes are described in the following section.

4 RESULTS

Six themes were identified with thematic analysis (Figure 1). Each theme had two to six subthemes, denoting different aspects of each theme. However, several themes were not directly related to teacher needs or visualization. Our aim is to work towards extracting a number of metrics for visualizing student progress within software engineering teaching. Thus, we take a data-oriented approach on our results by focusing on three main themes as they best address teacher needs from a visualization viewpoint. The remaining three themes (social aspects and communication in teaching, teaching resourcing, and tooling and the use of data) left out of the scope of this study relate to teaching experiences with communication, resourcing, and the use of tools. The rest of this section describes the selected themes and the visualization opportunities and metrics discussed by the interviewees.

[3] https://zoom.us/

[4] The interviewee demographics:
https://docs.google.com/spreadsheets/d/1Y43zwlLPSSjzoS9S3TZByh6q0pVZWFpEDekgVTgQET8/edit?usp=sharing

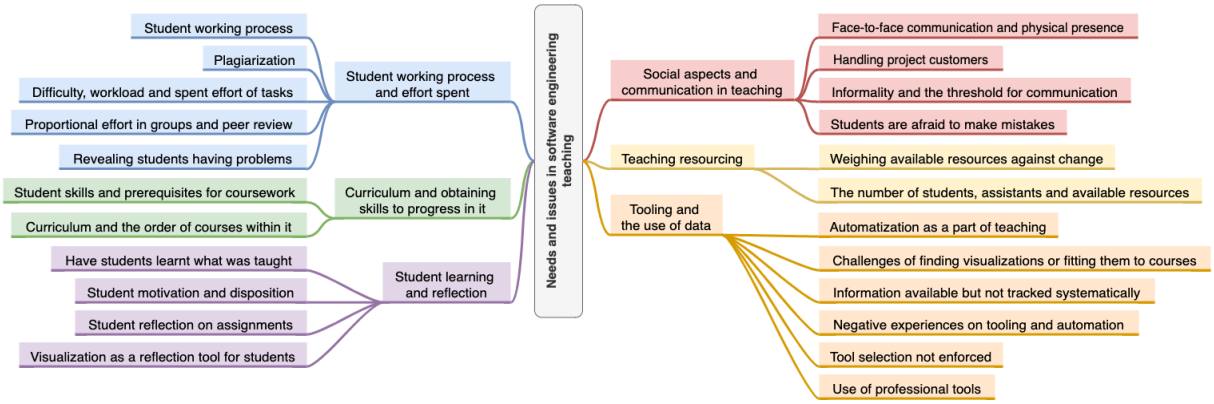

Figure 1: The codebook resulting from the thematic analysis of software engineering teacher interviews. The six boxes originating from the center represent the identified themes, whereas the outermost items originating from themes represent individual codes.

4.1 Theme one: Student Working Process and Effort Spent

The results suggest that software engineering teachers lacked a higher-level view of student or group progress and placed significant interest in tracking the students' working process – what is happening and what is not happening. Several interviewees had come up with different strategies to try to have students work in a continuous manner. In the first stages of a course, teachers were also interested whether work had even begun for groups or individual students. Later on, the interviewees sought statistics and trend-related information about participation and assignment-related points collected by students. Programming related courses often utilized the number and timing of commits in version control system (VCS) as an easy indicator whether actual work was being done. Project-based courses also tracked the ratio of not started, currently working on and done tasks. Several interviewees described they would like to know how points were distributed between students, how student's points accumulated overall during the course, how the points were distributed per week, and how many students were projected to pass the course at given time. Tracking presence in lectures and exercises and the number of assignment submissions also served the interviewees in judging how many students were actively participating in the course.

The findings show that one of the main reasons for tracking student working process and activity was to identify the students, groups or projects having problems or whether some students had stopped working and disappeared from the course. Several interviewees mentioned that they would like to have a tool that allowed them to proactively contact students having problems or those otherwise falling behind. It was also considered important to discover what were the problems students were having. The interviewees felt that such a tool could help them focus their resources in helping those who need it by resolving student issues before they silently dropped out of the course.

The results indicate a clear need to understand how much time students used for each task and that the effort was reasonable. Concrete work-related metrics such as how many students completed specific tasks overall and how much time they used for each task were often used to determine how much effort assignments required from students. Additionally, the time spent on tasks was considered as an indicator of how difficult the tasks were for students. However, it was noted that students tend to report hours in a sporadic manner.

> "I cannot see their working hours. Sometimes they share the Google document, but no. We see them, they deliver it every week, when they collected all the things. Because not every student is recording the hours every day. Before the actual presentation they try to fill up those things. And for me as a teacher, it will be a nightmare to follow all the working hours every day for them."

The results further suggest that the interviewees wished to prevent freeloading in group assignments by evaluating the distribution of work within groups. While programming courses could use commit

information from VCS, courses without programming assignments often featured peer review as a method of ensuring fair evaluation for all students. When students were assigned to communicate with each other, the interviewees described that they would like to see statistics about communication activity. These statistics included whether students post their answers on time, the number of comments to peers, how fast comments were made in response and the imbalance of communication between groups. Finally, several interviewees hoped for a way of automatically detecting plagiarization, as currently providing proof of plagiarization was difficult and often a manual process.

4.2 Theme two: Curriculum and obtaining skills to progress in it

The results suggest a need for a higher view of teaching on a degree program level. Teachers felt that collective yearly reports did not provide early or accurate enough information to intervene before students get into problems. Higher level visualizations would allow teachers to be able to gain continuous information, see how students are progressing, and even contact students pre-emptively in case of problems. One interviewee proposed an idea of visualizing degree programs as stories, where students would understand what a software professional would do in different roles, and how courses related to each others as a story towards being a professional developer. Another interviewee proposed a similar solution where the whole degree program would be visualized.

> "But we should get that kind of graph, that here is the, let's say study module, take its course, and show what needs to be in front of it, form just like this chain, that we don't have. But all of the information is there. This could be a great tool for students so that they, we'd show a kind of tree, that here are your courses, and as you complete them the color changes. Here's how you progress. And in this spot comes the bachelor's thesis and like that."

Furthermore, the results suggest that visualizing thesis work as a part of higher-level visualizations would benefit various stakeholders such as teachers, students and university staff. A few interviewees suggested a similar tooling solution that would allow students to report their individual thesis progress on a template, allowing teachers to structure thesis work for students, track their individual progress, see whether students actually worked on their thesis, and increase transparency on estimating realistic graduation dates. Besides reporting progress, students would use said tool to reflect the bigger picture of thesis work and how much work is done and still needs to be done.

The results further suggest that students may lack the necessary skills or knowledge when enrolling to more advanced courses. Interviewees often expected students to have an understanding of topics introduced in prior courses as they were necessary for performing well in their courses. Lacking the required prerequisites is challenging for students as well, as interviewees noted that students without necessary skills tend to silently drop out of their courses without telling anyone.

4.3 Theme three: Student learning and reflection

The results indicate a need to see whether students had learnt what was taught. Correct execution of tasks was seen as one of the simplest indicators, such as whether students were able to complete weekly assignments. For project based courses, a few interviewees discussed that correctly performing the process itself was an indication of learning.

> "That we have different step of the different processes, and then they are able to implement the different step of the different processes. So if we go to requirement elicitation, I just check if we are able to write the user stories correctly, (...) to write them in the correct format, and then if they are able to split them and not to make (-) too big user stories."

Other interviewees discussed essays and learning diaries as good ways to gauge learning. However, some remarks were made that not everyone is capable of expressing themselves in written assignments, making discussions and face-to-face sessions a good way to find out how well students had understood what was taught. Overall, interviewees considered automatic analysis of often free-form qualitative assignments challenging. Both concrete and abstract metrics such as number of sections, length of

text and indication of a reflective thought process were proposed to aid in analysis of text-based assignments.

The results further indicate that visualization from a student viewpoint could serve as a reflection tool for students. Examples of this include providing assignment specific visualizations or a visualization where students could benchmark their own progress against others in the same course. Additionally, some interviewees were interested in learning the general disposition of students towards the course. Concrete examples were provided as collecting feedback, checking whether students completed optional assignments and whether they thought the assignments were useful.

5 DISCUSSION

5.1 Visualization opportunities

Interviewees described possible visualization needs and opportunities in two different fashions: 1) detailed descriptions of what kind of data they had or would want to have available to them, and what kind of information they would like to see based on that data, and 2) visionary visualizations to help understand complex and multi-variate concepts such as progression in degree program. We will primarily take a data-oriented focus, in order to extract a number of metrics and envision visualizations based on them.

Tracking time spent on tasks helps understanding required effort to complete tasks, which in turn may help understanding difficulty level of tasks. Understanding the needed effort and difficulty of tasks could also be used to determine whether the required workload from students was reasonable in practice. Visualizing actualized effort would be beneficial both for the teachers as well as for students. Interviewees mentioned that some students had difficulties in understanding and estimating the effort required to complete courses and were sometimes surprised of the required effort.

Tracking succession rate may also benefit understanding how difficult tasks are. Particularly with programming assignments, students may commit several times before their code passes the tests. If the number of commits for particular tasks is significantly higher than on average, it may indicate a more difficult task or that the teacher should address a certain topic more in depth. Similarly, if significantly fewer students submitted answers to a particular task, it could be considered more difficult than others.

Level of participation may be an indicator of commitment towards finishing the course. Some interviewees already mentioned having explored participation and point related data statistically with varying results. Examples of the relationships explored include steady assignment completion, participation in consecutive exercises and the relationship between points and exercise projects and exam. Overall, possible metrics are distribution of points between students, cumulative points and distribution of points over time, and the completion of optional assignments.

Tracking progress and trends would allow teachers to get an overall view of how many students and groups have difficulties keeping up from early on, and also get a realistic idea of the pace with which students advance during the course. Teachers could also benefit from visualizations of student peak activity periods to determine whether students work systematically or in spurts near the end of deadlines. Some interviewees also mentioned that students could be motivated by providing them visualizations that would allow them to compare their progress with others in the same course. Both teachers and students could also benefit from a projection of passing the course with the current trend.

The presented results set the stage for developing software that visualises student progress in software engineering, forming a base for prototype implementations and further research within the project. Combined, metrics for tracking time, success rate, progress and participation could be presented using visualizations. These visualizations could be implemented with a tool that would allow teachers to investigate the visualizations, learn of students in trouble (clear deviations in the visualizations), and be able to proactively contact such students. Lastly, several different viewpoints for framing and interpreting data were mentioned in the interviews such as student, group, course and between

courses. This means that each of the presented metrics might look slightly different when viewed from varying perspectives. The interviewees also mentioned teaching assistants, teachers themselves, and staff such as study advisors as potential users (stakeholders) of such metrics and visualizations.

5.2 Threats to Validity

We will consider threats to validity as described by Wohlin [32] and cover the points which are relevant to our study. Conclusion validity concerns the correctness of conclusions drawn, and internal validity concerns threats that may affect the variables with respect to causality. For both the key issues are elements in data collection and analysis. We selected interviewees based on their expertise, availability, and relevance. We had no prior knowledge as to how they would consider the questions or what their attitude would be towards the topic. The interviewees did not volunteer directly (which might give falsely positive results), but they did all individually agree to partake in the study. The same interview protocol was followed for all interviewees. The only difference was that one interview was conducted via Zoom, with an additional question on distance learning.

We also need to consider the threats posed by having the themes validated by authors only. Here the majority of interviews were performed by other authors than the one creating the codebook. Further, one of the evaluators of the codebook was not part of performing the interviews. The validating authors were given the quotes and themes separately and independently, and no indication was given as to how the first author had done the initial mapping.

Construct validity concerns how well the results are generalizable to the concept or theory behind the experiment, and external validity regards generalizability to industrial practice. Our results are not meant to be generalizable over the whole software engineering teaching related field but indicate starting points for further research in visualizing useful information and data for teachers of software engineering. Finally, there are elements common to software projects within university setting and projects in the industry. Having visualizations applicable for multiple stakeholders and environments is also a goal of the research project. However, for the purpose of this study, we are only aiming at results for teaching.

6 CONCLUSIONS

This paper provides a report of our initial analysis on what kind of views software engineering teachers want for their courses and what kind of information they value in following learners' progress. We found that software engineering teachers lack higher-level views for tracking student progress in both course and degree program level. Additionally, our results indicate that software engineering teachers lack tools for identifying and contacting students that are lagging behind or having problems completing assignments. Thus, it is important to create initial metrics that can help teachers towards understanding the required effort and difficulty of tasks, sensibility of student workload and the pace of work. This creates a starting point for further research. Based on the results presented, we are currently building the first prototype implementations of a system that can visualize student progress.

7 ACKNOWLEDGEMENT

The authors thank the teachers for the time and expertise they provided for the interviews. The work was supported by the ITEA3 project VISDOM. The authors thank Business Finland for the funding.

References

[1] Jared Keengwe and Terry T Kidd. Towards best practices in online learning and teaching in higher education. *MERLOT Journal of Online Learning and Teaching*, 6(2):533–541, 2010.

[2] Tharindu Rekha Liyanagunawardena, Andrew Alexandar Adams, and Shirley Ann Williams. Moocs: A systematic study of the published literature 2008-2012. *International Review of Research in Open and Distributed Learning*, 14(3):202–227, 2013.

[3] Ville Isomöttönen, Antti-Jussi Lakanen, and Vesa Lappalainen. Less is more! preliminary evaluation of multi-functional document-based online learning environment. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE, 2019.

[4] Ville Karavirta, Petri Ihantola, and Teemu Koskinen. Service-oriented approach to improve interoperability of e-learning systems. In *2013 IEEE 13th International Conference on Advanced Learning Technologies*, pages 341–345. IEEE, 2013.

[5] Kirsti Ala-Mutka and H-M Jarvinen. Assessment process for programming assignments. In *IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings.*, pages 181–185. IEEE, 2004.

[6] Petri Ihantola, Essi Isohanni, Pietari Heino, and Tommi Mikkonen. Criterion-based grading, agile goal setting, and course (un)completion strategies. In *Agile and Lean Concepts for Teaching and Learning*, pages 207–230. Springer, 2019.

[7] S.C. Santos, P.A. Tedesco, M. Borba, and M. Brito. Curriculum guidelines for undergraduate degree programs in software engineering. In *Software Engineering 2014*, pages 205–214, 2015.

[8] S.C. Santos, P.A. Tedesco, M. Borba, and M. Brito. Curriculum guidelines for graduate degree programs in software engineering. In *Graduate Software Engineering 2009(GSwE2009)*, pages 205–214, 2009.

[9] Pierre Bourque and Richard E. Fairley, editors. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition, 2014. ISBN 978-0-7695-5166-1. URL `http://www.swebok.org/`.

[10] S.C. Santos, P.A. Tedesco, M. Borba, and M. Brito. Innovative approaches in teaching programming: A systematic literature review. In *CSEDU 2020 - Proc. 12th Int. Conf. Comput. Support. Educ. 1*, pages 205–214, 2020.

[11] D. Oguz and K. Oguz. Perspectives on the gap between the software industry and the software engineering education. In *Graduate Software Engineering 2009(GSwE2009)*, pages 117527–117543. IEEE Access, 2019.

[12] T. Nurkkala and S Brandle. Software studio: Teaching professional software engineering. In *Agile Processes in Software Engineering and Extreme Programming*, pages 126–142. XP 2014, 2014.

[13] X. Wang, I. Lunesu, J. Rikkila, M. Matta, and P Abrahamsson. Self-organized learning in software factory: Experiences and lessons learned. In *Agile Processes in Software Engineering and Extreme Programming*, pages 126–142. XP 2014, 2014.

[14] Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *ACM Trans. Comput. Educ.*, 13(4), November 2013. doi: $10.1145/2490822$. URL `https://doi.org/10.1145/2490822`.

[15] Sanja Čisar, Robert Pinter, A. Radosav, and Petar Čisar. Effectiveness of program visualization in learning java: a case study with jeliot 3. *International Journal of Computers, Communications and Control*, 6:669–682, 12 2011. doi: $10.15837/\text{ijccc}.2011.4.2094$.

[16] Rainer Oechsle and Thomas Schmitt. Javavis: Automatic program visualization with object and sequence diagrams using the java debug interface (jdi). In *Software Visualization*, pages 672–675. Springer Berlin Heidelberg, 2002. doi: $10.1007/3\text{-}540\text{-}45875\text{-}1\_14$. URL `http://dx.doi.org/10.1007/3-540-45875-1_14`.

[17] Weimin Wu, Yongfeng Cao, Baocai Chen, Qing Su, and Kailun Li. Anyviewc: A visual practice platform for data structures course. In Mark Burgin, Masud H. Chowdhury, Chan H. Ham, Simone A. Ludwig, Weilian Su, and Sumanth Yenduri, editors, *CSIE 2009, 2009 WRI World Congress on Computer Science and Information Engineering, March 31 - April 2, 2009, Los*

Angeles, California, USA, 7 Volumes, pages 493–497. IEEE Computer Society, 2009. doi: 10.1109/CSIE.2009.212. URL https://doi.org/10.1109/CSIE.2009.212.

[18] M. Vanhoucke, A. Vereecke, and P. Gemmel. The project scheduling game (psg): Simulating time/cost trade-offs in projects. *Project Management Journal*, 36:51 – 59, 2005.

[19] Filip Deblaere, Erik Demeulemeester, and Willy Herroelen. Rescon: Educational project scheduling software. *Computer Applications in Engineering Education*, 19:327 – 336, 06 2011. doi: 10.1002/cae.20314.

[20] Lorenzo Salas-Morera, A. Arauzo-Azofra, L. García-Hernández, Juan M. Palomo-Romero, and C. Hervás-Martínez. Ppcproject: An educational tool for software project management. *Comput. Educ.*, 69:181–188, 2013.

[21] J. S. Collofello. University/industry collaboration in developing a simulation-based software project management training course. *IEEE Trans. on Educ.*, 43(4):389–393, November 2000. ISSN 0018-9359. doi: 10.1109/13.883347. URL https://doi.org/10.1109/13.883347.

[22] Christoph Matthies, Thomas Kowark, Keven Richly, Matthias Uflacker, and Hasso Plattner. How surveys, tutors, and software help to assess scrum adoption in a classroom software engineering project. *CoRR*, abs/1809.00650, 2018. URL http://arxiv.org/abs/1809.00650.

[23] Pekka Mäkiaho, Katriina Vartiainen, and Timo Poranen. Mmt: A tool for observing metrics in software projects. *International Journal of Human Capital and Information Technology Professionals*, 8:27–37, 10 2017. doi: 10.4018/IJHCITP.2017100103.

[24] Robert K Yin. *Case study research and applications: Design and methods*. Sage publications, 2017.

[25] Lorna Hamilton and Connie Corbett-Whittier. *Using case study in education research*. Sage, 2012.

[26] Helen Gavin. Thematic analysis. *Understanding research methods and statistics in psychology*, pages 273–282, 2008.

[27] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 2006.

[28] Juliet M Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13(1):3–21, 1990.

[29] Anthony J Viera, Joanne M Garrett, et al. Understanding interobserver agreement: the kappa statistic. *Fam med*, 37(5):360–363, 2005.

[30] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[31] Mary L McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer-Verlag, Berlin-Heidelberg, Germany, 2012.