Software Engineering Expectations

An Explorative Case Study of Three University Courses

Jussi Kasurinen
LUT University
Lappeenranta, Finland
jussi.kasurinen@lut.fi

ABSTRACT

Software engineering as a discipline covers a large spectrum of concepts, from the very detailed technical aspects of system programming, to the high-end abstract concepts of the organizational software processes. Fundamentally, this also indicates that the software engineering researchers can focus their work on any of the several sub-disciplines, but also imposes an identity problem; how do the Master's and Bachelor's level students learn the fundamentals of software engineering, and how should they be taught these aspects? In this study, we observe three implementations of courses focusing on software engineering methods, and identify different approaches and recommendations on what works, and how software engineering courses should be constructed to promote student motivation and interest towards the discipline. Based on our observations, the fundamentals should focus more on the small-scale practical applications of the software engineering tools and practices, and promote real-life-applicability, since the students expect applicable experience, and software-industry relevance.

CCS CONCEPTS

• **Social and professional topics → Software engineering education**; Model curricula;

KEYWORDS

Software Engineering, Course content, student expectations, case study

ACM Reference Format:
Jussi Kasurinen. 2018. Software Engineering Expectations: An Explorative Case Study of Three University Courses. In *Proceedings of The 2018 Workshop on PhD Software Engineering Education: Challenges, Trends, and Programs (SWEPHD2018).* ACM, New York, NY, USA, Article 4, 6 pages.

1 INTRODUCTION

Software engineering as a discipline covers a large spectrum of different concepts, from the very detailed technicalities of system programming, to the highly abstract concepts of software organizations and their process models [20]. In this sense, this enables the software engineering researchers a freedom to focus on many different types of aspects within their discipline, but on the other hand it also promotes an identity problem since the discipline might not be very clearly defined for the university students [5].

The ACM curricula for software engineering [6] recommends that the discipline is approached by first introducing the different software process models, methods and tools which are commonly applied in the software engineering world. Even if there are studies which question the value of these tools and process models in the actual software industry [9], they provide the fundamental models and base foundation, from which other concepts and modern software engineering approaches are derived or developed.

In this paper, we conduct a qualitative case study on three courses following the general guidelines and recommendations of the ACM 2004 curricula for software engineering, and based on their feedback, assess the student expectations and general needs of the software engineering courses in the university-level education. First one of the case study courses was aimed towards Master's level students and heavily applied distance learning approaches, focusing on the software engineering methods and their application via software engineering design tools such as the UML2 (for example [19]). The second course focused on the fundamental process models and concepts of the software engineering, and was aimed towards Bachelor's level students, organized with more traditional lectures and weekly exercise events. The third course was fully long-distance-enabled course in software testing and quality assurance, intended for Bachelor's level students, where the student was required to only take the final exam as an local event.

The results indicate that there are concerns on how the students perceive software engineering as a discipline, and recommendations on what actions should be taken to enable better learning outcomes, and better motivation towards the software engineering as an area of academic interest. Overall, our research questions are *How should the fundamental software engineering courses be taught to enable more interest towards the subject?* and *What are the current areas of interest for the student body in software engineering?*.

This paper is also a continuance study which follows our software engineering research group's prior case studies on software engineering and university-level education, for example in the topics of application of video-based course lectures [11], how organizations adopt new practices of methods [15], how international standards can be applied in the self-assessment processes [14], or how computer science course curricula applies in the games industry context [13].

Rest of the paper is constructed as follows: In the Section 2 we discuss the recent research studies with the similar topics and concepts, while in Section 3 the applied research approach is described. Section 4 presents the case study results, while Section 5

SWEPHD2018, September 17th, 2018, St. Petersburg, Russia

provides discussion, observations and implications of these results. The Section 6 closes the paper with the conclusions.

2 RELATED RESEARCH

The software engineering discipline covers several different areas of interest, which makes the development of the course curricula difficult. To address this issue of expectations vs. the course contents, there are some studies which address the preferred methods of approaching the software engineering course contents, and two model curricula which include recommendations for the taught topics.

The most comprehensive model for software engineering curriculum is the ACM SE curriculum, first published in 2004 and supplemented with additions on the 2014 edition [6]. This model defines ten major categories for the software engineering from computing essentials and computer science, to the software design, human-computer interaction to the software quality, security and professional practices. Overall, the curriculum has definite emphasis on the design aspects of the software development, but shares several features with other computing-related disciplines. in fact, the ACM SE 2014 identifies that the capstone projects can be focused also into the computer science aspects, to emphasize the programming expertise if programming proficiency is expected in the latter doctoral or master's level programs.

The other more common model for software engineering is the IEEE Software Engineering Book of Knowledge (SWEBOK) with the version 3 being the most recent model available [2]. The SWEBOK model divides the software engineering discipline into 15 knowledge areas, such as Software Testing, Management or Professional practices. The model includes topics and definitions of "commonly accepted best practices and methods" by peer review process, and is constantly maintained with updates. However, the model does not include a separate curriculum, nor does it recommend a degree structure to cover the presented content.

In the assessment of these models, Alarifi et al. [4] have defined a method for assessing the actual development needs for the software engineering courses, which follow the general guidelines such as SWEBOK v3 or the ACM SE2004. Their study observes that the courses which follow the principles of the more recent model curriculas tend to perform better based on the course feedback, and identify five key areas which are critical success factors: 1) application of the latest SE tools and techniques, 2) Coverage of the key concepts identified in the SWEBOK model, 3) involving the actual software industry and practical assignments in the course, 4) application of software engineering standards and finally, 5) following accredited best practices such as NCAAA-accreditation in the course contents. Interestingly, the intrinsic student motivation was not found to be one of the critical factors, similarly as for example observed by [18] with the fundamentals of programming and LEGO robots; the students who are interested in the subject might even consider some forms of 'motivational activities' to have a negative impact.

On the flip side of the coin, Kuang and Han [16] discuss the common reasons why software engineering courses fail. Their study indicates, that the more common reasons are problems such as that the taught methods and course content in general is outdated, the given exercises do not prepare the students to work in the software industry, and that the faculty members simply lack relevant industry experience to provide real-world-equivalent learning experiences. Similar to Alarifi et al., Kaung and Han promote the need to constantly maintain and revise the course contents to match the industry expectations and requirements to provide a realistic view and realistic expectations for the students who are considering software engineering as their main profession. This expectations and capability gap has been also identified by Almi et al. [5] in software engineering, and for example also by McGill [17] in the software engineering -related domain, entertainment software engineering, games industry. In this sense, it can be argued that the expectation gap is more universal problem, but in any case it is present in the software engineering discipline.

3 RESEARCH METHOD

Assessing the feasibility and student motivation towards university courses is not straightforward, since there might be several underlying phenomena which might affect the outcomes. To compensate for these issues, we applied an explanatory case study approach following the general principles presented by Eisenhardt [8], applying open coding as defined by Corbin and Strauss [7] in the available qualitative data. As the objective was to assess the feasibility of the case course implementations and to identify possibly problematic areas, we found that this exploratory approach was sufficient to provide the data needed, and to allow us to compare our observations against the prior research results discussed in the previous section.

To study the student expectations and the course contents offered in Software Engineering courses, three implementations of the fundamental software engineering courses delivered by one faculty in one university were selected for in-depth case study. The first course which was selected for the analysis was Bachelor's level course *Fundamentals of software engineering* (Course A), arranged in the academic year of 2017-2018, the second course Master's level course *Software Engineering Methods* (Course B), from the academic year of 2013-2014 and the third one *Fundamentals of Software Testing* (Course C), arranged in 2014-2015. The course contents included in all courses included topics such as the common software process models, commonly applied software standards, software design tools -especially UML2-based-, software development environments and an overview of the common software product architectures, with boundary-control-entity-based (BCE) approach on the system module modeling. In all courses, the students were not expected to write algorithms or solve programming-based problems, but were expected to understand source code, and be able to create software engineering-related documents and designs based on the given materials and project repositories.

Combined, these three courses had over two hundred students, representing the different study majors from computer science, to industrial engineering and management, and electrical engineering from one university. The Course A was arranged as a traditional university course, with local weekly lectures and exercise events, whereas the Courses B and C were arranged as a long-distance-enabled courses applying course recordings and a separate tutorial video archives. All courses had exercise events which affected the course grade, plus at least one larger project involving teamwork and a mandatory separate exam. These courses and their components are defined in more detail in the Table 1.

From these three case courses, the data analysis on the expectations and the student considerations towards software engineering was collected via course feedback, course activity, submitted works, student-teacher communication and from the completion statistics. The open feedback and other eligible components were analyzed following the principles of open coding from the Straussian grounded theory [7] to identify and classify the common themes, concepts and problematic areas of these courses. The amount of material did not warrant a full GT analysis, used to pinpoint the topics of interest and to allow us to compare the results against the statistical data, and known issues identified from the related research. In this paper, we report only the issues which were identified both from the qualitative feedback, and from the quantitative sources.

4 RESULTS

The case courses were organized three years apart, and applied two different approaches on the course organization. The Course A was more or less traditional university course with the focus on the face-to-face teaching events with the course contents following a textbook, whereas the Courses B and C were organized as an long-distance-enabled course promoting self-study availability to select your own working hours. In terms of traditional course performance metrics, all of these courses performed very similarly; most of the students who actually started the course -submitted at least one graded exercise or project - completed all of the mandatory tasks, and the course average in all three cases was similarly around the midpoint of the grading scale. Similarly, the feedback indicated that the students felt that the course time consumption and overall difficulty was at an acceptable level. Interestingly, even though the courses were similarly laborious and difficult, the "traditionally organized" Course A had over 1 full grade worse student-given grade (3.0 vs. 4.3/4.1), and from the observed cases, it also was the only one where the amount of negative feedback superseded the positive.

Besides student workload and difficulty, the students also performed on these courses at an acceptable level. The courses B and C had over 75 percent pass rates when considering the students who actually made an effort to participate and pass the course. Similarly, the Course A had over 80 percent of the students eligible for the final grade, provided that they pass the final exam at some point in the future. The general course statistics and the amount of collected feedback is summarized in the Table 2.

In the feedback for the courses, the students were requested to list the topics they were expecting to learn from the course, and how the course contents matched with the student expectations. For example, in the Course C, the students were expecting to learn real-life applicable experiences (59% of the feedback), Management and design of software quality practices (50%), actual testing work techniques (45%) and testing exercises with a real software project and source code (36%). Similarly, the students in Course B ranked the actual exercise events more important than the lectures (grade 3.95 vs. 3.86 on scale 1-5, 5 best), the demo cases more important than the lectures (4.45 vs. 3.86) and both of the project works more important than the lectures (4.45 and 4.23 vs. 3.86). The most important difference between the separately graded course components was between the video lectures and tutorial archive versus the traditional lectures; the video-based approach was considerably more favored than the face-to-face lecturing (4.68 vs. 3.86). Similar observations were also made in both courses A and C; for example in the Course A, the summary video was watched by 35 percent of the students before the first exam, whereas in the last two lectures only less than ten percent of the student population attended.

In all three feedback surveys the students were also asked comments on how they felt concerning the course arrangements and the taught topics. To generally understand how students perceive software engineering, the feedback were analyzed and coded for observations as one dataset, where the differences between the course topics, and feedback concerning some course components such as the exercise submission systems, were abstracted or ignored.

In general level, there were more students which were interested in the software engineering as a topic, than those who found the topic to be boring or uninteresting (19% vs. 7% of given feedback). The most common feedback theme, which was expressed by 25 percent of the respondents was that they take software engineering courses because they want to learn practical skills, which can be applied in a real-life scenario.

[When asked on why did the student enroll to the course] "The course really seems like it is training us to work in real-life, not just showing things on the theoretical, abstract level." - Student from Course C

"It was nice to find out that the UML tools are not some mystery box, but a set of practical and simple illustrations." -Student from Course B

This sentiment was somewhat similar to the fourth and fifth most common items, "I want to learn about managing software processes" (15% of feedback) which meant that the student was interested in the management aspects of the software projects, and "I want to learn about <tool>" (15%), which means that the student was expecting to learn about some tools, technique or programming language on that course.

[What part would you keep unchanged if you could only select one'] "The actual software testing work in the exercises." - Student from Course C

"[The UML2 diagrams] are the most useful part of this course." -Student from Course A

Interestingly, the third most commented feature of the case study courses was that the students actually would have preferred project-based grading over the course exams, and because of that were complaining that the projects do not have enough impact on the course. Some of the comments declared that the project work was very efficient way to learn, and 10% of the feedback actually mentioned this separately.

"I personally found that demo assignments and projects were really good way to learn. -Student from Course B

"The project assignments really explained everything to me. I think we could drop the separate exercises. Or the exam." -Student from Course A

On the negative feedback, it was interesting to observe, that the most commented aspect was not the lack of motivation or lack of interest towards the subject, but that the students considered the

Table 1: Comparison of the case study course arrangements

Module	Course A	Course B	Course C
Course name (in English)	Fundamentals of Software Engineering	Software Engineering Methods	Fundamentals of Software Testing
Course level	Bachelor's 2nd year	Master's 1st year	Bachelor's 3rd year
# of participants	136	58	46
Lectures	11 * 2h traditional +3 * 2h video lectures	6 * 2h traditional + 6 * 2h demonstrations	2 * 2h traditional + 10 * 2h video lectures
Exercises	Mandatory; 70 tasks in total, 33% required to pass	Voluntary; 20 tasks and 6 case studies in total	Voluntary, 30 tasks in total
Projects	3 mandatory projects	2 mandatory projects, 1 voluntary extra credit project	2 mandatory projects, 1 voluntary extra credit project
Primary course manual	Textbook supplemented with lecture slides	Lecture slides and list of recommended reading	Textbook supplemented with lecture slides
Exam	Mandatory, 50% of the grade	Mandatory, 80% of the grade	Mandatory, 80% of the grade
Other features	Additional videos on topics of interest, Conclusions lecture recorded for exam preparation	Lectures recorded and available online, tutorial video archive and Facebook group	Online lecture videos and tutorial video archive and Facebook group

Table 2: Course statistics

	Course A	Course B	Course C
Enrolled students	136	58	34
Nothing done	11	13	12
All mandatory assignments done besides exam	115	37	20
Pass-% (of those who started)	40 (43) % *	64 (82)%	50 (77)%
Average grade (1-5 grading, 5 best)	2.9 *	3.1	2.8
Students answering to (% of all) the feedback survey	26 (19%)	22 (38%)	20 (59%)
Student-given grade to the course (1-5 grading)	3.0	4.3	4.1
# of Very positive comments	9	16	12
# of Very negative comments	21	3	1

*After 1st exam, exams available until 09/2018

course content or applied tools obsolete, or at least old-fashioned. Other common complaint was that the applied textbooks and exercises derived from those course materials were obsolete, or at best borderline cases. This concern was raised by 13 % of the feedback.

"I expected more advanced topics in software engineering like mobile apps engineering or web applications engineering or even consideration about future of software engineering." - Student from Course B

"Usage of better tools to create the diagrams. [Tool 1 name] is not the best idea. Personally, I have found a tool called [Tool 2 name] which is easier, with more functionalities and more stable. -Student from Course B

"Questions straight from the 70's way of working with software were unnecessary and in no way relevant to anything." -Student from Course A

"Besides being a student, I also work as a developer in [company]. None of the stuff shown here is no longer used, at least in my domain." - Student from Course A

Besides these items, the feedback also frequently mentioned aspects such as that the student expected to be working with a real-life software project (Open source or such) (13%), the course theory-practice-balance was somehow off (7%), or that they lost motivation to some other aspects of the course implementation, such as unclear instructions (9%), dislike of the lectures or lecturing style (6%) or that they simply did not find the topic interesting (7%), usually because they were forced to take the course as a mandatory part to their study program.

"I simply do not find these exercises interesting" -Student from Course C.

5 DISCUSSION AND IMPLICATIONS

Observing the student expectations and the success factors of the software engineering courses based on three implementations of university courses discussing the fundamental topics of software engineering is obviously not enough to declare any new theories or very deep insight, but it does provide insight into the student perspective and provides implications on the possible areas of interest. First of all, the given feedback promotes a number of clear implications for the areas of interest:

(1) The students expect practically applicable skills, whether it means new programming languages, development tools or simply experience transferable to real-life scenarios.
(2) The preferred way of learning software engineering discipline aims towards project and case exercises, instead of simple programming tasks or theory and examinations.
(3) The most problematic area was not the intrinsic motivation or general interest towards the software engineering as a discipline, but the impression that the taught materials were outdated and obsolete, making the learning effort meaningless.

These items more or less are in line with the observations made by [4] and [5]; the software engineering students expect to learn applicable information, and the course contents need constant revisions and cooperation with the software industry partners to stay relevant. Besides the open feedback, in our case courses this observation was also observable from the course statistics: the Course A which was also materials-wise oldest, was by students considered the worst even if the overall difficulty and required effort was the same. Even if there was a concern over the applied teaching methods of local events versus online events, as discussed for example in [11], the open feedback really did not reflect this. The students did not voice concerns over the possibility for long-distance studies, but were more critical towards grading methods, the relatively low importance of project assignments and the fact that they want to learn relevant, modern and practical skills. Few of the project-related comments also included concerns over the workload of the project, indicating that at least some of the participants were very critical towards the possibility of social loafing [12], and wanted more challenge and responsibility over their coursework.

Considering the research questions, *How should the fundamental software engineering courses be taught to enable more interest towards the subject?* and *What are the current areas of interest for the student*

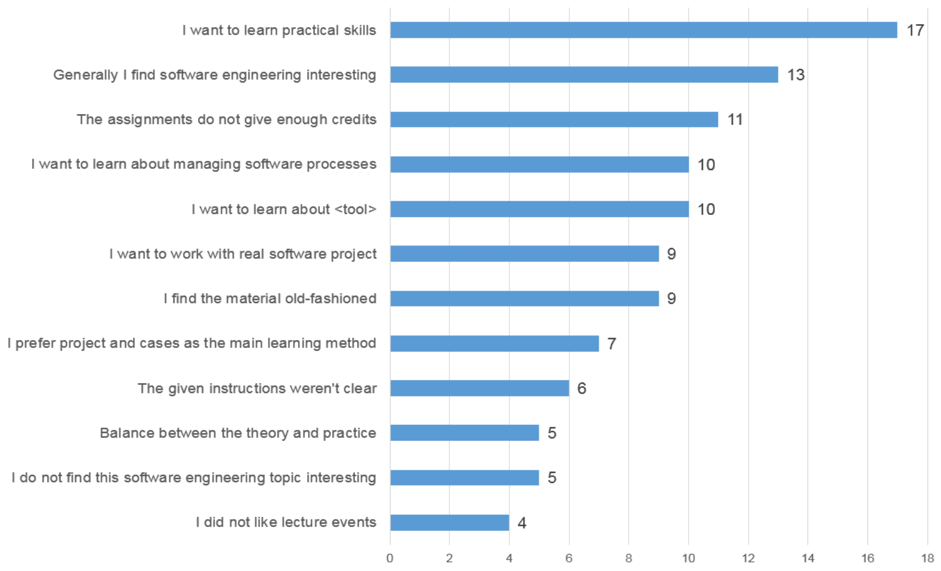

Figure 1: Most frequent feedback types from the combined course feedback. N = 68

body in software engineering?, the implications of this case study are clear. The students who are interested towards software engineering in general, seem to be more interested towards practical work and applicable experience, with the critical factor being that the students perceive the course contents to be useful and relevant. The approach into the teaching methods, long-distance enabled learning environment nor the course arrangements, are the most relevant concern if the course contents are seen something less than useful. Based on our observed cases, the students in general do not require a cutting edge digital infrastructure, if the materials and presented course content is sensible and relevant. Similar observations and considerations have been also identified and presented earlier, for example by [3].

However, it should be remembered that the course feedback was given by 68 (29.8 %) students from the total of 228 course enrollments, a minority of the entire pool of course participants. The first impression might be that the feedback was only given by the most vocal or active participants, but since the feedback surveys were voluntary and to a varying degree anonymous, a response rate of 29.8% can be considered acceptable for this type of usage, similarly as described by Fink [1]. In general, the qualitative analysis of the feedback data surfaced a number of aspects which should be taken into account in the future in the revisions of the software engineering courses. As with all qualitative studies, these observations are not strong, confirmatory results, but more like guidelines or suggestions on what aspects to take into account in the future work [10]. Similarly, there are inherent risks in the qualitative studies, which in any case mean that the qualitative data, even when triangulated against the quantitative data sources, cannot completely and universally explain the observations, only describe the phenomena in that particular organization or socio-technical ecosystem [21][22].

6 CONCLUSIONS

Software engineering as a discipline covers a vast area of different concepts, and from the viewpoint of student, this might cause confusion or identity problems for the software engineering students. Furthermore, the modern student expect certain level of services, which promotes motivation and interest towards the taught topics and concepts. In this paper, we observed three university courses, which all taught different fundamental software engineering concepts.

Based on our results, the software engineering students first and foremost expect practically applicable skills and experience, and require that the course materials are up-to-date and modern. A bit surprisingly, the student feedback when comparing online-enabled courses and a traditional lecture-based course, was focused on the relevance of the data, not on the ability to participate long-distance, or work at their own pace and schedules. Besides relevance, the students expect to learn how to manage real-life software projects, and prefer project-based approach over the theory-based lectures.

These results obviously do not form a strong theory or validate any prior works, but like qualitative studies in general, identify and pinpoint potential problems, and provide suggestions or recommendations which should be taken into account in the future works in similar conditions. In our case, the future work would be to modernize certain parts of the Course A analyzed in this study, and observe the changes in the student feedback and considerations. Other approach would be to include more advanced courses into the analysis, and develop a framework or recommendations on how the software engineering course design should be approached, or how the current ACM SE-curricula matches the modern student expectations.

ACKNOWLEDGMENTS

I would like to thank the course participants for their feedback and effort, and acknowledge the support from both Lappeenranta University of Technology (LUT) and South-Eastern Finland University of Applied Sciences (XAMK).

REFERENCES

[1] Arlene Fink (Ed.). [n. d.]. *The survey kit* (2nd ed ed.) (2003). Sage Publications.

[2] [n. d.]. SWEBOK V3 âĂć IEEE Computer Society. https://www.computer.org/web/swebok/v3

[3] Frank Achtenhagen. [n. d.]. Criteria for the development of complex teaching-learning environments. 29, 4 ([n. d.]), 361–380.

[4] Abdulrahman Alarifi, Mohammad Zarour, Noura Alomar, Ziyad Alshaikh, and Mansour Alsaleh. [n. d.]. SECDEP: Software engineering curricula development and evaluation process using SWEBOK. 74 ([n. d.]), 114–126. https://doi.org/10.1016/j.infsof.2016.01.013

[5] Nurul Ezza Asyikin Mohamed Almi, Najwa Abdul Rahman, Durkadavi Purusothaman, and Shahida Sulaiman. [n. d.]. Software engineering education: The gap between industry's requirements and graduates' readiness. IEEE, 542–547. https://doi.org/10.1109/ISCI.2011.5958974

[6] Mark Ardis, David Budgen, Greg Hislop, Mark Sebern, Jeff Offutt, and Willem Visser. [n. d.]. Software Engineering 2014 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering A Volume of the Computing Curricula Series. https://www.acm.org/binaries/content/assets/education/se2014.pdf

[7] Juliet M. Corbin and Anselm L. Strauss. [n. d.]. *Basics of qualitative research: techniques and procedures for developing grounded theory* (fourth edition ed.). SAGE.

[8] Kathleen M. Eisenhardt. [n. d.]. Building Theories from Case Study Research. 14, 4 ([n. d.]), 532–550. https://doi.org/10.5465/amr.1989.4308385

[9] Ana M. FernÃąndez-SÃąez, Marcela Genero, and Michel R.V. Chaudron. [n. d.]. Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. 55, 7 ([n. d.]), 1119–1142. https://doi.org/10.1016/j.infsof.2012.12.006

[10] Nahid Golafshani. [n. d.]. Understanding Reliability and Validity in Qualitative Research. 8, 4 ([n. d.]), 597–606. https://nsuworks.nova.edu/tqr/vol8/iss4/6

[11] Antti Herala, Antti Knutas, Erno Vanhala, and Jussi Kasurinen. [n. d.]. Experiences from Video Lectures in Software Engineering Education. 9, 5 ([n. d.]), 17–26. https://doi.org/10.5815/ijmecs.2017.05.03

[12] Rune HÃÿigaard, Reidar SÃďfvenbom, and Finn Egil TÃÿnnessen. [n. d.]. The Relationship Between Group Cohesion, Group Norms, and Perceived Social Loafing in Soccer Teams. 37, 3 ([n. d.]), 217–232. https://doi.org/10.1177/1046496406287311

[13] Jussi Kasurinen, Saeed Mirzaeifar, and Uolevi Nikula. [n. d.]. Computer Science Students Making Games: A Study on Skill Gaps and Requirement. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research* (2013) *(Koli Calling '13)*. ACM, 33–41. https://doi.org/10.1145/2526968.2526972

[14] Jussi Kasurinen, Per Runeson, Leah Riungu, and Kari Smolander. [n. d.]. A Self-assessment Framework for Finding Improvement Objectives with ISO/IEC 29119 Test Standard. In *Systems, Software and Service Process Improvement* (2011-06-27) *(Communications in Computer and Information Science)*. Springer, Berlin, Heidelberg, 25–36. https://doi.org/10.1007/978-3-642-22206-1_3

[15] J. Kasurinen, O. Taipale, and K. Smolander. [n. d.]. How Test Organizations Adopt New Testing Practices and Methods?. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (2011-03). 553–558. https://doi.org/10.1109/ICSTW.2011.63

[16] Li-Qun Kuang and Xie Han. [n. d.]. The Research of Software Engineering Curriculum Reform. 33 ([n. d.]), 1762–1767. https://doi.org/10.1016/j.phpro.2012.05.282

[17] Monica M. McGill. [n. d.]. Learning to Program with Personal Robots: Influences on Student Motivation. 12, 1 ([n. d.]), 4:1–4:32. https://doi.org/10.1145/2133797.2133801

[18] William Isaac McWhorter and Brian C. O'Connor. [n. d.]. Do LEGOÂő Mindstorms Âő motivate students in CS1?. In *ACM SIGCSE Bulletin* (2009), Vol. 41. ACM, 438–442.

[19] Dan Pilone and Neil Pitman. [n. d.]. *UML 2.0 in a nutshell* (1st ed ed.). O'Reilly Media. OCLC: ocm58999170.

[20] Roger S. Pressman. [n. d.]. *Software engineering: a practitioner's approach* (6th ed ed.). McGraw-Hill. OCLC: ocm53848343.

[21] Colin Robson and Kieran McCartan. [n. d.]. *Real world research: a resource for users of social research methods in applied settings* (fourth edition ed.). Wiley.

[22] Robin Whittemore, Susan K. Chase, and Carol Lynn Mandle. [n. d.]. Validity in Qualitative Research. 11, 4 ([n. d.]), 522–537. https://doi.org/10.1177/104973201129119299