Experiences from a Web-based Course in Software Testing and Quality Assurance

Jussi Kasurinen
South-Eastern Finland University of Applied Sciences (XAMK)
Pääskysentie 1
48220 Kotka

ABSTRACT
Fundamentally, computer science and its courses are considered difficult to learn, since so many concepts has to be grasped before anything worthwhile can be achieved. To make the things even more difficult, there also is a drive to cut costs on the teaching work, to minimize the amount of teaching staff and in general, steer the course modules towards web-based learning and assisted self-study. In this study, the objective is to assess the different tools and approaches available for constructing an online-enabled course on software testing and quality assurance (QA), based on two different course implementations to provide experiences and information. Based on our observations, the most important factors in teaching a course in software testing with an assisted self-study approach is to offer practical exercises using real software projects, discuss real-world scenarios in the lectures to maintain the student motivation, offer equal services to both online and offline students and discuss both the management and practical work aspects of the testing work.

General Terms
Software and its Engineering, Software creation and Management, Software verification and validation, Computing education, Software Engineering education.

Keywords
Software testing; quality assurance; case study; experiences.

1. INTRODUCTION
Pedagogically programming work and software development should be compared to problem solving. Software engineers design a product to solve a problem, solve the problem of how to make our design work with the given technical infrastructure, and solve the problem of proving that the product works correctly. However, due to the nature of computer science and software development work in particular, the approach in teaching has to extensively teach structures, processes, concepts and programming languages to the students before any practical result can be reached [15]. Overall, the attention in designing of the introductory computer science courses should be in defining the desired learning objectives and methods of exposing students to meaningful, but simple, case studies.

Even with this large drive to develop better learning outcomes, it is unfortunate that students seem to be generally disinterested on the computer science topics, even if they know that programming is really important skill to possess. On the programming courses, this problem has already been acknowledged to be tied closely to the motivational aspects in the loss of focus in the course topics [5, 21], but how should the issue of motivation and lack of interest in the course discussing software quality assurance and testing work be addressed? This topic is important, since teaching topics such as low level unit testing or building test cases are not very far from programming work [6]; so close in fact, that the computer science education curricula 2013 [8] for software engineering emphasizes verification and validation – testing activities - more than the construction of the software. In practice, the testing activities such as code reviews or module integration testing are more or less programming work, since they involve direct manipulation, or at least direct proof-reading, of the source code.

This paper describes a course design project to teach software testing as a part of computer science education curricula with the minimal amount of local teaching events or teaching resource needs. The concept is to create the new course for demand, where the ratio between teaching personnel and students does not allow for personal training or tutoring sessions. On the other hand, this study also focuses on the aspect of how to effectively apply online services and tools to substitute for tutoring sessions and teacher-administrated course events. Hence, the research questions for this study are the following: "*What are the beneficial on-line services for successful testing course?*" and "*To what extent can a technically challenging CSE course be offered online*?."

Rest of the paper is constructed as follows: The Chapter 2 discusses the related research and concepts of this study, and the Chapter 3 introduces the applied research method. Chapters 4 and 5 introduce the results and summarize their implications, Chapter 6 discusses the shortcomings and limitations of the study and finally, Chapter 7 closes the paper with conclusions.

2. RELATED RESEARCH
The fundamental concepts in computer science are a field, which extensively develops different types of tools and services to enhance the learning experiences. Several studies from the last thirty years indicate that learning computer science is actually very difficult [1,20], and usually benefits from any support or tool it can apply. These tools come in several sizes, offering wide range of different teaching approaches. Telling example of the diversity of these different learning environments and teaching tools is that the basic work group report on learning environments for computer science curricula [16] combined with the basic taxonomy for long distance learning ecosystems [12] cover over hundred different examples of different types of systems, which all are plausible, diverse and completely functional learning tools.

Besides right tools, the other aspect of designing computer science courses is the student motivation. For example, studies by Shell et al [20] or Guzdial and Soloway [5] discuss these problems in a programming context. The modern students expect to have more meaningful assignments than traditional source code based command prompt assignments, and are only learning if positively reinforced through motivation to achieve results. Similarly, a study by Krutz et

al. [13] puts this into the testing perspective; even if testing is usually the most costly phase of software development and simultaneously the largest influence to the product profitability, the students tend to think testing work as boring and unnecessary. In Krutz et al. study, this problem was addressed by applying real open-source cases as the course assignments. Based on their results, 85 percent of their students considered this approach to be positive, with student feedback also indicating improved motivation and learning results. Finally, a study by Smith et al. [17] discusses similar requirements for goals of developing testing course: testing activities in a university course have to be fun and competitive activity, allow students to learn from each other, demonstrate the importance of doing testing work, and provide a mechanism to evaluate the demonstrated testing skills.

Another concern for designing a testing course is discussed by Kazemian and Howles [11]. Their study points out, that testing-related courses tend to have additional problems with the course infrastructure. Since most of the industry-applied testing tools are commercial, they usually are not available for academic institutions to use without expensive licensing deals. In addition, since large amount of testing work in the industry is related to creating and following plans to systematically ensure product quality, testing course should also address these issues, in addition of the traditional mechanical testing work of running use cases in the test environment. In another example by Harrison [6], the testing course actually consisted of two parts: first learning the low-level testing techniques from the viewpoint of software developer, and later managing testing work and test documentation from the viewpoint of software tester. This approach, and practical assignments instead of purely theoretical ones are needed, as the lack of interest towards developing the teaching approaches of software testing, and the differences between academic and industrial interest in the testing work, are considered so widespread, that they start to hinder each other [6]. In wider context, a study by Eldh and Punnekkat [3] discuss the general needs of development for computing science curricula in academia. In their study a list of topics which should be addressed more detail is presented; topics such as professionally applied tools, industry de-facto standards of working, agile teams in large projects and most importantly, "Testing at all levels of software".

In any case, there are several observations on how software testing course module should be constructed. For example, earlier studies into the design and revisions of computer science course modules (for example withheld for review) have indicated that the course infrastructure and seamless integration of all different components of the course is really important to maintain the student motivation. By applying all these observations, our testing course was defined based on the recommendations by the literature reviews as follows: the course will apply access to network and social media [5], apply practical project assignments [9], promote student project works [3], teach both management and testing work in practice [6], use open source or freely available tools [11], promote some form of team work [17] and address the motivational aspects [13].

3. RESEARCH METHOD

The approach on developing the testing course relied on the systematic process improvement, and it had two objectives: 1) to develop a functional infrastructure for teaching and learning the fundamentals of software testing and 2) to create indicators to aide continuous assessments and cyclic reviews to develop the course module. In one previous work by our research group [10], it was established that this approach is appropriate for developing computer science course modules. In this scenario, the approach included a prior student data set collected from a similar course development project with software engineering methods, the statistical comparison of course feedback surveys conducted before and after the course, the statistics collected from the course material repository, and a case study analysis of the collected open feedback and course project work. A visualization of the data sources and primary research methods are summarized in Figure 1.

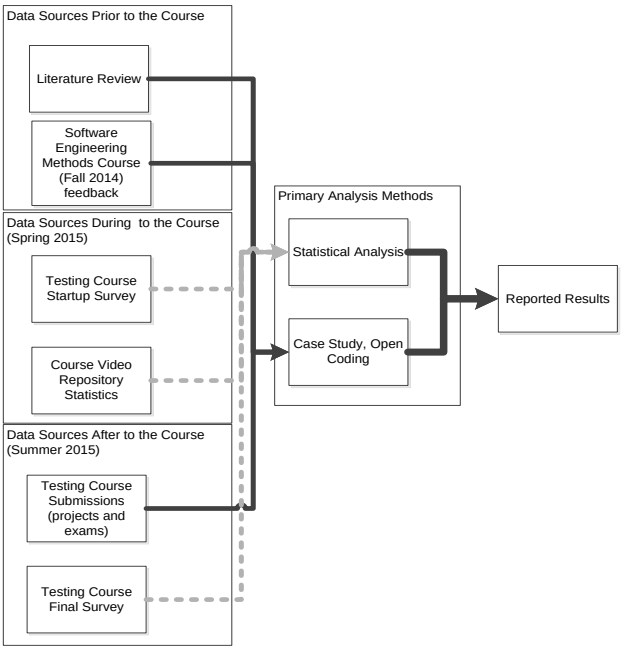

Fig 1: Main data sources and primary analysis methods

The objective of these approaches was to assess the student performance during the course, and collect information on student background and experience on the topic (Testing Course Start-up Survey), student activity during the course (Testing Course Video Repository Statistics, enrolment information), learning results (Testing Course Submissions) and motivational aspects (Testing Course Final Survey). The statistical analysis of the student data was mostly conducted by assessing the key indicators, such as enrolment records (local and online statistics), course statistics (drop rate, grades) and survey data (background information, prior experience on software development) to find metrics which could indicate problems or potential enhancement points for the course. These observations were further studied and validated with the qualitative data to ensure proper observational triangulation which is essential to this type of mixed-method approach. In addition, to assess the usability of the qualitative data, a chi squared-test was conducted to establish that the student bodies of both of the courses were results-wise representative of the same population. Therefore the testing course student population and software engineering student population were comparable against each other, since the differences between the course results were statistically insignificant with p= .05. Data collection instruments for surveys are available at the address http://tinyurl.com/ksm5wu6.

The collected course feedback and especially the qualitative data from the surveys was classified and codified following the principles of the open coding method from the Grounded Theory [4, 18]. The open coding and case analysis was done

to collect observations and identify repeated themes from the data; the amount of observations did not warrant a full Grounded Theory analysis, but pinpointed major themes and was used to understand the different observations from the quantitative data. For this paper, only observations which were present both in the qualitative and quantitative sources are reported to establish more confidence in the results. The actual codification and analysis work was done using office tools such as Excel.

4. RESULTS

The resulting course had the following objectives: "*After finishing the module, the student is familiar with the most common work methods and tools of software testing. The student is capable of conducting independent testing work under normal project administration and is able to design and prepare for testing work-related aspects. The student knows how testing work is done, and how quality assurance and software development are related.*"

In the lectures, the course focused on the topics such as testing tools, test phases, test levels, different test methods, testing-related standards and certifications, developing testing work, measurement of quality and defining quality. In the exercises, different testing tools and testing work activities such as unit testing, integration testing and system testing were practiced with separately created scenarios, which applied real-world open source software projects and industrially applied testing tools. More detailed information about the course components is in the Table 2. In addition, in Table 3 there are more statistics concerning the course outcomes.

Table 2: The course infrastructures

Component	2014 Fall SEM	2015 Spring Testing
Course lectures, also recorded video archive	6 weeks * 2 hours traditional, 6 weeks * 2 hours demonstration	12 weeks * 2 hours
Course exercises	5 * 2 hours traditional, 6 * 2 hours demonstration lectures. Voluntary attendance.	11 * 2 hours, voluntary attendance. Weekly summary video recorded for the course video archive.
Tutorial video archive	39 videos	18 videos
Social media tools	Course videos, lecture archive on YouTube, University courseware system.	Course videos, lecture archive on YouTube, University courseware system, Facebook group
Course projects	2 mandatory group projects, 1 voluntary extra credit project.	2 mandatory group projects, first on actual testing work, second on planning testing work.
Availability of teacher consultation	1 hour per week, 12 h total	1 hour per week, 12 h total
Course manual	None, lecture slides and additional reading material. A separate course book available but not mandatory.	Yes, 80 pages; also lecture slides A separate course book available but not mandatory.
Exam	Mandatory. Possibility to gain points with course assignments.	Mandatory. Possibility to gain points with course assignments.

4.1 Course feedback and statistics

Besides statistics on the course outcomes, feedback was collected with three surveys: 1) Course end survey for SEM in December 2014, 2) Course starting survey for Testing in January 2015 and 3) Course end survey for Testing in April 2015. Based on the collected course feedback, we can make several observations how the course structures worked.

Table 3: Course outcome on Introduction to Software Testing, with Software Engineering Methods for comparison

Metric	Fall 2014 SEM	Spring 2015 Testing
Number of students enrolled (Number of students starting[1])	58 (45)	34 (22)
Percentage with programming experience on commercial software project or organization.	N/A	23 %
Percentage with previous testing-related experience	N/A	18 %
Passing grades given	37	17
Pass rate (Pass rate from students starting the course[1])	64 % (82 %)	50 % (77 %)
Nothing done[2]	13	12
Withdrawals during the course	1	2
Average grade (0-5 scale)	3,1	2,8
Passed with grade "1", the worst grade	2	2
Passed with grade "5", the best grade	18	5
All mandatory project tasks returned	37	20
All mandatory and extra credit project tasks returned	30	16
%-of all enrolled students, who filled the feedback survey	38 %	44 %

[1]Students enrolled minus the students with nothing done.
[2]Student did not do anything beyond registering for the course.

For example, in the course feedback for *"Software Engineering Methods"*, the most disliked feature of the course were the traditional lectures, which were graded 3,86 (on scale 1-5, 5 best grade) while the traditional exercise events gained a grade 3,95. All the other parts of the course scored at least 0.2 higher, while the course average grade for structures was 4.27. For the lectures, the low grade can be partially explained by the unappealing schedule, as demonstrated by this feedback:

"I think the time of lecture at 8AM on Fridays was not good and I preferred to watch lectures in you tube instead of participating to class."

Considering that the lecture videos and topical video archive was the highest graded feature of the course (4.68, scale 1-5), and the demo lectures (combining work and lecturing) was also well-received (4.2). Overall, some additional observations can be made based on the given open feedback from the course. For example, the project works were criticized for being unclear and too extensive:

"However the projects themselves alone can potentially be fairly extensive. Something that most courses would only have one of."

"Second project wasn't very clear, because it was supposed to do in parts weekly but at the end I wasn't sure what I should return and what my project work should consist of. I suggest that there should be clearer list of what to do."

In addition, the lectures and exercises in the first period were considered to be repetitive and redundant. In exercises particular, also the applied software tools were also criticized:

"Usage of better tools to create the diagrams."…" A small tutorial in class could be very helpful."

"The subjects are quite repetitive. I expected more advanced topics"

However, taking into account all of the negative feedback, it should also be mentioned that most of the open feedback for

the course was positive and the course seemed to be well-liked. A clear majority (71%) of the feedback was positive or had something positive to say about the course or the teaching style.

On the design of the course in software testing, the students were requested to list their expectations on the implementation of the course "*Fundamentals of Software Testing*". On the starting survey, students were requested to explain, what they wanted to learn from the course, besides the obvious "the fundamentals of testing work". In Figure 2, the most common topics from the student requests are summarized. Out of 22 submitted responses, a majority (59 %) expected to learn about the real life applications and experiences on the software testing. From the feedback, this expectation seemed prevalent mostly because the software testing has been discussed in the earlier courses, but not in any detail:

"Testing is the area of software engineering on which I have the least amount of experience, so any practical information would be most welcome."

In addition, management of the test processes from the perspective of a test manager, and conducting software testing from the viewpoint of software tester were considered almost identically important topics (11 requests for management, 10 for tester's work). The application of different documents, certain testing tools (such as automation suites or unit testing frameworks) and certain methods (test automation, stress testing) were also mentioned several times.

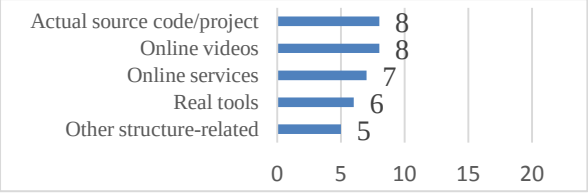

Fig 3: Course structure expectations for the "Fundamentals of Software testing" (N = 22)

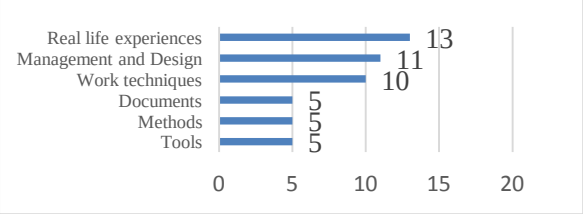

Fig 2: Student learning expectations for the "Fundamentals of Software testing" (N = 22)

In a separate item, the expectations for the course structure and teaching tools were also requested, and they are summarized in the Figure 3. In this category, the collected feedback was not as focused as in the learning expectations, but also in this category the practical experience was the most requested feature. 8 (36% of all) comments requested that the course exercises and project works should be done with real source code, taken from actual software development projects. Following the similar trend, 6 students requested "real, industry-applied tools". On the learning tools, 8 students requested some form of online recordings of the lecture and exercise events, while 7 requested other online features such as slide sets, tutorial videos or open access learning material. Rest of the feedback, both in learning expectations and course structure, were random remarks or other observations.

The types of feedback reflected the student opinions and grades given for the different parts of the course; the most liked components were lecture recordings (4.5 on 1-5 scale, 5 best grade), lecture presentations (4.3) and the course content (4.2). Interestingly, both projects received a grade of 3.9 which was a bit below the average grade for the course implementation, which was 4.1.

The end survey also collected information on the aspects the students considered to need revision for the future implementations. The most common criticism was over the exercise events. Since these events were not mandatory, several students did not participate on them, but in the end survey indicated that they would have liked them more if they would have been mandatory, or at least given extra credits for the final grade. This was in line with the observation that the exercises themselves were also the least liked feature of the course (3.18 on 1-5 scale, 4,01 course average) by a large margin.

"Weekly exercises should be 'more' mandatory. This would make more students participate, and they would be more useful for learning."

"Exercises should somehow be made mandatory or at least more integral for learning stuff. For example, could the exercises somehow lead to the completion of the project works?"

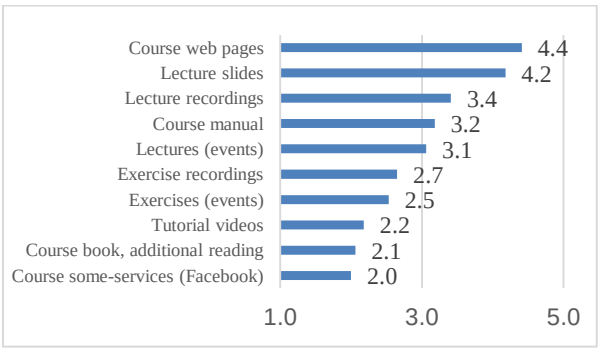

Fig 4: Most applied learning tools and services according to students (N = 17, 1 = Did not use at all, 5 = Used constantly)

Finally, the last part of the end survey collected information on the most applied learning tools and course services by the students (Figure 4). Unsurprisingly the course webpages were the most applied service (4.4, on scale 1-5 where 1 =did not use at all, and 5 = used constantly). The other applied tools were lecture slides (4.2), lecture recordings (3.4) and course manual (3.2). The least applied were social media services (2.0), course books and additional reading material (2.1) and a bit surprisingly, the tutorial videos (2.2). Local teaching events were also less applied than their online equivalents, lectures got grade 3.1 and exercises 2.5. In open comments, some student feedback indicated that the students felt surprised that they did not feel penalized for having to use the online resources:

*"The quality of the given material and the fact that nothing is withheld as a punishment from the online participants, gave the feeling that we actually *are* studying at the university."*

When combined with the data from starting survey on what learning tools the students usually apply, there are some

observations: For example the startup survey mentioned lecture recordings at 3.7 (0.3 difference) and exercise events at 3.0 (0.5 difference). In addition, social media services were originally rated at 1.5, but were at 2.0 at the end survey. The application of social media tools was somewhat divisive; most of the students did not use the service at all, but few students considered it to be one of the most important tools for the course. The statistics from the course video archive for lecture and exercise recording and for the tutorial videos reveal that the service was applied to some degree, but was not a major success. Overall, testing course-related videos (33 videos) got total of unique 603 views during the course. However, it should be remembered that these numbers reflect the amount of students on the course: 22 active students, who view majority of the 33 course videos once, is in the ballpark of the amount of views received.

4.2 Implications

Since there are several data sources and many statistics related to this study on designing a course on software testing, it would be relevant to present a summary of implications made from the collected data. Based on the collected feedback and case analysis over the two courses, the data indicates following implications for the results:

- The lectures can be replaced completely with the pre-recorded lectures, especially if the scheduled timeslot is unappealing (early in the morning or late in the evening).
- Supporting self-study is more efficient than offering face-to-face learning events, but based on the feedback some students need at least the theoretical possibility for personal tutoring even if this option is never used.
- For online-enabled course, it is an important motivational factor that the students who rely solely on the online materials feel equal to the students participating to the local teaching events.
- Exercise events should offer a tangible benefit, such as extra credits for the exam or easier way to accomplish course projects.
- Course-administered social media integration is not absolutely required, but students need one focused online location for all information and material, which is actively maintained and updated.
- In the testing context, the fundamentals-level course should offer both management skills and practical testing skills in the curricula.
- In the testing context, the possibility or at least the illusion of possibility, to apply the course learning experiences directly in the practice is an important motivational factor for the students.

5. DISCUSSION

These results obviously cannot be explained with any single action, so-called a silver bullet [2], but approach the issue from the Software Engineering Curriculum [8] point of view which claims that the success of an educational program depends on three elements: faculty, student body, and the infrastructure. This approach indicates, that each course module has three irreplaceable, and always present elements; the faculty who teach and administer the course, student body which enrolls to the course and works towards passing the course, and the infrastructure which enables the faculty to teach and the students to learn. Since affecting the faculty or the student body is difficult, and because people involved could not be changed or selected, the best aspect in this equation was to enhance and improve the course infrastructure. Removing the small problems in the course infrastructure and tuning the course based on the prior feedback from similar modernization efforts made a big difference for the students. These explanations are supported by the general change research, which claims that major changes lead to performance dip [14] and the motivation research which claims that employee motivation is complemented by 'hygiene factors' that cause dissatisfaction among employees and distract them from the actual work [7]. In practice this means that the biggest causes of dissatisfaction in the work are the small irritants, which cause unnecessary problems and divert the learning focus from learning the substance to learning to cope with the given tools. In this study this problem can be demonstrated with the negative feedback caused by the tools used in the SEM course. The tool used to draw the UML diagrams was causing problems, so improving the hygiene factors of this course would mean that the tool has to be changed to something more functional.

In the infrastructure, the recommended changes proposed by our literature review were applied. In this study, the results of [9] and [6] were replicated in almost every aspect. It seems, that for the motivational aspects the illusion of learning practically applicable skills is very important. In addition, the first course in software testing seems to need to address both the management and testing work aspects, since even if the students are aware of the testing work as a subject of software engineering, it seems that these topics may not be covered in detail in the software engineering courses. Overall, based on the observations this course should not focus on certain level, method or tool of software testing, but focus on covering the basics of the entire software test process and quality assurance work both from the viewpoint of management and testing tasks. From the organizational viewpoint the results indicate, that the students do prefer online sessions over traditional teaching, and also bit surprisingly also prefer mandatory – or at least grade-affecting - exercises. One important observation on the importance of the motivational aspects however, was the first project work, in which students conducted real explorative testing on a real open source game. This project was very well-received, and had 100 percent retention rate of students.

Obviously the results of this case study are open for discussion, should the experiences be transferred to another environment. To maintain the validity of this study against the common threats (for example [19]) our student groups were compared against each other with a chi squared test to ensure that they represent the general student population. The results and the collected data was discussed with peers to avoid personal bias, several passive and active data sources was applied and finally, the collected data applied both qualitative and quantitative approaches to triangulate the collected data. Finally, only observations which were present both in the qualitative and quantitative data were reported. In any case, the results of a qualitative study cannot be generalized since every ecosystem has unique features, but the results are useful indicators or guidelines, when observing new ecosystems.

6. CONCLUSIONS

In this paper the development of a web-based course on software testing was discussed. The objective of this study was to understand which the most beneficial course structures are, and which course design approaches enable the students to learn with minimal face-to-face interaction with a teacher. In addition, this study also focused on constructing a course module on software testing, based on the existing experiences on teaching programming courses and general course on software engineering methods.

Based on the collected data and observations, the web-based approach was functional and there were no immediate or critical problems with the course infrastructure. The most important aspects from the viewpoint of the students were the possibility to get everything from the online sources without feeling that the online-only participants are withheld information, and maintaining the illusion of possibility to use the course experiences in real-world projects. For example, the first project involving explorative testing with an open source software was completed by 100 percent of the students who started the course. In general, the students preferred online sessions over the traditional in-class sessions. The online components such as the lecture archive, video tutorials and social media services were not heavily applied, but served their purpose. Overall, the course results indicate acceptable teaching outcome at 2.7 average (on 0-5 scale) with 77 percent pass rates from the students actually starting the course. In addition, the data collected so far indicates that the infrastructure was well-received and the course contents matched the student expectations.

As for the future research, the next action should therefore be to seek if the course infrastructure introduced here is transferable and feasible in other computer science domains. For example, can the same infrastructure be applied in more technically oriented context, such as programming-focused course module? Other interesting option would also be to replicate the course with a larger student body, to gain confidence on the results reported in this publication.

7. REFERENCES

[1] Avison, D. & Fitzgerald, G. 2003. Information systems development: Methodologies, techniques and tools, 3rd edition., Berkshire, England: McGraw-Hill Education.

[2] Brooks, F.P. Jr., 1987. No Silver Bullet Essence and Accidents of Software Engineering, Computer Vol 20(4), doi: 10.1109/MC.1987.1663532

[3] Eldh, S. & Punnekkat, S. 2012. Synergizing industrial needs and academic research for better software education. In Proceedings of the First International Workshop on Software Engineering Education Based on Real-World Experiences (EduRex '12). IEEE Press, Piscataway, NJ, USA, 33-36.

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

[5] Guzdial, M. & Soloway, E., 2002. Teaching the Nintendo Generation to Program. Communications of the ACM, Vol 45(4), pages 17-21.

[6] Harrison N.B. 2010. Teaching software testing from two viewpoints. J. Comput. Small Coll. 26, 2 (December 2010), 55-62.

[7] Herzberg, F. 1968, One more time: how do you motivate employees?, Harvard Business Review, Vol. 46(1), pages. 53–62.

[8] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society. 2013. Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. ACM, New York, NY, USA.

[9] Jones, E.L. & Chatmon, C.L. 2001. A perspective on teaching software testing. J. Comput. Sci. Coll. 16, 3 (March 2001), 92-100.

[10] Kasurinen, J., & Nikula, U. (2016). Just Passing Courses or Learning: Building Bayesian Classifier to Assess Outcomes. International Journal on Information Technologies & Security, 8(4).

[11] Kazemian, F. & Howles, T. 2005. A software testing course for computer science majors. SIGCSE Bull. 37, 4 (December 2005), 50-53. DOI=10.1145/1113847.1113876 http://doi.acm.org/10.1145/1113847.1113876

[12] Kelleher, C. & Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, Volume 37(2), pages 83 – 137.

[13] Krutz, D.E., Malachowsky, S.A. & Reichlmayr, T. 2014. Using a real world project in a software testing course. In Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14). ACM, New York, NY, USA, 49-54. DOI=10.1145/2538862.2538955 http://doi.acm.org/10.1145/2538862.2538955

[14] Nikula, U., Jurvanen, C., Gotel, O. and Gause, D.C., 2010. Empirical validation of the Classic Change Curve on a software technology change project, Information and Software Technology, Vol. 52(6), doi: 10.1016/j.infsof.2010.02.004

[15] Pears A., Seidman S., Malmi L., Mannila L., Adams E., Bennedsen J., Devlin M. & Paterson J. 2007. A survey of literature on the teaching of introductory programming, ACM SIGCSE Bulletin, Volume 39(4), pages 204-223.

[16] Guido Rößling, Mike Joy, Andrés Moreno, Atanas Radenski, Lauri Malmi, Andreas Kerren, Thomas Naps, Rockford J. Ross, Michael Clancy, Ari Korhonen, Rainer Oechsle, and J. Ángel Velázquez Iturbide. 2008. Enhancing learning management systems to better support computer science education. *SIGCSE Bull.* 40, 4 (November 2008), 142-166. DOI=http://dx.doi.org/10.1145/1473195.1473239

[17] Smith, J., Tessler, J., Kramer, E. & Lin, C. 2012. Using peer review to teach software testing. In Proceedings of the ninth annual international conference on International computing education research (ICER '12). ACM, New York, NY, USA, 93-98. DOI=10.1145/2361276.2361295 http://doi.acm.org/10.1145/2361276.2361295

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

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

[20] Winslow, L.E., 1996. Programming pedagogy – A psychological overview. SIGCSE Bulletin, 28, pages 17-22.

[21] Duane F. Shell, Leen-Kiat Soh, Abraham E. Flanigan, and Markeya S. Peteranetz. 2016. Students' Initial Course Motivation and Their Achievement and Retention in College CS1 Courses. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). ACM, New York, NY, USA, 639-644. DOI: http://dx.doi.org/10.1145/2839509.2844606