Designing Early Testing Course Curricula with Activities Matching the V-Model Phases

Timo Hynninen*, Antti Knutas** and Jussi Kasurinen**
*South-Eastern Finland University of Applied Sciences / Department of Information Technology, Mikkeli, Finland
**LUT University / LUT School of Engineering Science, Lappeenranta, Finland
timo.hynninen@xamk.fi, antti.knutas@lut.fi, jussi.kasurinen@lut.fi

***Abstract*—This work addresses the gap between software engineering process terminology in formal education, and the practical skills relevant to testing related work. The V-model is a commonly referenced description of how the software engineering processes are tied to the different software testing levels. It is used in software engineering education to illustrate which type of testing work should be carried out during a certain development stage. However, the V-model is mainly conceptual and tied to the steps in the Waterfall model, leaving the students with little knowledge about what is actually done. To solve this problem, we propose an approach to map the V-Model development phases and testing levels with corresponding, actual testing techniques. We then evaluate the approach by designing the weekly topics, learning goals and testing activities for a 7 week introductory course on the basics software testing and quality assurance. Based on the course outcomes and recent literature, we discuss the strengths and weaknesses of the proposed curriculum.**

I. Introduction

Software testing and quality assurance (QA) form an integral part of software engineering processes and therefore should be an equally integral part of software engineering education. Testing education improves software quality, as testing-savvy students learn techniques that lead to more reliable program code [1]. For example, the ACM curricula for software engineering [2] integrates testing and quality assurance into other domains of computing education. However, the ACM curricula has been criticized for not conveying a strong enough testing and quality assurance mindset [3]. Our proposal to address this issue is to design concrete learning objectives and testing activities that follow the principles of constructive alignment [4], [5]. This approach carries over from our previous work in using constructive alignment, creating high-level guidelines for software testing education from the industry practices [6].

The motivation behind designing a dedicated undergraduate testing course and its activities was that currently software testing education research has mainly focused on the implementations of such courses and not in course content design. Although previous research has also established approaches to integrating testing and QA work into larger projects [7], [8], many institutions organize an undergraduate course in the methods and models of software testing separately. In addition, the software industry leaves a lot of responsibility in QA work to the shoulders of individual employees, while acknowledging that personnel do not always have the necessary skills in testing beforehand [9]. We therefore feel that the objectives of this study are of interest to many in higher education.

In order to place the testing activities into a software engineering context, we contrast them with the phases in the V-Model [10], [11]. The V-Model (see Figure 1) is a generic software development process model where requirement analysis, specification, architectural design, and detail design are linked with the levels of testing, namely acceptance testing, system testing, integration testing, and unit testing. These development process phases and testing levels are often referenced in software engineering education. However, in the education and training context, the practical impact of these activities may play an auxiliary role or even be neglected. Hence, students might be familiar with the development process phases on an abstract level but fail to understand which practical activities should happen within them.

To summarize, our research questions in this paper are as follows.

- RQ1. What learning activities can we map to the high-level testing concepts?
- RQ2. Which actual testing techniques can be utilized?

In order to answer the research questions, we used the principles of constructive alignment [4], [5] to design and implement an undergraduate course on the fundamentals of software testing. We planned the topics and activities for a 7-week (one period), first-year freshman course. The course had no other prerequisites except the freshman course on introductory programming. Various techniques for testing were adapted from Swebok [12] and the ISO/IEC Software testing standard, which covers a multitude of testing techniques in ISO/IEC 29119 Part 4 [13]. These testing techniques were used as a starting point for designing assignments demonstrating the practical testing work on each testing level. Additionally, these topics and learning activities were mapped to fit the V-model, so that students would be able to relate the practical testing work conducted during this course with software engineering process steps in their later studies and working life.

The rest of this paper is organized as follows. Recent studies on software testing education are presented in

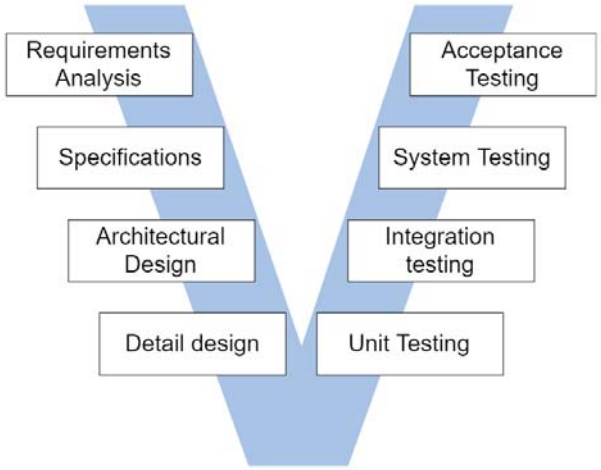

Fig. 1. The software testing V-Model, adapted from [11]

Section 2. Section 3 introduces our course implementation and its results. Discussion and implications are discussed in Section 4. Finally, we conclude in Section 5.

II. Related Work

Educators face challenges when it comes to testing concepts. For example, students find it difficult to digest testing concepts unless they are introduced one at a time [14]. In addition, many instructors do not have the necessary knowledge that should be taught to students [1]. Additionally, the motivational aspects especially in technically challenging topics are well-known factors influencing the outcomes of a learning scenarios to a significant degree [15].

Testing course content has been studied in different countries, for example by Šošić in Serbia [16], Bin in China [17] and Kasurinen in Finland [18]. According to Kasurinen, students want software testing education to be practice oriented, using real-world tools with a real software project to promote the motivational aspects of learning something practical, which provides skills applicable in real-life software development work.

Experiences from running traditional university courses in software testing, or software verification and validation, have been previously reported by Mishra et al. [19] and Lopez et al. [20]. Van Eijck et al. [21] designed a flipped classroom version of the testing course in addition to bundling the course contents around Microsoft's developer software [22].

Gamification in education is a rising trend in the testing field. Fraser [23], Valle [24], Fu [25], and Soska [26] all present recent approaches for gamifying software testing education. In general, gamification can be used to increase student motivation and communication [27].

Other approaches for increasing student motivation on the testing course include using large, real-world projects. For example, Krutz et al. [7] used open-source projects and Garousi industrial software projects [8]. Another interesting approach employed by Chen et al. infused research topics into the testing curriculum [28].

All in all there is an extensive body of related work to software testing education. The studies focusing on testing education at university level have been mapped in, for example, [20] or [29]. However, these curriculum design or testing course implementation studies have not been carried out with introductory software development courses.

This research gap in early software testing education has previously been addressed by Scatalon et al. [30], who point out that early testing helps build better understanding of the programming process, and testing is widely recommended to be addressed in the beginning of studies. However, the study also reveals that there are potential drawbacks to integrating testing into the first programming courses, such as additional workload on course staff, or students' negative attitude towards testing, favoring programming over testing. In this context we feel that a curriculum for a separate, early testing course is relevant to the software engineering educators.

III. Course Implementation

In the academic year 2017-2018 the course Principles of Software Testing was arranged in parallel with a basic course on C programming. The course population consisted of first year computer science students, and also students majoring in other technology programmes such as electrical engineering, mechanical engineering, computational engineering, and industrial engineering management. The objective of the course was to cover the most common software testing methods, give the students an overview of how testing and software engineering are related, and give the students the transferable skills to perform testing related work autonomously or as part of an organization.

We created the course syllabus by taking the high-level objectives and relating the required testing-related skills to the ones that can be acquired by mastering the V-Model. During the process we also mapped the development phases and testing levels from the V-Model to the set of weekly lecture topics on software testing practices.

During the mapping process we used the theory of constructive alignment as the guiding principle when setting learning goals and designing course activities. Constructive alignment is an outcomes-based approach to teaching in which the learning outcomes that students are intended to achieve are defined before teaching takes place [5]. Teaching and assessment methods are then designed to best achieve those outcomes and to assess the standard at which they have been achieved. The teaching environment, practices and evaluation should support learning goals and the student's future environment [4]. We summarize the principles of constructive alignment [4], [5] as follows:

- Learning goals should be clear, serve a purpose, and set in advance.
- Students need to be placed in situations and environments that elicit the required learnings, with declarative teaching minimized.

- Students are then required to provide evidence, either by self-set or teacher-set tasks, as appropriate, that their learning can match the stated objectives.

In the next phase, we took different testing activities and test techniques, and placed them under the weekly lecture schedule. The testing techniques were taken from the ISO/IEC 29119 Software testing standard. The different testing activities were carefully selected to fit the development phase and test levels according to the V-Model activities. For example, Black-box testing and exploratory testing techniques were used as exercises on the system testing level, whereas the unit testing level used the White-box testing approach. Similarly, state transition testing, scenario testing and random testing were used as the approach to specification and requirement analysis, while the classification tree method was used on the integration testing level.

In total, the course consisted of seven weeks of instruction in the form of lectures and voluntary exercise (tutoring) sessions. The weekly course topics are presented in Table I.

The concepts and skills covered in the course material were assessed in two parts: First, the students performed, reported and planned a small-scale testing project using a small console application. The students were given the source code, and a manual detailing the different modules and source code files. The assignment was completed in groups of 2-3 and it accounted for 35% of the total course grade. A written exam worth 25% of the grade formed the second part of course assessment. Voluntary weekly exercises, also completed in groups, formed the rest of the course grade, but the emphasis in grading was on the project and the exam.

The testing project was graded by the head teaching assistant based on the completion of five individual parts. Parts 1 and 2 consisted of system testing activities. First, the tests were completed manually using exploratory testing as the main method. Then in part 2 the task was to automate some of the test cases developed in part 1 by recording the inputs and program outputs during the test.

In part 3 of the project we tasked the students with writing unit tests for individual modules of the software. Part 4 was an exercise in testing work from a managerial point of view, and students were to develop a testing plan for the project software as if it was a real product by a real software company. Part 5 consisted of reporting the whole project and documenting in the write-up which test cases they had developed, which of the test cases were automated, and what unit tests were added to the project repository. Additionally, the report included testing logs and bug reports from the manual system testing phase.

The project's problem description did not specify which testing methods or approaches should be used in the different parts as one objective of the project was that students select a suitable method and justify it in the test plan. The various testing techniques had been covered previously in the weekly exercises, where tools for unit testing and test coverage were also introduced.

The final exam consisted of two essay questions about the concepts presented in the lecture material. The exam was graded by the lecturer.

Descriptive course statistics are presented in Table II. In the end a total of 124 students worked actively on the course assignments. The average project grade was 3.4 and the average exam grade 3.8 (on a scale of 1-5 in passing grades). In addition to the course deliverables, a post-course survey was also conducted. Unfortunately, even though the survey got a 15% response rate in relation to the course population, the number of actual respondents remained low, and only 10 students gave written feedback.

Observations from the student testing projects are summarized in Table III. The themes listed were collected from the written feedback on the project given by the teaching assistant to the students. Overall, 37% of the comments in the feedback were positive, and 63% negative pointing out flaws in the implementation or clear misconceptions in the report.

IV. Discussion and Implications

In assessing the proposed curricula, it is necessary to highlight the importance of an early testing course. Some approaches to software testing education emphasize using big, real world projects as the basis to prepare graduates for work in the industry (for example [7], [8]). However, as students first go work in the industry as early as 1.5 years into their studies [31], this approach can be difficult to employ early on.

The ACM Curricula Recommendation for undergraduate software engineering programmes infuses software verification and validation into larger projects and courses [2]. We take a step back and ensure the students have familiarized themselves with testing, verification and validation on the dedicated testing course. Our approach to teaching the testing discipline complements the approach of the ACM Curricula Recommendation: Once the students have acquired the appropriate testing mindset the testing skills can be used in other software engineering and computer science projects.

Next, we summarize the findings from our course by addressing the individual research questions. To answer RQ1, what learning activities can we map to the high-level testing concepts, it is possible to complete activities on all testing levels. The activities can vary from black-box to white-box testing, system testing to unit testing, or something in between.

For RQ2, which actual testing techniques can be utilized, we created weekly assignments for students employing a variety of different techniques taken from relevant literacy, such as Swebok [12] or the ISO/IEC Software testing standard [13]. We incorporated a number of different techniques for deriving test cases on different levels, and additionally covering static testing methods and code reviews. In our course we used exploratory testing, equivalence partitioning, boundary value analysis, combinatorial methods, state transition testing, scenario testing, random testing and static testing in conjunction

TABLE I
THE WEEKLY ACTIVITIES, COVERED TOPICS, AND TESTING TECHNIQUES

Week	Development phase (V-Model)	Test level (V-Model)	Weekly covered topic(s)	Activities and testing techniques applied in them	Learning goals
1	Specification and requirement analysis	System testing	Introduction to testing. Objectives of testing	Black-box system testing. Exploratory testing. Boundary value analysis. Defect reporting.	Understand the objectives of testing work. Student is able to create (Black-box) test cases. Student understands the scope, and limitations of the black box methods.
2	Detail design	Unit testing	Testing levels. Unit testing	White-box testing. Test case reporting. Equivalence partitioning.	Understand the concept of unit / module test. Understand the difference between Black-box and White-box testing.
3	Architectural design	Integration testing	Integration testing	Combinatorial methods and the classification tree method. Test stubs.	Understand the infeasibility of ”testing everything.” Student is able to select a technique for deriving test cases. Student understands the scope, and limitations of the software testing in the real world software projects.
4	Specification and requirement analysis	System testing. Acceptance testing	System testing	State transition testing. Scenario testing. Random testing.	Understand the objectives of system-level testing. Student is able to select an appropriate testing technique for system testing. Student understands the scope, and limitations of the system-level testing methods.
5	Detail design	Unit testing	Test automation and tools	Implementing unit tests in code, using a unit testing framework	Student is able to use a programming framework / library to implement module tests. Student understands the scope, and limitations of the unit testing tools.
6	Architectural design	System testing	Testing processes, documentation and planning	Creating test plans. Code review and static testing methods. Test coverage analysis.	Student understands the purpose of static testing methods and code review practices.
7	Specification, architectural design, detail design	System, integration, unit testing	Visiting lecture from a software company	Course project: Plan, design, implement and document testing for a small software item.	Student is able to demonstrate their knowledge by applying the course's activities autonomously in the testing project. Student is able to explain how test process activities would relate to the whole software project.

TABLE II
DESCRIPTIVE STATISTICS FROM THE INTRODUCTORY TESTING COURSE

Students working in the course	124
Students with passing grade	117 (94.4%)
Group projects returned	43
Average project grade (median)	3.4 (4)
Average exam grade (median)	3.8 (4.5)
Respondents in the post-course survey (%)	19 (15%)
”The course implementation helped me to achieve the learning outcomes of the course” (1 - very poorly, 3 - neutral, 5 - very well)	3.33
”The teaching methods used on the course supported my learning” (1 - very poorly, 3 - neutral, 5 - very well)	3.39

TABLE III
THE MOST COMMON TYPES OF PROJECT FEEDBACK GIVEN TO STUDENTS BY THE TA

Unit tests did not check that the functions manipulated data correctly, only that their return value reported 'success'	49% (21)
Manual system testing was comprehensive	37% (16)
Objectives of the testing project were unclear or undefined	35% (15)
Unit tests were implemented without the use of a testing framework. The results of the tests were often presented in a way which required the tester to verify the results manually.	16% (7)
The tests seemed to only concentrate on crashing the program using only bad/sketchy inputs.	14% (6)
Unit tests were comprehensive, and tested the actual data manipulation	5% (2)
Amount of generally positive feedback comments given	37% (16)
Amount of generally negative feedback comments given	63% (27)

with unit testing, integration testing, system testing, writing test stubs and drivers, and analyzing test coverage.

In short, we introduced different fundamental testing methods and techniques which the students are likely to encounter in the real world software development to

give them some practical experience on the methods, as recommended in for example [17] or [19]. Our seven week course content is aligned with the V-model. This in our opinion makes it easier for students to grasp how the software engineering processes presented in theory relate to work with real projects, and bridges the gap between formal software engineering terminology and the real world.

Finally we discuss the course outcomes in the context of our original objective, creating learning activities for testing education early in the curriculum. The project as a demonstration of learning worked generally well. Especially in the first part of the project, exploratory system testing, the project reports presented testing comprehensively. Students were able to use the different techniques and approaches combined with intuition and creativity to sufficiently cover possible errors. Even if the reports did not directly name a particular method which was used in order to arrive to the test cases, it appears that the students were either formally or informally adequately familiar with these techniques.

However, there are still some issues which need addressing in our course. The other parts of the project proved to be more challenging for the students. For example in the second part nearly all groups successfully employed an automated system testing pipeline, but it was unclear what the students had set as the objectives for automated testing. In most cases the students had simply recorded test cases which they knew would fail, resulting in nearly all tests failing.

We can see that the problems revolve around defining the objectives and implementations of testing. In part 4 of the assignment, drafting a testing plan and test reports, the actual testing objectives were either shallow or neglected to a large degree. It seems that for the students the testing meant finding errors and making the program crash - and not ensuring that the program works, or conducting internal tests, not just checking the I/O interface. Based on these observations, it can be argued that the students mostly understood the concept of testing work itself as defined by Myers [32], but not the concept of quality assurance or quality control practices as defined by Kaner et al. [33]. However, testing levels were often referenced in the reports, meaning that students were able to place the testing work within the V-model, and based on activities, most teams also grasped the fundamental concepts of using testing frameworks as development tools.

On the other hand the course outcomes were, at least from our perspective, positive: Over eighty percent of the student groups successfully used a testing framework and automated tools in their project, with two thirds of the projects having defined objectives and purposes for the testing work, regardless of their quality. In addition, almost 95 percent of the students who started the course also gained a passing grade. As the learning outcomes were set beforehand in accordance with the constructive alignment theory, we can say that after completing the course most students demonstrated at least an adequate level of knowledge in the software testing field.

V. Conclusions

The objective of this study was to map high-level learning objectives into concrete testing activities, and to ground the testing activities firmly into the software engineering processes by using the V-Model as a starting point. In order to accomplish this objective we designed an introductory software testing course, and using the principles of constructive alignment, mapped learning goals to actual weekly activities and testing techniques.

We assessed the course curriculum by examining the outcomes of our seven week testing course. Student's practical assignments were used as demonstrations of learning. We observed from the projects that students were able to adopt the testing mindset and carry out comprehensive and systematic testing on the system testing level. On the other hand, this systematic approach to testing work was mainly carried out on the system level, while many projects had problems with unit tests, integration tests and reporting of the project.

The limitations of the study and the validity of the results warrant some discussion. The assignment reports written as group work are of course not the perfect instrument to measure the learning of individual students. However, as the assignment was split into multiple subsections with each section focusing on individual activities (exploratory testing, system testing, unit testing, test planning and documentation), it was easy to see which concepts the students excelled in or struggled with.

Additionally, the lecture material and the reference book [34] used in the course will most certainly have had a significant impact on the student's perception of testing as a whole. If the reference material is biased towards some topics or does not cover a concept well enough, it can be expected that the student reports follow the same shortcomings. In our case, the course material covers all the concepts expected in the assignments. On the other hand, due to the practical limitations some content discussing more advanced topics had to discussed only on a high level.

As future work, one promising approach would be incorporating a knowledge acquisition measurement algorithm such as ACT-R or BKT to assess the student performance and learning during the course in order to establish which course components require more refinement. Other prospective area of interest would be the integration of the advanced topics, and including a larger project work, which integrates both the software engineering and software testing methods into one capstone assignment.

References

[1] O. A. L. Lemos, F. F. Silveira, F. C. Ferrari, and A. Garcia, "The impact of software testing education on code reliability: An empirical assessment," *Journal of Systems and Software*, vol. 137, pp. 497–511, 2018.

[2] The Joint Task Force on Computing Curricula, "Curriculum guidelines for undergraduate degree programs in software engineering," New York, NY, USA, Tech. Rep., 2015.

[3] P. H. D. Valle, E. F. Barbosa, and J. C. Maldonado, "Cs curricula of the most relevant universities in brazil and abroad: Perspective of software testing education," in *Computers in Education (SIIE), 2015 International Symposium on*. IEEE, 2015, pp. 62–68.

[4] J. Biggs, "Enhancing teaching through constructive alignment," *Higher education*, vol. 32, no. 3, pp. 347–364, 1996.

[5] ——, "Constructive alignment in university teaching," *HERDSA Review of higher education*, vol. 1, no. 1, pp. 5–22, 2014.

[6] T. Hynninen, J. Kasurinen, A. Knutas, and O. Taipale, "Guidelines for software testing education objectives from industry practices with a constructive alignment approach," in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 2018, pp. 278–283.

[7] D. E. Krutz, S. A. Malachowsky, and T. Reichlmayr, "Using a real world project in a software testing course," in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 49–54.

[8] V. Garousi, "Incorporating real-world industrial testing projects in software testing courses: Opportunities, challenges, and lessons learned," in *Software Engineering Education and Training (CSEE&T), 2011 24th IEEE-CS Conference on*. IEEE, 2011, pp. 396–400.

[9] T. Hynninen, J. Kasurinen, A. Knutas, and O. Taipale, "Software testing: Survey of the industry practices," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018, pp. 1449–1454.

[10] P. Rook, "Controlling software projects," *Software Engineering Journal*, vol. 1, no. 1, pp. 7–16, 1986.

[11] S. Mathur and S. Malik, "Advancements in the v-model," *International Journal of Computer Applications*, vol. 1, no. 12, 2010.

[12] P. Bourque, R. E. Fairley *et al.*, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[13] "Software and systems engineering – software testing – part 4: Test techniques," International Organization for Standardization, Geneva, CH, Standard, 2015.

[14] D. Mishra, S. Ostrovska, and T. Hacaloglu, "Exploring and expanding students' success in software testing," *Information Technology & People*, vol. 30, no. 4, pp. 927–945, 2017.

[15] M. Gagné and E. L. Deci, "Self-determination theory and work motivation," *Journal of Organizational behavior*, vol. 26, no. 4, pp. 331–362, 2005.

[16] S. Šošić, O. Ristić, K. Mitrović, and D. Milošević, "Software testing course in it undergraduate education in serbia," *Information Technology*, vol. 4, no. 6, p. 8, 2018.

[17] Z. Bin and Z. Shiming, "Curriculum reform and practice of software testing," in *International Conference on Education Technology and Information System (ICETIS 2013)*, 2013, pp. 841–844.

[18] J. Kasurinen, "Experiences from a web-based course in software testing and quality assurance," *International Journal of Computer Applications*, vol. 166, no. 2, 2017.

[19] D. Mishra, T. Hacaloglu, and A. Mishra, "Teaching software verification and validation course: A case study," *International Journal of Engineering Education*, vol. 30, pp. 1476–1485, 2014.

[20] G. Lopez, F. Cocozza, A. Martinez, and M. Jenkins, "Design and implementation of a software testing training course," in *122nd ASEE Annual Conference & Exposition*, 2015.

[21] J. van Eijck, V. Zaytsev *et al.*, "Flipped graduate classroom in a haskell-based software testing course," in *Pre-proceedings of the Third International Workshop on Trends in Functional Programming in Education (TFPIE 2014)*, 2014.

[22] G. Lopez and A. Martinez, "Use of microsoft testing tools to teach software testing: An experience re-port," in *Proceedings of the American Society for Engineering Education Annual Conference and Exposition*, 2014.

[23] G. Fraser, A. Gambi, and J. M. Rojas, "A preliminary report on gamifying a software testing course with the code defenders testing game," in *Proceedings of the 3rd European Conference of Software Engineering Education*. ACM, 2018, pp. 50–54.

[24] P. H. D. Valle, A. M. Toda, E. F. Barbosa, and J. C. Maldonado, "Educational games: A contribution to software testing education," in *Frontiers in Education Conference (FIE)*. IEEE, 2017, pp. 1–8.

[25] Y. Fu and P. Clarke, "Gamification based cyber enabled learning environment of software testing," *submitted to the 123rd American Society for Engineering Education (ASEE)-Software Engineering Constituent*, 2016.

[26] A. Soska, J. Mottok, and C. Wolff, "An experimental card game for software testing: Development, design and evaluation of a physical card game to deepen the knowledge of students in academic software testing education," in *Global Engineering Education Conference (EDUCON), 2016 IEEE*. IEEE, 2016, pp. 576–584.

[27] K. Seaborn and D. I. Fels, "Gamification in theory and action: A survey," *International Journal of human-computer studies*, vol. 74, pp. 14–31, 2015.

[28] Z. Chen, A. Memon, and B. Luo, "Combining research and education of software testing: a preliminary study," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM, 2014, pp. 1179–1180.

[29] P. Lauvås Jr and A. Arcuri, "Recent trends in software testing education: A systematic literature review," in *UDIT (The Norwegian Conference on Didactics in IT education)*, 2018.

[30] L. P. Scatalon, J. C. Carver, R. E. Garcia, and E. F. Barbosa, "Software testing in introductory programming courses: A systematic mapping study," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: ACM, 2019, pp. 421–427. [Online]. Available: http://doi.acm.org/10.1145/3287324.3287384

[31] The Joint Task Force on Computing Curricula, "Curriculum guidelines for baccalaureate degree programs in information technology," New York, NY, USA, Tech. Rep., 2017.

[32] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The art of software testing*, 2nd ed. John Wiley & Sons, 2004.

[33] C. Kaner, J. Bach, and B. Pettichord, *Lessons learned in software testing: a context-driven approach*. Wiley, 2002.

[34] J. P. Kasurinen, *Ohjelmistotestauksen käsikirja [Handbook of Software Testing]*. Docendo, 2013.