Recent Trends in Software Testing Education: A Systematic Literature Review

Per Lauvås jr and Andrea Arcuri
Westerdals Oslo ACT, Norway,
and University of Luxembourg, Luxembourg

Abstract

Testing is a critical aspect of software development. Far too often software is released with critical faults. However, testing is often considered tedious and boring. Unfortunately, many graduates might join the work force without having had any education in software testing, which exacerbates the problem even further. Therefore, teaching software testing as part of a university degree in software engineering and is very important. But it is an open challenge how to teach software testing in an effective way that can successfully motivate students. In this paper, we have carried out a systematic literature review on the topic of teaching software testing. We analysed and reviewed 30 papers that were published between 2013 and 2017. The review points out to a few different trends, like the use of *gamification* to make the teaching of software testing less tedious.

1 Introduction

In 2017, 606 documented[1] software fails did impact half of the world's population (3.7 billion people), costing $1.7 trillion in assets. And that is only a small sample of all different software faults that happen regularly on a daily base in all different kinds of existing software systems.

Software testing is the most common technique used to reduce the amount of faults during software development before the software is released to customers. Unfortunately, software testing is often considered tedious and boring. Furthermore, many graduates joining the work force as software developers do have a background in computer science, where proper software engineering foundations like software testing might be lacking or covered only superficially.

Teaching software testing in a university poses many challenges, as lecturers need to find ways to overcome the tediousness of testing, to motivate students, and to show them why software testing is such a critical and fundamental aspect of software development.

In this paper, we have carried out a systematic review of recent (2013-2017) work on *software testing education*. Systematic literature reviews are a method to synthesise existing research regarding specific research questions in a systematic,

This paper was presented at the NIK-2018 conference; see http://www.nik.no/.

[1] https://www.tricentis.com/software-fail-watch/

comprehensive, and unbiased way [18]. Our systematic literature review is based on a selection of 30 papers.

The main contributions of this paper are:

- A systematic literature review of software testing education based on 30 papers published between 2013 and 2017.
- A categorisation and analysis of current research, where for example experience reports on pedagogical approaches are the most common kind of research article.
- A meta-analysis of learned lessons on what to do to improve the teaching of software testing.

The paper is organised as follows. Section 2 describes how the 30 papers for the systematic literature review were selected. Section 3 poses three research questions, which are answered based on the content of the systematic literature review. Finally, Section 4 concludes the paper.

2 Paper Selection

To select our sample of papers to study current trends in software testing education, we made a query on Google Scholar. We selected papers published during the span of five years, from 2013 to 2017. The papers must contain the keywords *software* and *testing* in either their title or abstract. Furthermore, each paper in its title/abstract must contain at least one of the following keywords: *education*, *student*, *course*, *learn*, *curriculum* and *teaching*.

Such query was carried out on the 30th of January 2018, and resulted in 57 hits. We were unable to retrieve two book chapters and one article within these hits. The remaining selection was independently reviewed by the two authors of this paper to exclude:

- out of topic papers.
- papers with title/abstract in English, but with the main text in a different language (e.g., Chinese and Portuguese).
- BSc and MSc theses.
- papers presenting a tutorial done in a conference.

After such filtering phase, the resulting sample was reduced to 30 papers.

3 Paper Analysis

In this paper, we aim at answering the following research questions:

RQ1: What topics are addressed in the recent literature on software testing education?

RQ2: What kind of contributions are present in papers published on software testing education?

RQ3: What insight can be provided to lecturers that want to introduce software testing as part of their teaching?

Table 1: Topic categorisation for the analysed 30 papers.

Topic	# Papers	References
Pedagogical Approaches	12	[9, 27, 5, 4, 1, 19, 30, 8, 10, 14, 7, 16]
Tooling	7	[24, 11, 17, 15, 32, 31, 13]
Gamification	4	[29, 12, 26, 25]
Benefits and Impact	2	[21, 20]
Outside Higher Education	2	[23, 22]
Course Evaluation	2	[2, 3]
Trends	1	[28]

RQ1: What topics are addressed in the recent literature on software testing education?

To answer **RQ1**, each author read through the paper selection, and independently proposed his own categorisation for each paper. The two categorisations were then discussed, and merged into a single one, resulting in the following:

***Pedagogical Approaches*:** *how* to deliver a course on software testing.

***Tooling*:** *tools* which can be used to facilitate the delivery of software testing courses.

***Gamification*:** introducing game elements when teaching software testing to create engagement and stimulate motivation for learning.

***Benefits and Impact*:** expected benefits of teaching software testing concepts.

***Outside Higher Education*:** teaching software testing concepts outside higher education.

***Course Evaluation*:** how to evaluate software testing courses.

***Trends*:** status on teaching software testing within higher education.

Table 1 shows the topic categorisation of the analysed papers. We see that *Pedagogical Approaches* is clearly the largest category with 12 out of 30 papers. There are also a significant amount of papers describing *Tooling* (7) and *Gamification* (4). *Gamification* and *Pedagogical Approaches* are clearly related. They both concern *how* to deliver a course. In fact, gamification can be regarded as a pedagogical approach [6]. The three largest categories may therefore tell us *how* to deliver courses on software testing and *what tools* could help us in the process. These three categories constitute 77% of the papers in our review.

Some of the pedagogical papers have a foundation from pedagogical theory. Within a blended classroom setting, educators may use an adaptive approach to teaching grounded in Vygotsky's zone of proximal development [4]. Teachers may aim for an increased depth of understanding according to Blooms Taxonomy levels by spending less time on lectures [5]. There are multiple good reasons to flip the classroom [30]. Two papers describe how courses may be designed according to the educational philosophy CDIO (Conceive - Design - Implement - Operate)[8, 16].

Other papers within the pedagogical category do not have the same strong *theoretical* foundation within pedagogy, but they do address *how* to teach software testing. With so many openly accessible open source projects, educators should use these real-world projects in their teaching [19, 10]. As Data Structures is a well known topic within Computer Science and Software Engineering studies, why not use the topic to teach software testing [9]? Multiple papers recommend to place students in teams. They can also be placed in pairs to perform pair testing [1]. A specific subtopic, metamorphic testing, has the potential to really engage students in creative classroom activities [27]. Using exploratory teaching, students learn via negative experiences in practical situations [14].

Multiple existing tools may be used to support the teaching of software testing. Educators can use a collaborative virtual environment (CVE) [24] or a cyberlearning environment, such as WRESTT-CyLE [11, 31]. Teachers may develop web-based courses using multiple learning tools [17]. Some describe and evaluate tools where students may upload program code, including tests, to get an evaluation automatically [15, 13]. With HoliCoW [32], student code can be manipulated (mutated) so that instructors can run regression tests and evaluate the robustness of the original software.

Introducing game elements can make software testing education more enjoyable. Such elements can be digital games. In Code Defenders [12], the students learn and practice testing concepts as they play the game. Students playing the Testing Game [29] will more specifically learn functional, structural and mutation testing. The games do not have to be digital. Using a custom, physical card game can also strengthen the learning of software testing [26]. Gamifying software testing education does not have to include actual games. With HALO - ''Highly Addictive sociaLly Optimized Software Engineering'' [25] software testing techniques are disguised as quests. The student can complete quests to receive rewards in the form of achievements, titles and experience points.

The remaining categories have one or two articles in our review. Both papers within *Benefits and Impact* describe how software testing knowledge leads to more reliable code [21, 20]. Two papers on teaching software testing outside higher education were found. One involves K12 education [23], and one describes a training course for software developers [22]. Two papers suggest how teachers may evaluate the delivery of a software testing course through direct [2] and indirect [3] assessment. A single paper was found in the category *Trends*. In this paper we find a description on how software testing education has been approached in Brazil and abroad [28]. The authors identify a lack of software testing on the curricula. Few lectures are assigned to the teaching of software testing. Furthermore, software testing is seldom integrated with other possible disciplines. 11 different initiatives to support software testing education, found within existing research, are described.

RQ1*: There are three main topics in recent research litterature on software testing in education:* ***Pedagogical Approaches****,* ***Tooling*** *and* ***Gamification****. Other less described topics include* ***Benefits and Impact, Outside Higher Education, Course Evaluation*** *and* ***Trends****.*

RQ2: What kind of contributions are present in papers published on software testing education?

To answer **RQ2**, we defined the following types of contribution given by the papers:

Table 2: Types of contribution for the analysed 30 papers. The papers are grouped by topic: *Pedagogical Approaches* (P), *Tooling* (To), *Benefits and Impact* (I), *Outside Higher Education* (O), *Course Evaluation* (E) and *Trends* (Tr).

Topic	Reference	Experience	Experiment	Survey	Interview	Review
P	[27]	✓				
	[8]	✓				
	[10]	✓				
	[14]	✓				
	[4]	✓				
	[19]	✓		✓		
	[5]	✓			✓	
	[1]		✓			
	[9]					
	[30]					
	[7]					
	[16]					
To	[32]	✓				
	[31]	✓				
	[13]	✓				
	[24]		✓			
	[11]	✓		✓		
	[17]	✓		✓		
	[15]	✓		✓		
G	[25]	✓				
	[29]			✓		
	[26]			✓		
	[12]					
I	[20]		✓			
	[21]		✓	✓		
O	[22]	✓		✓	✓	
	[23]					
E	[2]	✓				
	[3]			✓		
Tr	[28]					✓
Total	30	16	4	9	2	1

***Experience*:** reports on anecdotal personal experience in teaching software testing. Papers may include some statistical analyses.

***Survey*:** questionnaires to students and/or professionals.

***Interview*:** interview with students, instructors or other persons involved in a course delivery.

***Review*:** literature review or review of current practices.

***Experiment*:** controlled empirical study involving human subjects (typically students).

A paper could provide none, one or more kinds of contribution. Papers that provide none of these contributions are usually short *position papers* discussing for example possible plans for future work, or ideas/tools not evaluated yet.

Table 2 shows our categorisation. *Experience* is the most common type of contribution within our findings. These papers are stories from software testing course deliveries or structural changes within software testing education. With these stories, other educators may learn from the experience and pick up good ideas to use in their own teaching. ''*This paper presents some reflections on a number of classroom experiences of five academics, working in different teaching contexts, all of whom share an expertise in the software quality assurance area of software testing(..)*'' [27]. Perceived increased learning outcome and happier students are often described results within these papers, but they are not always backed up by scientific evidence. ''*We obtained good teaching results and reached the goal of modern engineering education reform*'' [8]. Some articles within the category did include statistical data to support the reported experiences [4, 5, 31, 13, 2, 11, 15, 17], but not within a controlled experiment: ''*Mixed model repeated measures ANOVAs were conducted for both statement and branch coverage for unit, subsystem, and system testing to compare the effect of availability and knowledge of the use of code coverage tools (IV) on percentage of code coverage (DV) across the Fall 2012 and Fall 2013 semesters*'' [11].

Some of the experience reports did also include *Survey*, which is our second largest type of contribution. Data from a course delivery and reflections from the educators can be combined with a survey with the students or trainees [11, 17, 15, 22, 19].

When students are asked to fill out a questionnaire, they normally do so as a part of a course evaluation. But we also find examples of surveys completed at the start of a course [17]. There is also a survey with professors that teach introductory programming courses to evaluate their level of software testing knowledge [21]. We find different topics within the different student surveys:

- The perception of the usefulness of testing tutorials within a tool [11].
- Usefulness of the collaborative features in a tool [11].
- Course-start survey [17].
- Course-end survey [17, 3, 22].
- Survey on reflective thinking [15].
- Evaluation of a testing game [29, 26].
- Achievement test [22].

Not all details about practicalities are reveiled about the different surveys, but the Likert scale is mentioned multiple times[29, 26, 22].

We found *Experiment* to be somewhat well represented within our findings. Two of the papers [21, 20] (from the same research project) investigate the possible impact of software testing knowledge on the production of reliable code. They found that code written after exposition to software testing knowledge was 20% more reliable. The concept of pair programming can be applied to students who write test cases [1]. Students write better test cases when they write them in pairs. The test

cases will detect more errors and have better code coverage. Collaborative Virtual Environments (CVEs) can be used for software testing courses [24]. Results suggest that students working virtually through a CVE can obtain equivalent results as those students who do not.

Interview (2) and *Review* (1) were less represented within our findings. The interviews were with students and a manager. The students [5] were interviewed in groups "*to learn the students' opinion, attitude and behavior towards their learning*". The manager [22] was interviewed three months after the software developers had finished their software testing course. This was done to investigate "*if there had been any changes to the processes or activities performed by this unit(...)*". The single paper in the *Review* category provide an overview of CS curricula in and outside Brazilian universities. It also includes a systematic mapping to describe initiatives within existing research to support software testing education.

RQ2*: Recent papers on software testing in education include many experience reports. Another major type of contribution is survey. There are also papers providing controlled experiments. Interviews and reviews also appear, but less frequently. Position papers are quite common.*

RQ3: What insight can be provided to lecturers that want to introduce software testing as part of their teaching?

To answer **RQ3**, we analysed which types of suggestions and shared learned lessons are present in the selected 30 papers. Here we discuss the one that most often appeared among these papers.

***Motivate Students*:** Software testing may be perceived as boring by the students. What can we as educators do to address that?

Multiple papers state that software testing is not well accepted among students. Many students find it dull and not very interesting:

- "*Students often view testing as a boring and unnecessary task, and education is usually focused on building software, not ensuring its quality.*" [19]
- "*(...) standard testing techniques are often perceived as boring and difficult when compared to creative programming and design activities, which dominate education.*" [12]
- "*(...) students remain averse to software testing as there is low student interest in software testing.*" [25]
- "*(...) many students feel unmotivated to learn contents related to software testing.*" [29]
- "*A major challenge was to dispel the stigma of software testing and maintenance as an unholy alliance of arguably the two least favored tasks within the realm of the software life cycle*" [4]

Learning is not easy when motivation is low. Increasing the motivation for software testing could therefore be a natural aim when teaching the topic. Multiple papers within our findings discuss motivation.

All papers within the *Gamification* category do, naturally enough, involve motivation. The game elements are there to increase motivation in the first place. The papers with empirical data do find that introducing games or game-like elements increase motivation or the level of excitement: "*Through the feasibility study performed, we observed that the Testing Game have good quality regarding motivation (...)*" [29]. "*(...) brings benefits in the form of group learning and enjoyable learning*" [26]. "*Finally, a fun, informal, and collaborative classroom was used to prevent creating stereotypes that CS is boring*" [25].

But it is possible to increase motivation without using gamification. Some recommend to use real-world projects [11, 19, 10] and industry testing tools [11, 1, 19, 10]. The real-world projects can be selected within open-source projects [19, 10]. By doing so, the students will see that testing skill and knowledge can solve real problems on existing projects. And if the students are placed in teams, they will be in a typical real-world setting, and they may experience the benefits of collaborative learning [11]. The students may also be the active part when the projects are to be selected to further increase the motivation [19]. The use of code coverage tools motivate the students to improve their test suites [11].

Specific topics within software testing can also stimulate motivation. Mutation testing involves making small changes in a computer program. A mutated version is called a mutant. When good test cases exist for a program, they will manage to *kill* the mutant. A live mutant can guide further test generation efforts. The robustness of the program can be measured by evaluating how many mutants the written test cases will detect. Multiple papers mention mutation testing specifically [29, 12, 21, 19]. Killing mutants can be a game in itself [12], but even without gamification, the introduction of inserted mutants can engage the students. They can see how important test cases are, and they can see how the program robustness is increased when a test suite is expanded and refined. Some describe a tool "*that simulates production environments through forced logical error injections into student projects.*" [32] without referring to it as mutation testing.

RQ3*: Software testing education does not have to be dull and boring. We, as educators, can use real-world projects, industry tools, mutation testing and gamification to create enthusiasm and motivation among our students.*

4 Conclusion

In this paper we have carried out a systematic literature review on software testing education. The review was based on a selection of 30 papers, published between 2013 and 2017.

The papers give an impression that software testing is an important topic, but does not receive enough attention in Computer Science and Software Engineering education. When students do approach software testing concepts, they find it to be dull and of little interest. But these issues can be addressed. This review shows that many aspects within software testing education are discussed in recent literature. The papers offer many forms of contributions. The most common suggestions involve getting the students motivated for learning, using different kinds of techniques and tools to achieve it.

Future work will study how the trends change through time, e.g., by looking at older articles from before 2013. There are also many research questions that, for

reason of space, were not addressed in this paper, like for example: In which venues are this type of research papers published? Are there conflicting reccomandations in different experience reports? What is the empirical or anecdotal evidence to reccomand teaching software testing in its own dedicated courses instead of being part of more general programming ones (or vice-versa)? Etc.

Acknowledgment

This work is supported by the Research Council of Norway (project on Evolutionary Enterprise Testing, 274385), and by the National Research Fund, Luxembourg (FNR/P10/03).

References

[1] I. Alazzam and M. Akour. Improving software testing course experience with pair testing pattern. *International Journal of Teaching and Case Studies*, 6(3):244--250, 2015.

[2] A. Alelaiwi. Direct assessment methodology for a software testing course. *Life Science Journal*, 11(6s), 2014.

[3] A. Alelaiwi. Indirect assessment of student learning in a software testing course. *Life Science Journal*, 11(6s), 2014.

[4] M. Allison and S. F. Joo. An adaptive delivery strategy for teaching software testing and maintenance. In *Computer Science & Education (ICCSE), 2015 10th International Conference on*, pages 237--242. IEEE, 2015.

[5] R. Aziz. Improving high order thinking skills in software testing course. *International Journal of Computer Science and Information Security*, 14(8):966, 2016.

[6] J. Banfield and B. Wilkerson. Increasing Student Intrinsic Motivation And Self-Efficacy Through Gamification Pedagogy. *Contemporary Issues In Education Research - Fourth Quarter*, 7(4), 2014.

[7] Z. Bin and Z. Shiming. Curriculum reform and practice of software testing. In *International Conference on Education Technology and Information System (ICETIS 2013)*, pages 841--844, 2013.

[8] Z. Bin and Z. Shiming. Experiment teaching reform for software testing course based on cdio. In *Computer Science & Education (ICCSE), 2014 9th International Conference on*, pages 488--491. IEEE, 2014.

[9] I. A. Buckley and W. S. Buckley. Teaching software testing using data structures. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 8(4):1--4, 2017.

[10] 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*, pages 1179--1180. ACM, 2014.

[11] P. J. Clarke, D. L. Davis, R. Chang-Lau, and T. M. King. Impact of using tools in an undergraduate software testing course supported by wrestt. *ACM Transactions on Computing Education (TOCE)*, 17(4):18, 2017.

[12] B. S. Clegg, J. M. Rojas, and G. Fraser. Teaching software testing concepts using a mutation testing game. In *Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), 2017 IEEE/ACM 39th International Conference on*, pages 33--36. IEEE, 2017.

[13] D. M. de Souza, B. H. Oliveira, J. C. Maldonado, S. R. Souza, and E. F. Barbosa. Towards the use of an automatic assessment system in the teaching of software testing. In *Frontiers in Education Conference (FIE), 2014 IEEE*, pages 1--8. IEEE, 2014.

[14] B. Doersam. Teaching of software testing and quality issues. In *EDULEARN14 Proceedings*, pages 80--88. IATED, 2014.

[15] E. Fridge and S. Bagui. Impact of automated software testing tools on reflective thinking and student performance in introductory computer science programming classes. *International Journal of Information and Communication Technology Education (IJICTE)*, 12(1):22--37, 2016.

[16] S. Jia and C. Yang. Teaching software testing based on cdio. *World Transactions on Engineering and Technology Education*, 11(4), 2013.

[17] J. Kasurinen. Experiences from a web-based course in software testing and quality assurance. *International Journal of Computer Applications*, 166(2), 2017.

[18] K. Khan, R. Kunz, J. Kleijnen, and G. Antes. *Systematic reviews to support evidence-based medicine*. Crc Press, 2011.

[19] 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*, pages 49--54. ACM, 2014.

[20] O. A. L. Lemos, F. C. Ferrari, F. F. Silveira, and A. Garcia. Experience report: Can software testing education lead to more reliable code? In *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*, pages 359--369. IEEE, 2015.

[21] 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*, 2017.

[22] 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.

[23] T. Michaeli and R. Romeike. Addressing teaching practices regarding software quality: Testing and debugging in the classroom. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, pages 105--106. ACM, 2017.

[24] J. P. U. Pech, R. A. A. Vera, and O. S. Gómez. Software testing education through a collaborative virtual approach. In *International Conference on Software Process Improvement*, pages 231--240. Springer, 2017.

[25] S. Sheth, J. Bell, and G. Kaiser. A gameful approach to teaching software design and software testing - assignments and quests. Technical report, Technical Report cucs-030-13, Dept. of Computer Science, Columbia University, 2013. http://mice. cs. columbia. edu/getTechreport. php, 2013.

[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*, pages 576--584. IEEE, 2016.

[27] D. Towey, T. Y. Chen, F.-C. Kuo, H. Liu, and Z. Q. Zhou. Metamorphic testing: A new student engagement approach for a new software testing paradigm. In *Teaching, Assessment, and Learning for Engineering (TALE), 2016 IEEE International Conference on*, pages 218--225. IEEE, 2016.

[28] 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*, pages 62--68. IEEE, 2015.

[29] P. H. D. Valle, A. M. Toda, E. F. Barbosa, and J. C. Maldonado. Educational games: A contribution to software testing education. *XLVII Annual Frontiers in Education (FIE). IEEE*, 2017.

[30] 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.

[31] P. Yujian Fu and P. J. Clarke. Integrating software testing to cs curriculum using wrestt-cyle. 2015.

[32] P. Zhang, J. White, and D. C. Schmidt. Holicow: automatically breaking team-based software projects to motivate student testing. In *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*, pages 436--439. IEEE, 2016.