Using an Incremental Testing Strategy to Improve Students' Perception of Software Quality

Italo Santos[1], **Allan Mori**[1], **Simone R. S. Souza**[1]

[1]University of São Paulo (ICMC/USP)
P.O. Box 1212 – São Carlos – SP – Brazil

italo.santos@alumni.usp.br, allanmori@usp.br, srocio@icmc.usp.br

Abstract. *Teaching software testing should include a broad view of the main techniques, criteria, and tools. In general, students have few opportunities to test their code-projects suitably during the undergraduate course and, therefore, teaching software testing in practice is crucial to students recognize the advantages and limitations of different testing techniques. This paper reports the experience of teaching software testing in practice, with students applying an incremental testing strategy to validate their software projects. Students selected a software project developed during their undergraduate and an incremental testing strategy, including testing criteria learned in the class. The students should choose the testing techniques, apply them and write a report with the results and perceptions. Through this experience, it was possible to show to the students, in practice, the importance of combining more than one technique during the software testing activity.*

1. INTRODUCTION

Software testing is an important activity to assure the reliability of software. According to [Offutt et al. 2011], much has changed in the way we develop software in the last two decades, but unfortunately, very little has changed in how we teach software development. Software testing is a wide field and there is more material available than what is possible to cover in a one-semester course at any level. It is best if an undergraduate course focuses on the basics of software testing in the areas of terminology, test generation, test assessment, and test automation [Wong et al. 2011]. In most undergraduate programs, students get little practical training in how to test their code and often have poor skills (and even poorer expectations) in this area [Edwards 2003].

Teaching software testing in a software engineering course is usually a challenge for the following reasons [Clarke et al. 2014]: (i) there are usually too many other topics to cover, (ii) many instructors are not familiar with software testing techniques, and (iii) students and instructors are not familiar with the tools available to support software testing. Some universities have worried about it and have included a software testing course to a degree in computing so that test concepts can be taught in-depth and students can have an extensive view of tools and software techniques. Teaching software testing poses many challenges, as lecturers need to find alternatives to motivate students and to show them why software testing is such a critical and fundamental aspect of software development [Lauvås Jr and Arcuri 2018]. There is a need to pay more attention to testing and to improve the effectiveness of education in the testing field [Bijlsma et al. 2020].

Andrade et al. [Andrade et al. 2019] report the teaching process in two undergraduate courses in two different contexts: (i) computer science students, who can dedicate more time to study during the day, and (ii) information systems students, who attend evening classes. To engage and motivate the students in the context of software testing learning, real industry problems were used in the classroom to adopt the Problem-Based Learning (PBL) approach and results show that students positively perceived the inclusion of an active methodology in the learning process. [Smith et al. 2012] used peer review to teach software testing. The peer-review process motivated students to frame testing as a fun and competitive activity, allowing students to learn from each other. The study evaluated students' performance through a survey summarizing student attitudes taken throughout the course.

A gamification strategy was proposed by Fraser et al. [Fraser et al. 2019], to integrate Code Defenders game as a semester-long activity of an undergraduate and graduate course on software testing. Students compete over code under test by either introducing faults ("attacking") or by writing tests ("defending") to reveal these faults. Results showed that the integration of Code Defenders was well received by students and led them to thoroughly testing practice.

Lemos et al. [Lemos et al. 2015] investigated the impact of software testing knowledge on the production of reliable code. They conducted a controlled experiment involving 28 senior-level computer science students, results showed that code delivered after the exposition to software testing knowledge is, on average, 20% more reliable, which indicates that knowledge in software testing can make developers produce more reliable code with no additional overhead in terms of program size.

In this paper, we report our experience of combining different testing techniques and apply them to students' code-projects to improve the learning in a software testing course. This could help professors who teach software testing to decide whether it is worth adopting the combination of testing techniques to help students' learn concepts of software testing. This course has 12 credit-hours and 180 hours of class time in a 15-week semester. It is offered by the Computer Science and Computational Mathematics Program at the University of Sao Paulo. Our main motivation was to make students aware of the importance of testing activity. We used common knowledge (e.g., complementary use of testing techniques) to motivate students about the importance of testing their code. Other courses try to innovate the way testing can be taught [Smith et al. 2012, Fraser et al. 2019, Paschoal et al. 2019]. We took a step to innovate by using a simple strategy and work more in the students' perceptions of the need for testing. This paper provides the following contributions: (i) description of how the software testing course was conducted through the design of a final project that required students to apply the theoretical concepts learned, and (ii) a report on students' experience of using the combination of testing techniques and how it helps them to learn testing concepts.

2. COURSE CONTEXT

Course Goals and Contents: The main goal of the course is to provide an overview of software verification, validation, and testing with an emphasis on software testing strategies, techniques and criteria, and tools that can be applied in software development. The course contents are divided into thirteen classes. Each class has a practical activity (fix-

ation exercises) associated with it. Two more sessions were employed for students to present their project results. Details about the course content and this study are available in a repository [1].

Students: This course has a mixed attendance, with undergraduate and graduate students. A total of 40 students were enrolled in this course throughout the semester. The majority of undergraduate students pursue a bachelor's degree in computer engineering at senior year; students from computer science and information systems can follow this course. The graduate students are predominantly Ph.D students, the research field of these students are from different backgrounds (e.g., human-computer interaction, artificial intelligence, software architecture, software engineering).

Assignments: Practical activities were conducted in each class addressing the subject taught. After the execution of the activities, some discussions were made in class to solve the doubts, and to review the answers of the students.

Assessment: Students were assessed by conducting two practical assignments and a final exam. The first practical work consisted of developing test cases and apply functional criteria (e.g., equivalence class partitioning and boundary value analysis) to create a set of test case, then students should use the Junit[2] to execute the test cases created and make a report with information about the expected/generated outputs. In the second practical assignment, students should select a code-project developed during their undergraduate to apply the testing techniques learned in the course. The exam applied at the end of the term involved all the covered concepts in the course.

3. RESEARCH METHOD

Students were assigned to work in pairs and perform a final project that was divided into two main parts. The first part consists of choosing a software project, which should be authored by the students themselves or some project available on GitHub[3]. The team must write a report that specifies project details, such as: when it was developed, development context (e.g., discipline activity), project functionalities and include information regarding the code metrics (e.g., cyclomatic complexity, lines of code, number of methods, number of classes). The chosen project was analyzed by teacher and the teacher assistant to decide if project are suitable for the activity.

The second part of the project consists of the incremental application of three testing techniques *(functional - structural - fault-based)*. Regarding the application of the functional testing, students must apply the criteria Equivalence Class Partitioning and Boundary Value Analysis in the chosen software. This application must be made based on the software specification and the results should be reported with the information of the two testing criteria used. Then, students should add new test cases to improve the coverage of the structural testing criteria. Depending on the complexity of the code-project, students could choose only some classes or modules to apply structural testing. The goal is to generate test cases to achieve 100% coverage for structural testing criteria using

[1] https://zenodo.org/record/4569149#.YDwqiWhKjIU
[2] https://junit.org/junit5/
[3] https://github.com/

EclEmma[4] to measure the code coverage. After that, students should use the PIT[5] mutation testing tool to execute the previously generated test cases, i.e. test cases appropriate to the functional and structural test. The goal is to verify the quality of generated test suites in relation to mutation testing.

After the design of the final project, students present their results describing the learned lessons, difficulties and limitations. Furthermore, a survey was applied to collect students' opinions on the course and the project. The survey received 40 responses and it was anonymously fulfilled to allow students to feel free to express their opinions.

The data gathered through the opinion survey were quantitatively and qualitatively analyzed. In the quantitative analysis, we used descriptive statistics to represent and describe the characterization data. In the qualitative analysis, we used Grounded Theory methods [Strauss and Corbin 2007], which aims to create a theory from the data collected. The coding process stops when no new data adds new knowledge to the categorization analysis process. Although Grounded Theory proposes to build a theory, Strauss and Corbin illustrate that the researcher can use only a few steps to achieve the research goal [Strauss and Corbin 2007]. In this study, only open and axial coding was used to identify emerging topics, difficulties, strengths, and weaknesses. This analysis involves creativity, experience, and researchers' bias. To avoid the bias effect, the coding process was generated by the first author and validated by the other two, we also held meetings to discuss the data to ensure that all authors agreed with the analysis.

4. RESULTS AND DISCUSSION

In this section, we present the results and discuss the main findings obtained from students' opinions survey.

4.1. Course Details

Q1 - Did you know any concept of software testing before the course?

As previously mentioned, the course has a mixed attendance, with undergraduate and graduate students. Students' backgrounds are composed of undergraduate students from senior year of Computer Engineering, Computing Science, and Information systems courses. The graduate students are Ph.D. students and master students.

The results indicated that 77% already had some knowledge, and 23% had no knowledge and decided to enroll in the course to know more about software testing. We investigated what kind of knowledge the students who answered this question positively had. We noticed that they only had a basic notion about software testing (from software engineering course) or have seen the subject in internship experiences.

Q2 - Has your opinion of the importance of testing activity changed since the beginning of the course?

Based on the analyzed data, we observed that the perception of the software testing importance changed for the students. For the students, the main aspects of software testing activity are related to industry practices (e.g., tools, techniques, difficulties), the need to

[4]https://www.eclemma.org/

[5]https://pitest.org/

improve software quality and the importance of considering testing throughout software development practices.

Several answers highlighted that the software testing skills is of great importance for the quality of the developed software. Some quotes from students: "*In the industry, I felt in the skin how important testing is. Then the course just confirmed what I have seen in the real world*", "*My view completely changed, given the importance of this activity in software development*" and "*After doing the practical activities, I realized this is an essential activity to ensure the minimum software quality. Also, I believe that having test concepts in mind helps developers during the software construction and maintenance*".

Q3 - Check whether you agree or disagree with the statements presented (Course in general):

Students were asked to indicate among the presented statements what would be their level of agreement. (Figure 1) presents the results and the statements are below.

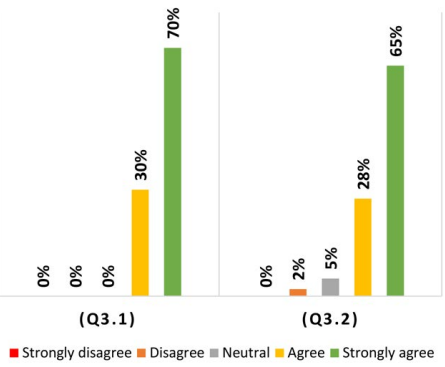

Figure 1. Students' opinions about the course in general.

- **Q3.1 - The exercises were appropriate for the class taught -** Results indicate that all students agree (30%) and strongly agree (70%) that the proposed practical activities are appropriate. Some quotes: "*I believe that the assignments were essential for applying the knowledge obtained in class*", "*The class assignments matched the topic taught in class so concise while the final project covered all the knowledge developed throughout the term as a whole*'.
- **Q3.2 - The knowledge obtained was put into practice -** Most students agreed with this statement (93%), some remained neutral (5%) or disagreed (2%). Some quotes: "*There were exercises for all theoretical content always at the same level addressed in class*". "*I had some difficulties but I always got help from both the monitor and the teacher. The project was good to put the theory into practice, making me learn a lot*".

These results indicate the efficiency of the assignments prepared for the students, helping them to explore the knowledge covered in class, encourage students to learn and apply the acquired knowledge into practical activities.

Q4 - What were the strengths of the course?

The students mentioned some positive aspects of the course: content, critical thinking, theory and practice, learnability. In the content category, we address several

testing topics to spark students' interest in this activity. The classes were designed to generate discussions between students and foster critical thinking about content related to software testing activity, increasing student engagement and participation. Concerning the learnability and theory & practice category, it shows the concern during the course development was to ensure that students put into practice the addressed theoretical knowledge.

Some quotes: "*Class level and content presentation. Many theoretical and practical exercises. The project encompassing the contents covered throughout the course.*", "*Well explained lessons and assignments that exercised the content learned.*", "*There were several exercises, in my opinion, the course was quite practical. It is no longer a boring and theoretical class, making students engage more.*"and "*The practice allowed us to put all our knowledge into the project and comprehend how complex it is to test a software.*".

Q5 - What were the weaknesses of the course?

The students pointed out that little class time, lack of diversity of programming languages, and lack of laboratory practices were the weaknesses of the course. It is very important to discuss these points in order to improve new course editions.

The lack of time indicated by the students happens due to the diverse amount of topics to address of software testing, making impossible to cover all of them. This raises the need to include a second software testing course for students that have the interest in this area. Regarding the lack of a diversity of programming languages, it happened because students have interest for languages like *python* and *ruby* and the course concentrated in Java language. This aspect can easily be settled in next editions. Concerning laboratory practices was indeed a weakness. Due to environmental restrictions, it was not possible to conduct all classes in the laboratory. However, we can guarantee that students were not harmed in the execution of activities. For instance, they could use their computers in the class or did the activities in teams.

4.2. Applying testing techniques

Q6 - Which testing technique was the most efficient? Why?

Regarding the execution of the proposed project, we asked students which testing technique was the most efficient to find bugs and could increase the code coverage. Students have indicated that the structural testing was the most efficient (53%), following of the functional testing (30%) and fault-based testing (17%).

Some quotes: "*Through structural technique there is a clearer view of what is being tested and greatly assists in building test cases that can identify bugs*", "*I think the ability to analyze the code provides a better performance of testing activity than relying solely on the documentation of what the software does*"."*Black-box testing is the most objective technique, focused on the functionality of the software, which will generate sufficiently good test cases, while the other techniques help improve testing, but if used alone may not be as efficient*" "*During the final project, the fault-based test helped to develop more accurate and unnoticed test cases that using the previous techniques*".

Q7 - What do you think is important to consider when selecting a testing technique?

Some aspects were considered by students as important to support the selection

of testing technique, such as complexity that involves aspects as software size, critically, and difficulty to automate the tests. The knowledge of the tester team also influences the testing tool selection. The scope of the project also influences the selection considering the software architecture and the used technologies. Another factor is the cost of applying a testing technique and the time available within the project.

Some students quotes:"*Review the complexity of the software and its size, since the bigger the software, the more complex it would be to automate the tests*", "*I think the technique should be chosen based on the knowledge of testers. It's not worth choosing a technique that no one in the test team knows about.*", "*It is necessary to analyze the project, how it was developed, architecture, used technology and code structure, as it will not always be possible to realize all types of techniques in the project. Another important point is the main goal of the project and to define well what is most important, sometimes a white-box test may be more appropriate than a black-box, and vice versa*".

Q8 - Check whether you agree or disagree with the statements presented (Final project)

Two questions were elaborated to evaluate the students opinion about the combination of testing techniques. The questions and analysis are presented below (Figure 2).

- **Q8.1. The software project to be tested influences the choice of the type of testing technique -** Most students agreed (87%) that the software directly influences the choice of testing technique. A quote: "*The context of the application is essential for the definition of the method to be used, and the combination of different types of testing approaches brings better results because they complement and treat different aspects*".
- **Q8.2 - The combined use of the 3 techniques yielded better results for the testing activity than the use of only one -** The results were positives (87%) with some neutral responses (13%). Students declared that the combination of testing techniques yielded positive results than just using one of them. A quote: "*It is possible to see that one technique complements the other since with each variation of technique, the code coverage increased and different bugs were revealed*".

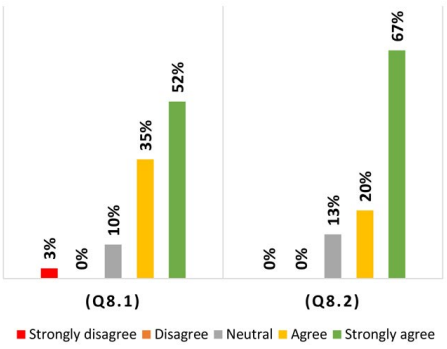

Figure 2. Students' opinions about the final project.

Testing coverage. Available testing techniques display different and often complementary characteristics [Santos et al. 2020]. We asked students to apply the testing techniques incrementally to achieve full coverage for structural and mutation testing. Figure 3 presents an overview of the coverage results achieved by students. The first column

presents the coverage using functional technique criteria; the second column, the coverage for functional and structural techniques; the third column, the mutation score using the three testing techniques. More details are available in repository material. Some teams have not applied all the testing techniques. We believe these teams found difficulties to cover all requirements (e.g., due to infeasible elements or equivalent mutants). Testing coverage increased with the use of more than one technique, highlighting the importance of the combined use of testing techniques. The testing coverage increased by 25% when the combination of the three testing techniques was applied.

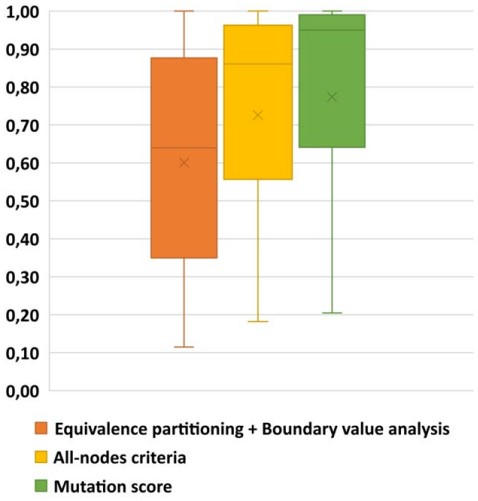

Figure 3. Testing coverage values.

5. LESSONS LEARNED

- Students' Perspective:

Testing skills can help to improve the code development: the conduction of the software testing activity helps to further improve the code developed. This practice should be considered essential in companies to ensure successful and better performance to the developed software projects;

Influences of good didactic and classes content: students provided good feedback on the classes, emphasizing the teacher's didactic and the covered content, as well as the practical assignments that were proposed describing that it makes the content assimilation more natural in each class;

Difficulties to learn how to use testing tools: students indicated that had difficulties with some testing tools. The raised doubts were resolved by the assistant professor. Occasionally, the solution to the problem relied on factors that were not clear in the tools documentation;

Software quality improvement: students indicated that the combination of testing techniques further improved the quality of the software under test, and they were able to achieve a higher level of code coverage through the new test cases. Also, they perceive better how the techniques complement each other by improving bug detection.

- Suggestions for Professors

Feedback about the practical activities: most students indicated that the performed assignments were adequate. Some students suggested more constant feedback during the course to help students who have more difficulties.

General suggestion: based on students' suggestions, the course could have more laboratory classes to combine theory and practice. Also, the time has been indicated as a limiting factor, which requires teachers to prioritize the content that will be covered in the course and which content will take more time for students to assimilate and practice;

Selection of testing techniques: some factors can be considered to support the selection of testing techniques to apply in a software project. These factors may contribute to new research related to improving the selection of testing techniques by considering testers' opinions on a practical situation.

6. THREATS TO VALIDITY

The potential threats are discussed to guide the interpretation of this work:

1) Construct validity: a possible threat can be the survey used to collect students' opinions. To mitigate this, we adapted a survey that was used in [Coutinho et al. 2019].

2) Internal validity: we used the open and axial coding from grounded theory to analyze the data. To mitigate any bias or misinterpretations, the analysis were validated and discussed by authors until achieving an agreement.

3) Conclusion validity: a possible threat might be the coverage testing results obtained by the students. Towards mitigating this threat, we checked the results reported by the students before to generate the study analysis.

7. FINAL REMARKS

This paper reports our experience using the combination of testing techniques to improve students learning. The study was carried out in the context of graduate and undergraduate students attending a software testing course. We describe the course context highlighting the student's background, the assignments and assessments proposed to evaluate students.

The results indicated that it is promising to practically teach software testing. The course was able to motivate students to test more their code, highlighting the importance of the testing activity. We identify the strengths and weaknesses of the course and these data are important to improve future editions. We presented the testing coverage achieved by the students when using the testing techniques and observed how efficient is the use of the combination of testing techniques against the application of just one technique.

As future work, we will apply improvements in the testing course through the lessons learned in this study, and also replicate this activity in another edition of the course, complementing the techniques with testing tools for evaluating non-functional aspects, such as security and performance.

ACKNOWLEDGMENT

The authors acknowledge the financial support from FAPESP (2018/10183-9 and 2019/06937-0) and CNPq (312922/2018-3). Thanks to the students for their involvement.

References

Andrade, S. A., Oliveira Neves, V., and Delamaro, M. E. (2019). Software testing education: dreams and challenges when bringing academia and industry closer together. In *XXXIII Brazilian Symposium on Software Engineering*, pages 47–56. ACM.

Bijlsma, A., Doorn, N., Passier, H., Pootjes, H., and Stuurman, S. (2020). How do students test software units? In *https://arxiv.org/abs/2102.09368*.

Clarke, P. J., Davis, D., King, T. M., Pava, J., and Jones, E. L. (2014). Integrating testing into software engineering courses supported by a collaborative learning environment. *ACM Transactions on Computing Education (TOCE)*.

Coutinho, E. F., Santos, I., Moreira, L. O., and Bezerra, C. I. (2019). A report on the teaching of software ecosystems in software engineering discipline. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 130–139. ACM.

Edwards, S. H. (2003). Teaching software testing: automatic grading meets test-first coding. In *Conference on Object Oriented Programming Systems Languages and Applications: Companion of the 18 th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*.

Fraser, G., Gambi, A., Kreis, M., and Rojas, J. M. (2019). Gamifying a software testing course with code defenders. In *50th ACM Technical Symposium on Computer Science Education*, pages 571–577.

Lauvås Jr, P. and Arcuri, A. (2018). Recent trends in software testing education: a systematic literature review. In *Norsk IKT-konferanse for forskning og utdanning*.

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

Offutt, J., Li, N., Ammann, P., and Xu, W. (2011). Using abstraction and web applications to teach criteria-based test design. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, pages 227–236. IEEE.

Paschoal, L. N., Turci, L. F., Conte, T. U., and Souza, S. R. (2019). Towards a conversational agent to support the software testing education. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 57–66.

Santos, I., Furlanetti, A. B., Melo, S. M., de Souza, P. S. L., Delamaro, M. E., and Souza, S. R. (2020). Contributions to improve the combined selection of concurrent software testing techniques. In *Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing*, pages 69–78.

Smith, J., Tessler, J., Kramer, E., and Lin, C. (2012). Using peer review to teach software testing. In *Proceedings of the ninth annual international conference on International computing education research*, pages 93–98.

Strauss, A. and Corbin, J. (2007). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, 3rd edition.

Wong, W. E., Bertolino, A., Debroy, V., Mathur, A., Offutt, J., and Vouk, M. (2011). Teaching software testing: Experiences, lessons learned and the path forward. In *24th IEEE Conference on Software Engineering Education and Training*, pages 530–534.