A Systematic Literature Review on the Use of Software Testing and Programming with Novice Students

Anália Cristina B. T. Meira
Federal University of Campina Grande
Campina Grande, PB, Brazil
analia.cristina@copin.ufcg.edu.br

Dalton D. S. Guerrero
Federal University of Campina Grande
Campina Grande, PB, Brazil
dalton@computacao.ufcg.edu.br

Wilkerson L. Andrade
Federal University of Campina Grande
Campina Grande, PB, Brazil
wilkerson@computacao.ufcg.edu.br

***Abstract*—This Research Full Paper investigates the use of software testing in introductory programming courses (CS1 and/or CS2) as a supportive and facilitating technique for novice students, identifying adopted strategies and tools used. The literature commonly emphasizes problem-solving skills in programming education or strategies for intervening in educational practices. This paper conducts a Systematic Literature Review (SLR) and aims to understand the impact of introducing tests on novice students' comprehension of programming problems. More research is needed to understand how to improve programming teaching materials and methodologies and how novice students absorb all of this. A study was conducted to answer the following research questions: (RQ1) How are tests being used to facilitate novice students' understanding of programming problems? Additionally, (RQ2) What strategies and tools are being adopted to incorporate software testing into teaching programming for beginner students? The responses highlighted the challenges faced in teaching CS1 and/or CS2, evidencing controlled experiments, alternative methods, and tools used. Finally, we critically analyzed the impact on this target audience and found through the research that there is no consensus on the software testing skills necessary to learn programming. Therefore, these skills were grouped as follows: (1) skills related to introducing software testing tools - consolidated or new - and (2) skills related to education with testing activities. By cross-referencing the information, improvements in student performance can be seen when software tests are integrated into the learning process. Introducing tests by customizing materials adopted in programming courses, activities, and evaluations can promote more effective learning for this group.**

***Index Terms*—Programming introduction; teaching programming; software testing; novice students**

I. INTRODUCTION

Progress in Information Technology has intensified concern about the quality of software produced, boosting expectations for professionals who are skilled in dealing with new technologies and solving problems through software development.

In the Programming Introduction course, some objectives are to enable students to solve everyday problems, improve reasoning, develop analytical thinking skills, and shape competencies to be adopted in the development of applications. Specifically, solving problems demands understanding, breaking down the problem, collecting and interpreting data, questioning assumptions, evaluating knowledge and solution strategies, and validating and verifying the solution.

Developing programs requires students to plan how to solve the problem and creatively write code. For many students, the barrier to identifying and fixing bugs or extending and refactoring code is a negative experience and they simply "abandon all hope of solving the problem on their own" [13].

By explicitly teaching problem-solving techniques, better performance is observed among students in introductory programming courses [8], helping to address this scenario. Adding software testing practices to the student activities and exercises can greatly enhance students' understanding of the problems they need to solve. By making them regularly formulate hypotheses about their programs' behavior and conducting experiments to confirm or refute such hypotheses in the form of tests, students can develop a deeper grasp of programming processes, tools and underlying concepts.

Implementing testing practices can enhance novice students' critical thinking during programming activities and contribute to the ongoing development of testing skills. Given this context, a Systematic Literature Review (SLR) is needed to understand better of what tools are being used or what strategies have been adopted and how testing is being used to facilitate students' understanding of problems.

Therefore, an SLR was planned adhering to the guidelines outlined [7], which involved searching for scientific papers in the ACM Digital Library (DL)[1], IEEE Xplore[2], Science Direct[3] and Scopus[4] databases, covering papers that address this trend in CS1 and/or CS2. The Escritha online tool was used to collect data, enabling collaboration in a shared workspace to conduct the research. The purpose was to analyze the published works in programming education that focus on integrating software testing with programming instruction, either through assessment using testing criteria or through incorporating testing activities during the development process

[1]*ACM Digital Library - https://dll.acm.org/*
[2]*IEEE Xplore Library - https://ieeexplore.ieee.org/*
[3]*Sciencedirect Library - https://www.sciencedirect.com/*
[4]*Scopus Library - https://scopus.com/*

in various iterations of the test-driven development approach, targeting novice programmers.

This paper is structured as follows. In Section II, the methodology applied to this paper is presented. Section III outlines the results obtained in SLR, followed by their discussions in Section IV. Finally, Section V presents the conclusions and suggestions for future work.

II. RESEARCH METHODOLOGY

A Systematic Literature Review was carried out following the guidelines proposed by Kitchenham *et al.* [7], to determine whether using tests in programming courses improves the understanding of software problems that beginner students need to solve. The subsequent steps were followed To establish the research protocol and conduct the study:

- Definition of the scope of the review;
- Search and selection of pertinent papers;
- Data extraction from chosen papers.

A. OBJECTIVES

This SLR aimed to identify whether adopting tests in introductory programming courses has improved students' understanding of problems and to identify which test-related strategies are being adopted when teaching programming to beginner students.

B. RESEARCH QUESTIONS

In order to meet the proposed objectives, the following research questions (RQs) were defined:

1. How are tests being used to facilitate students' understanding of problems?

2. What strategies and tools are being adopted to incorporate software testing into teaching programming for beginner students?

C. SEARCH STRATEGY

Our search approach involved an online investigation in the four widely recognized digital libraries: ACM Digital Library, IEEE Xplore, Science Direct and Scopus.

Search keywords play a crucial role in the comprehensiveness of the results, and careful definition is essential when conducting searches in online digital libraries. The query consists of a raw string of four terms, including two standard terms and another two consisting of a combination of synonyms of the identified keywords.

The first term refers to Beginner Programmers or CS1, the second refers to comprehension and its synonyms, the third refers to programs, and the last to tests. An initial search was carried out in the ACM DL database to analyze the results obtained from our search. In the end, the following query was selected:

(novice OR CS1) AND (comprehension OR learning) AND test AND program

D. SELECTED STUDIES

The selected papers address software testing in higher education, specifically in the programming disciplines of the initial semesters, in accordance with the parameters outlined by the research questions.

After obtaining the papers from the automatic search, duplicate papers, those not written in English, and those outside the context of higher education or focused on advanced computer science courses were excluded.

The following inclusion and exclusion criteria were defined for this SLR. The number of papers obtained can be seen in Table I

- **Inclusion criteria (IC):** studies that include tests in CS1/CS2;
- **Exclusion criteria (EC):** short papers or incomplete studies;
- **Exclusion criteria:** not written in English;
- **Exclusion criteria:** published before the last fifteen years;
- **Exclusion criteria:** duplicates.

E. DATA EXTRACTION

TABLE I
SELECTED STUDIES

Steps	Papers
Step 1: Identify and organize the documents retrieved from the search databases based on the inclusion criteria;	51,193
Step 2: Refine by exclusion criteria EC1, EC2, EC3 and EC4;	666
Step 3: Review of titles and abstract removing papers that fit the exclusion criteria;	22
Step 4: Review of the complete papers that fit the exclusion criteria.	16

Data extraction was carried out to summarize information from the primary studies. The following elements were extracted from each selected paper: year, publication vehicle, abstract, methodology adopted, categories, evaluation form, and results obtained. The process described in the previous section led to 51,193 papers in the four databases. Then, the inclusion and exclusion criteria were applied, which resulted in 666 papers. Only papers published in the last fifteen years were selected because papers published earlier than that would not represent current educational practices. After reading the titles and abstracts, 22 studies were selected for detailed reading. Finally, after this stage, 16 papers were defined to be correlated with this SLR, as shown in Figure 1.

The selected papers were published in a wide variety of journals and conferences, and most of the studies were published in Computer Science Education libraries (ACM DL, IEEE Xplore, Science Direct and Scopus). The details of the selected studies are presented in Table I, categorized by library.

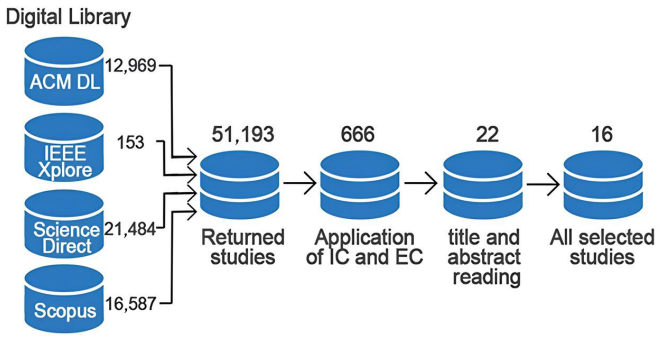

Fig. 1. Number of papers returned and selected.

F. CLASSIFICATION

The selected studies were categorized to facilitate the analysis of the extracted data and meet the defined research questions:

- approaches examined;
- tools investigated;
- topic investigated;
- evaluation methods.

The classifications of the examined approaches and the investigated tools provide a comprehensive overview of the strategies adopted, helping to answer RQ1. In short, we sought to identify concepts that represented the collaboration of each paper.

The evaluation methods identified were **Systematic Mapping Study**, which reviews and synthesizes existing literature on a topic; **Qualitative Study**, which includes the analysis of qualitative data, explores meanings, perspectives and contexts; **Experimental**, including quasi-experiments and case studies experiments; and **Experience Report**, which reports on an experience of applying an intervention, detailing a practical situation. The topics investigated highlight the use of tests to understand students' difficulties, helping to answer RQ2.

III. RESULTS

This paper presents the results achieved in the SLR carried out from September to November 2023. Section IV will present a discussion of the analyzed studies. The distribution by year of the selected studies is shown in Figure 2, and it can be seen that most of the papers were published in the last ten years.

As shown in Table II, most of the papers were published in journals and conferences on Computer Science Education, such as SIGCSE, ITiCSE, ICER, and ACE, totaling eight occurrences, followed by studies published in sources dealing with Software Engineering Education, totaling six occurrences, such as in ICSE, SIGSOFT/FSE, ACM SEEM, and ACM EASE. Finally, studies were identified in places with an emphasis on education in curricula related to computing, including FIE and the Journal Computing Sciences in Colleges.

Regarding the evaluation of each study, four classifications were identified, distributed between Systematic Mapping Study, Qualitative Study, Experimental Study and Experience Report. Trying to identify whether the studies are usually aimed at CS1 students or CS2 students, can be seen in Figure 3 that the majority focused on CS1 classes, with no studies found in our search aimed only at CS2 students.

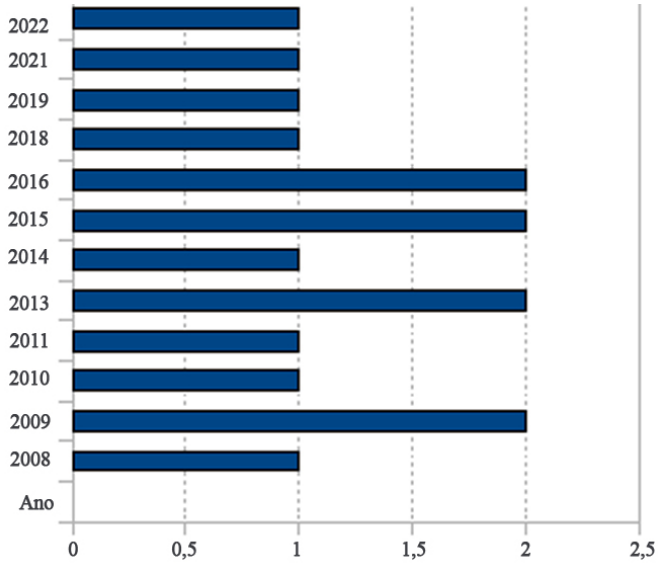

Fig. 2. Number of papers X Year of publication.

TABLE II
DISTRIBUTION OF PUBLICATION VENUES

Venue Name	Venue Type	Ref	Total
SIGCSE	Conference	[4], [5], [15], [17], [18]	5
ICSE	Conference	[11], [19]	2
FIE	Conference	[12]	1
ACE	Conference	[20]	1
Journal of Computing Science in Colleges	Journal	[10]	1
SIGSOFT/FSE	Conference	[14]	1
ACM SEEM	Conference	[16]	1
ACM EASE	Conference	[3]	1
ITiCSE	Conference	[9]	1
TSE	Journal	[6]	1
Total			16

Three papers did not specify in which classes the study took place, Pham *et al.* [14] specified the number of students who took part in the research, but without making it clear whether they were CS1 or CS2 students.

Table III shows the mapping of works by evaluation method and category topics; some works fall into more than one category as seen in the image. The papers that relate to each category and evaluation method can be seen in Table III.

The **curriculum** category suggests and recommends integrating testing into the curriculum of computer science courses, seeking to address testing from introductory subjects. Three papers fell into this category, concluding their approach with experiments, such as Proulx, which presented a curriculum and teaching approach for introductory object-

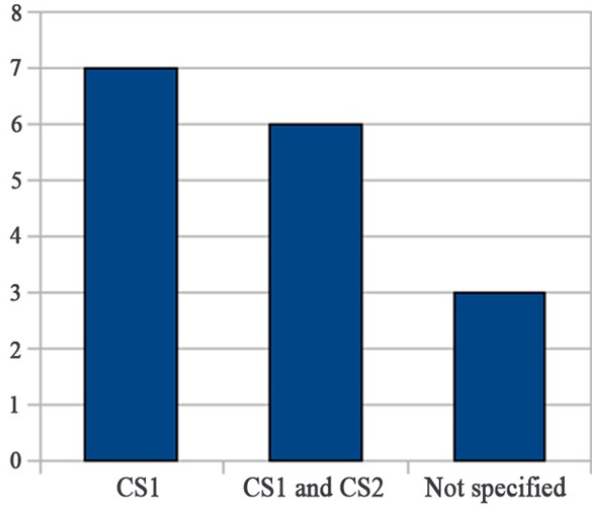

Fig. 3. Number of studies aimed at CS1 and CS2 students.

oriented programming projects, where Test-Driven Development (TDD) is applied consistently from the outset [15].

TABLE III
MAPPING OF THE RESEARCH BETWEEN EVALUATION METHODS AND PUBLICATION CATEGORIES.

Category	Evaluation method	Ref
Curriculum	Systematic Mapping	[17]
Curriculum	Experience report	[1], [18]
Curriculum	Experimental study	[6], [15]
Concept comprehension	Experimental study	[10]
Tools	Experimental study	[3], [9], [11], [12], [16]
Teaching methods	Qualitative study	[14]
Teaching methods	Experimental study	[4]–[6], [11], [15] [20]
Teaching methods	Experience report	[1], [18], [19]
Program/test quality	Qualitative study	[1]
Program/test quality	Experimental study	[3]–[6], [9] [10], [12]
Program/test quality	Experience report	[1]

Only two studies classified as **concept comprehension** were selected; they aimed to investigate assessments of programming concepts, including software testing concepts. Li and Morreale analyzed programming productivity after a comprehensive review of testing concepts being studied alongside programming [10].

Eight papers classified as **program/test quality** [1], [3]–[6], [9], [10], [12] address student performance in programming activities, usually by analyzing whether they can correctly meet the requirements and using metrics that involve an analysis of the source code.

Those categorized as **teaching methods** examine ways of teaching programming alongside software testing, addressing how these two subjects can be taught together. Most of the selected papers propose integrating TDD into subject materials, such as [1], [4]–[6], [15], [18], [20] and/or introducing a method such as Test-Driven Learning (TDL), a method that teaches programming through unit testing, as presented by Janzen and Saiedian [5].

Those categorized as **tools** use them to automate testing practices. Some studies focus on providing new tools or those already available on the market, which ends up adding a workload to the students, as they will have yet another tool to learn to use, as in the work of [9], [11]. The research by Lappalainen *et al.* [9] focused on integrating TDD with the use of the ComTest tool, falling between the two types of studies mentioned above. In addition to the indication of TDD or TDL, there is a study analyzing the use of mutation testing in a programming course in order to add valuable knowledge to the learning process, as in [12].

A common feature of modern approaches to teaching programming is the introduction of testing activities in the early stages, before implementation, so some studies focus on carrying out an analysis and diagnosis of teaching difficulties, especially in problem-solving, and evaluating the improvement in performance after introducing tests at the start of learning programming, as in [10], [14], [16]. Test cases at this stage are used to stimulate student reflection on the problem and promote problem-solving.

Studies focusing on student skills such as problem-solving, programming, and testing predominate among the selected works [3], [21]. One work carried out a systematic mapping study [17] investigating the integration of tests in this context of introductory programming courses and providing an overview of the research carried out in the area.

In general, the effects of adopting software testing in introductory programming courses have been positive. Although some results indicate that the workload increases [2], an improvement in student performance has been identified [2], [5], [10], [16]. Some authors even make pedagogical recommendations [10], [20]. In some cases, it has been suggested that the gradual introduction of testing concepts not only benefits initial courses but also motivates students to explore dedicated software testing courses [16].

IV. DISCUSSION

Analyzing the distribution of studies in CS1 and CS2 classes, it is observed that 43.75% of the studies were conducted in CS1; no studies were found to focus solely on CS2 classes, but 37.5% focused on both CS1 and CS2 classes, as can be seen in Figure 3.

Regarding the observation on the carrying out of controlled experiments, alternative teaching methods, and tools adopted, i.e., the categories investigated, it can be seen that some papers belong to more than one of the categories identified, with only 3.44% (1) of the studies addressing the topic of concept comprehension and 13.79% (4) addressing the curriculum category. These less explored areas may suggest fields that require further investigation; for example, if there were a change in the curriculum by adding new subjects and materials to the disciplines in order to integrate tests, this could help improve student performance.

However, the items on teaching methods, with 34.48% (10) of the papers, and program/test quality - with 31.03% (9) - cover the majority of the research carried out in the area, followed by the tools category with 17.24% (5).

Regarding the defined evaluation strategies, empirical studies, such as experimental studies, comprise 68.96%, those classified as experience reports represent 20.68%, qualitative studies 6.89%, and systematic mapping 3.44%. No systematic literature reviews were found.

Regarding the research question: How are tests being used to facilitate students' understanding of problems? Among the selected studies are strategies such as the analysis of the use of mutation tests being incorporated into programming courses as a way of contributing valuable knowledge to the learning process, such as the work by Oliveira *et al.* [12]. Mutation testing is a fault-based testing criteria that involves making small changes to a Software Under Test (SUT) by creating mutants. The aim is to identify introduced faults and analyze faulty code through test cases. This study [12] explored the effects of mutation testing in teaching programming fundamentals to novice students in a first-year undergraduate class.

Janzen and Saiedian [5] introduced Test-Driven Learning (TDL) as an educational tool for integration into Computer Science (CS) and Software Engineering (SE) curricula. The authors provided guidance on how to apply it in the classroom. This approach highlights the importance of using automated unit testing throughout the curriculum.

Similarly, Desai *et al.* [2] addressed how Test-Driven Development (TDD) can be incorporated into existing course materials without detracting from the comprehensiveness of the topic. Other studies propose similar techniques, aiming to familiarize students with software testing concepts and practices implicitly in the curriculum [1], [10]. These approaches seek to integrate testing elements into the existing curriculum to educate students about improved software development.

Still, on the first research question, regarding the tools to be adopted, when approaching software testing, many studies turn to using or creating tools to support the process. Neto *et al.* [11] proposed the use of a teaching method called Problem-Oriented Programming and Testing (POPT) for introductory courses, supported by the TestBoot tool, which enables students to create test cases without the need to learn test framework APIs. Likewise, Pham *et al.* [14] have provided the ComTest tool as a solution that uses a simple syntax to formulate and automate test code generation.

In relation to the second research question: What strategies and tools are being adopted to incorporate software testing into teaching programming for beginner students? Research shows that problems are mostly addressed by creating test cases before implementing the solution, which makes it easier for students to understand software problems. Some studies suggest that the reason for these improvements lies in the practice of testing, which contributes to developing students' comprehension and analysis skills. These skills, when associated with unit testing as proposed, help to improve productivity and complement programming approaches that cannot be dealt with in isolation with testing, as pointed out by Fucci *et al.* [3] and Yang *et al.* [21]. The role of existing skills is relevant in studying the effects of developers' efforts in unit testing on productivity.

Implementing tests provides students with valuable information about their programming and testing progress before the final deadline for activities. This aspect of feedback is a consistent motivation for using or implementing automated assessment tools, as demonstrated by [3], [9], [11], [16], which provide feedback on student performance.

Automated feedback can reduce the need for teacher intervention, helping students progress when facing difficulties or identifying when they need to seek teacher support. Sychev [19] reports that the number of questions received by teachers and teaching assistants decreased significantly, with most attempts at training questions resulting in finding the correct answer without faculty assistance.

Programmers who conducted tests beforehand reported greater confidence in the quality of the code they developed and felt more secure in their ability to modify the code without causing issues. Additionally, they were more confident they could reuse the code in future projects (Reusability), as noted in [5].

Buffardi and Edwards indicate a positive correlation between the prior use of tests and more favorable student outcomes in programming tasks, such as higher scores and a lower likelihood of late submissions [1]. Similarly, Proulx notes that teachers reported good preparation of students, which resulted in significantly better performance in the course assessments [15].

According to Ramasamy *et al.* [16], with increased use of tests, there was a proportional decrease in failing grades. These observations suggest that the early use of tests benefited student performance.

Regarding testing practices in programming language course assignments, students can be involved to different extents:

- analyzing test results from submission tools;
- working with tests provided by the instructor, such as acceptance tests or unit tests;
- students writing their own tests. Typically, a support tool is involved in the study, primarily a submission system that provides automated assessment.

V. CONCLUSION

This paper aimed to present an SLR to identify whether there is an improvement in the understanding of software problems to be solved in programming by beginner students. To this end, we collected information from empirical studies on this research topic, obtained from a systematic literature review that we carried out on the subject. The SLR protocol was defined, the search was performed, and the results of this review were presented.

They reinforce the advantages of incorporating software testing into programming teaching for novice students, as it improves the understanding of the problems analyzed and,

consequently, the students' performance in the programming activity. As observed in Ramasamy *et al.* [16], the integration of testing with computer programming shows an improvement in the performance of students in the treatment group, compared to students in the control group who did not receive additional knowledge about testing. They also suggested that the gradual introduction of testing concepts not only benefits the initial courses, but also motivates students to explore specific software testing courses.

Few studies address how software testing contributes to developing problem-solving skills. However, approaches have been identified that integrate a variety of software testing concepts into programming education, noting the inclusion of testing in several stages of the introductory sequence, which contributes to enhancing the quality of education in this introductory discipline. This suggests that it is possible to integrate software testing holistically into the computing curriculum [23].

Additionally, testing in this context is primarily conducted in Java [4], [6], [9], rather than other popular used languages in programming courses, such as Python [24].

There are a wide variety of methods for incorporating testing into introductory courses, as evidenced by the chosen papers. As a result, it was identified that, in recent years, several studies have shown and addressed the use of different tools in teaching programming; some studies are still focused on students' pre-existing skills, suggesting that they teach programming skills based on different educational theories.

Many studies use software development practices, such as TDD and TDL, as a way to help software understanding in the programming activity.

The current SLR has limitations because, although the sources used in this protocol include influential journals and conferences, there is no guarantee of complete coverage. For future work, the intention is to expand the SLR to include research in Portuguese, as well as the proceedings of the main Computer Science conferences and journals. Additionally, practical studies are planned with classes in a controlled environment, introducing testing concepts to one group and not to another, aiming to observe which students find it easier to grasp software problems.

ACKNOWLEDGMENT

The third author was supported by CNPq/Brazil (process 303773/2021-9).

REFERENCES

[1] Buffardi, Kevin and Edwards, Stephen, "Effective and ineffective software testing behaviors by novice programmers,". ICER 2013 - Proceedings of the 2013 ACM Conference on International Computing Education Research, pp. 83–90, 2013.

[2] C. Desai, D. Janzen and J. Clements, "Implications of Integrating Test-Driven Development into CS1/CS2 Curricula,". John Clements, 2009.

[3] D. Fucci, B. Turhan and M. Oivo, "On the Effects of Programming and Testing Skills on External Quality and Productivity in a Test-Driven Development Context,", 2014.

[4] M. A. S. Graciotto and B. Camara, "A Strategy to Combine Test-Driven Development and Test Criteria to Improve Learning of Programming Skills,", 2016.

[5] D. Janzen and H. Saiedian, "Test-Driven learning in early programming courses," ACM SIGCSE Bulletin, 2008.

[6] I. Karac, B. Turhan and N. Juristo, "A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development," IEEE Transactions on Software Engineering, 2019.

[7] K. Barbara, "Procedures for performing systematic reviews,". Keele, UK, Keele University, 33(2004):1–26, 2004.

[8] T. Koulouri, S. Lauria and R. D. Macredie, "Teaching introductory programming: A quantitative evaluation of different approaches," Transactions on Computing Education, 14(4):26:1–26:28, 2015.

[9] V. Lappalainen, J. Itkonen, V. Isomöttönen and S. Kollanus, "ComTest: a tool to impart TDD and unit testing to introductory level programming", pp. 83–90, 2010.

[10] J. Li and P. Morreale, " Enhancing CS1 curriculum with testing concepts: a case study,". Journal of Computing Sciences in Colleges, 31, pp. 36–43, 2016.

[11] V. Neto, R. Coelho, L. Leite, D. Serey and A. Mendonca, "POPT: A Problem-Oriented Programming and Testing approach for novice students", Proceedings - International Conference on Software Engineering, 2013.

[12] R. Oliveira, L. Oliveira, B. Cafeo, Bruno and V. Durelli, "Evaluation and assessment of effects on exploring mutation testing in programming courses,". 2015.

[13] D. N. Perkins, C. Hancock, R. Hobbs, F. Martin and R. Simmons, "Conditions of Learning in Novice Programmers,". E. Soloway and J. C. Spohrer, (eds.), Studying the Novice Programmer, pp.261–279. 1989.

[14] R. Pham, S. Kiesling, O. Liskin, L. Singer and K. Schneider, "Enablers, inhibitors, and perceptions of testing in novice software teams,". pp. 30–40, 2014.

[15] V. Proulx, "Test-Driven Design for introductory OO programming," SIGCSE'09 - Proceedings of the 40th ACM Technical Symposium on Computer Science Education. 41. pp. 138–142. 2009

[16] V. Ramasamy, H. Alomari, J. Kiper and G. Potvin, "A Minimally Disruptive Approach of Integrating Testing into Computer Programming Courses," 2018.

[17] 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, pp. 421-–427, New York, NY, EUA. ACM. 2019

[18] S. Schaub,, "Teaching CS1 with web applications and test-driven development" SIGCSE Bull. 41, 2 (June 2009), pp. 113-–117.

[19] O. Sychev, "Write a Line: Tests with Answer Templates and String Completion Hints for Self-Learning in a CS1 Course,". 2022.

[20] J. Whalley and A. Philpott, "A unit testing approach to building novice programmers skills and confidence," Conferences in Research and Practice in Information Technology Series. pp. 113-118. 2011.

[21] T. C. Yang, G. Hwang, S. Yang and G. Hwang, "A two-tier test-based approach to improving students computer-programming skills in a web-based learning enironment," Educational Technology and Society. 18. 198-210. 2015.

[22] R. P. Medeiros, G. L. Ramanho and T. P. Falcão, "A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education," in IEEE Transactions on Education, vol. 62, no. 2, pp. 77-90, May 2019.

[23] E. L. Jones. Software testing in the computer science curriculum – a holistic approach. In Proceedings of the Australasian Conference on Computing Education, ACSE '00, pages 153–157, New York, NY, USA, 2000. ACM.

[24] T. Koulouri, S. Lauria, and R. D. Macredie. Teaching introductory programming: A quantitative evaluation of different approaches. ACM Transactions on Computing Education (TOCE), 14(4):26:1–26:28, Dec. 2014.

[25] J. G. Politz, S. Krishnamurthi, and K. Fisler. In-flow peerreview of tests in test-first programming. In Proceedings of the Tenth Annual Conference on International Computing Education Research, ICER '14, pages 11–18, New York, NY, USA, 2014. ACM.

[26] J. L. Whalley and A. Philpott. A Unit Testing Approach to Building Novice Programmers' Skills and Confidence. In Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114, ACE '11, pages 113–118, Darlinghurst, Australia, Australia, 2011. Australian Computer Society, Inc.

[27] M. Thornton, S. H. Edwards, R. P. Tan, and M. A. Perez-Quinones. Supporting Student-written Tests of Gui Programs. In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '08, pages 537–541, New York, NY, USA, 2008. ACM.

[28] M. Hertz and S. M. Ford. Investigating factors of student learning in introductory courses. In Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13, pages 195–200, New York, NY, USA, 2013. ACM.

[29] M. H. Goldwasser. A Gimmick to Integrate Software Testing Throughout the Curriculum. In Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, SIGCSE '02, pages 271–275, New York, NY, USA, 2002. ACM.

[30] S. M. Rahman and P. L. Juell. Applying Software Development Lifecycles in Teaching Introductory Programming Courses. In 19th Conference on Software Engineering Education Training (CSEET'06), pages 17–24, Apr. 2006.