Teaching Practices of Software Testing in Programming Education

Lilian Passos Scatalon
University of São Paulo (ICMC-USP)
São Carlos-SP, Brazil
lilian.scatalon@usp.br

Rogério Eduardo Garcia
São Paulo State University (FCT-Unesp)
Presidente Prudente-SP, Brazil
rogerio.garcia@unesp.br

Ellen Francine Barbosa
University of São Paulo (ICMC-USP)
São Carlos-SP, Brazil
francine@icmc.usp.br

***Abstract*—This Research Full Paper presents an overview of the practices that have been used to integrate software testing into programming education. Introductory programming courses compose the core of several undergraduate programs, since programming is a crucial technical skill for professionals in many areas. Given the subject importance, researchers have been conducting several studies to investigate teaching approaches that can help overcoming students' learning difficulties. In particular, studies on introducing software testing into this context present evidence that testing practices can improve students' programming performance and habits. There are many teaching approaches in programming education, which involve different choices of programming paradigm and language, support tools and development practices, such as version control. Likewise, the integration of software testing into such diverse context can also happen in many different ways. Therefore, investigating the ways to teach programming and testing at the same time can help instructors with informed choices. In this sense, we identified teaching practices that have been adopted to integrate software testing into programming education. To do so, we further analyzed a subgroup of 195 papers that returned in our systematic mapping on this research domain. We selected papers describing empirical studies (e.g. survey, qualitative studies, experiments, case studies and experience reports), since this kind of study involves applying a given teaching practice in order to collect evidence or report the observed experience. Overall, our results shed light on how the integration of software testing has been done in different classroom contexts of programming education. We discuss the practices in terms of their application context (i.e. the course), how testing was introduced in theory and practice, and the adopted support tools. We also discuss an important gap regarding the lack of instruction in testing concepts, even when students are responsible to write their own tests.**

***Index Terms*—Software Testing, Programming Fundamentals, Teaching Practices, Computer Science Education**

I. Introduction

The sequence of introductory programming courses provides students the required grounding to learn advanced computing concepts. Depending on the institution or degree program, it may have a variable number of courses and cover different set of topics, but it usually includes at least two courses, known as CS1 and CS2 [1]. The choices for instructional design of such courses encompass different programming languages, tools, development practices (e.g. version control, unit testing) and pedagogical approaches [2, 35, 45, 49].

In this scenario, it is possible to notice that the integration of software testing stands out among teaching approaches in the context of programming education. Results of previous studies suggest that testing practices can improve students' programming performance and habits [38, 69, 76]. Furthermore, there are topics of software testing as recommended content of programming fundamentals [2], such as *test case generation* and *unit testing*.

However, software testing can be integrated into programming education in many different ways. For example, it can be firstly introduced since CS1 or at at a later stage [3, 20]. Students can adopt either a test-first or a test-last development approach during assignments [37]. Also, test cases can be expressed by using input/output text files, the assert command or a testing framework. Hence, choosing an approach to integrate software testing also involves deciding on these aspects, among others.

In this paper we identified how the integration of software testing into programming education has been implemented in the classroom. We describe the teaching practices along three aspects: testing concepts, testing practices and support tools. We extracted information from 195 papers reporting empirical studies, which are a subgroup of selected papers from our systematic mapping on this subject [65].

We extracted the following information regarding teaching practices for each study: (i) the course where the empirical study was carried out or that students were taking then; (ii) the programming language used in the study; (iii) the testing concepts presented to students in course materials, lectures or training; (iv) the testing practices adopted by students in programming assignments and (v) the support tools used throughout the study.

Results show a clear predominance of object-oriented languages and tools that support such practices. Also, it is possible to note that software testing is much more present in programming assignments than in course materials. Very few studies address how students learn testing concepts in programming courses.

The remainder of the paper is organized as follows. Section II describes how we gathered the information on teaching practices. Section III provides an overview of several aspects that comprise teaching practices on the integration of software testing into programming education. Section IV presents conclusions and prospects of future work.

II. Method

We performed a systematic mapping on the integration of software testing into programming education in a previous work [65], providing us a set of papers on this research topic. In a nutshell, we performed an automated search in digital libraries and a backward snowballing arriving at a total of 293 selected papers.

We classified papers in terms of evaluation method used in the reported study, based on the categories adopted by Al-Zubidy et al. [4]. Namely, we considered the following categories:

- *Literature review*: when the paper presents a review of existing studies in a given topic.
- *Exploratory study*: if the paper involves observation and model building.
- *Descriptive/persuasive study*: when there is an overview of the current situation in a given topic.
- *Survey*: if the paper described a study in which subjects are surveyed about some intervention;
- *Qualitative study*: if it involves the analysis of qualitative data.
- *Experimental*: papers describing experimental studies include *experiments* (which involve the researcher's intervention and manipulation of the investigated phenomenon), *quasi-experiments* (experiments without randomization) and *case studies* (the phenomenon is monitored while it happens naturally in its own environment).
- *Experience report*: when the paper describes an empirical study that is not previously planned, but a report about the researcher's experience of applying an intervention.
- *Not applicable*: when the paper presents a proposal, but without evaluation.

Then, we set apart papers describing empirical studies, i.e. papers mapped to *survey*, *qualitative study*, *experimental* and *experience report*, resulting in a subgroup of 195 papers. Empirical studies involve applying the investigated approach with students in order to collect evidence or report experience, so it is a source of information on teaching practices.

Hence, for each empirical study, we extracted information on the following aspects:

- the **context** of programming education (by means of the course and the adopted programming language),
- how **testing concepts** were integrated into the programming course materials,
- how **testing practices** were integrated into programming assignments, and
- **tools** used to ease either the teaching of testing concepts or the conduction of testing practices by students.

Still, for testing practices, we extracted the *programming process* recommended by the instructor (if any) and the *testing tasks* for which students were responsible while working on assignments. According to Ammann and Offutt [8], the testing tasks are the following: *test design*, *test automation*, *test execution* and *result evaluation*. Considering the description of how students worked in the assignments in each study, we assessed whether students were responsible for each testing task.

III. Teaching Practices

This section describes the overview of teaching practices on the integration of software testing into programming education, according to the information extracted from empirical studies.

A. Context

Aiming to characterize the context of programming education in each paper, we extracted the course (i.e. the study setting) and the adopted programming language. Since there was a wide variety of course names, we sorted them into the categories specified in Table I.

Considering such categories, Figure 1 provides an overview of courses in which the empirical studies were carried out or that students were taking at the time of the study. 145 papers mention the involved courses, with some indicating more than one course. Considering separately all occurrences, there is a total of 196 mentioned courses.

Overall, it is possible to observe the integration of software testing across several courses. The occurrences of CS1 (24) and CS2 (28) had a similar number. A significant number of occurrences (62) mention more generic course names, using terms such as "introductory" and "programming". Object-oriented programming (13) and design (8) had a smaller representation. Still, there where three occurrences of CS0, which is a preliminary course focused on retention, and one occurrence of CS3.

Note that some courses fall outside the range of introductory programming courses, such as Software Engineering (19), Software Testing (4) and the courses classified as "other" (15). This means there are studies addressing relevant questions to programming education, which can be investigated using different contexts, such as the studies in [5, 19, 23, 25, 26, 34, 39, 54].

There are studies reporting experiences both of introductory courses and other intermediate and advanced courses [32, 58, 71]. Also, many studies investigated automated assessment tools, which can be used throughout the computing curriculum, not limited to introductory courses [17, 46, 68, 74, 77].

The programming language is another important aspect of programming education, since it determines the paradigm, the involved concepts and possible support tools, such as testing frameworks and automated assessment tools. Figure 2 provides an overview of the languages used in the empirical studies. Not all studies indicate the programming languages involved, 140 studies do so. Again, there are studies indicating more than one language.

It is possible to notice the prevalence of Java. Object-oriented languages (Java, C++, Python, C# and Ruby) are more representative of languages used in this context, considering that seven of the 15 languages are object-oriented, mentioned throughout 123 studies. Next, C and Pascal are present in 22

TABLE I
COURSE CATEGORIES IDENTIFIED IN THE EMPIRICAL STUDIES

Course category	**Examples of course names mentioned in the empirical studies**
CS0	*"introductory computing course (CS0)"*, *"Introduction to Computing (CS0)"*
CS1	*"CS1 course"*, *"Programming 1 (CS1)"*
CS2	*"CS2 course"*, *"Data structures (CS2) course"*, *"CS2 (Software Design and Data Structures) course"*, *"Java-oriented CS2 course"*
CS3	*"CS3 programming course"*
Intro/Programming	*"first introductory course"*, *"freshman programming course"*, *"C programming course"*
Data Structures	*"Data Structures and Algorithms"*
Object-oriented Programming	*"introductory object-oriented programming course"*
Object-oriented Design	*"Objects and design course"*, *"software design course"*, *"Object-Oriented (OO) Design in Java course"*
Software Engineering	*"first year software engineering course"*, *"software engineering course"*, *"Soft. & Syst. Eng."*
Software Testing	*"Software Testing course"*, *"Verification, Validation and Testing"*
other	*"Parallel programming"*, *"Comparative Languages"*, *"Programming of Large Systems"*, *"Web development course"*

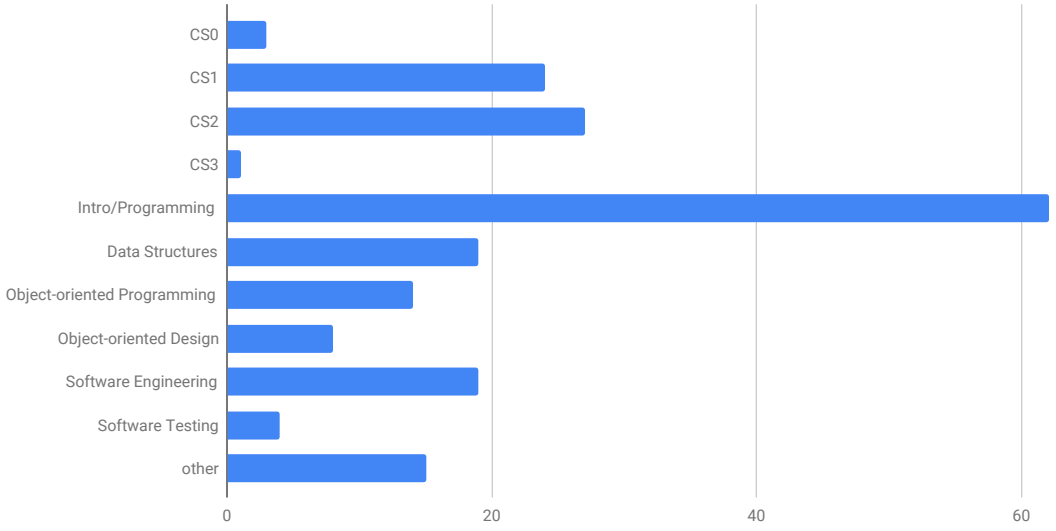

Fig. 1. Courses involved in the empirical studies

studies, showing that the procedural paradigm is the second most common.

A small amount (three studies) use functional languages (Haskell and Scheme) and some studies use variants of other well-known languages, such as Pyret (a variant of Python) and Squeak (a variant of Smalltalk). All the remaining mentioned languages (MPD, OCL, Pep/8 and Prolog) come from the study of Tremblay and Lessard [74], where the authors describe the experience of applying their automated assessment tool, which can evaluate assignments in these different languages.

B. Testing Concepts in Course Materials

Very few papers address how instructors teach testing concepts in programming courses. Only 22.56% (44) of the empirical studies mention some kind of instruction in testing concepts. In general, the authors provide a brief description about it, like suggested in the following text snippets from the papers: "*students are taught testing best practices*", "*the lecture introduced automated unit testing*", "*instructor led introduction to the running and reading of test units*", "*the role of unit testing is introduced early*" and so on. However, this kind of description does not allow to identify what testing concepts were presented to students.

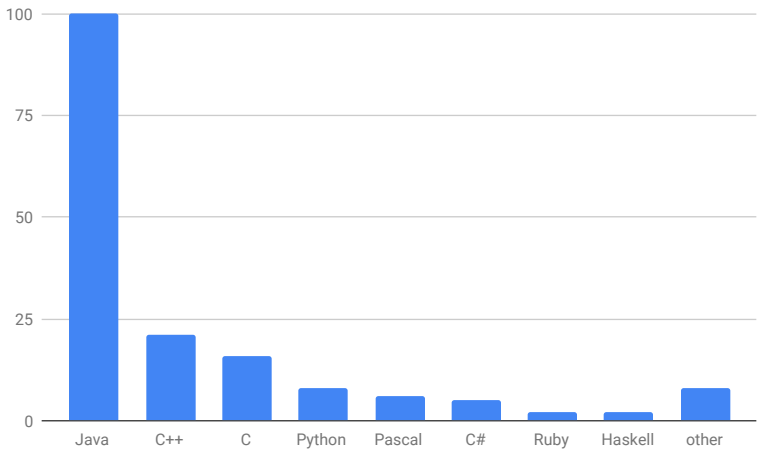

Fig. 2. Programming languages used in the empirical studies

In contrast, the study of Elbaum et al. [28] is a good example of paper that provides a thorough description of testing concepts. Their tool, named *Bug Hunt*, includes a tutorial composed of four lessons about testing concepts: (1) basic testing concepts and terminology, (2) black box testing, (3) white box testing and (4) testing automation and efficiency.

Wick et al. [80] also show the testing concepts to be introduced in stages across different courses in the computing curriculum: (1) unit testing in a first programming course, (2) testing design rules for a second course, (3) integration and use case testing in a software engineering course, and (4) design patterns of testing frameworks in a senior-level design course. The authors provide test code examples in each suggested stage, strengthening their proposal from a practical viewpoint.

It is interesting to notice that even fewer empirical studies (12 studies, i.e. 6.15%) mention the teaching of testing techniques/criteria to students. So, it is not clear how students learn to select input/output values and design test cases in the remaining studies.

C. Testing Practices in Programming Assignments

We also investigated how testing practices have been integrated into programming assignments. In this direction, we identified how testing can be merged into each aspect of an assignment: description/problem specification, assignment steps, deliverables and grading.

First, testing can be a part of the problem specification as *acceptance tests*, i.e. a resource provided to help students validate their solutions at the system level [30, 50, 51, 63, 64]. Also, the tests supplied by the instructor can serve as a *test harness* or *scaffolding*, in the lower level of *unit tests*, providing more detailed support to implement their programs [36, 56, 76].

Regarding the assignment steps, the instructor can provide guidelines to students about the *programming process* to be adopted while working on the assignments. 20% (39) of the empirical studies mention that students were instructed to use a programming process, which are supposed to guide students on how to bind the activities of programming and testing.

Figure 3 shows the distribution of adopted programming processes. Note the prevalence of TDD (61% – 24) in studies that mention the recommendation of a process. Moreover, all the remaining approaches are influenced by TDD somehow, such as test-first processes designed specifically for novice programmers, e.g. [60], [52], [18].

In particular, the identified processes do not provide details on how the testing activity should be conducted. Because of that, we investigated how the testing activity has been conducted by students in terms of testing tasks.

In this way, we considered students responsible for **test design** when they had to choose input/output values and design their own test cases. **Test automation** involved students coding test cases. **Test execution** involved students executing test cases against their programs. Finally, **result evaluation** involved students evaluating test results and getting feedback about their programs. Figure 4 shows the distribution of each testing task conducted by students in the empirical studies.

Considering the configurations of such tasks in the studies, we identified four possible situations:

- **Students only receiving test results:** When they only participate of the *result evaluation* task, usually by submitting their program to an automated assessment system that provides information about failed input/output values. Studies such as [6, 15, 29, 40, 43, 62, 73, 78] adopted this approach.
- **Students working with instructor-provided tests:** When they only participate of *test execution* and *result evaluation*, such as the studies in [50, 56, 76, 79]. They receive a ready-made test suite and are supposed to execute it against their programs in order to get tests results.
- **Students using support mechanisms to design and execute tests:** When they only participate in *test design* and *test execution*. Students are responsible to choose in-

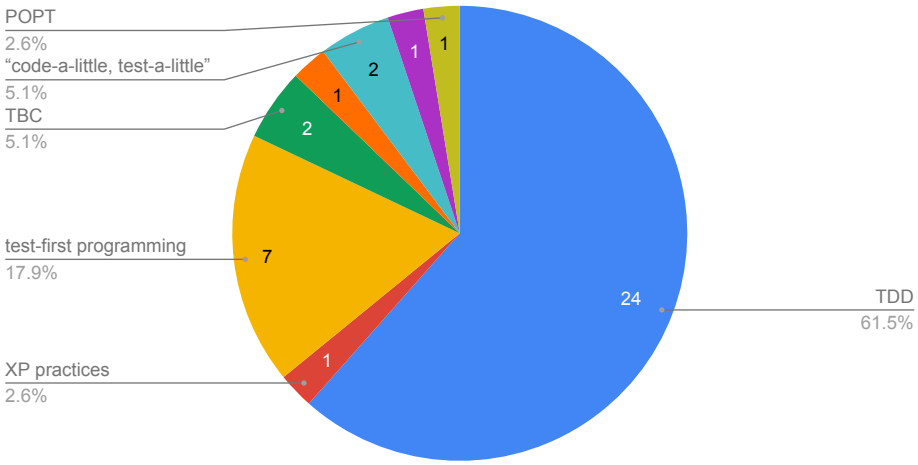

Fig. 3. Programming processes used in the empirical studies

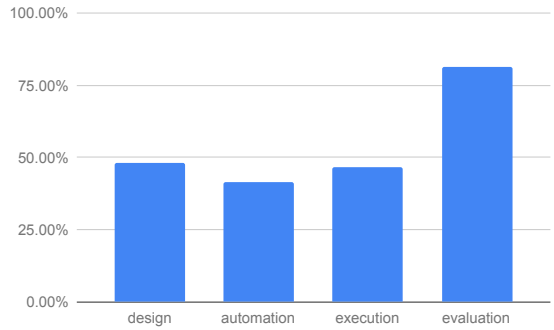

Fig. 4. Testing tasks performed by students in the empirical studies

put/output values and insert them into a support tool that will automate and execute them, returning the test results. This is the case of studies like [22, 28, 36, 44, 47, 48, 81].

- **Students writing their own tests:** When they participate of all testing tasks: *test design*, *test automation*, *test execution* and *result evaluation*. This situation takes place when programming assignments require a test suite as a deliverable along with the program, like studies in [10, 16, 24, 31, 33, 39, 52, 59, 70].

D. Support Tools

74.87% (146) of the empirical studies mention the use of a support tool related to the integration of testing. We identified the adopted tools according to the categories we established in [65]:

- **Testing frameworks/libraries:** testing libraries similar to xUnit frameworks, which are designed for novice programmers [9, 13, 66, 72]
- **IDEs' testing facilities:** IDEs can present features that aid students testing their programs, like BlueJ [55].
- **Submission and testing systems:** These systems usually are responsible for compiling the submitted program, executing tests and providing feedback to students. In some cases these tools also grade students' submitted code according to tests results in a (semi-)automatic manner. In general, these systems are web-based [27, 68, 70, 77] or plug-ins to other widely used systems such as IDEs [7] or LMSs [75].
- **Online judges:** Tools composed of a catalog of problems, where students submit their solutions to be assessed by means of testing. It is common to use these tools in programming competitions [57, 62].
- **Games:** Some tools aim to motivate students through fun and competition. They introduce software testing in different ways, such as implicit testing to solve programming "quests" [12], or hints in the format of unit tests, which help students to guess a "secret implementation" in code duels [11].
- **Tutor systems:** Tools that combine materials and exercises, providing automatic guidance while students learn programming and testing. The materials usually are presented through slides or hypertext and exercises are evaluated with an automated assessment tool that test students' programs [21, 28, 32, 38].

Figure 5 shows the distribution of tools used in the studies, according to the these categories. Again, it is important to note that some studies adopted more than one category of tool.

Submission and testing systems, such as Web-CAT [26],

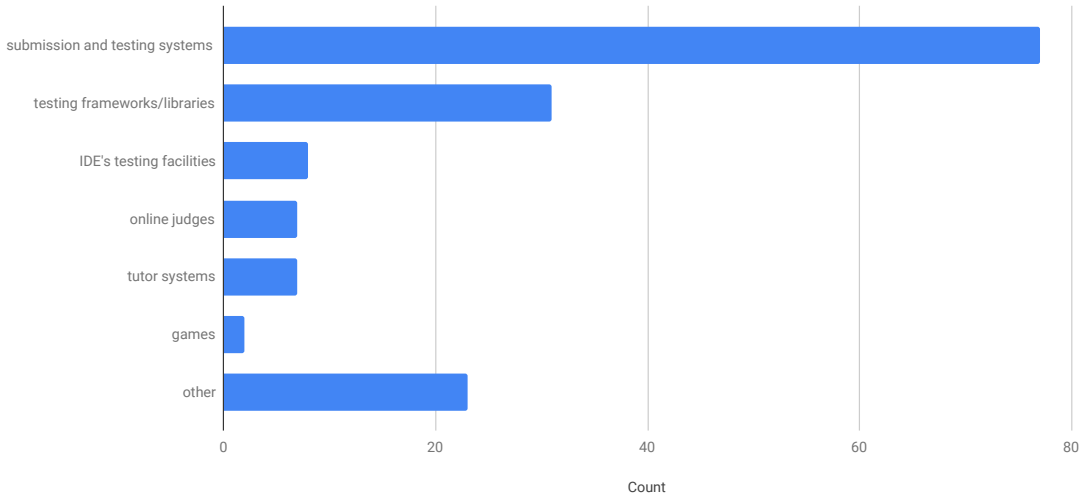

Fig. 5. Supporting tools used in the empirical studies

Marmoset [70] and ProgTest [67], are the most commonly used tools, mentioned by 77 empirical studies. It is natural to adopt this kind of automated assessment tool in a programming course, especially when software testing is involved. Surprisingly, testing frameworks/libraries are only mentioned by 31 studies, even though students are responsible for test automation in 81 studies, as discussed in Section III-C. This is because authors often do not report details of the testing practice adopted by students.

The remaining categories of tools, i.e. IDEs testing facilities (8), online judges (7), tutor systems (7) and games (2), are less employed in the studies. The category "other" includes tools such as *Emma* [17], *EasyAccept* [63], *SpecCheck* [41], *TestBoot* [52], *CaptainTeach* [58], *Pascal Mutants* [53], *BugFixer* [61] and *Ante Up* [14].

IV. Conclusion

In this paper we investigated teaching practices of software testing in the context of programming education. More specifically, we gathered information on how instructors/researchers have addressed this approach in the classroom.

To do so, we extracted information from empirical studies on this research topic, which were obtained from a systematic mapping that we conducted on this subject [65]. We described the practices along three aspects: the programming education context, the testing concepts addressed in materials/lectures, the testing practices in programming assignments and the support tools used by students.

Firstly, it is possible to note the presence of testing in different points of the introductory sequence, what indicates that is possible to adopt a holistic approach to integrate software testing into the computing curriculum [42]. Also, testing in this context is predominantly presented with Java, in detriment of other programming languages widely used in programming courses such as Python [45].

Moreover, results show that testing is much more present in programming assignments than in course materials, which is expected since testing is used as a support practice to programming in this context. However, if students are supposed to test their programs, they need guidance in how to do so, hence there is a need to further investigate how testing concepts should be introduced into programming courses.

As to testing practices in programming assignments, students can be involved to different extents: analyzing test results from submission tools, working with instructor-provided tests (e.g acceptance tests or a test harness in the lower level of unit testing), using support mechanisms to design tests (e.g. plugins where students insert inputs and expected outputs) and, finally, students writing their own tests. Usually there is a support tool involved in the study, mostly a submission and testing system that provides automated assessment.

Results draw attention to how testing concepts have been addressed in the context of programming education. The studies describe this aspect very briefly. However, instruction in testing concepts and techniques is particularly important in approaches that students are supposed to write their own test cases. Nonetheless, this problem can be more about study reporting than about the teaching practices per se.

As future work, we intend to further investigate such teaching practices by surveying instructors of programming courses. The information gathered in this study (which was accessible in the research papers) can help to precisely obtain information in more detail on issues identified in our results. For example, why testing practices are concentrated in the Java language? Is Java also the predominant adopted language in programming courses in general? Or how exactly instructors that choose to integrate testing in their courses teach testing concepts? Addressing questions like these can help to expand in practice the integration of software testing into programming education.

Acknowledgment

This work was supported by FAPESP (São Paulo Research Foundation) grants 2014/06656-8 and 2018/26636-2.

References

[1] ACM/IEEE-CS. Computing curricula 2001, 2001. Joint Task Force on Computing Curricula.

[2] ACM/IEEE-CS. Computer science curricula 2013, December 2013. Joint Task Force on Computing Curricula.

[3] J. Adams. Test-driven Data Structures: Revitalizing CS2. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 143–147, New York, NY, USA, 2009. ACM.

[4] A. Al-Zubidy, J. C. Carver, S. Heckman, and M. Sherriff. A (updated) review of empiricism at the sigcse technical symposium. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE'16)*, pages 120–125, New York, NY, USA, 2016. ACM.

[5] E. Allen, R. Cartwright, and C. Reis. Production Programming in the Classroom. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 89–93, New York, NY, USA, 2003. ACM.

[6] A. Allevato and S. Edwards. Dereferee: Instrumenting C++ pointers with meaningful runtime diagnostics. *Software - Practice and Experience*, 44(8):973–997, 2014.

[7] A. Allowatt and S. H. Edwards. Ide support for test-driven development and automated grading in both java and c++. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, eclipse '05, pages 100–104, New York, NY, USA, 2005. ACM.

[8] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 2 edition, 2016.

[9] S. K. Andrianoff, D. B. Levine, S. D. Gewand, and G. A. Heissenberger. A Testing-based Framework for Programming Contests. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, eclipse '03, pages 94–98, New York, NY, USA, 2003. ACM.

[10] E. G. Barriocanal, M.-A. S. Urban, I. A. Cuevas, and P. D. Perez. An Experience in Integrating Automated Unit Testing Practices in an Introductory Programming Course. *SIGCSE Bull.*, 34(4):125–128, Dec. 2002.

[11] J. Bell, S. Sheth, and G. Kaiser. Secret Ninja Testing with HALO Software Engineering. In *Proceedings of the 4th International Workshop on Social Software Engineering*, SSE '11, pages 43–47, New York, NY, USA, 2011. ACM.

[12] J. Bishop, R. N. Horspool, T. Xie, N. Tillmann, and J. de Halleux. Code Hunt: Experience with Coding Contests at Scale. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 398–407, Piscataway, NJ, USA, 2015. IEEE Press.

[13] D. Blaheta. Unci: A C++-based Unit-testing Framework for Intro Students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 475–480, New York, NY, USA, 2015. ACM.

[14] M. K. Bradshaw. Ante Up: A Framework to Strengthen Student-Based Testing of Assignments. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 488–493, New York, NY, USA, 2015. ACM.

[15] C. Brown, R. Pastel, B. Siever, and J. Earnest. JUG: A JUnit Generation, Time Complexity Analysis and Reporting Tool to Streamline Grading. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 99–104, New York, NY, USA, 2012. ACM.

[16] K. Buffardi and S. H. Edwards. Reconsidering Automated Feedback: A Test-Driven Approach. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 416–420, New York, NY, USA, 2015. ACM.

[17] R. Cardell-Oliver, L. Zhang, R. Barady, Y. H. Lim, A. Naveed, and T. Woodings. Automated Feedback for Quality Assurance in Software Engineering Education. In *2010 21st Australian Software Engineering Conference*, pages 157–164, Apr. 2010.

[18] M. E. Caspersen and M. Kolling. STREAM: A First Programming Process. *Trans. Comput. Educ.*, 9(1):4:1–4:29, Mar. 2009.

[19] H. B. Christensen. Systematic Testing Should Not Be a Topic in the Computer Science Curriculum! In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '03, pages 7–10, New York, NY, USA, 2003. ACM.

[20] J. Clements and D. Janzen. Overcoming Obstacles to Test-Driven Learning on Day One. In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 448–453, Apr. 2010.

[21] J. Collofello and K. Vehathiri. An environment for training computer science students on software testing. In *Proceedings Frontiers in Education 35th Annual Conference*, pages T3E–6, Oct 2005.

[22] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx. CodeWrite: Supporting Student-driven Practice of Java. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 471–476, New York, NY, USA, 2011. ACM.

[23] S. H. Edwards. Improving Student Performance by Evaluating How Well Students Test Their Own Programs. *J. Educ. Resour. Comput.*, 3(3), Sept. 2003.

[24] S. H. Edwards. Teaching Software Testing: Automatic Grading Meets Test-first Coding. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, pages 318–319, New York, NY, USA, 2003. ACM.

[25] S. H. Edwards. Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In *International Conference on Education and Information Systems: Technologies and Applications (EISTA?03)*, 2003.

[26] S. H. Edwards. Using Software Testing to Move Students from Trial-and-error to Reflection-in-action. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '04, pages 26–30, New York, NY, USA, 2004. ACM.

[27] S. H. Edwards and M. A. Perez-Quinones. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, pages 328–328, New York, NY, USA, 2008. ACM.

[28] S. Elbaum, S. Person, J. Dokulil, and M. Jorde. Bug hunt: Making early software testing lessons engaging and affordable. In *Proceedings - International Conference on Software Engineering*, pages 687–697, 2007.

[29] E. Enstrom, G. Kreitz, F. Niemela, P. Soderman, and V. Kann. Five years with kattis: Using an automated assessment system in teaching. In *2011 Frontiers in Education Conference (FIE)*, pages T3J–1–T3J–6, Oct. 2011.

[30] H. Erdogmus, M. Morisio, and M. Torchiano. On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, March 2005.

[31] C. Fidge, J. Hogan, and R. Lister. What vs. How: Comparing Students' Testing and Coding Skills. In *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136*, ACE '13, pages 97–106, Darlinghurst, Australia, Australia,

2013. Australian Computer Society, Inc.

[32] G. Fischer and J. W. von Gudenberg. Improving the Quality of Programming Education by Online Assessment. In *Proceedings of the 4th International Symposium on Principles and Practice of Programming in Java*, PPPJ '06, pages 208–211, New York, NY, USA, 2006. ACM.

[33] 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.

[34] O. S. Gómez, S. Vegas, and N. Juristo. Impact of cs programs on the quality of test cases generation: An empirical study. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 374–383, New York, NY, USA, 2016. ACM.

[35] 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.

[36] V. Isomottonen and V. Lappalainen. Csi with games and an emphasis on tdd and unit testing: Piling a trend upon a trend. *ACM Inroads*, 3(3):62–68, Sept. 2012.

[37] D. Janzen and H. Saiedian. Test-driven Learning in Early Programming Courses. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 532–536, New York, NY, USA, 2008. ACM.

[38] D. S. Janzen, J. Clements, and M. Hilton. An Evaluation of Interactive Test-driven Labs with WebIDE in CS0. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1090–1098, Piscataway, NJ, USA, 2013. IEEE Press.

[39] D. S. Janzen and H. Saiedian. On the Influence of Test-Driven Development on Software Design. In *19th Conference on Software Engineering Education Training (CSEET'06)*, pages 141–148, Apr. 2006.

[40] P. Jezek, M. Malohlava, and T. Pop. Automated evaluation of regular lab assignments: A bittersweet experience? In *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 249–258, May 2013.

[41] C. Johnson. SpecCheck: Automated Generation of Tests for Interface Conformance. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 186–191, New York, NY, USA, 2012. ACM.

[42] 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.

[43] M. Joy, N. Griffiths, and R. Boyatt. The boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.

[44] M. Kolling, B. Quig, A. Patterson, and J. Rosenberg. The bluej system and its pedagogy. *Computer Science Education*, 13(4):249–268, 2003.

[45] 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.

[46] S. Krusche and A. Seitz. Artemis: An automatic assessment management system for interactive learning. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 284–289, New York, NY, USA, 2018. ACM.

[47] V. Lappalainen, J. Itkonen, V. Isomottonen, and S. Kollanus. ComTest: A Tool to Impart TDD and Unit Testing to Introductory Level Programming. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, pages 63–67, New York, NY, USA, 2010. ACM.

[48] C. Leska and J. Rabung. Refactoring the CS1 Course. *J. Comput. Sci. Coll.*, 20(3):6–18, Feb. 2005.

[49] A. Luxton-Reilly, Simon, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo. Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, page 55106, New York, NY, USA, 2018. Association for Computing Machinery.

[50] M. Missiroli, D. Russo, and P. Ciancarini. Teaching test-first programming: Assessment and solutions. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 420–425, July 2017.

[51] M. Morisio, M. Torchiano, and G. Argentieri. Assessing quantitatively a programming course. In *10th International Symposium on Software Metrics, 2004. Proceedings.*, pages 326–336, Sept. 2004.

[52] V. L. Neto, R. Coelho, L. Leite, D. S. Guerrero, and A. P. Mendonca. POPT: A Problem-oriented Programming and Testing Approach for Novice Students. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1099–1108, Piscataway, NJ, USA, 2013. IEEE Press.

[53] R. A. P. Oliveira, L. B. R. Oliveira, B. B. P. Cafeo, and V. H. S. Durelli. Evaluation and assessment of effects on exploring mutation testing in programming courses. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Oct. 2015.

[54] B. H. Pachulski Camara and M. A. Graciotto Silva. A Strategy to Combine Test-Driven Development and Test Criteria to Improve Learning of Programming Skills. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 443–448, New York, NY, USA, 2016. ACM.

[55] A. Patterson, M. Kölling, and J. Rosenberg. Introducing unit testing with bluej. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '03, pages 11–15, New York, NY, USA, 2003. ACM.

[56] J. Paul. Test-driven Approach in Programming Pedagogy. *J. Comput. Sci. Coll.*, 32(2):53–60, Dec. 2016.

[57] J. Petit, O. Gimenez, and S. Roura. Jutge.Org: An Educational Programming Judge. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 445–450, New York, NY, USA, 2012. ACM.

[58] J. G. Politz, S. Krishnamurthi, and K. Fisler. In-flow peer-review 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.

[59] V. K. Proulx. Test-driven Design for Introductory OO Programming. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 138–142, New York, NY, USA, 2009. ACM.

[60] 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.

[61] L. Reynolds, Q. Mayo, D. Adamo, and R. Bryce. Improving Conceptual Understanding of Code with Bug Fixer. *J. Comput. Sci. Coll.*, 31(2):87–94, Dec. 2015.

[62] M. Rubio-Sanchez, P. Kinnunen, C. Pareja-Flores, and A. Velazquez-Iturbide. Student perception and usage of an automated programming assessment tool. *Computers in Human Behavior*, 31:453 – 460, 2014.

[63] J. P. Sauve and O. L. Abath Neto. Teaching Software Development with ATDD and Easyaccept. In *Proceedings of the 39th*

SIGCSE Technical Symposium on Computer Science Education, SIGCSE '08, pages 542–546, New York, NY, USA, 2008. ACM.

[64] J. P. Sauvé, O. L. Abath Neto, and W. Cirne. Easyaccept: A tool to easily create, run and drive development with automated acceptance tests. In *Proceedings of the 2006 International Workshop on Automation of Software Test*, AST '06, pages 111–117, New York, NY, USA, 2006. ACM.

[65] 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*, SIGCSE'19, pages 421–427, New York, NY, USA, 2019.

[66] J. Snyder, S. H. Edwards, and M. A. Perez-Quinones. LIFT: Taking GUI Unit Testing to New Heights. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 643–648, New York, NY, USA, 2011. ACM.

[67] D. M. d. Souza, J. C. Maldonado, and E. F. Barbosa. ProgTest: An environment for the submission and evaluation of programming assignments based on testing activities. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–10, May 2011.

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

[69] J. Spacco, D. Fossati, J. Stamper, and K. Rivers. Towards Improving Programming Habits to Create Better Computer Science Course Outcomes. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 243–248, New York, NY, USA, 2013. ACM.

[70] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. Padua-Perez. Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITICSE '06, pages 13–17, New York, NY, USA, 2006. ACM.

[71] R. Teusner, T. Hille, and C. Hagedorn. Aspects on finding the optimal practical programming exercise for moocs. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, Oct 2017.

[72] 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.

[73] G. Tremblay, F. Guerin, A. Pons, and A. Salah. Oto, a generic and extensible tool for marking programming assignments. *Software: Practice and Experience*, 38(3):307–333, 2008.

[74] G. Tremblay and P. Lessard. A marking language for the oto assignment marking tool. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 148–152, New York, NY, USA, 2011. ACM.

[75] L. C. Ureel and C. Wallace. WebTA: Automated iterative critique of student programming assignments. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Oct. 2015.

[76] I. Utting, A. E. Tew, M. McCracken, L. Thomas, D. Bouvier, R. Frye, J. Paterson, M. Caspersen, Y. B.-D. Kolikant, J. Sorva, and T. Wilusz. A Fresh Look at Novice Programmers' Performance and Their Teachers' Expectations. In *Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-working Group Reports*, ITiCSE -WGR '13, pages 15–32, New York, NY, USA, 2013. ACM.

[77] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Partel. Scaffolding Students' Learning Using Test My Code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 117–122, New York, NY, USA, 2013. ACM.

[78] T. Wang, X. Su, P. Ma, Y. Wang, and K. Wang. Ability-training-oriented automated assessment in introductory programming course. *Computers & Education*, 56(1):220 – 226, 2011. Serious Games.

[79] 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.

[80] M. Wick, D. Stevenson, and P. Wagner. Using Testing and JUnit Across the Curriculum. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '05, pages 236–240, New York, NY, USA, 2005. ACM.

[81] B. V. Zanden, D. Anderson, C. Taylor, W. Davis, and M. W. Berry. Codeassessor: An Interactive, Web-based Tool for Introductory Programming. *J. Comput. Sci. Coll.*, 28(2):73–80, Dec. 2012.