# A Minimally Disruptive Approach of Integrating Testing into Computer Programming Courses

Vijayalakshmi Ramasamy, Hakam W. Alomari, James D. Kiper
Dept. of Computer Science and Software Engineering
Miami University
Oxford, OH, USA
Email:{ramasav,alomarhw,kiperjd}@miamioh.edu

Geoffrey Potvin
Department of Physics
Florida International University
Miami, Florida, USA
gpotvin@fiu.edu

## ABSTRACT

The problem of finding and evaluating effective ways of integrating software testing concepts and related techniques into introductory programming courses is still an open research question. In this paper, we present multiple studies that assess our approach to integrating software testing in Computer Science (CS) and Software Engineering (SE) courses. Each study uses SEP-CyLE (Software Engineering and Programming Cyberlearning Environment), an external, web-based learning tool to help instructors integrate testing concepts into their courses. These empirical studies were conducted in eight CS/SE course sections at a medium-sized public university. The results show (1) SEP-CyLE can be efficiently used in the classroom to impact the testing knowledge gained by students, and (2) students find that SEP-CyLE is a useful learning resource that effectively helps them complete course tasks and better master course concepts.

## CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; • **Social and professional topics → Computing education**; **Computer science education**; **CS1**;

## KEYWORDS

SEP-CyLE, Management System, Software Testing Concepts and Techniques, Empirical Study, Learning Objects

**ACM Reference Format:**
Vijayalakshmi Ramasamy, Hakam W. Alomari, James D. Kiper and Geoffrey Potvin. 2018. A Minimally Disruptive Approach of Integrating Testing into Computer Programming Courses. In *Proceedings of IEEE/ACM International Workshop on Software Engineering Education for Millennials (SEEM'18).* ACM, New York, NY, USA, Article 8, 7 pages. https://doi.org/10.475/123_4

## 1 INTRODUCTION

Cognitive Load Theory [22] offers a method of constructing course pedagogy to help students learn effectively by using an awareness of the limited amount of information that working memory can hold at one time. From the students' learning perspective, the process of how the mind procures information, stores knowledge in the long-term memory and retrieves it later into working memory is an intricate process. Therefore, instructional methods should avoid overloading the working memory with additional activities that do not contribute directly to learning.

Students in foundation-level STEM courses, especially introductory computer programming, typically need a wide range of multimodal instructional techniques for constructive engagement in order to acquire more detailed knowledge and skills in programming. When developed well with an awareness of cognitive load, such a range of instructional techniques can have many positive effects not only on students' problem solving, critical thinking, and perseverance but also on students' academic achievement [6][17][19]. Moreover, it has always been challenging to design curricula that meets the requirements of large and heterogeneous groups of students, often leading to high dropout rates and high student failure rates in programming courses [7][16].

The curricula of foundational undergraduate programming courses such as CS1 and CS2 typically are designed to focus on writing effective code without much emphasis on possible testing procedures for these programming constructs. Developing practical solutions that would benefit teaching and learning to write programs includes helping the students understand the basics of the behavior of programs and the abstract concepts of programming.

A vital area of software engineering research is to develop pedagogy and instructional techniques to equip students with the fundamental skills needed to define requirements and specifications, to develop an application, and to examine whether or not the code meets those requirements and specifications. It is evident from research studies that the process of testing code forces the integration and application of theories and skills of software analysis and is considered as an essential skill for writing efficient computer programs [2][15][18]. Furthermore, we believe (and will present evidence) that a deeper knowledge of software testing concepts and methods helps to develop a firmer foundation of the students' understanding of fundamental programming constructs in addition to skill in testing programs with all possible usage scenarios. Thus, identifying methods to integrate the basic concepts of software engineering (including testing) into foundational undergraduate programming courses has been an emerging area of study for many researchers [10][21].

While software industries expend more than 50% of their project resources on testing the software, the students rarely understand the need for testing at any point in undergraduate curricula [1][21]. To evaluate this hypothesis about the value of integrating software


*SEEM'18, 2018, Gothenburg, Sweden*

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

testing with computer programming, we introduced the basic concepts of software testing to the undergraduate students in a CS1 course. We used an external website tool called SEP-CyLE [5][7] for software testing learning objects and tutorials. We then assessed experimentally the degree to which the students have gained additional and useful knowledge of testing along with learning the fundamental concepts in programming from the use of this pedagogical tool.

The remainder of this paper is organized as follows. Section 2 provides an overview of the literature along with a description of SEP-CyLE. Section 3 presents the design of the empirical study using SEP-CyLE including data acquisition and possible research questions. Section 4 presents the analysis of the data organized around the study goals and a detailed discussion of the experimental results. Section 5 discusses the study limitations and threats to the results' validity followed by concluding remarks and future directions in Section 6.

## 2 BACKGROUND

### 2.1 Related Work

Members of both academic communities and software industries have recognized a need to integrate software testing practices into curricula, and there have been several investigations into the topic. For example, Goldwasser [12] proposed requiring students to submit test sets along with their source code. This technique can be seamlessly integrated into any new or existing programming assignment and requires students to think of software testing as a natural part of their program development. Edwards [10] indicates that learners consider testing as both a boring activity and a significant overhead, owing to the time that they spend on performing the tests and writing of test plans. Ellen et al. [3] investigated combining programming and testing by developing an educational module.

Janzen et al. [14] proposed the idea of TDL, or test-driven learning, as an educational tool that could be incorporated into CS and SE curricula. This approach emphasizes the use of automated unit tests throughout the curriculum. Other findings [8][11][13][20] propose similar techniques that all seek to familiarize students with the concepts and practices of software testing implicitly in the curriculum. These approaches seek to blend the concepts and practices of software testing into existing curriculum in the hopes of teaching students how to develop better software, rather than the typical trial-and-error approach that Edwards [9] and others have frequently observed.

However, while these proposed techniques show promise, there is still few resources and tools available for instructors to adopt these approaches. For the integration of software testing into existing curriculum to be effective, instructors must possess the necessary knowledge and skill. In addition, they need appropriate assisting tools and course materials in order to achieve successful curriculum integration.

This paper presents an initial investigation of the impact of integrating the software testing principles with fundamental programming courses by applying cognitive load theory. This investigation aligns with the goals of the study presented in [7] that focuses on using the SEP-CyLE cyberlearning environment in integrating software testing concepts and testing techniques and tools into students' learning process. Moreover, it has been easier to integrate this approach into the curriculum by selectively using the needed learning objectives and learning materials.

The following subsection provides an overview of the tool and elaborates the features of SEP-CyLE used in this study.

### 2.2 SEP-CyLE

SEP-CyLE was initially designed and developed by Clarke, et al. in 2010 [7] as a Web-Based Repository of Testing Tutorials (WReSTT) [4] with a cyberlearning environment. It intends to improve the testing knowledge of computer science and software engineering students by providing a set of learning objects and testing tool tutorials to satisfy learning objectives, ordering these learning objects and tutorials sequentially based upon the difficulty level. Subsequently, WReSTT has evolved into a collaborative learning environment with social networking features such as the ability to award virtual points for student social interaction about testing. We call these virtual points since an instructor may choose not to use these as a part of students' grades, but only for the motivation that collaborative learning and gamification may provide. The current version of WReSTT, used in this paper, now called SEP-CyLE, is a configurable learning and engagement cyberlearning environment that contains a growing amount of digital learning content in many STEM areas. Currently, the learning content primarily focuses on software engineering and programming. Unlike other learning management systems, SEP-CyLE uses embedded learning and engagement strategies (ELESs) to involve students more deeply in the learning process. These ELESs currently include Collaborative Learning, Gamification, Problem-Based Learning and Social Interaction.

The learning content in SEP-CyLE is packaged in the form of Learning Objects (LOs) and Tutorials. LOs are chunks of multimedia learning content that should take the learners between five to fifteen minutes to complete and contains both a practice (formative) assessment and a second summative assessment. The collaborative learning features allow students to upload their user profile, gain virtual points after completing a learning object, post comments on the discussion board, and monitor the activities of peers.

The choice of learning objects used in a given course is based on the level of the course, the course learning outcomes, and instructor preferences. A variety of learning objects and tutorials are available in SEP-CyLE. For example, SEP-CyLE can be used in both undergraduate and graduate courses by having the instructor assign students learning contents with appropriate levels of difficulties.

## 3 EMPIRICAL STUDY

In this section, we start by presenting the study motivation and identify the research objectives and its related research questions, describe the subject selection, experimental design and instrumentation.

### 3.1 Motivation

The natural ties between software testing and fundamentals of problem-solving and programming were identified while teaching eight sections of the CS1 course during Fall semester 2016. In that semester, there was no testing instruction provided. However,

students were required to include simple test cases for their programming assignments. This is a common approach in CS1 courses. Students are taught to create correct programs with little instruction about how to thoroughly test. However, students found it challenging to design appropriate test cases. They were frustrated with the reduction in the points for their programs that failed on different possible test cases. Although we did not explicit teach about testing, the students were expected to test their code. Thus, they developed their own informal theories of testing. Therefore, it was appropriate for us to ask students in the Fall of 2016 about their knowledge of testing to establish a baseline of student testing knowledge.

WHAT?

To improve upon this lack of testing knowledge, our idea was to introduce the basic concepts of software testing and to do this in a way that does not impose much additional workload into our CS1 courses. An additional challenge is that there is little or no time to teach testing in these courses since there are usually many topics to be covered and learning outcomes to be achieved.

To facilitate this, we introduced the cyberlearning external tool, SEP-CyLE, in our CS1 course sections during Spring 2017. Our goal is to validate the idea that the fundamental knowledge in testing is essential for understanding the foundational concepts of programming and thereby write effective programs. The instructor makes no or little changes to the content of her course syllabus or her teaching style to integrate software testing into the CS1 course. The only redesign needed was using the SEP-CyLE materials to integrate testing-related programming assignments using the cognitive load theory. Our hypothesis and hope is that this integration can improve the performance of the students in learning programming languages in comparison with the conventional way of teaching programming courses. Of course, a side effect is an improved understanding of testing concepts and methods.

### 3.2 Research Objectives and Questions

The study consisted of one control group with eight sections of the CS1 course from Fall 2016 semester and one treatment group with eight sections of the same course from Spring 2017 semester. Table 1 shows the details of the students in treatment and control groups used in the analysis.

The objective of this study is twofold. First, we want to examine the impact of combining foundation level programming courses with necessary software testing assignments provided by SEP-CyLE on the overall learning of the students. The second objective is to demonstrate that our integration approach is improving the performance of the students in learning programming languages. Together, these objectives lead to two primary research questions this study tries to address:

**RQ1**: Does the integration of SEP-CyLE have a significant and quantitative impact on the programming and testing knowledge gained by the students?

**RQ2**: Are SEP-CyLE testing-related assignments aligned with the needs and interests of the learners in understanding underlying programming concepts?

### 3.3 Design

The data used in this study was collected from eight sections each in Fall 2016 and Spring 2017 courses of an introduction to programming course (CS1) at a medium-sized public university. Sections in Fall 2016 semester are considered as a control group, and sections in Spring 2017 semester represent the treatment group. This data include: (1) pre/posttest instruments, (2) SEP-CyLE learning materials or objects, (3) SEP-CyLE heuristic survey from the treatment group students and (4) students' final grades from both control and treatment students.

The pre/posttest instrument used in this study and the user survey are available for download at https://stem-cyle.cis.fiu.edu/publications. The pre/postest used to assess how software testing knowledge may help to augment the critical thinking and problem-solving skills of the students as they learn the basic programming constructs. The pretest was administered to the students in the treatment group at week 1 of the Spring 2017 semester and the posttest was administered for the same group at the conclusion of the same semester (week 14). The pretest was administrated prior to being exposed to the learning objects in SEP-CyLE.

To introduce the basic concepts and methods of testing, we selected only eight learning objects that were assigned periodically throughout the course to avoid imposing an unreasonable workload on the students. The first three learning objects, LO1 to LO3, center on an introduction to software testing concepts including software testing definition, verification versus validation, test case development, test case selection, exhaustive testing, and debugging. The subsequent four learning objects, LO4 to LO7, contain learning material about fundamentals of black-box testing and its application at various levels of testing including system-level testing, regression testing, and acceptance testing. The last learning object, LO8, is on white-box testing including integration and unit testing (statement, branch, and path coverage). The SEP-CyLE assignments included eight weekly assignments, each requiring a maximum of two hours of learning.

In addition to all these data, the final grades of students in both groups were collected. The final grades from the control group students are considered as baseline data for comparison. It may be noted that the control group students were not explicitly assigned or instructed in learning any testing concepts or testing assignments. From the treatment group (those that exposed to SEP-CyLE), we used both their final course grades and the data collected from SEP-CyLE assignments for evaluation.

Finally, the user surveys are used for evaluating the usability and effectiveness of this tool from the perspective of the treatment group students.

## 4 RESULTS AND ANALYSIS

Thirty two questions were included in the pre/posttests and are used to assess the programming knowledge along with the related testing knowledge of the treatment group students. We used the same instrument in both pretest and posttest.

The pre/posttest instruments of the treatment group are compared using the post-hoc t-test (two-tailed) and the results are shown in Table 2. The mean differences are computed with 95% confidence interval where the null hypothesis $H_0$ (*that the mean*

**Table 1: Details of Participating Students**

| Class[a] | Enrollment | Pre/Posttest Participation | % | Survey Participation | % | Final Grades Average | SD |
|---|---|---|---|---|---|---|---|
| Treatment Group: CS1 SP17 | 199 | 169 | 84.9 | 169 | 84.9 | 81.68 | 10.58 |
| Control Group: CS1 Fa16[b] | 197 | - | - | - | - | 77.57 | 13.65 |
| Total: | 396 | | | | | | |

[a]SP17 - Spring 2017; Fa16 - Fall 2016
[b]Only Final Grades data used

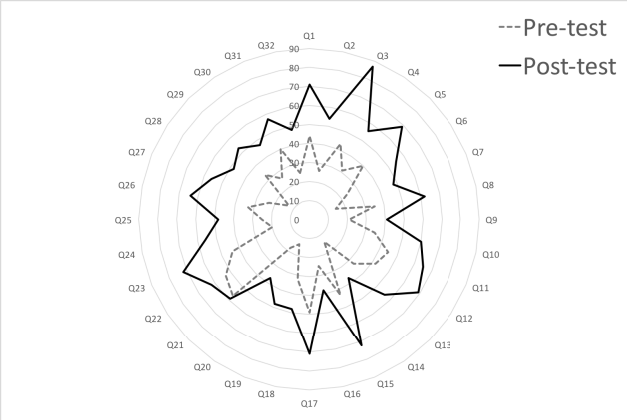


**Figure 1: Pretest and Posttest Scores of Treatment Group Students. The Survey Questions are numbered Q1 - Q32**

*values of pre/posttests of treatment students are the same*) is rejected against the alternative hypothesis $H_1$ (*that the mean values are significantly different between the two tests*).

The average scores of the 32 questions of the pretest and posttest assignments taken by the treatment group students are compared and the results are shown in Fig. 1. Although it is difficult to determine causality, there is clearly a strong relationship between the improved average scores of the posttest and increase in knowledge of software testing gained by the use of SEP-CyLE.

The final grades of all the students in both groups are considered for analysis. As shown in Table 1, it can be noted that the overall final grade of the treatment group students is approximately 4% higher than those of the control group.

To evaluate the relationship between posttest and final grades of the treatment group quantitatively, we used the least squares linear regression method that helps to predict the value of a dependent variable, final grade ($Y$), based on the independent variable, posttest score ($X$). Based on the performance of the posttest (independent variables $x_i$) and the final grades (dependent variables $y_i$) of the treatment group students, we discuss three research questions using the linear regression analysis: (1) what linear regression equation best predicts the programming course performance of the students, based on posttest scores?, (2) if the posttest score of a given student were 80, what final grade would we expect her to make in the course?, and (3) how well does the regression equation fit the data?.

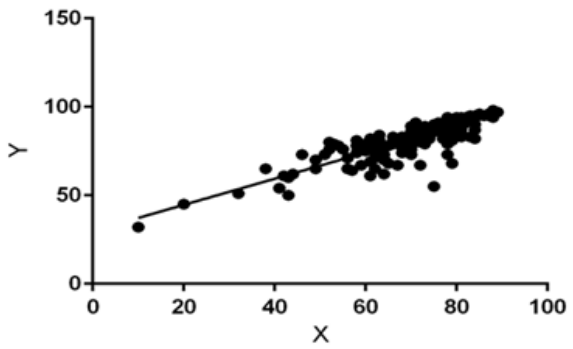


**Figure 2: $R^2$ plot of final grades ($Y$) Vs. posttest scores ($X$) of Treatment Group ($p < 0.0001$)**

Given $x_i, y_i$, and the scores of the posttest and the final exam for 169 students in the treatment group, the regression equation solved for $b_0$ and $b_1$ is of the form: $\hat{y} = b_0 + b_1 x$, where $\hat{y}$ = estimated value for the dependent variable (final grade), $x$ = value of the independent variable (posttest score), and $b_0 = \overline{y} - b_1 * \overline{x}$ and $b_1 = \sum[(x_i - \overline{x}) * (y_i - \overline{y})]/\sum(x_i - \overline{x})^2$. The regression equation that predicts the course performance based on posttest scores is computed as: $\hat{y} = 0.7389 * x + 29.88$. Thus, it can be inferred that, if a student made an 80 on the posttest, the estimated programming course final grade ($\hat{y}$) would be $\hat{y} = 88.99$.

To further determine how well the regression equation fits the data, the coefficient of determination $R^2$ is computed using the following formula: $R^2 = ((1/N) * \sum[(x_i - \overline{x}) * (y_i - \overline{y})]/(\sigma_x * \sigma_y))^2$. Here, the coefficient of determination $R^2$ equal to 0.7201 indicates that about 72% of the variation in final course grades (the dependent variable) can be explained by the relationship to posttest scores (the independent variable). We can consider this as a good fit to the data, in the sense that it would substantially help to predict student performance in the programming course. Fig. 2 shows the fitted $R^2$ plot of final grades ($Y$) versus posttest scores ($X$) of treatment group.

This study takes into consideration a systematic approach to the design of learning materials, so that they present information at a pace and a level of complexity that the learner can fully understand reflecting cognitive load theory [3][22]. The average scores obtained by the treatment group students in the weekly SEP-CyLE

**Table 2: Statistical Validation of mean differences between pre/post-test using Post-Hoc T-Test**

| Number of Participants | Mean diff. | 95% CI | p-value |
|---|---|---|---|
| 169 (Treatment Group) | -22.7813 | [-32.6357, -16.9268]* | 8.5351e-12 |

(*Mean difference is significant at $\alpha < 0.05$)

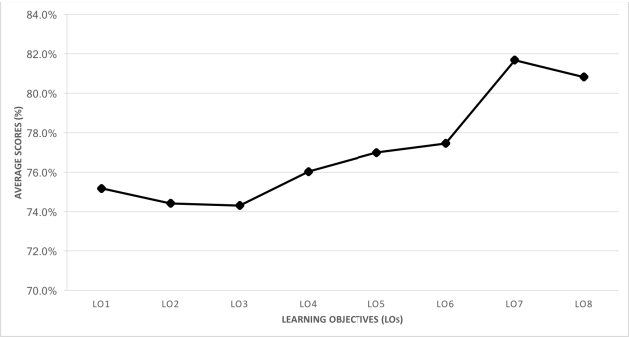


**Figure 3: Performance Evaluation of Treatment Group Average Scores in SEP-CyLE Assignments from LO1 to LO8**

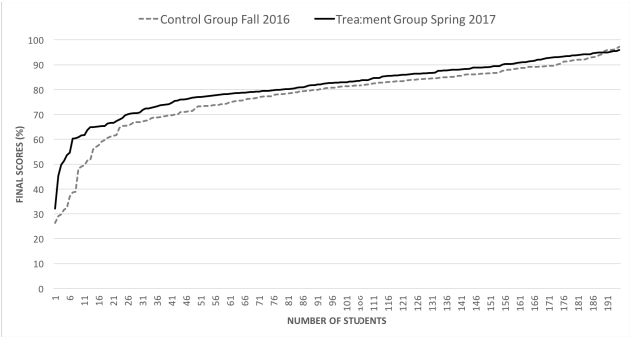


**Figure 4: Comparison of Final Scores of Treatment Group Vs. Control Group**

assignments show a positive trend as depicted in Fig. 3. As shown, the average scores have increased significantly starting from LO1 - Introduction to Software Testing to LO8 - White Box Testing despite the fact the the LOs used generally increase the depth and complexity of software testing concepts.

The testing assignments were carefully designed with the goal of understanding the cost and benefit of this pragmatic approach in improving the final grades of the above-average category of students and simultaneously encouraging the mid- to lower-performing students. The students have appreciated the effectiveness of software testing knowledge to acquire insights while working with various programming constructs such as loops, arrays, and methods. The group activities including debugging, and hand-tracing as part of active-learning strategy received the appreciation of the students' community. To evaluate the effectiveness of imparting software testing knowledge using a variety of activities in enhancing the skills acquired in programming quantitatively, the final scores of the treatment and the control groups are compared. Fig. 4 shows the final scores (in %) of the treatment group (Spring 2017) plotted against the control group (Fall 2016). Identical exams are provided for both groups of students. As a result of evaluation, the average score of all the students in the treatment group was found to be comparatively higher than that of students in the previous semester. The only substantive difference between these two semesters was application of their testing learning and experience in all the assignments and exams.

To further quantify the degree of variation in the scores, the final grades of the treatment and control group students are analyzed. The results reveal the significant differences in the overall performance of the treatment group students as shown in Fig.5. As obvious from this figure, we perceive that the rate of high performers (with Grades A and B) has increased by 17% along with the failure rate (with Grade F) being reduced by 5%. Furthermore, the student feedback from a subsequent study showed a very positive response.

The magnitude of changes between the final grades of the treatment and control groups with means ($\mu$) and standard deviations ($s$) (as shown in Table1 above) of 81.6, 10.58 and 77.57, 13.65 respectively is further quantified using the Cohen's d measure. The Cohen's d measure is computed as follows: Cohen's d = $\frac{\mu_1 - \mu_2}{S_p}$, where $S_p = \sqrt{\frac{s_1^2 + s_2^2}{2}} = 0.330007$. It can be inferred that there is a "*small*" difference between means of the treatment and control group final grades that is considerably more than one fifth the standard deviation.

From the extensive experimental analyses, it is evident that despite an increased workload for the students due to additional assignments on software testing, they can successfully develop a thorough testing knowledge while learning to write programs. Thus, we conclude that it is a better approach to interleave computer programming and software testing concepts. This new strategy does impose an additional cost (graded SEP-CyLE assignments) of sixteen hours of work over the fourteen weeks of the semester.

The user survey analysis results reveal a positive implication of using the SEP-CyLE assignments along with the regular course materials. The survey was taken by 169 students (84.9% of the class enrollment), and the results reveal interesting findings. Around 75% of the students recognize that they have learned the concepts of software testing related to understanding the essential programming constructs at the expense of only a two hours of learning as a web-based assignment. Nearly 74% of the students realize that SEP-CyLE tutorials provided them a comprehensive knowledge of testing concepts. In addition, 71.5% of the students acknowledged that the virtual points awarded inspired them to visit the website more regularly. The usability survey results reveal that the SEP-CyLE website has been a fruitful tool by being a student-centered learning strategy despite the considerable amount of time the students employ in the assignments.

The inferential statistical analysis is performed to evaluate the positive impact of software testing knowledge in learning basic

**Table 3: Statistical Validation of Mean Differences between Final Grades of Treatment Versus Control Groups using Post-Hoc T-Test**

| Number of Participants | Mean diff. | 95% CI | p-value |
|---|---|---|---|
| 396 (Treatment & Control groups) | -5.0977 | [-7.8422, -2.3532]* | 0.0003 |

(*Mean difference is significant at $\alpha < 0.05$)

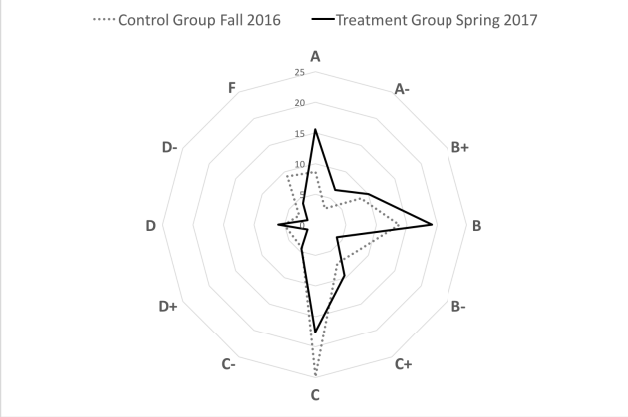


**Figure 5: Performance Evaluation of Final Grades of Treatment Group versus Control Group**

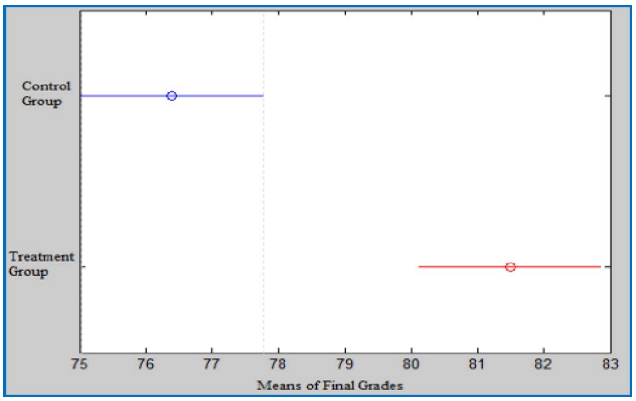


**Figure 6: Significant Difference between the Means of Control and Treatment Groups**

programming courses. The final grades of the treatment group and the control group are compared using the post-hoc t-test (two-tailed). The t-test results computed on the two groups of grades is shown in Table 3. The mean differences of the final grades of the treatment and control groups are computed with 95% confidence interval. The null hypothesis $H_0$ (*that the mean values of final grades of treatment and control group students respectively are the same*) is rejected against the alternative hypothesis $H_1$ (*that the mean values are significantly different between the two groups*). The actual differences in means of the final grades of treatment and the control groups are shown in Fig. 6.

These statistical results validate the fact that integration of software testing in programming courses in early computer science and software engineering education is effective in improving the performance of the students in acquiring the basics of programming principles better. Furthermore, the results validate the principles of cognitive load theory. That is, the results demonstrate that an underlying knowledge of testing (obtained through the web-based learning experience) has allowed the students to quickly move programming practices into working memory on demand. In summary, all these empirical results show that the additional knowledge of basic software testing helps students to learn the insights of programming language fundamentals more effectively.

## 5 THREATS TO VALIDITY

In this paper, we presented an empirical study at a medium-sized public university. We aimed at improving software testing and programming knowledge of computer science and software engineering students from undergraduate level by using an external web-based Software Engineering and Programming CyberLearning Environment (SEP-CyLE). The dataset were obtained from (1) a pre/postest instruments, (2) SEP-CylE LOs, (3) a users experience survey and (4) students' final scores. While the pre/posttest, the user survey, and LOs results were used to measure the knowledge gained just in the treatment group, the students' final scores of both treatment and control groups were used for performance evaluation study.

The major threat to the result's validity comes from studying the pre/posttest results only in the treatment group. As shown in Table 1 above, the control group students did not participate in pre/posttest during the Fall 2016. This situation can be explained through the designing process we chose to conduct this study. We used the pre/postest results of treatment group to answer the first research question **RQ1**, as shown in Fig. 1 and Fig. 2 above. Additionally, we used the students' final scores of both control and treatment groups to answer the second research question **RQ2**, as shown in Fig. 3, Fig. 4, and Fig. 5 above. Furthermore, while R-squared presents an assessment of the strength of the correlation between the posttest and the final grades, it does not provide a formal hypothesis test for this relationship. We have planned to extend this study by considering many other instruments such as an exploratory usability survey.

## 6 CONCLUSION

The empirical results of the initial investigation on the effects of combining testing with computer programming demonstrate the ability of the treatment group students to perform better in the programming courses than the control group students who were not provided any additional knowledge of testing. The assertion that the treatment group students learned programming concepts

better by acquiring testing knowledge is well established from the performance evaluation of the final grades. Further, the outcomes from this study reveal a positive relationship in reinforcing software testing concepts gradually in CS1/CS2 courses. This study will be extended to leveraging a web-based learning environment that makes learning less traumatic and more interesting. The conclusion of this initial investigation is that this pragmatic approach not only helps students to learn essential software testing strategies along with programming but also inspires the students in learning software testing courses.

In the near future, we plan to perform additional studies at different academic institutions to validate our findings and experience.

## 6.1 SUPPLEMENTARY MATERIALS

The data used in this paper was collected using a pre/posttest instrument and the SEP-CyLE survey. The pre/posttest instrument used in this study and the user survey are available for download at https://stem-cyle.cis.fiu.edu/publications.

## REFERENCES


[1] Hakam W. Alomari, James D. Kiper, Gursimran S. Walia, and Katharine Zaback. 2017. Using Web-Based Repository of Testing Tutorials (WReSTT) with a Cyber Learning Environment to Improve Testing Knowledge of Computer Science Students. In *124th ASEE Annual Conference Exposition (Computers in Education Division)*. ASEE. https://doi.org/10.1145/971300.971312

[2] Ellen Francine Barbosa, José Carlos Maldonado, Richard LeBlanc, and Mark Guzdial. 2003. Introducing testing practices into objects and design course. In *Software Engineering Education and Training, 2003.(CSEE '03). Proceedings. 16th Conference on*. IEEE, 279–286.

[3] Ellen Francine Barbosa, M. Silva, C. Corte, and J Maldonado. 2008. Integrated teaching of programming foundations and software testing. In *Proceedings of the 38th Annual Frontiers in Education Conference (FIE '08)*. IEEE, NY, USA, 5–10. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.690.19&rep=rep1&type=pdf

[4] R. Chang-lau and Peter J. Clarke. 2017. *Web-based repository of software testing tutorials a cyberlearning environment (WReSTT-CyLE)*. http://wrestt.cis.fiu.edu/.

[5] R. Chang-lau and Peter J. Clarke. 2018. *Software engineering and programming cyberlearning environment (SEP-CyLE)*. https://stem-cyle.cis.fiu.edu/instances.

[6] Wei Kian Chen and Brian R. Hall. 2013. Applying Software Engineering in CS1. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. ACM, New York, NY, USA, 297–302. https://doi.org/10.1145/2462476.2462480

[7] Peter J. Clarke, Andrew A. Allen, Tariq M. King, Edward L. Jones, and Prathiba Natesan. 2010. Using a Web-based Repository to Integrate Testing Tools into Programming Courses. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA '10)*. ACM, New York, NY, USA, 193–200. https://doi.org/10.1145/1869542.1869573

[8] Chetan Desai, David S. Janzen, and John Clements. 2009. Implications of Integrating Test-driven Development into CS1/CS2 Curricula. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)*. ACM, New York, NY, USA, 148–152. https://doi.org/10.1145/1508865.1508921

[9] Stephen H. Edwards. 2003. Rethinking Computer Science Education from a Test-first Perspective. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '03)*. ACM, New York, NY, USA, 148–155. https://doi.org/10.1145/949344.949390

[10] Stephen H. Edwards. 2004. 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)*. ACM, New York, NY, USA, 26–30. https://doi.org/10.1145/971300.971312

[11] Sebastian Elbaum, Suzette Person, Jon Dokulil, and Matt Jorde. 2007. Bug Hunt: Making Early Software Testing Lessons Engaging and Affordable. In *Proceedings of the 29th International Conference on Software Engineering (ICSE '07)*. IEEE Computer Society, Washington, DC, USA, 688–697. https://doi.org/10.1109/ICSE.2007.23

[12] Michael H. Goldwasser. 2002. A Gimmick to Integrate Software Testing Throughout the Curriculum. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE '02)*. ACM, New York, NY, USA, 271–275. https://doi.org/10.1145/563340.563446

[13] David Janzen and Hossein Saiedian. 2008. Test-driven Learning in Early Programming Courses. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)*. ACM, New York, NY, USA, 532–536. https://doi.org/10.1145/1352135.1352315

[14] David S. Janzen and Hossein Saiedian. 2006. Test-driven Learning: Intrinsic Integration of Testing into the CS/SE Curriculum. *SIGCSE Bull.* 38, 1 (March 2006), 254–258. https://doi.org/10.1145/1124706.1121419

[15] Edward L. Jones. 2000. Software Testing in the Computer Science Curriculum – a Holistic Approach. In *Proceedings of the Australasian Conference on Computing Education (ACSE '00)*. ACM, New York, NY, USA, 153–157. https://doi.org/10.1145/359369.359392

[16] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A Study of the Difficulties of Novice Programmers. *SIGCSE Bull.* 37, 3 (June 2005), 14–18. https://doi.org/10.1145/1151954.1067453

[17] Juan Jenny Li and Patricia Morreale. 2016. Enhancing CS1 Curriculum with Testing Concepts: A Case Study. *J. Comput. Sci. Coll.* 31, 3 (Jan. 2016), 36–43. http://dl.acm.org/citation.cfm?id=2835377.2835384

[18] Andrew Patterson, Michael Kölling, and John Rosenberg. 2003. Introducing Unit Testing with BlueJ. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '03)*. ACM, New York, NY, USA, 11–15. https://doi.org/10.1145/961511.961518

[19] McDaniel A Popkess AM. 2004. Are nursing students engaged in learning? A secondary analysis of data from the National survey of student engagement. *Nursing Education Perspectives* 32, 2 (2004), 89–94. https://www.ncbi.nlm.nih.gov/pubmed/21667789

[20] Stephen Schaub. 2009. Teaching CS1 with Web Applications and Test-driven Development. *SIGCSE Bull.* 41, 2 (June 2009), 113–117. https://doi.org/10.1145/1595453.1595487

[21] Terry Shepard, Margaret Lamb, and Diane Kelly. 2001. More Testing Should Be Taught. *Commun. ACM* 44, 6 (June 2001), 103–108. https://doi.org/10.1145/376134.376180

[22] John Sweller. 1988. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science* 12, 2 (April 1988), 28:257–28:285. https://doi.org/10.1207/s15516709cog1202_4


## ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under grants DUE-1225742 and DUE-1525112. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.