Teaching Software Testing in Introductory CS Courses and Improving Software Quality

Syed M Rahman and Akram Salah

Department of Computer Science, North Dakota State University
258 IACC Building, Fargo, North Dakota 58105, USA

Abstract. Undergraduates in computer science typically begin their curriculum with a programming course or sequence. Many researchers found that most of the students who complete these courses, and even many who complete a degree, are not proficient programmers and produce code of low quality. In this paper, we have addressed this problem by proposing a *cultural shift* in introductory programming courses. The primary feature of our approach is that software testing is presented as an integral part of programming practice; specifically, a student who is to write a program will begin by writing a test suite. Our initial results show that this approach can be successful. Teaching basic concepts how to test a program and writing test cases do not take much time, it helps beginning students to understand the requirements, and it helps them produce better-quality code.

1 Introduction

An industry survey [2] reported that more than 50% of a software project's budget is spent on activities related to improving software quality. Industry leaders claimed that this is caused by the inadequate attention paid to software quality in the development phase. Another multi-national, multi-institutional [1] assessment showed that students who completed one or two computer-programming classes' on average scored only 22.89 out of 110 points on the general evaluation criteria. Universities in USA, Canada, and elsewhere found that 50% of students failed, withdrew or earned D-grades in introductory programming courses [3, 4]. These disappointing and alarming research results concluded that many students do not know how to program at the end of their introductory programming courses.

In this paper, we have addressed this problem and proposed two different models for two introductory programming courses. Our initial finding shows that our approach can be successful. We run the experiment in the Department of Computer Science. In our first model, CS-I (Introduction to Java Programming) students write test cases as a prerequisite of writing programs. Students learn how to write test cases and how to test their own code. Students draw context diagrams, answer a few general questions, write test suites before writing code, and submit all these to the instructor. After writing code, students execute their test cases and submit test results and the test program with the main program. In our second model, CS-II (Data Structure using Java), students apply Test-driven Development (TDD) or test-first programming as a

M Rahman S. and Salah A. (2005).
Teaching Software Testing in Introductory CS Courses and Improving Software Quality.
In *Proceedings of the 3rd International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 126-128
DOI: 10.5220/0002577201260128

software development and testing methodology. In TDD, one always writes test cases before adding new code. It promotes incremental development and gives students a great degree of confidence in the correctness of their code, helps them understand the requirements and design better, makes it easier to change requirements, and helps to build reusable code [5].

2 Evaluation Procedure

2.1 Teaching Software Testing

In the CS-I classroom, we explained how to write test cases. We spent only 25 minutes and showed one example of how to write test cases. Right after the presentation, we provided a similar problem and asked students' to write test cases. More than 70% of the students came up with test cases. Of course, they did not know in detail about software testing or its different techniques.

2.2 Measuring Program Quality

In our experiment, the same instructor taught both section-I and section-II of CS-I class. We collected students' projects for both sections. Section-I students did not follow our approaches. Section-II students followed our approaches and submitted test suites one week earlier than their final submission. We created a test suite following different testing techniques such as boundary value analysis and equivalence partitioning. We executed all test cases in all students' programs in section-I and section-II. Executing a black box test suite, we found that our model-1 was effective.

2.3 Conducting Surveys and Interviews

In our approach, students' involvement plays a vital role. We conducted a pre-test and post-test survey of students' understanding and achievement from the experiment. We took several in depth interviews to faculties who teach computer classes.

3 Results

In this paper, we have proposed two models in two introductory programming classes to improve software quality. Our approach is to make a *cultural shift* in teaching programming languages by making testing an integral part of programming practices. Students not only need to produce correct output but also needs to understand how to test their programs. In our opinion, if you know how to write the program then you better know how to test it and make sure that your program is doing what you expected to do.

We spent only 25 minutes teaching students how to write test cases, and more than 70% of the students came up with test cases in the classroom. We collected students' projects and measured the quality of the program by applying the same test suites to both sections' code. Student feedback was very positive about our model.

We have not completed the experiment in this semester yet. We will run the same experiment again in next semester. However, our initial finding shows significant improvements in students' program quality. We found teaching basic concepts and terminology of software testing, does not take much time. Our approach helps students to understand the problem better. Students like testing their code and it boost their confidence.

4 Conclusions

In this paper, we proposed two different models for two introductory programming classes. In the first model, course CS-I students would get a preliminary idea about how to write test cases and test program. In our experiment, the same instructor teaches the same course in two different sections. We eliminated many factors that vary from instructor to instructor. One section instructor followed our model used testing as a prerequisite of writing code and other section did not. Keeping all factors the same, we measured how our model can play a role in improving programming quality.

Our initial experimental result shows that our model can be successful. We found that our approach helps students to understand the problem and improves software quality. We also found that teaching basic concepts of software testing to beginner students does not take much time, and student feedback was very positive.

References

1. McCracken, M., Almstrum, V., Laxer, C., and others: A Multi-national, multi-institutional study of assessment of programming skills of first year CS students, volume 33, issue COLUMN: ITiCSE 2001 working group reports, Pages: 125 – 180, 2001
2. Townhidnejad, Hilburn: Software quality: a curriculum postscript?, Technical Symposium on Computer Science Education, Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, Austin, Texas, United States Pages: 167 – 171, 2000
3. Nagappan, N., William, L., Ferzil, M., Wiebe, E., Yang, K., Miller, C., and Balik, S.: Improving the CS1 Experience with Pair Programming. Proceedings of the 34th SIGCSE Technical Sympoisum on Computer Science Education, Reno, Nevada, USA, March 2003.
4. Hermann, N., Popyack, J., Char, B., Zoski, P., Cera, C., and Lass, R.N.: Redesigning computer programming using multilevel online modules for mixed audience. Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, Reno, Nevada, USA, March 2003.
5. Edwards, E.: Using test-driven development in the classroom: Providing students with concrete feedback on performance. In Proceedings of the EISTA'03, August 2003