A PRELIMINARY REPORT ON IMPROVING STUDENT MOTIVATION AND PERSISTENCE IN COMPUTER PROGRAMMING COURSES WITH SOFTWARE INSPECTION

Rajendran Swamidurai, Uma Kannan
Alabama State University/Auburn University
Department of Mathematics and Computer Science/ Department of Computer Science and Software Engineering
rswamidurai@alasu.edu/uzk0002@auburn.edu

Abstract

In recent years, due to aging workforce, and growth of new fields such as cloud computing, cybersecurity, big data and mobile networks, the demands for the computer scientists are tremendous. Exacerbating the current problem is the anticipated increase in demand for computing and STEM professionals in the near future. Employment of computing profession is expected to increase up to 27 percent over the next decade, making it one of the fastest-growing professions in the country. The enrollments in computing majors have reached the peak levels seen at the end of the dot-com era after a precipitous decline that lasted almost a decade. Though enrollments in computing majors have started climbing, the attrition is still a problem. Indeed, if left unaddressed, high attrition rates could easily negate these enrollment gains. Various reports indicate that the attrition rates in computing courses are as high as 30% nationally. At Alabama State University, we have seen attrition rates similar to the national trends and the failure rate of various computer programming courses are very high compared to non-programming courses for the computer science sophomores. Motivating and engaging students in computer programming courses are vital tools to address the above said attrition problem and retain students in computing majors. In this paper, we describe the design and implementation of a flipped classroom model that incorporates software inspection to motivate and engage students in computing, and then present the results of an evaluation of the model.

1. Introduction

In recent years, due to *aging workforce* [1], and *growth of new fields* such as cloud computing, cybersecurity, big data, and mobile networks [2], the demand for the computer scientists are tremendous. Exacerbating the current problem is the anticipated increase in demand for computing and STEM professionals in the near future. Employment of computing profession is expected to increase up to 27 percent from 2014 to 2024, making it one of the fastest growing professions in the country [2]. Difficulties in finding skilled workers; Growth of cloud computing, cybersecurity, and mobile networks; Increased data needs of companies; Need to create innovative solutions to defend cyber networks; Demand for computer software; The growing popularity of mobile devices and ecommerce are some of the reasons cited behind this demand [2].

The enrollments in computing majors have reached the peak levels seen at the end of the dot-com era after a precipitous decline that lasted almost a decade (2000 to 2009). The Taulbee Survey shows an approximate 61 percent increase in enrollment at U.S. computing majors between 2010-2011 and 2013-2014. [3]

Though enrollments in computing majors have started climbing the *attrition*, the dropout and failure rates, are still a problem. Indeed, if left unaddressed, high attrition rates could easily negate these enrollment gains. Various reports indicate that the attrition rates in computing courses are as high as 30% nationally [4,5] and most of the attrition happens during freshman and sophomore years [4]. At Alabama State University, we have seen attrition rates similar to the national trends. Our STEM sophomore classes continued to have an average failure rate of about 30% to 35% and the failure rate of various computer programming courses are very high compared to non-programming courses for the computer science sophomores. This average rises to even more alarming rate when we include those students who earned a D and therefore these attrition rates are affecting a significant number of potential STEM graduates.

Per-course attrition rates are, of course, only one part of the picture. Student persistence, viz. remaining in and graduating from a STEM degree program, is the second and equally alarming facet of this problem. National estimates show that less than half of those students who begin in a STEM major graduate [6].

All of these results call for a fundamental rethinking of undergraduate computing education with systematic innovative approach to teaching and learning techniques, which enable students to become more proficient in programming, problem-solving and reasoning skills. Engaging and effective approaches to teaching computer programming courses from the very first computer programming courses that majors encounter are critical to reversing these negative trends. In this context, this paper presents a study to reduce the high attrition rates found in computer programming courses by using software inspection in a flipped classroom environment.

Software inspection is a static testing method used to verify that the software satisfies its requirements [7], and used to find errors in design and code [8]. In addition to finding 60-90 percent defects in design and code, the feedback provided by the inspection process helps programmers in avoiding injecting defects in their future work [7]. Evidence [7] shows that programmers who participate in the inspection of their own code actually create fewer defects in their future work.

2. Software Inspection

Software inspection, also known as Fagan Inspection, is a static testing method used to verify that the software satisfies its requirements [7], and proven industry best practice for finding defects in design and code [8]. Software inspection is a formal rigorous review method, which is more effective than any other error-removal strategy, including testing [9]. It commonly removes up to 90 percent of defects from a software product before the first test case is run [7,9].

2.1. The Inspection Team

According to Fagan [8], the circumstance will dictate the team size; but a team of four developers is a good-sized inspection team: one moderator, one author, and two testers.

- *Moderator:* The moderator is the key person in the inspection process. The moderator must be a competent programmer. He should be a specially trained player-coach and jobs include manage inspection team, offer leadership, inspection meeting scheduling, reporting inspection results, and follow-up the rework.
- *Author/Coder:* The author of the source code being inspected.
- *Tester:* The developer who writes and/or executing the test cases or testing the author's code.

2.2. The Inspection Process

The Fagan inspection process is described below [8, 10]:

- *Overview (whole team):* This is an optional phase for code inspection and a mandatory phase for the design inspection. The designer first describes the overall design and then specific detail design that he has designed through logic, paths, dependencies, etc. At the conclusion, the design documentation is distributed to the participants.
- *Preparation (individual):* It is the homework process in which the participants try to understand the intent and logic of the design document(s). In order to increase their error detection process, the participants should study the recent inspections ranked distribution of error types and checklists of clues on finding these errors.
- *Inspection (whole team):* First the reader, usually the coder, describes his implementation plan for the given design. After the design is understood, the reader paraphrases the code line by line and each member of the inspection team has an opportunity to ask for clarification or point out a defect in the current item. Once an error/defect is identified, the recorder wrote down the defects, and after the moderator classified the error type and severity (major or minor), the inspection process will continue.
- *Rework (author/coder):* All defects/errors noted in the inspection report are resolved by the coder.
- *Follow-Up (moderator):* It is the responsibility of the moderator to ensure that all defects/errors found in the inspection process must be resolved by the code/author. If more than 5 percent of the material has been reworked then a reinspection is necessary, otherwise the moderator can verify the quality of the rework by himself or request for a re-inspection.

3. Implementing Software Inspection in a Computer Programming Course

The CSC 211- Programming Concepts, Standards, and Methods course at Alabama State University is a typical CS1 course, focusing on basic computer programming concepts. The instructional language is C++ and the instructional IDE is jGRASP (jgrasp.org). CSC 211 is a 4 credit hour course and meets for 2 clock hours of lecture per week and 2 clock hours of lab per week. All students meet together for the same lecture, but they meet separately in small lab sessions. The course is normally offered in every semester and each class has no more than 30 students.

We have adapted the Fagan's inspection process described in [10]. Unlike other software inspection processes, the clear structure the Fagan inspection process enables us to study and conduct the inspection process in an effective manner. The inspection was accomplished by a team of seven people (because of the class size), a *moderator* and 6 students. The course instructor worked as the moderator, and one student acted as a recorder, one student acted as a reader, going through the code one line at a time, and the remaining 4 students acted as testers.

The inspection was preceded by a *preparation period* of 15 minutes. The preparation process consists of two activities: a *group overview* in which the moderator gave an overview of the requirements and the author of the source code was asked to provide an overview of his/her design, and an *individual study*, in this time each inspector individually analyzes the source code in order to become familiar with it. After each inspector completed their preparation, the team then met to inspect the program. Each inspection session lasted about an hour. During the inspection, the reader went through the code one line at a time and each member of the inspection team has an opportunity to ask for clarification or point out a defect in the current item. The recorder wrote down the defects.

The recorded defects were classified as *minor* or *major* by the moderator/instructor. A minor defect could be a syntax error such as a missing semicolon in code and a major defect could be a failure to implement a requirement either through logic error or omission [10]. The minor defects were turned over to the author of the source code for rectification. The major defects were noted down by the instructor for going through or explain the concept(s) behind the logic again in the class.

4. Results

To assess the true effectiveness of the software inspection based instructional model, we empirically compared the inspection-based offerings of the CS1 with a traditional (non-inspection) offering of the same course. To facilitate a rigorous experimental comparison, we compared the CS1 course offered in the traditional format during the fall 2011 semester and then in the inspection-based format during the summer 2015 semesters by the same instructor.

The data collected in both semesters includes, pre/post course content tests, all graded items from the course, and the exit interviews of selected students from the top, middle, and bottom of the class. The full analysis of this data is not yet complete, and for this paper we will focus only on the evaluation of student performance, that is their scores on the exams and programming assignments. The t-test data is summarized in Figure 1.

As we examined the graded items from the course (exams and programming assignments), we saw significantly higher performance from students who took the inspection-based offering. The average score in the inspection-based semester was 81, while the average score in the traditional semester was only 61. All Computer Science courses in all three semesters (Fall, Spring, and Summer) at Alabama State University are taught by regular instructors only and there is no difference in course duration as well. Due to the shortage of instructors, some courses are offered only during summer terms of some years and the students who enrolled these courses are regular students. Since there is no difference in instructor, course duration and nature of students

between fall and summer semesters, this difference of 20 percentage points could be only due to the difference in instructional mode (traditional vs. inspection-based) of the course. Though the t-test result is not statistically significant, the increase in 20 percentage points suggests that students in the inspection-based offering were able to perform at a higher level on exams than the students in the traditional offering of the course.

Unpaired t test results

P value:

The two-tailed P value equals 0.2247

Confidence interval:

The mean of Inspection minus Traditional equals 19.5500

95% confidence interval of this difference: From -14.7168 to 53.8168

Intermediate values used in calculations:

t = 1.3156

df = 8

standard error of difference = 14.860

Review of Data:

Group	Inspection	Traditional
Mean	80.8000	61.2500
SD	9.1700	35.6800

Figure 1: Summary of student performance measures.

5. Summary

We have adapted a software inspection-based learning model for CS1 course and evaluated its effectiveness. In our model of inspection-based learning, students get experience in solving computational problems, critically analyzing each other's solutions in inspections, and reflecting on and learning from these collaborative sessions over the course of a semester. A formal evaluation in which the inspection-based learning model was compared to a traditional offering of a CS1 course revealed that the inspection-based learning model offered significant benefits to students in terms of both course content mastery and programming achievement. We are quite optimistic that the inspection-based learning model will prove to be an effective approach to reinvigorating computing education.

6. References

1. Industries Studies 2006 Report, Industrial College of the Armed Forces (ICAF), National Defense University.
2. Bureau of Labor Statistics, U.S. Department of Labor, "Occupational Outlook Handbook", 2016-17 Edition.
3. Lecia Barker, Tracy Camp, Ellen Walker, and Stu Zweben, "Booming Enrollments – What is the Impact?" Computing Research News, Computing Research Association, MAY 2015, VOL. 27/NO.5
4. Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for computer science students: Some thoughts and observations. ACM SIGCSE Bulletin, 37(2), 103-106.
5. Guzdial, Mark, and Elliot Soloway. "Teaching the Nintendo generation to program." Communications of the ACM 45.4 (2002): 17-21.
6. Xianglei Chen and Matthew Soldner, STEM Attrition: College Students' Paths Into and Out of STEM Fields: Statistical Analysis Report, NCES 2014-001, U.S. Department of Education
7. Fagan, Michael E. "Advances in Software Inspections," IEEE Transactions on Software Engineering, Vol. 12, No. 7, Jul. 1986, pp. 744-751.
8. Fagan, Michael E. "Design and Code Inspections to Reduce Errors in Program Development, "IBM Systems Journal, Vol. 15, No. 3, 1976, pp. 182-211.
9. Robert L. Glass, Frequently forgotten fundamental facts about software engineering, IEEE Software 2001.
10. James E. Tomayko and James S. Murphy, Materials for Teaching Software Inspections, CMU/SEI-93-EM-7, 1993