IMPROVING CONCEPTUAL UNDERSTANDING OF CODE WITH *BUG FIXER* *

Lisa Reynolds, Quentin Mayo, David Adamo, and Renee Bryce
Department of Computer Science and Engineering,
University of North Texas
Denton, TX 76207
(940) 565-2767
{Lisa.Reynolds2, QuentinMayo, DavidAdamo}@my.unt.edu, renee.bryce@unt.edu

ABSTRACT

Bug Fixer is a web-based application that complements lectures with hands-on exercises that encourage students to better understand logic in programs. Students view code with several bugs that they must fix. The process of fixing the bugs forces students to conceptually think about the code and reinforces their understanding of the logic behind algorithms. We conducted a study with undergraduate students at University of North Texas using our system as part of their CS2 classes. *Bug Fixer* exercises helped some students to improve their conceptual understanding of the algorithms, but we experienced a large variation in this result from week to week. Many students enjoyed using **Bug Fixer** and recommend the system for future use. We observed a slight increase in the passing grades of students who participated in our study compared to students in other sections of the course with the same instructor who did not use *Bug Fixer*. The students who did not report a positive experience provided comments that we plan to address in future work. Some of these comments included suggestions such as giving hints and providing more performance feedback throughout the sessions.

INTRODUCTION

Computer Science students must understand many concepts in order to implement solutions to problems. Students that pass introductory programming courses typically

---

move beyond syntactic errors, but many still struggle with logic errors [3]. Fixing logic errors usually requires students to trace and conceptually understand code. For instance, Lopez et al. [5] report that there is a relationship between code tracing skills and the ability to write conceptually accurate code. In this paper, we develop and examine a web app, *Bug Fixer*, with the goal to help students trace code and improve their conceptual understanding of algorithms. We run weekly Bug Fixer sessions with problem sets that are related to the weekly course material and homework assignment in order to reinforce the material.

Background

Bug Fixer attempts to improve students' conceptual understanding of algorithms through the use of weekly exercises in which students modify code to fix bugs. Each *Bug Fixer* problem set relates to their weekly lecture topic and assignment. Many others have explored novel approaches for similar goals. Odekirk-Hash et al. [6] present the InSTEP tutoring system for beginning C programmers. The tutoring system provides automated feedback to beginning programming students. They found that students who used InSTEP spent less time asking teaching assistants for help compared to others who did not use the system. InSTEP succeeds in minimizing the need for one-on-one interactions with students in an introductory programming class. Levitin et al. [4] use puzzle-like problems to teach design and analysis of algorithms. Puzzles help students think about algorithms on a more abstract level. However, it also deprives them of much needed hands-on experience with the computer programming skills required to properly understand and implement algorithms in a particular programming language. Tillman et al. [9] present an interactive-game-based teaching and learning platform called Pex4Fun. The platform provides automatic grading of programming assignments using automatic test generation, symbolic execution, and automated feedback. Pex4Fun automatically generates test cases and compares the output from a student's implementation to that of a secret correct implementation. The student solves a programming problem based on the results from test cases. Pex4fun begins with a template but enables users to edit code. In comparison, our *Bug Fixer* approach is more constrained by removing some of the freedom available in Pex4fun because we highlight specific lines of code seeded with logic errors. *Bug Fixer* encourages students to trace through the code and fix specific lines of code. Stiller's [8] *bricolage* approach for introductory students focuses on student experimentation and exploration. Students gain familiarity with a program and then try to adapt the given program to achieve a new specified behavior. *Bug Fixer* closely resembles the *bricolage* approach. However, our aim is to improve students' conceptual understanding of algorithms through the process of code tracing and debugging. Students trace through the code of an incorrect algorithm implementation and modify specific lines of code in attempt to correct it and change its behavior.

Delev et al. [1] implement the Code system that provides a special environment for students to write code and test their solutions. Their system logs student attempts and allows them to closely analyze their students. Students that participate in their experiments generally show a better understanding of material. Further, the system makes it easy for them to detect cheating and intervene quickly. Our system differs as we give problem sets in a competitive lab setting where students all view the same code and try

to fix bugs. Bishop et al. [2] created *Code Hunt*. This system is similar to *Bug Fixer* in that they focus on problem solving in a competitive environment. Rather than providing requirement specifications like *Bug Fixer*, they provide unit tests for students to work from as they solve puzzles.

BUG FIXER OVERVIEW

Student Functionality

Bug Fixer is open source and available for download [7]. Figure 1 shows a student

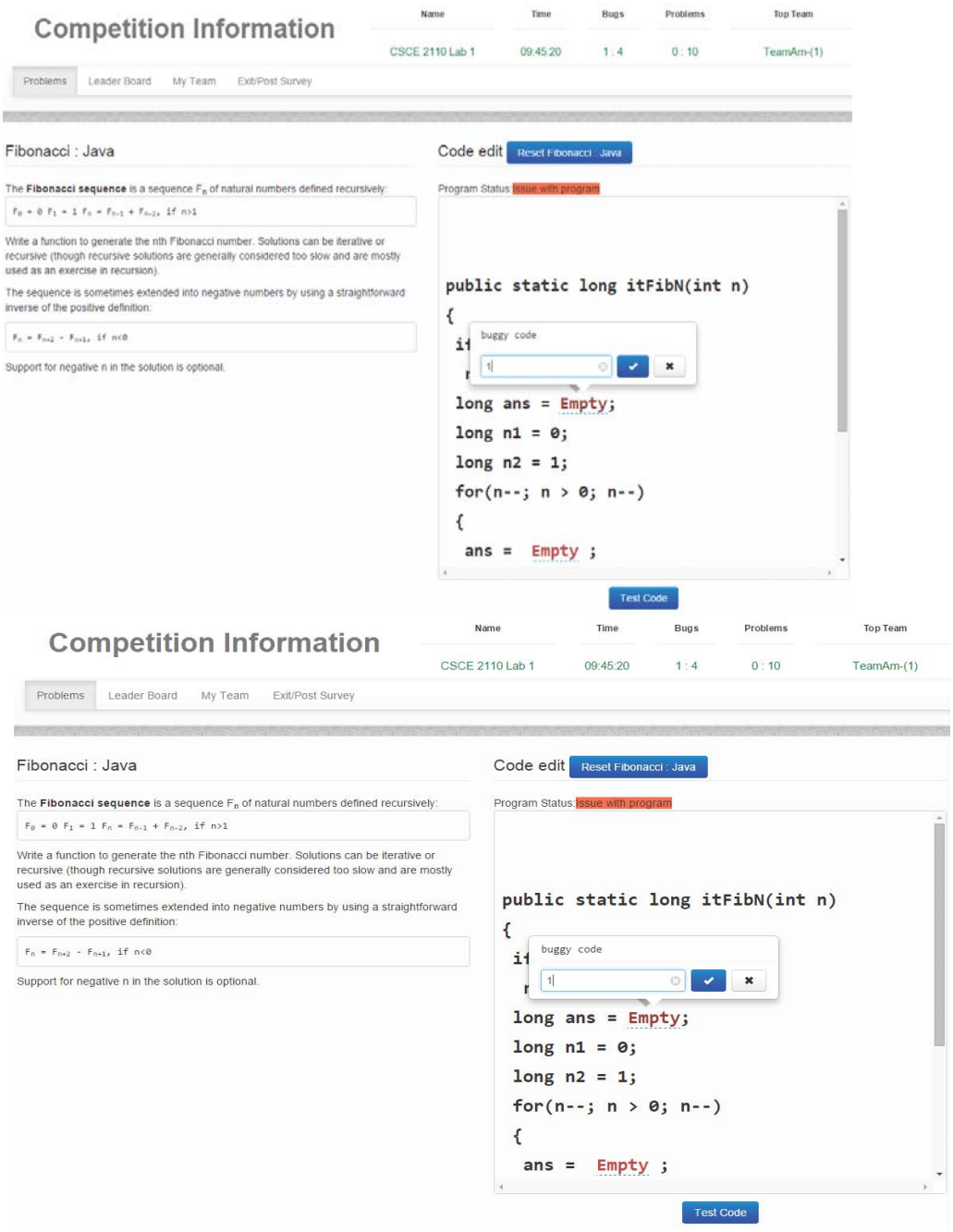

Figure 1: Main screen of algorithm fixing competition in Bug Fixer.

using *Bug Fixer* for a Fibonacci problem that has been seeded with logic errors. The header displays a *timer*, *total number of errors fixed* and *the participant who has corrected the largest number of bugs*. Below the header, the left side shows an explanation of how a correct implementation of the algorithm should work. The right side shows the buggy code with editable areas that enable students to change specific lines of code in an attempt to fix the algorithm. Students may click on the lines that are in a bold red and underlined font to edit code. A box to edit the line of code will appear. Students need to understand the logic of this code in order to fix the bugs. In the *Bug Fixer* environment, once students modify the code and choose to "Run Code," it will tell them if the test cases return the same output for their code and notify the user whether they found a bug.

Table 1 shows sample code that was seeded with logic bugs for the Fibonacci problem. The problem set includes four logic bugs that require the student to trace the code and conceptually understand the code in order to fix the bugs.

Buggy Code for Fibonacci Java	Correct Implementation for Fibonacci

public static long itFibN(int n){
if (n <4)
 return n;
long ans = 0;
long n1 = 1;
long n2 = 2;
for(n--; n > 0; n++){
 ans = n1 - n2;
 n1 = n1+n2;
 n2 = ans;}
return ans;}
	public static long itFibN(int n){
	if (n < 2)
	 return n;
	
	long ans = 0;
	long n1 = 0;
	long n2 = 1;
	for(n--; n > 0; n--){
	 ans = n1 + n2;
	 n1 = n2;
	 n2 = ans;}
	return ans;}

Table 1: Sample problem set for Bug Fixer with four bugs that are in red font and underlined.

Administrator Functionality

We do not show the administrator screen due to space constraints. The interested reader is encouraged to download Bug Fixer [7] to see this part of the system. We provide administrator features to allow students to work individually or in teams. Administrators may add new problem sets and bugs to the system or assign pre-existing ones. We specify a difficulty level for each problem so that administrators can tailor competitions to the experience level of the participants. They can also control the amount of information available to the students such as showing the bugs remaining in each problem. Bug Fixer may be used as a stand-alone activity. We also implemented it so that it may be used for competitions. Administrators can set a time limit for each competition. Bug Fixer also provides a scoreboard of student rankings based on the time that it takes them to solve the problems.

EXPERIMENTS

Research Questions

We examine the following research questions:

- **RQ1:** *Do the students report an increased understanding of the course topics?*
- **RQ2:** *Do the students show improvement as measured by final grades?*
- **RQ3:** *Do the students recommend this exercise for future students?*

Problem Sets

We conducted an experiment with students from the CS2 course at the University of North Texas during a weekly recitation session for 12 weeks. Table 2 summarizes the course topics that we incorporated into *Bug Fixer* problem sets. The problem sets contain 14 to 153 lines of code (LOC) and 3 to 11 bugs. We do not provide the problem sets in this paper due to space constraints. However, they are available for download [7].

Lab Topic	LOC	# Bugs
N/A (first week)	37	3
Structures	73	4
Pointers	73	3
Recursion	14	5
Divide and Conquer and Hashing	34	9
Type Manipulation	89	5
Lists Comprehension	103	7
String Comprehension	56	7
Dates Calculations	153	11
Encryption	69	6
Letter Pairs	124	5
Permutation	222	5
C++ Input/output	132	5
Classes and Objects	116	5

Table 2: Bug Fixer Competition Descriptions for CS2 Spring 2015.

RESULTS

RQ1: *Do the students report an increased understanding of the course topics?*

Students were asked to rate their understanding of the course topics each week in a pre and post-activity survey. Table 3 shows that on a weekly basis, a small number of

students reported that the *Bug Fixer* exercises increased their understanding of the course material each week. In the lowest weeks, only 1% of the students reported an increased understanding of course material. We attribute this in part to the difficulty of the material and believe that we should have broken the exercises into smaller logical pieces. During the best week, 29% of students reported an increased understanding of course material. We observe that future work will require redesigning many of our problem sets to be more focused on specific concepts with fewer lines of code when possible to better help students. We may also only want to use *Bug Fixer* for specific topics rather than using it every week.

Week	Participants	% Pos
1	110	4%
2	107	7%
3	108	5%
4	108	1%
5	74	29%
6	96	14%
7	141	1%
8	85	11%
9	88	5%

Table 3: Survey Results for "How well do you understand the course so far?"

In addition to the specific course topics, some students reported a gain in knowledge that was not directly listed as a course topic. Some students acknowledged that finding specific types of bugs in Bug Fixer problem sets made them more aware of common bugs that they could avoid in their own code. Some students also appreciated the opportunity to see someone else's code in the Bug Fixer problem sets so that they could see examples of code that helps to shape their own coding practices.

RQ2: Do the students show improvement as measured by grades in the course?

We compare the final course grades for students that had the same instructor for lectures. The students are split into two groups based on whether they participated in weekly *Bug Fixer* sessions in different semesters. Table 4 shows that the grades did not increase drastically, but there is a slight increase in the number of A's for students that participated in *Bug Fixer* sessions. Our future work will examine a larger sample size of students while also addressing constructive feedback that we discuss shortly.

Grade	Section with *Bug Fixer*	Section without *Bug Fixer*
A (90-100)	40 (32.26%)	15 (27.27%)

B (80-89)	28 (22.58%)	14 (25.45%)
C (70-79)	22 (17.74%)	9 (16.36%)
D (60-69)	10 (8.06%)	4 (7.27%)
F (0-59)	23 (18.55%)	7 (12.73%)

Table 4: Grade Distribution in sections with and without Bug Fixer (No. of students, % of students).

RQ3: *Do the students recommend this exercise for future students?*

We asked students whether they would recommend *Bug Fixer* to future students and whether they had suggestions for improvements. A total of 48.7% of the students recommended the activity for future students and only 3.8% said that they would not recommend it. It is motivating that almost half of the students would recommend *Bug Fixer* for future students. On the other hand, many of the other students provided constructive feedback that our future work will address. For instance, important comments include that we need to modify problem sets to allow more flexibility for modifying code and provide more hints. Of course, the results from RQ1 indicate that we also need to examine our problem sets as there was a lot of variation in the value that the students reported for the different exercises each week.

CONCLUSIONS AND FUTURE WORK

Students frequently encounter *logic* bugs. When students leave lectures without a clear understanding of an algorithm, such conceptual misunderstanding causes them to struggle with programming assignments. Our work shows that *Bug Fixer* is a promising approach to help students better understand material as they focus on tracing and debugging small problems. We found slightly positive results to our research questions that focus on students' reported value and actual grades. We also received feedback indicating that some students felt that *Bug Fixer* exercises exposed them to common bugs that they avoided in their own homework assignment solutions and that they appreciated the opportunity to see sample code that provides guidance to improve their coding style. On the other hand, we also received constructive criticism. This feedback may be general to many Computer Science Education projects that attempt to help students with conceptual understanding of algorithms. For instance, while *Bug Fixer* provides a good review of course material and allows them to edit another programmer's code, we found that some students were still frustrated with course material and wanted smaller and more focused problem sets, hints, notification of the exact bugs that they corrected instead of a message that they fixed X out of Y bugs, and the option to rate problem sets. Our future work will address these issues and experiment with a larger sample size.

REFERENCES

[1] Bishop, J., Horspool, R., Xie, T., Tillmann, N., Code Hunt: Experience with Coding Contests at Scale, ICSE, pages to appear, 2015.

[2] Delev, T., Gjorgjevikj, D., A study on implementation and usage of web based programming assessment system: Code, ICT Innovations, 76-85, 2014.

[3] Hall, M., Laughter, K., Brown, J., Day, C., Thatcher, C., Bryce, R., A Study of the Types of Programming Bugs that Introductory Students Request Help Solving, Journal of Computing Sciences in Colleges, 28(2):87-94, 2012.

[4] Levitin, A., Papalaskari, M., Using puzzles in teaching algorithms, *Proceedings of the 33rd SIGCSE technical symposium on Computer Science Education*, New York, NY, USA: ACM, 292-296, 2002.

[5] Lopez, M., Whalley, J., Robbins, P., Lister, R., Relationships between reading, tracing and writing skills in introductory programming, *Proceedings of the Fourth International Workshop on Computing Education Research*, New York, NY, USA: ACM, 101-112, 2008.

[6] Odekirk-Hash, E., Zachary, J. L., Automated feedback on programs means students need less help from teachers, *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, New York, NY, USA: ACM, 55-59, 2001.

[7] Bug Fixer, https://www.cse.unt.edu/~reneebryce/BugFixer, retrieved July 9, 2015

[8] Stiller, E., Teaching programming using bricolage, *Journal of Computing Sciences in Colleges*, 24(6), 35-42, 2009.

[9] Tillmann, N., De Halleux, J., Xie, T., Gulwani, S., Bishop, J., Teaching and learning programming and software engineering via interactive gaming, *Proceedings of the 2013 International Conference on Software Engineering*, Piscataway, NJ, USA: IEEE Press, 1117-1126, 2013.