BUG WARS: A COMPETITIVE EXERCISE TO FIND BUGS IN CODE

Renee Bryce
Utah State University
Computer Science
Renee.Bryce@usu.edu

ABSTRACT

Software bugs are a common problem that students encounter in any Computer Science program. "Bug Wars" is a fun and competitive class exercise for student teams to identify bugs in code. To prepare for the competition, the instructor provides several code examples that contain bugs. Each student team also develops code that has a bug. All of the code examples are placed on a table in the classroom at the beginning of class. The competition then begins by each team taking one problem to solve and checking with the author of the respective code to ask whether they correctly identified the bug. If they solve the bug, we update the score and they swap the problem that they solved for a new problem. The team that identifies the most bugs wins the competition. The majority of students reported that this activity increased their interest in software testing and made them more aware of bugs that they should avoid on future assignments.

1 INTRODUCTION

Students in introductory programming courses often struggle with programming bugs. For instance, we conducted a small survey of students from our Computer Science I course in two separate semesters so that we could understand their frustrations as a CS1 student. A total of 92 students completed the survey. We asked the open-ended question of whether anything in the course was frustrating and if so, what the most frustrating issue was for them. Programming bugs and debugging were the biggest issues reported by 38%
of the students. This motivates our work to create fun activities that help students to think about programming bugs so that they can avoid them on future assignments.

Locating bugs is often more difficult than correcting bugs [7, 8]. Software testing is one way to help students with bugs. For instance, Edwards examines the relationship of student performance and how well they test their programs [5]. Another topic in software testing is that of Test Driven Development (see [4] for one example study). Cole et al. propose teaching students to use static analysis to find bugs [2]. Fleming et al. have more advanced students work on corrective maintenance of concurrent software [9]. Williams suggests defect prevention [11]. Bryce et al. collect data about student programming bugs, share the data with students and instructors, and create on-line movies and games about the most common bugs [1]. Elbaum et al. teach students about software testing using an interactive web application called Bug Hunt [6]. Our work here differs as we make software testing a team-based activity in a fun, competitive environment by having students work in teams to (1) create code that contains one bug for others to solve and (2) competitively locate bugs in problems that are created by the instructor and other student teams. Our activities emphasize working in teams since many studies such as pair programming demonstrate that students often enjoy working together. Another example by DeClue et al. reinforces that a sense of community among students is helpful [3].

In the remainder of this paper, Section 2 describes our "Bug Wars" activity, Section 3 provides example problems, Section 4 discusses results of student feedback on the activity, and Section 5 concludes.

2 BUG WARS

The activity spans over two 70 minute class sessions as follows:

Class session 1: In the first class session, students form teams of size 3 to 4 students. Each team then develops code that contains one bug and a solution key. They submit these at the end of the class period. After class, the instructor reviews the problem and solution key from each team to ensure that they are correct and usable for the remainder of the activity. The instructor then creates several additional problems of buggy code that complement those that the students created.

Class session 2: The class session begins by the instructor placing all of the papers with buggy code on a table in the classroom. Each team then takes one problem. Once a team believes that they solved a problem, they check with the author of the code to confirm whether they correctly identified the bug. If they answer correctly, they receive a point and swap their problem for a new one. If they do not answer correctly, they are also allowed to swap for a different problem at any time. The team that locates the most bugs wins.

3 EXAMPLE PROBLEMS

This activity may be used in any programming course, but we used it in a CS3 course. The problems should be tailored to material in the specific course. When we conducted this activity, students were learning about recursion. Therefore, the example problems that the instructor created involved recursion and code that had been reviewed
in class and their textbook, "Data Structures and Other Objects Using Java" [10]. For instance, our first problem in Figure 1 includes a bug in the Mergesort algorithm. This code is modified from the course textbook [10]. We told the students to assume that the "merge" method works correctly and that the bug is in the "mergesort" method. The bug is on line 11. The starting index for the right half of the array is incorrect.

```java
// Assume that the merge method works correctly as we
// reviewed in class
1. public static void mergesort(int[] data, int first, int n)
2. {
3.     int firstHalf;
4.     int secondHalf;
5.     if (n > 1)
6.         {
7.             // Compute sizes of the two halves
8.             firstHalf = n / 2;
9.             secondHalf = n - firstHalf;
10.            mergesort(myArray, first, firstHalf);
11.            mergesort(myArray, first, secondHalf);
12.            merge(myArray, first, firstHalf, secondHalf);
13.         }
14. }
```

**Figure 1:** Example 1: Bug in the MergeSort Merge algorithm (The bug is on line 11, the starting index for the right half is incorrect)

The second example in Figure 2 is also code that students are familiar with from lecture and their textbook [10]. In this problem, the method takes an integer value and prints each digit of the number on an individual line. We inserted a bug on line 6. The wrong value is passed to the recursive call and can easily be found if the students trace the method.

```java
 45
```
// This code is supposed to take a number and print it vertically.
// For instance, an input of "1234" should result in the output:
// 1
// 2
// 3
// 4
// 1. static void writeVertical(int num)
// 2. {
// 3. if(num10)
// 4. cout << num << endl;
// 5. else{
// 6. writeVertical(num%10);
// 7. cout << (num%10) << endl;
// 8. }
// 9. }

Figure 2: Example 2: Bug in the formula that computes the value inside of the recursive call (The bug is on line 6, the mod should be divide)

The third example in Figure 3 is supposed to compute the factorial of a number using a recursive method. We inserted a bug on line 6 in which we sum the values instead of multiplying.

// This code is supposed to compute the factorial of a number
// 1. public static int factorial(int n)
// 2. {
// 3. if(n == 0) {
// 4. return 1;
// 5. }
// 6. else {
// 7. return n + factorial(n-1);
// 8. }
// 9. }

Figure 3: Example 3: Bug in the return statement (The bug is on line 6, the '+i' should be '*i')

Finally, some students in the course were still weak on the recently covered topic of recursion, so we gave an easier problem in order to boost everyone's confidence that they could solve at least one problem. In the example shown in Figure 4, the code should compute the sum from 1 to n. A common bug that was brought to office hours during this particular semester was that students were off-by-one in their loops. The bug in this
problem is on line 4 and includes an off-by-one bug. Further, we gave a bonus point if the students could improve the efficiency of the code. Earlier in the semester, we talked about this example and how a formula could replace the loop.

```java
// This code is supposed to calculate the sum from 1 to n.
// Bonus: The code has a bug for you to find. Is there a way to
// implement this more efficiently without a loop?
1. public static int calculateSum(int n)
2. {
3.     int totalSum = 0;
4.     for(i=1; i<n; i++)
5.         totalSum += i;
6.     return (totalSum);
7. }
```

**Figure 4:** Example 4: Bug in a for-loop (The bug is on line 4, the loop is off-by-one)

### 4 RESULTS

At the conclusion of the activity, we administered a survey to students and then discussed the results of the survey. This section summarizes the results and class discussion.

#### 4.1 Student survey

The survey was optional, but all 26 students completed the survey. Table 1 shows the results. The activity increased interest in software testing for 77% of the students. Approximately 92% of the students said that the activity increased their awareness of bugs that they should avoid on future assignments. Indeed, this is important as other researchers report that students often have more difficulty locating bugs rather than fixing them [7, 8]. All of the students with the exception of one recommended that we hold this activity again in the future.
Table 1: Summary of survey questions

In addition to the three questions above, we asked students to provide any comments that they had. All of the comments were positive and include:

- This was fun! We should do it more often.
- I've made so many bugs that I was able to think of really good ones for this game.
- I like working in teams and class time went by fast.
- We need more time!
- I would have made our bug tougher after seeing some of the other team's bugs.
- Thanks! (This comment was made four times.)
- We should be able to put more than one bug in the problem. Muhaha!
- Can we do this again?

4.2 Class discussion about the survey results

During a class discussion about the activity, we asked the students why the activity increased their interest in software testing. Several students said that they chose this answer because they had fun working with their team members to create the most difficult buggy code that they could for the other teams. They also said that it was interesting to work as a team to find the bugs that other teams created. Students also said that testing their own code was "boring", but that testing code that they did not write on their own was fun! We also asked how the activity increased their awareness of bugs that they should avoid on future assignments. Students volunteered that they experienced many of the bugs already, but that the bugs related to the stopping cases for recursion or the values passed through the recursive methods were tricky. Finally, we asked why all of the students recommended the activity for future. Many students said that it was fun to work in a team. Many agreed that they enjoyed the competition. One student said that it is helpful to see the bugs that you may make before you make them. Many of the students in the class agreed. Another student said that we should do this exercise before every homework assignment so that they could guess the bugs that they would make on the assignment before it was due. Many student enthusiastically supported this. I concluded the discussion by asking the students if the following was a fair summary and they all agreed: "This activity was fun because students enjoyed working in teams. The skills and information that they learned from the exercise is helpful because it strengthens their
ability to avoid programming bugs in the future. Further, they recommend the activity in the future for both their class and other programming classes."

5 CONCLUSIONS

Previous research suggests that students often have more difficulty in locating bugs than fixing them. "Bug Wars" is an activity that makes it fun for students to think about bugs that they could encounter. A brief overview of the activity is as follows: (1) student teams create a problem with code that has exactly one bug and an answer key, (2) the instructor reviews the student code to make sure that there is indeed only one bug, (3) the instructor adds additional problems and buggy code to the pool of problems for students to solve, (4) the instructor shares the problems by placing them on a table in the classroom, (5) each team may take one problem at a time to solve, (6) for each bug that a team identifies, their score increases by one point, and (7) students may swap for a new problem at any time. This activity gives students the opportunity to brainstorm "difficult" bugs when their team creates their problem and code that contains a bug since they want to make it tough for the other teams to solve. It then gives students the opportunity to test code. Our students reported that it is often frustrating finding bugs in their own code, but it was fun to find bugs in code that others wrote. The majority of students reported in a survey that this activity increased their interest in software testing and that it increased their awareness of bugs to avoid on future assignments. All of the students recommended this activity for the future. While we initially meant for this question to ask whether they recommended this activity for future students, the students clarified in a class discussion that they also wanted to repeat this activity for each of their remaining homework assignments so that they could learn about bugs that they should avoid on those assignments as well.

REFERENCES


