A One Year Empirical Study of Student Programming Bugs

Renée C. Bryce, Alison Cooley, Amy Hansen, Nare Hayrapetyan
Utah State University, Renee.Bryce@usu.edu, ali.cool@aggiemail.usu.edu, amy.hansen@aggiemail.usu.edu, nare.hayrapetyan@aggiemail.usu.edu

Abstract - Students in introductory Computer Science courses often have difficulty with coding and problem solving which results in bugs. These bugs cause both student frustration and attrition of many of our CS majors. In this work, we seek to understand the problems that students believe they cannot solve on their own and for which they ask tutors for assistance. We collect and analyze 450 bugs that were brought to our tutor lab by our CS1 and CS2 students over a one year period. The results show that approximately 22% of the problems are due to problem solving skills, while the remaining problems involve a combination of logic and syntax problems for specific topics in the courses.

Index Terms - CS1, CS2, programming bugs.

INTRODUCTION

The enrollment of Computer Science majors has declined in recent years [6]. In addition, a recent report shows that as few as 44% of students are retained at some universities [6]. The retention data at our university are similar. At Utah State University, we observe that the majority of our CS students change majors after their freshman year. Table I shows that over a 5 year period, we have retained only 41.33% of our female freshman and 48.74% of our male students. While there may be several reasons for this decision, we believe that since CS1 and CS2 involve primarily coding, we need to know the programming bugs that our most frustrated students encounter so that professors can adapt their curriculum to each group of students. Further, sharing the data with undergraduates may show students that they are not isolated in experiencing problems on programming assignments. In this paper, we describe our process that is geared towards Computer Science departments that have open tutor labs. At Utah State University, our tutor lab is open 6 days/week from 10:30AM to 8:30PM and is staffed by upper division students. The tutor lab is open to all students that are enrolled in a Computer Science course and has no personal cost to the students. CS1 and CS2 students that visit the tutor lab complete their homework assignments in the C++ programming language.

In the remainder of this paper, the Related Work section discusses related work on collecting student programming bugs, a section on Data Collection presents the web application that we use to collect data, and a section on Analysis discusses the results of one year of data. We then provide sections on Threats to Validity, Future Work, and Conclusions.

TABLE I

<table>
<thead>
<tr>
<th></th>
<th>2002</th>
<th>2003</th>
<th>2004</th>
<th>2005</th>
<th>2006</th>
<th>5 year avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Males</td>
<td>40.0</td>
<td>38.46</td>
<td>57.89</td>
<td>52.17</td>
<td>55.17</td>
<td>48.74</td>
</tr>
<tr>
<td>Females</td>
<td>50.0</td>
<td>0.0</td>
<td>50.0</td>
<td>66.67</td>
<td>40.0</td>
<td>41.33</td>
</tr>
</tbody>
</table>

RELATED WORK

Several approaches exist to collect student programming bugs. We briefly review a few examples of such work. Fenwick et. al. [2] study the behaviors of novice programmers in CS1 and CS2 courses by using their ClockIt Data Logger to track student coding patterns. Their tool tracks the student experience by recording programming events while a student is coding. It logs data such as time between compilations, object instantiation and invocations, and compiler errors. They compare the most common errors to those reported in similar work by Jadud [3]. The ClockIt experiments find that unknown variables, unknown methods, and missing semicolons rank as the top three errors, whereas, Jadud reports that missing semicolons, unknown variables, and missing brackets are the top three errors respectively [2]. Further, Fenwick et. al. show that students that begin their assignments early and work incrementally to complete their assignments earn higher grades. Allevato et. al. [1] also use a toolkit to analyze student programming bugs. Their program does not record the same event based data as the ClockIt Data Logger [2], but rather helps students to find problems such as null pointers, uninitialized pointers, deleted pointers, and out of bounds pointers. Their students use a library extension (called Dereferee) that tracks individual pointers, where pointers are referenced, and their states (i.e., live, null, out of scope). They find that Dereferee helped students to track down their own pointer related bugs and submit better quality code for their assignments.
In addition to automated approaches for data collection, Ko et al. take a different approach and study the cognitive causes of programming bugs [4]. They use a video camera and observe users that program in Alice. They classify the bugs using Reason's latent failure model of error which include attentional, strategic, and knowledge problems [5]. They find that most errors are due to attentional and strategic problems. While we will later show that we find many similar results, our work differs from these previous works as we use a web form to collect data from students that visit our tutor lab. The students describe their bugs “before and after” their interactions with a tutor in our tutor lab. The students describe their bugs “before and after” their interactions with a tutor in our tutor lab. The student records (1) their personal understanding of their bugs before a tutor helps them and then (2) their personal understanding of the problem and solution after the tutor helps them. These data provide a different view point from previous studies as our students that visit the tutor lab document problems that they think they cannot solve on their own. Further, the students that we study may have different characteristics than those in the previous related work. On average, our students spend 2 to 2.7 hours attempting to solve their programming problems but are unsuccessful on their own and then visit our tutor lab. We do not collect data from students that are able to fully work through problems on their own since those students do not visit the tutor lab.

Capturing and Analysis Process

Our research involves a two step process. First, we collect data from students that visit our tutor lab using a web application that we developed. This web application is freely available to other universities upon request. Second, we analyze the data by classifying it into 20 categories. We discuss these two steps next.

Data Collection Process

Figures 1 and 2 show our data collection process. Figure 1 shows the web form that students fill out before the tutor helps them. Students enter their class year, course (i.e., CS1, CS2), the programming language, number of lines of code, number of instance variables, number of methods, and the amount of time that they spent before asking a tutor for help. Next, they provide a brief description of the assignment and the problem that they encountered. The tutor then sits with the student to review the information that they entered into the web form. Once this is done, the tutor walks through the code with the student and helps them to understand their problems. During the tutor session, students sometimes find that they described the problem incorrectly or that there were additional problems that they did not know before the tutor helped them. This is clarified in the second step of our data collection process. After the tutor helps the students, they complete the second part of the web form as shown in Figure 2. On this form, they
summarize the bug(s), the solution(s), and optionally provide one or more test cases that would have exposed their bug(s). They submit their data and it is stored on our server.

**ANALYSIS**

In our analysis, we (1) provide the bug classification categories and brief examples of each, (2) provide a table that summarizes the classification of the 450 bugs, and due to space limitations, (3) discuss the top 5 most common bugs in each course which account for 63.8% of the CS1 bugs and 67.1% of the CS2 bugs.

**BUG CLASSIFICATION CATEGORIES**

We classify the 450 bugs from the past year into 20 categories. To identify these categories, we reviewed example classification schemes such as those summarized in [4] and then narrowed down the best categories for our study after reviewing the actual data brought to our tutor lab. The categories with brief descriptions are as follows:

1. **Computer Environment**: These problems involve the configuration of a machine. Examples include problems with unplugged speakers if a program is supposed to play sound, problems with class paths, or other machine configurations that cause a correctly implemented program to not run.

2. **Problem solving**: These problems involve the inability to understand a problem and/or how to use a programming language to implement a solution. A common example is that a student reads the assignment and does not understand what is being asked.

3. **Pointers**: These include any problems involving pointers. The most common are accessing null or incorrect pointers.

4. **Loops and switch statements**: These include all for, while, do-while, and switch statements. The most common are off-by-one mistakes, infinite loops, and switch statements without default cases.

5. **Arrays**: These include declaring, using, and deleting arrays. Common mistakes include off-by-one problems or going out of bounds.

6. **If statements**: These include both logic and syntax problems with if-statements. A common mistake is placing statements in a wrong order and/or using incorrect logic for the statements.

7. **File I/O**: These include open, close, read, and write functions. The most common problems include attempting to read from files that do not exist, not understanding the syntax associated with file i/o calls, and difficulty with file format conversions.

8. **Functions**: These include the creation and use of functions. The most common problems involve passing variable incorrectly.

9. **Pass by reference**: These include the passing of variables. Common problems include understanding how to pass arrays and strings.

10. **Formatting**: These include the format of output to the screen or file. Common problems include missing include statements.

11. **Classes**: These include problems with classes, methods, and instance variables. The most common problems...
involves confusion over private and public class members. Another common problem involves instantiating objects.

12. **Algorithms**: These include the use of algorithms built into the programming language. The most common problems involve a lack of understanding of how the algorithms work, including appropriate input and expected output.

13. **Vectors**: These include the STL vector. The most common problems involve adding or changing items in a vector.

14. **Strings**: These include manipulating strings. Common problems include parsing strings or converting them.

15. **Abstract Data Types (ADTs)**: These include any ADTS. However, the most common brought to the lab involve using stacks and queues.

16. **GUIs**: These include basic layout problems and calls from GUI widgets. Common problems involve the implementation of listeners.

17. **Overloading**: These include logic and syntax problems. Common problems involve incorrect syntax.

18. **Recursion**: These include problems with recursive calls and stopping cases. Common problems involve both of these.

19. **Try/Catch Exceptions**: These include problems related to exception handling. The most common problem includes throwing exceptions incorrectly.

20. **Search/Sort**: These include any search and sort algorithms that students implement on their own. The most common problem involves implementing the steps of the algorithm out of order.

In the remainder of this section, we refer to these categories as we examine the most common bugs and the amount of time that students spent on bugs in each of these categories.

**CS1: THE TOP 5 MOST COMMON BUGS**

Table II shows the distribution of the 210 bugs that CS1 students brought to the tutor lab in the past year. **Problem Solving** bugs are most common as 48 students encountered such bugs. We found that the most common problem in this category is that students do not understand the requirements of their homework assignments. This shows us that students would benefit from CS1 instructors spending more time reviewing the assignments in class. **Loops and switch statements** were the second most common problem and experienced by 28 students. The most common problems in this category involve loops that are off-by-one or that are infinite. In addition, students often forgot default cases in their switch statements. The third most common bug experienced by 24 students involves **arrays**. While students generally understand the concept of arrays, specific implementation problems include allocating memory for the arrays and iterating through arrays without being off-by-one in the indices or going out of bounds. **File input/output** problems are fourth most common and experienced by 23 students. Most of these students did not understand the order of file operations, specifically, opening a file before reading/writing to it and then remembering to close it when done. The fifth most common bugs involve **strings** and were brought to the tutor lab by 11 students. The assignments involving strings required students to parse and convert strings, both of which involved logic and syntax problems for students.

**CS2: THE TOP 5 MOST COMMON BUGS**

The last three columns of Table II show the distribution of bugs that were brought to the lab by CS2 students, along with the amount of time that students spent working on their own before asking a tutor for help and the number of lines of code that they wrote on their own. As we saw for CS1, **problem solving** bugs are also the most common bug for the CS2 students as 54 students brought this type of bug to the tutor lab. Again, we saw that students had difficulty understanding the requirements of their homework assignments. The next most common bug is **File i/o** and was brought to the lab by 38 students. This bug was also quite common in CS1, so we see that this topic continued to pose difficulties on students throughout their first year of CS1 and CS2. The third most common problem involves **classes** and was brought to the lab by 28 students. This topic was not covered in CS1 and was new to the CS2 students. We saw that both the logic of object-oriented programming and syntax was problematic for these students. **Pointers** were also new in CS2 and 21 students visited the lab for help on mismanagement of pointers, including many null references. The fifth most common problems involve **recursion**. This topic is new in CS2 and we saw 20 students struggle, mainly with the logic of how recursion works. This lack of understanding often resulted in bugs with both recursive calls and stopping cases.

**CS1 AND CS2: STUDENT REPORTED TIME SPENT ON BUGS AND LINES OF CODE WRITTEN BEFORE SEEKING HELP**

Students reportedly spent an average of approximately 2 hours in CS1 and 2.7 hours in CS2 trying to solve their own bugs before asking a tutor for help. Table II shows the average times for each of the 20 categories of bugs. In CS1, the most time consuming bugs include those involving if-statements as 9 students spent an average of over 10 hours trying to solve their problem on their own before asking for help. The next most time consuming were 3.25 hours for pass by reference bugs by 5 students, 3 hours for overloading by 2 students, and just less than 3 hours for problem solving by 48 students.
In CS2, 10 students spent an average of 8.4 hours on their own before asking tutors for help when the bugs involved functions. This was followed by 2 students spending an average of 8.25 hours on formatting problems, 4 students spending 6.75 hours on bugs involving types, and 38 students spending 6.2 hours on file i/o bugs.

It is also important to notice that students begin to write code before visiting a tutor. Table II shows that the average CS1 students writes approximately 58 lines of code before they visit the lab and the average CS2 student writes approximately 159 lines of code before they visit the tutor with a problem that they do not think that they can solve on their own. We emphasize that the substantial lines of code and time is important as our study focuses on problems that students “try” to solve, but for which they give up and visit a tutor. Of course, we refer the reader to our section on threats to validity for a discussion of issues associated with student reported times and lines of code.

**SUMMARY**

In summary, we examined 450 problems that CS1 and CS2 students brought to our tutor lab. The students spent an average of 2 to 2.7 hours trying to solve their own problems before seeking assistance from a tutor. The problems that students felt they could not solve on their own (without a tutor) fell into 20 categories. By far, the most common frustration involved problem solving, i.e., students had difficulty understanding the problem statement on their assignment and designing a solution. The remaining problems stem from misunderstanding of specific topics that are covered both in class and in the textbook. For instance, file i/o was also a difficult topic for both CS1 and CS2 students. Loops and arrays were particularly problematic in CS1, but we saw bugs involving these topics decrease in CS2. In CS2, classes, pointers, and recursion were new topics to the students and caused many bugs.

Overall, this data allows us to better understand bugs that many of our students encounter and will allow us to continually improve our curriculum by addressing the issues. For instance, our current and ongoing work shares the data with both students and instructors. First, we share our data with students so that they see that they are not isolated in encountering bugs. For instance, Figure II shows the number of bugs from Jan. 1, 2009 to Dec. 31, 2009. Due to space limitations, we do not present the results on a monthly basis in this paper, but in practice, we provide these monthly results to students and instructors in our department. For instance, CS2 student may see that 27 students also made the bug related to file i/o in the month of September. Students may also use this data to look up the most common bugs in attempt to avoid them in the future. Instructors use this data to continuously improve their curriculum. For instance, if an
instructor learns that 23 CS1 students spent an average of 87 minutes stuck on a bug involving a for-loop, they may modify their curriculum to go more in-depth on this topic and provide examples of the common mistakes that we logged in our system. Our current and future work will use this data to improve our curriculum.

**Threats To Validity**

As with any empirical study of student programming bugs, we suffer from threats that prevent us from generalizing our results to all students at every university. First, an internal threat is that our study uses data about programming bugs that are written by students with their tutors. Students are not always as descriptive as they should be and sometimes have additional bugs that they do not know about. We minimized this threat by training our tutors to fill out the forms and having the tutors assist the students with their forms. Second, our data is only collected from students that visit the tutor lab. While we intentionally target the study of students that visit the tutor lab since we want to know the problems that students believe they cannot solve on their own, we suggest that other techniques are more inclusive of all students. For instance techniques such as Jadud and ClockIt can automatically collect data from all students through their IDE. However, we have found that the students that visit our tutor lab are a group that struggle with their assignments and help us to better understand the problems that students find they cannot solve on their own. The Jadud and ClockIt approaches do not provide the same level of detail that our tutor lab process provides. Third, the effectiveness of different instructors may skew our results since an instructor can directly impact student learning. Finally, our results are based on one year of data. We believe that the data may evolve as instructors become aware of our data analysis and modify their curriculum to address the documented problems.

**Future Work**

We plan to extend several aspects of this work. First, we plan to extend our web application to auto-generate reports for professors and students. This requires two main tasks (1) the ability for tutors and other approved users to classify the bugs and save this data for each form and (2) the functionality to automatically generate reports for professors. (Currently, we classify and create reports manually.) We plan to report the data on a class-by-class basis and then provide more advanced reporting that compares (1) how the bugs for specific courses change in a single semester, (2) how the bugs change in subsequent offerings (i.e., a professor can address problems from the previous semester and possibly reduce certain bugs by spending more class time on a topic), and (3) how the types of bugs vary by course (i.e., as students progress through our curriculum, the types of bugs may change). Second, we plan to make animated movies about the most common bugs. We are calling it "Mystery Bug Theater". This on-line theater will provide movies that are tailored to the data from our tutor lab logs and will show students the most common problems, how to identify such problems, and how to solve them. We plan to distribute “movie tickets” to the CS1 and CS2 students next fall so that they can visit our on-line movies. Finally, we plan to document and study curricula changes that are motivated by this data. This includes both quantitative data on student performance and surveys of student options on the value of sharing this data with them and providing on-line resources for the most common bugs.

**Conclusions**

Many Computer Science programs suffer from a high attrition rate of students [6]. One cause for attrition is that many students become frustrated with programming bugs. In this work, we collected data from students that visited our tutor lab with problems that they felt that they could not solve on their own. We found that problem solving logic impacts the largest number of students that visit the tutor lab, accounting for 22.85% of the problems in CS1 and 22.5% of the problems in CS2. Indeed, this data indicates that spending extra class time to review the homework requirements may be useful to our students. The other top problems involved a mix of both logic and syntax misunderstandings. For instance, arrays, loops, file i/o, and string topics contributed to 40.95% of the bugs that CS1 students brought to the lab. While there was a decrease in bugs involving arrays and loops, CS2 students continued to have difficulty with file i/o. We also observed that CS2 students had difficulty with new material on classes, recursion, and pointers. These new topics accounted for 32.86% of the CS2 bugs in the tutor lab. While we highlighted the top 5 problems for CS1 and CS2 students, we also observed a total of 20 categories of bugs. This data pinpoints the exact problems that many students think they cannot solve on their own and is available to our faculty and students on a monthly basis. As a result, this data helps us to improve our curriculum by addressing the most common issues. In addition, we share this data with our CS1 and CS2 students as it may improve the confidence of our students to know that 450 problems were brought to our tutor lab and that they are not alone in their challenges.

**Acknowledgements**

Thank you to support from the Collaborative Research Experience for Undergraduates (CREU) project that is sponsored by the Computing Research Association Committee on the Status of Women in Computing Research (CRA-W) in conjunction with the Coalition to Diversify Computing (CDC). The CREU project is funded by the National Science Foundation. Also, thank you to the USU Computer Science department for allowing me collect the data in this paper during my first year as a faculty member.
REFERENCES


AUTHOR INFORMATION

Renée Bryce, Assistant Professor, Utah State University, Renee.Bryce@usu.edu.

Alison Cooley, Undergraduate Student, Utah State University, Ali.Cool@aggiemail.usu.edu

Amy Hansen, Undergraduate Student, Utah State University, Amy.Hansen@aggiemail.usu.edu

Nare Hayrapetyan, Undergraduate Student, Utah State University, Nare.Hayrapetyan@aggiemail.usu.edu