5-2011

An Empirical Study of Student Programming Bugs

Amy Hansen
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/honors

Part of the Computer Engineering Commons

Recommended Citation
https://digitalcommons.usu.edu/honors/81

This Thesis is brought to you for free and open access by the Honors Program at DigitalCommons@USU. It has been accepted for inclusion in Undergraduate Honors Capstone Projects by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.
AN EMPIRICAL STUDY OF STUDENT PROGRAMMING BUGS

by

Amy Hansen

Thesis submitted in partial fulfillment of the requirements for the degree of

DEPARTMENTAL HONORS

in

Computer Science
in the Department of Computer Science

Approved:

Thesis/Project Advisor
Dr. Renee C. Bryce

Departmental Honors Advisor
Dr. Donald H. Cooley

Director of Honors Program
Dr. Christie Fox

UTAH STATE UNIVERSITY
Logan, UT

Spring 2011
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 bugs that students encounter. We have two sources of data. First, we collect and analyze 450 bugs that were brought to our tutor lab by our CS1 and CS2 students. Second, we analyze bugs in CS1 homework assignments. The results show that the majority 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.

ACKNOWLEDGMENTS

This research was funded by 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. Thank you to those entities for their support. Thank you to the USU Computer Science department for allowing us to collect the data for this project. Also, thank you to Dr. Scott Cannon for allowing us use of his CS 1400 class for further data collection and analysis through the undergraduate teaching fellow program. Lastly, thank you to Dr. Renee Bryce for her continued support, mentoring, and help in research efforts and this honors thesis.
# Table of Contents

Abstract ........................................................................................................ ii
Acknowledgements ...................................................................................... ii

Introduction ................................................................................................ 1
Related Work ................................................................................................. 2
Capture and Analysis Process ...................................................................... 5
  Tutor Lab Study .......................................................................................... 6
  In-Classroom Experience ........................................................................... 6

Data Collection Process ................................................................................ 6
  Tutor Lab Study .......................................................................................... 6
  In-Classroom Experience ........................................................................... 8

Analysis ......................................................................................................... 9
  Tutor Lab Study .......................................................................................... 9
    Bug Classification Categories ................................................................. 9
    CS1: The Top 5 Most Common Bugs .................................................... 11
    CS2: The Top 5 Most Common Bugs .................................................... 14
    CS1 and CS2: Student reported time spent before seeking help ........... 14

In-Classroom Experience ............................................................................. 15
    Determining Best, Average, and Low Submissions ............................ 16
    Findings .................................................................................................. 16
      Best ....................................................................................................... 16
      Average ................................................................................................ 17
      Low ....................................................................................................... 18

Comparison to Tutor Lab Study ................................................................. 19

Summary ...................................................................................................... 20
Threats to Validity ......................................................................................... 22
Future Work ................................................................................................. 23
Conclusions ................................................................................................. 23
References .................................................................................................... 25
Author's Biography ....................................................................................... 26
LIST OF TABLES AND FIGURES

Table 1: Percentage of Male and Female Computer Science Students Retained in the Freshman Year at Utah State University ...........................................2

Table 2: Counts, Time Spent, and Lines of Code (LOC) for Bugs Brought to the Tutor Lab by CS1 and CS2 students from Jan. 1, 2009 to Dec. 31, 2009 ........13

Table 3: Categorized Bugs from Average and Low Assignments .........................................................20

Figure 1: Web form for data collection students complete before the tutors help them ....7

Figure 2: Web form for data collection students complete after the tutors help them.....8
INTRODUCTION

In the United States, the enrollment of computer science (CS) majors has declined in recent years [6]. In fact, a recent report shows that as few as 44% of students are retained beyond their freshman year at some universities [6]. The retention rate at Utah State University (USU) is no exception. At USU, we observe that the majority of our CS students change majors after their freshman year. Table 1 shows that over a five-year period, we 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 CS 1400, CS1, and CS 1410, CS2, involve primarily coding, frustration with programming bugs may be one important factor leading to the decision to leave the CS program. Thus, 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 seek to answer the research question – What are the most common programming bugs that students encounter? To answer this question, we use two sources of data. First, we describe a process geared towards computer science departments that have open tutor labs. Our tutor lab is open six days/week from 10:30AM to 8:30PM and is staffed by upper-division CS students. The tutor lab is open to all students that are enrolled in a computer science course and is paid for by class fees assessed at registration. CS1 and CS2 students that visit the tutor lab complete their homework assignments in the C++ programming language. Second, we analyze homework assignments from CS1 from the fall 2010 semester as they may contain bugs
that they do not report in the tutor lab. Through this in-classroom experience, I was able to gain further information about student bugs.

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

### TABLE 1: Percentage of Male and Female Computer Science Students Retained in the Freshman year at Utah State University

<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 collecting 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. The authors 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 Dereferree) that tracks individual pointers, where pointers are referenced, and their states (i.e., live, null, out of scope). They find that Dereferree helped students to track down their own pointer related bugs and submit better quality code for their assignments.

Additional work examines whether different teaching methods make students better programmers and debuggers from the beginning, thus preventing bugs among beginning programmers. Two major practices are test-driven development (TDD) [8, 9, 13] and placing more emphasis on debugging [10, 11, 14, 15].

In several studies, integrating test-driven development early proves to be beneficial. Such integration gives students the incentive to write higher quality code because the testing forces them to think of their code coverage [8]. Beyond immediate benefits, one study showed that incorporating test-driven development from the beginning better prepared students for upper-level courses. It also results in more competent and marketable students, having a deeper understanding of language models and programming practices [13]. Apart from the side effects, having the knowledge of testing prepared students to join industry, as it is a very crucial and common aspect of software development [8]. Because test-driven development takes time and resources to teach, the authors suggest that better prioritization of course content in CS1 and CS2
classes may be best in order to provide both introductory programming and testing materials [8]. If one decides to integrate test-driven development into introductory level CS class, this work can provide guidance on how to best prioritize material.

While debugging is a useful skill, many students are lacking in this key area [10, 11]. In a particular study, 21 students were given a syntactically correct program with a number of logic errors. Their task was to find the errors and correct them. Student debugging techniques were often ineffective and ad-hoc; some students introduced further logic errors during the debugging process. This showed that there is a need for teaching appropriate and useful debugging techniques to students early on [11]. Another study showed that students who are good debuggers are also good programmers, while the reverse is true less than 50% of the time. The authors attribute this to the knowledge of actual program implementation that is gained by debugging [15]. This lends credence to the claim that including debugging in introductory-level CS class course content could be very beneficial for students. If one decides to focus on debugging, one study suggests giving a set of questions to determine a student’s debugging skills. This may be useful for both pre- and post-surveying [14]. I believe it is quite possible that because students do not have knowledge of debugging, they all too often end up creating more errors than they fix, and either end up seeking help from the tutor lab, or submit their homework with bugs.

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 includes attentional, strategic, and knowledge
problems [5]. The authors find that most errors are due to attentional and strategic problems.

These studies took many different approaches to determining student skills and the benefits of different teaching methods, but the findings were consistent: early development of test-writing and debugging skills improve students’ programming abilities in both short term and long term perspectives.

While we later show that our findings are similar to those of the studies discussed here, our work differs in that we used 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. See Figures 1 and 2. Students record (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 viewpoint 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 studied may have different characteristics than those studied in previous related work. On average, the students we studied spent 2 to 2.7 hours unsuccessfully attempting to solve their programming problems before visiting our tutor lab. We did not collect data from students who were able to fully work through problems on their own since those students did not visit the tutor lab.

CAPTURE AND ANALYSIS PROCESS

Our research seeks to answer the research question: What are the most common programming bugs that students encounter? We collect and analyze data from two sources that we describe next, including tutor lab and in-class sources.
**Tutor Lab Study**

Our research involves a two-step process. First, we collect data from students that visit our tutor lab by 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 shortly.

**In-Classroom Experience**

The second source of data involved me attending one section of CS1 as an undergraduate teaching fellow. The class had 38 students. From these students, I was able to learn the problems that students experienced based on the questions that they asked in class and at weekly study sessions. These questions ranged from needing clarification of the assignment description to how to syntactically or logically express a concept. Further, I reviewed code that students submitted for their assignments and look at the correlation between course success and programming assignments, including the bugs found therein. I discuss this process below.

**Data Collection Process**

**Tutor Lab Study**

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(s) 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 found the bug(s). They submit their data, and it is stored on our server.

FIGURE 1: Web form for data collection students complete before tutors helps them
**FIGURE 2**: Web form for data collection students complete after the tutors help them.

**IN-CLASSROOM EXPERIENCE**

For this aspect of the study, I took a role as an undergraduate teaching fellow, an opportunity available in many disciplines throughout the university. I attended class sessions, observed, and took note of student questions, and examined the bugs that they submitted in homework assignments. Additionally, I held weekly meetings wherein students had the opportunity to ask me questions and receive one-to-one help and attention. The most beneficial component of this experience was reviewing the assignments that students submitted. For each assignment, I looked through students’ code and came up with an example of a best, average, and low submission. From looking at the students’ code, I was able to see common bugs. Over the course of the semester, I was able to see trends in the bugs that students submitted. The observations from attending class and holding weekly study sessions were closely reflected in the findings.
from the submitted assignments. I focus on this aspect of the experience in the analysis and further discussions.

ANALYSIS

TUTOR LAB STUDY

In our analysis, we seek to answer the research question: *What are the most common programming bugs that students encounter?* To evaluate this question, 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 (3) discuss the top five 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. Brief descriptions of the categories follow:

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 a 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 involve 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 Five Most Common Bugs

Table 2 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 were 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 were 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 23 students.

<table>
<thead>
<tr>
<th>Problem Category</th>
<th>Number of Bugs</th>
<th>Common Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Solving</td>
<td>48</td>
<td>Students do not understand the requirements of their homework assignments.</td>
</tr>
<tr>
<td>Loops and switch statements</td>
<td>28</td>
<td>Loops that are off-by-one or that are infinite.</td>
</tr>
<tr>
<td>Arrays</td>
<td>24</td>
<td>Allocating memory for arrays and iterating without being off-by-one in indices</td>
</tr>
<tr>
<td>File input/output</td>
<td>23</td>
<td>Problems with the order of file operations.</td>
</tr>
<tr>
<td>Strings</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>
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.

TABLE 2: Counts, Time Spent, and Lines of Code (LOC) for Bugs Brought to the Tutor Lab by CS1 and CS2 Students from Jan. 1, 2009 to Dec. 31, 2009

<table>
<thead>
<tr>
<th>Classification</th>
<th>CS1</th>
<th>CS1</th>
<th>CS1</th>
<th>CS2</th>
<th>CS2</th>
<th>CS2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of students</td>
<td>Avg. time</td>
<td>Avg. LOC</td>
<td>No. of students</td>
<td>Avg. time</td>
<td>Avg. LOC</td>
</tr>
<tr>
<td>Computer Environment</td>
<td>6</td>
<td>156.25</td>
<td>2.5</td>
<td>3</td>
<td>120.00</td>
<td>77.5</td>
</tr>
<tr>
<td>Problem Solving</td>
<td>48</td>
<td>179.22</td>
<td>48.8</td>
<td>54</td>
<td>270.49</td>
<td>173.27</td>
</tr>
<tr>
<td>Pointers</td>
<td>8</td>
<td>111.25</td>
<td>39.89</td>
<td>21</td>
<td>212.00</td>
<td>152.5</td>
</tr>
<tr>
<td>Language Constructs</td>
<td>6</td>
<td>117.00</td>
<td>63.6</td>
<td>1</td>
<td>240.00</td>
<td>200</td>
</tr>
<tr>
<td>Loops and switch statements</td>
<td>28</td>
<td>87.81</td>
<td>49.06</td>
<td>11</td>
<td>341.25</td>
<td>171.62</td>
</tr>
<tr>
<td>Arrays</td>
<td>24</td>
<td>117.68</td>
<td>68.25</td>
<td>4</td>
<td>246.25</td>
<td>227.75</td>
</tr>
<tr>
<td>if / if else</td>
<td>9</td>
<td>654.50</td>
<td>48.38</td>
<td>1</td>
<td>210.00</td>
<td>275</td>
</tr>
<tr>
<td>File i/o</td>
<td>23</td>
<td>96.53</td>
<td>48.76</td>
<td>38</td>
<td>373.04</td>
<td>214.58</td>
</tr>
<tr>
<td>Function</td>
<td>10</td>
<td>123.75</td>
<td>64.88</td>
<td>10</td>
<td>504.17</td>
<td>170.6</td>
</tr>
<tr>
<td>Pass by Reference</td>
<td>5</td>
<td>195.00</td>
<td>69.75</td>
<td>5</td>
<td>120.00</td>
<td>125</td>
</tr>
<tr>
<td>Formatting</td>
<td>5</td>
<td>82.50</td>
<td>74.5</td>
<td>2</td>
<td>495.00</td>
<td>315</td>
</tr>
<tr>
<td>Types</td>
<td>9</td>
<td>101.25</td>
<td>34.09</td>
<td>4</td>
<td>405.00</td>
<td>103.25</td>
</tr>
<tr>
<td>Classes</td>
<td>0</td>
<td>60.00</td>
<td>100</td>
<td>28</td>
<td>146.27</td>
<td>128.65</td>
</tr>
<tr>
<td>Algorithms/Advanced</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>8</td>
<td>246.11</td>
<td>121.38</td>
</tr>
<tr>
<td>Functions</td>
<td>5</td>
<td>53.33</td>
<td>49</td>
<td>5</td>
<td>187.50</td>
<td>117</td>
</tr>
<tr>
<td>Vectors</td>
<td>11</td>
<td>67.50</td>
<td>53.27</td>
<td>4</td>
<td>225.00</td>
<td>161.5</td>
</tr>
<tr>
<td>Strings</td>
<td>3</td>
<td>77.67</td>
<td>71.33</td>
<td>10</td>
<td>96.25</td>
<td>90.27</td>
</tr>
<tr>
<td>Abstract Data Types</td>
<td>2</td>
<td>180.00</td>
<td>50</td>
<td>0</td>
<td>0.00</td>
<td>245.71</td>
</tr>
<tr>
<td>Overloading</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>20</td>
<td>188.18</td>
<td>35</td>
</tr>
<tr>
<td>Recursion</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>5</td>
<td>294.00</td>
<td>180</td>
</tr>
<tr>
<td>Try/Catch &amp; Exceptions</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>5</td>
<td>225</td>
<td>161.5</td>
</tr>
<tr>
<td>Average</td>
<td>10</td>
<td>122.58</td>
<td>58.28</td>
<td>11.43</td>
<td>242.21</td>
<td>158.73</td>
</tr>
<tr>
<td>Median</td>
<td>6</td>
<td>101.25</td>
<td>51.64</td>
<td>5</td>
<td>225</td>
<td>161.5</td>
</tr>
</tbody>
</table>
CS2: THE TOP FIVE MOST COMMON BUGS

The last three columns of Table 2 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 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 and syntax of object-oriented programming 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 also 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 self-reported that they spent an average of approximately 2 hours in CS1 and 2.7 hours in CS2 trying to solve their bugs before asking a tutor for help. Table 2
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 note that students begin to write code before visiting a tutor. Table 2 shows that average CS1 students writes approximately 58 lines of code before visiting the lab, and average CS2 students writes approximately 159 lines of code before visiting the tutor with a problem that they do not think 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 before 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.

IN-CLASSROOM EXPERIENCE

In this analysis, I discuss (1) how best, average, and low programs were found, (2) the classifications of the bugs in these programs, and (3) how they compare to the tutor lab findings.
DETERMINING BEST, AVERAGE, AND LOW SUBMISSIONS

As expected, the quality of a student's fulfillment of an assignment is manifest in the grade given. The best submissions were submissions that received full points for the assignment. However, there is a level of quality that is not captured in this grade. There were times when a submission representing the average still received full points, but there were other submissions that were better for reasons such as code readability. For this reason, average submissions were not always those that received C grades. Low submissions, however, were consistently those that received low markings and had many bugs. Through review of student submissions, a representative best, average, and low submission were chosen for each assignment. Data was then gathered based on these assignments.

FINDINGS

Best

Overall, there were three characteristics that were routinely manifested in best submissions: effective commenting, encapsulation of common tasks, and code readability.

Effective commenting is widely known to be a good practice. Between 1992 and 1998, 75% of development costs were post-delivery maintenance [16]. Since so much time is devoted to maintenance in industry, commenting code during development can save a lot of time and confusion in the future. It is possible that early practice of commenting code would carry over to further documentation in software development, such as requirements, class diagrams, etc.
Encapsulation of common tasks was seen in many different ways. The most common of these was putting reused code into functions. Such encapsulation becomes very helpful to understand when students move on to object-oriented (OO) design principles.

Code readability was demonstrated in the best submissions in many ways. Beyond commenting, these include descriptive variable and function names, intentional placing of variable declarations, and intuitive use of loops, functions, etc.

These three practices were readily apparent and common to the best homework submissions. These students demonstrated not only a working knowledge of how to solve the problem at hand, but also a knowledge of how to write code that does more than just work.

Average

Average submissions sometimes contained bugs. When they did, they were often bugs that could be easily found and/or fixed. The most common of these were logic errors. The classifications and descriptions of the bugs in the average assignments include:

- **Loops and switch statements**: Performing input validation only once, but getting input multiple times.

- **Strings**: Unnecessary use of strlen and atof functions, thus overcomplicating the assignment.
• **Problem solving:** Math logic errors – using the wrong operators in the wrong context.

• **Arrays:** Use of one-dimensional array when a two-dimensional array was needed.

• **Problem solving:** Not taking into account all possible values and handling them.

The bugs that were present in the average submissions were not indicators of misunderstanding of fundamental programming principles. In many cases, they contributed to decreased efficiency and readability, but the average submissions still provided core functionality.

*Low*

Low submissions often received markings of < 70%. Some did not compile, and many did not provide basic functionality. They also contained bugs showing a lack of understanding of many programming fundamentals, even weeks after the students had learned those basics. These include:

• **Arrays:** Misunderstanding of arrays - capacity vs. size, 0-based indexing, and type:

• **Arrays:** Misuse of [] vs. ()

• **File I/O:** Misunderstanding of file I/O – ifstream vs. ofstream.

• **Problem solving:** Misunderstanding of variable storage and usage – using unchanged variables as if they had been changed.
• **Problem solving:** Sometimes students did not appear understand the problem description – they were told to use overloaded functions but failed to do so.

• **If statements:** Lack of understanding of characters – compared int to a character (and not as we sometimes do using an ascii value).

• **If statements:** Logical errors – incorrect use of && and || operators.

**COMPARISON TO TUTOR LAB STUDY**

The categorized bugs from average and low submissions are shown in Table 3. This data has a closer correlation to the tutor lab data than I expected. The top five bugs for CS1 in the tutor lab study were (from high to low prevalence) problem solving, loops and switch statements, arrays, file input/output, and strings. The five most common from this additional study (in the same order) were problem solving, arrays, if statements, and the remaining three tied: loops and switch statements, file I/O, and strings.

Both studies showed problem solving as the number one bug among CS1 students. Arrays were also in the top three from both studies. The five categories were the same in both studies, with the exception of if-statements being the third most common in the in-classroom experience.

Our collected data consistently shows that the five most common bugs from the tutor lab study are also common bugs to CS1 students. This holds not only for students who seek out help, but those that turn their assignments in with these bugs.
TABLE 3: Categorized Bugs from Average and Low Assignments

<table>
<thead>
<tr>
<th>Bug category</th>
<th>Number in average submissions</th>
<th>Number in low submissions</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loops &amp; Switch Statements</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Strings</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Problem Solving</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>If Statements</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Arrays</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>File i/o</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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 stemmed 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.

Further study and data collection showed that CS1 students who submit their code with bugs tend to have the same bugs as CS1 students who seek out help in the tutor lab. The only difference is that students who submitted their code with bugs had more problems with if-statements and the logic of the condition than those in the tutor lab study.

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 w 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 on our website. 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 may 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, we review the threats of the data from our tutor lab. One 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, future studies should contain techniques that 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 students who visit our tutor lab are a group that struggle with their assignments. Thus, focusing on them can 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.

Second, there are threats to validity for our experiments that are based on the classroom experience. A threat to validity for the in-classroom experience is that these
results are based on code students actually submitted. The reasons students submitted code with errors cannot be determined, and may be a result of one or more of the following: personal life issues (i.e., not enough time, life-changing events, etc.), laziness, lack of knowledge of the existence of bugs, or the inability to find and correct the bug. In my analysis, I assumed that bugs were submitted for the last reason. I believe that students generally do their best on their homework assignments and turn in work that is representative of their best work only restricted by their knowledge and abilities. Additionally, we only studied one class, and the results may be influenced by the instructor, a group of students that may not be representative of all students, the material covered, and programming language used in the course.

**Future Work**

As discussed in the related work section, studies have shown that courses that focus more on debugging and test driven development have improved student understanding and better prepared them for careers in industry. As one study suggested, intentional course alteration is necessary to allow time to teach these topics [8]. With the information that we have found in this study about the most common bugs for students, one could design a course to focus on these areas while eliminating other material and teaching debugging and test-driven development. Both debugging and test-driven development force students to think about what they are really doing, allowing them to understand how to produce optimal, more error free, and more robust code.

**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 they could not solve on their own, and we analyzed bugs from CS1 student homework submissions. 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. We also found the same indication in the homework submissions; problem solving was the most prevalent bug. 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. The most common CS1 bugs found from the tutor lab study were reflected in the bugs found in homework submissions, further solidifying these findings. 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. 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 students struggle with. 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.
REFERENCES


AUTHOR’S BIOGRAPHY

Amy Hansen grew up in Wyoming and Utah and graduated from Murray High School, Murray Utah, in 2008. Two months after high school graduation, Amy also graduated with an Associate of Science from Salt Lake Community College. She will graduate from Utah State University in the spring of 2011 with a bachelor’s degree in Computer Science with an emphasis in Software Development. Currently, Amy works remotely for Xactware, Inc. as a Software Engineer. Upon graduation, Amy plans to continue her career with Xactware in Orem, Utah.