Bug Catcher: A System for Software Testing Competitions

Renée Bryce
Computer Science and Engineering
University of North Texas
Denton, TX 76208
Renee.Bryce@unt.edu

Quentin Mayo, Aaron Andrews, Daniel Bokser, Michael Burton, Chelynn Day, Jessica Gonzalez, Tara Noble
Computer Science
Utah State University (REU Site Program)
Logan, UT 84341
USU.CS.REU@gmail.com

ABSTRACT

Bug Catcher is a web-based system for running software testing competitions. While programming competitions are a way to engage students, they require students to have coding experience. On the other hand, software testing competitions may reach high school students that do not have access to a programming course. In this paper, we present the Bug Catcher system and the results from four sessions of a competition that include a total of 94 high school students. Bug Catcher provides students with requirements, buggy code, and input fields to enter test cases. We observed that most students began entering test cases based on requirements, but then many took an interest in the code as time went on. Our results show that 90% of students would recommend this activity in the future and 72% of students report that the activity increased their interest in Computer Science. Students also provided feedback on the system from the perspective of students without background in Computer Science, allowing us to create and modify features for future use.

Categories and Subject Descriptors
K.3 [COMPUTERS AND EDUCATION]: Computer and Information Science Education—Computer science education

General Terms
Experimentation

Keywords
Computer Science outreach, software testing, software testing competition

1. INTRODUCTION

Competitions are a way to engage students. Many different types of competitions exist. Example competitions include the ACM International Collegiate Programming Contest [1], ACM Student Research Competition [2], National Collegiate Cyber Defense Competition [4], Facebook Hacker Cup [6], First Robotics competitions [7], Google Code Jam [8], Google Summer of Code [9], Annual Berkeley Programming Competition [10], SCORE [3], International Olympiad in Informatics [11], and the Microsoft Imagine Cup [12]. To our knowledge, our Bug Catcher system is the first specifically for Software Testing Competitions. We anticipate that this system can be used for students without programming experience, but also tailored to experienced students and programmers by modifying the problem sets. In this paper, we focus on the use of Bug Catcher for high school students that may not have programming experience. We attempt to engage them through the competition so that their interest in Computer Science increases.

2. BACKGROUND

Prior to the creation of Bug Catcher, we conducted a mock software testing competition by modifying the intended use of the “Bug Hunt” software developed by Elbaum et al. [5]. Bug Hunt is an educational tool that was not developed for software testing competitions, but rather as a tool to teach students about software testing topics, including black box testing, white box testing, code coverage, and automated unit testing. In addition to the educational material, Bug Hunt allows students to apply their new knowledge in a fun environment. Bug Hunt displays problems and allows students to create test cases by entering input and expected output. As students submit test cases, the system records the number of bugs found. For the mock competition, Bug Hunt was used with the following modifications: (1) there is a required quiz at the beginning of Bug Hunt - we provided the answers so that students could skip the quiz (particularly for those without programming or software testing experience who were unable to answer the questions), (2) students skipped the tutorials, (3) students did not use the module on JUnit, (4) each team was only allowed to log onto one machine using their team leader’s account, and (5) since the system was created to track the total bugs found, not the speed at which bugs are found, a team of volunteers did their best to record the approximate times at which students found the bugs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE ’13, March 6–9, 2012, Denver, Colorado, USA.
Copyright 2013 ACM 978-1-4503-1775-7/13/03 ...$15.00.
For the mock competition, we invited high school, undergraduate, and graduate students to participate. It was held on a Saturday and a total of 59 students competed. The event gave us firsthand experience running a software testing competition and provided feedback to consider when creating Bug Catcher. The follow-up survey was optionally completed by 55 of the participants. In this survey, the most common comments were on the limitations of Bug Hunt for competitions. Students indicated that they wanted the quiz, tutorials, and unused modules to be removed. They also wanted easy navigation between problem sets and real time views of their scores in comparison to the other teams. On the administrative side, volunteers requested easier methods for enrollment, including account creation for both students and teams. They also want the system to automatically compute the rates at which bugs are found and to sort the teams by rankings.

Though students provided constructive criticism, which we address in Bug Catcher, all 55 students reported that they would recommend the event to future students, 43 students communicated an increased interest in Computer Science, and 44 students said the event had increased their interest in Software Testing. In the comments section, many students wrote that they enjoyed the competition and wanted us to hold another event.

Many of the high school students, as well as their supervising high school teacher, commented that this software testing competition was more accessible for students who are new to programming than programming competitions. They felt that students need a lot of programming experience in order to compete in a programming competition, but students without any programming experience could still compete in a software testing competition.

3. BUG CATCHER

Bug Catcher is available at [http://renee.cse.unt.edu](http://renee.cse.unt.edu). Figure 1 shows a student competing. The header displays the timer, the top three teams, the current team’s name, and the total number of bugs that the team has found. The leftmost side shows a requirements, which we also refer to as the problem description. The middle shows the code with options to change the problem and enable code coverage.
highlighting. The right side has input boxes for the input and expected output. On the right side, where students enter test cases, they can hover over the test input and expected output boxes to see an example input and output for the problem. The lower right hand corner has a results box that displays important information such as administrator hints and feedback from test cases that they already entered. We observed that many students relied only on the problem statement (requirements), but some took an interest in the code as time went on. Only a few teams used the code coverage option to highlight executed lines of code.

Figure 2 shows the administrator view for setting up a competition. They select from a list of problems and add the desired problems to the competition. Each problem is ranked by difficulty so that they can plan accordingly for their participant group. The administrator may view the code and list of associated bugs before including them. The administrator also sets a time limit, competition name, competition description, whether the competition is visible to participants yet, and whether to allow code coverage. The password for a competition is optional and facilitates running multiple simultaneous competitions.

Figure 2 shows the administrator view during a competition. The administrator is able to view the scores and rankings for each team in real time. At the top, a timer counts down with the option to pause and resume the competition. Figure 4 shows the screen for the administrator to push hints to the student screens. We have prewritten hints for the problem sets in the system, but the administrator may also create their own custom hint.

While not shown here, the system also allows the administrator to upload new problem sets, including the code, bugs, and optional hints. We have a similar implementation to that of BugHunt [5]. For each test case, we then run the oracle and each of the buggy versions of code, compare the output, and identify bugs based on mismatches between the expected outputs. We require the administrator to upload a correct version of the code (i.e., the oracle) and then a file for each bug that contains compilable code with the respective bug. We require the use of one file per bug in order to minimize the threat of one bug possibly masking another bug. While our current version of Bug Catcher supports Java code, we believe that it will be easy to extend the system to allow other languages in the future.

4. EXPERIMENT

4.1 Research Questions

We examine four research questions:

- **RQ1**: How many bugs does each team find?
- **RQ2**: Do the students recommend this event for future students?
• RQ3: Do the students report an increased interest in Computer Science?

• RQ4: What are suggestions for improving the system?

4.2 Experimental Set Up

We conducted four sessions of a Software Testing Competition over a two day period with 94 high school students that visited campus. The students formed teams of 2 to 3 students. Each competition was 80 minutes long. The following is an example schedule:

- 8:00 - 8:20: Pre-surveys, instructions for the competition, and team registrations
- 8:20 - 9:05: Competition
- 9:05 - 9:20: Post-surveys and prize distribution

The competitions each had 6 to 8 problems that contained a total of 12 to 16 bugs. Each problem was written in Java. Problems range in difficulty and covered a variety of bugs. We used our judgement when classifying the level of difficulty of bugs and peer reviewed them with each other as we created them. Example bugs include simple mutations, for-loops that are off-by-one, division by zero, and not accounting for inputs of different types or sizes. For instance, Figure 5 shows our simplest program that adds two numbers, but with a mutation to subtract instead of add. This bug was found by every student team. This was a good problem because students that relied on the problem statement to develop ideas for tests were able to use their problem solving skills to figure out that the code was subtracting instead of adding and then they could point to the minus sign in the code and start to figure out how the code worked.

To simulate real world testing, the students were not told how many bugs were in each problem. This is a feature of Bug Hunt [5] that we adopt. Therefore, part of the competition is to strategically move between problems.

4.3 Results

RQ1: How many bugs does each team find? Table 1 shows the number of teams, problems, bugs, and percentage of possible bugs that each team found in the four different sections. All of the teams found some of the easy bugs while the most advanced team found as many as 94% of the bugs in their competition. The lowest ranked team in each of the four competitions found between 2 to 5 bugs. That is 17% to 38% of the possible bugs in the different sections. The highest ranked team in each of the four competitions found between 8 to 16 bugs. That is 67% to 94% of the possible bugs in the different sections. On average, teams in the four sections found 53% to 68% of the possible bugs. The median was close to the average and fell between 53% to 77% for the different sections. We interpret these results positively that most students were able to find over half of the bugs and none of the teams were under-challenged as no team found all of the possible bugs. On the other hand, we’ll discuss an observation shortly that the teams that found as few as 2 to 5 bugs during the 40 minutes may have felt frustration and it may be beneficial to insert a larger number of “easy” bugs into the programs to increase their interest and to motivate them to keep trying. Indeed, we observed student excitement when they accomplished finding new bugs.

RQ2. Do the students recommend this event for future students?

In our post-survey, 90% of students said that they would recommend this event for future students. While this is encouraging, we’ll discuss points for improvement shortly.

RQ3. Do the students report an increased interest in Computer Science?

Our post-survey results show that the activity increased interest in Computer Science for 73% of the students. This is lower than the number of students that said that they would
Table 1: Number of teams and the average, median, low, and high percentage of bugs found

<table>
<thead>
<tr>
<th></th>
<th>Competition 1</th>
<th>Competition 2</th>
<th>Competition 3</th>
<th>Competition 4</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of teams</td>
<td>9</td>
<td>12</td>
<td>7</td>
<td>8</td>
<td>36</td>
</tr>
<tr>
<td>Total no. of problems in this session</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>n/a</td>
</tr>
<tr>
<td>Total no. of bugs in the problems</td>
<td>16</td>
<td>13</td>
<td>17</td>
<td>12</td>
<td>n/a</td>
</tr>
<tr>
<td>Average % of bugs found per team</td>
<td>56</td>
<td>68</td>
<td>54</td>
<td>53</td>
<td>58</td>
</tr>
<tr>
<td>Median % of bugs found per team</td>
<td>56</td>
<td>77</td>
<td>53</td>
<td>58</td>
<td>61</td>
</tr>
<tr>
<td>Low % of bugs found per team</td>
<td>46</td>
<td>38</td>
<td>18</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>High % of bugs found per team</td>
<td>92</td>
<td>92</td>
<td>94</td>
<td>67</td>
<td>94</td>
</tr>
</tbody>
</table>

Figure 4: Bug Catcher: Administrator screen to push hints

RQ4. What are suggestions for improving the system?

Our post-survey collected comments on how to improve the event in the future. Some of these comments are encouraging as we see ways to possibly better engage students.

- **Practice problems:** We did not provide a practice round before the competition. Students would like to try the system on their own before the competition begins.

- **Provide information on the number of bugs in each problem:** Some students do not like our design choice that we do not tell them how many bugs are in each problem.

- **Talk more about Computer Science careers:** We spent approximately 5 minutes telling students about career paths in Computer Science, but some students would like us to spend more time on this and provide more time to ask questions before the competition.

- **Teach coding:** Some students wanted us to give a lesson on programming.

We will address two of these suggestions in our ongoing work. First, we created a new system called, “Bug Catcher Lite” which is available at (http://maman.cs.usu.edu/Lite). This program allows students to log on at any time and practice. They do not compete against other teams, but they have the opportunity to practice with sample problems so that they can familiarize themselves with the system and competitions.

We also plan to dedicate a longer amount of time to talk more about Computer Science and careers. We will add this discussion at the end of the competitions instead of at the beginning because we felt that students had more interest in Computer Science after they finished the competition.

We do not plan to provide information on the number of bugs in each problem because we feel that this is an important design decision for our competition and better mimics the real world.

We recommend the event to future students, so we see room for improvement in exciting students about Computer Science. We address some ideas for this shortly in our discussion of suggestions to improve the system.
Finally, we found it exciting that students wanted to know more about coding after the competition. We observed that many students initially ignored the code on their screens and entered test cases based on the requirements. However, as time went on, some students showed more interest in the code because they were motivated to do what they could to find more bugs. After the competition, some students complained that we should have taught them to code before the competition. We view this as a positive comment because it indicates that they want to learn more about programming!

We also asked our volunteers for suggestions to improve the system. These include:

- **Include more bugs**: Include problem sets with more bugs, including many that are at the “easy” level because many students became frustrated if they went more than approximately 10 minutes without finding new bugs.

- **Increase the time**: Increase the amount of time for the competition so that students have more time to find more bugs.

- **Offer the competition to college students and professionals**: Offer competitions to undertake students, graduate students, and professionals by using more complex problem sets and bugs.

We are currently addressing the comments from volunteers. First, it is straightforward to increase the number of “easy” bugs and the amount of time available during the competition. However, we are looking into more complex problem sets for more advanced students and professionals. Our future work explores competitions for these groups.

5. CONCLUSIONS AND FUTURE WORK

To our knowledge, Bug Catcher is the first system designed specifically for Software Testing Competitions. It is a web-based system that can be administered to anyone through a web browser. Surveys conducted during a prototype competition, prior to the development of Bug Catcher, revealed that students had a strong interest in Software Testing competitions and provided us with feedback to create a web application to administer competitions. We then created the Bug Catcher system with this feedback in mind. Using Bug Catcher, we administered four competitions to 94 high school students. The results showed that 67 high school students, or 73%, report an increased interest in Computer Science. Further, 90% recommend this competition for a future event.

Constructive comments include suggestions that we provide a training mode. We have created “Bug Catcher Lite” to address this concern. Students also reported that they wanted to learn more about programming after the event and that they wish that we would have spent more time telling them about career paths in Computer Science. We view these comments as positive indicators that their interest in Computer Science increased and we plan to look into opportunities to address these comments in our future work. Our volunteers suggested that we include more bugs, particularly “easy” bugs since students feel excitement when they find bugs and it may increase the engagement of the lower ranked teams. They also suggested increasing the amount of time for the competition. Our ongoing work seeks to not only address all of these issues, but also to better understand how to fine tune future competitions. We plan to examine the parameters of the problem sets, types and frequencies of bugs that we use, the amount of time that we allocate for competitions, the sizes of the teams, and how to tailor the competition to high school students with different backgrounds. Finally, our ongoing and future work will develop more complex problem sets and bugs that are geared toward college students and professionals.

6. ACKNOWLEDGEMENTS

Thank you to Sebastian Elbaum for providing us with access to Bug Hunt. Thank you to support from National Science Foundation grants #1139840 and #1156487. Thank you to Google for providing the prizes for these competitions and to Utah State University for their support. Thank you to Dr. Daniel Bryce for co-advising the students on this project and providing valuable feedback that helped to improve this paper.

7. REFERENCES


