BugHunt
Making Early Software Testing Lessons Engaging and Affordable

Sebastian Elbaum
K. Stolee, X. Guo, S. Trout, M. Jorde, S. Person, J. Dokulil
University of Nebraska - Lincoln

With support from NSF and UNL
With the patience of several generations of UNL students
What will your students do when they graduate?

“... testing is 50% of software development costs” Beizer

Compounded over the software lifetime
More as complexity and dependability requirements increase

National Institute for Standards and Technology
Cost of faults to US USS 59b
Existing testing techniques could save $22b

“... Microsoft in terms of this quality stuff – we have as many testers as we have developers. And testers spend all their time testing, and developers spend half of their time testing...”

Bill Gates
Once upon a time there was a new faculty member ...
The faculty got ready to take on the challenge only to realize that...
Common Themes

“Testing accounts for about half of the cost of software, but testing receives little treatment in most curricula.”

“Students eschew software testing as though it were poison.”

Teaching good testing practices early helps students (more realistic experience, less bad habits)
Challenges

- Lack of properly-trained instructors
- Lack of physical resources (labs, etc.)
- Early curriculum packed with material
- Not engaging topic for students
Motivation

- Testing is key to software development.
- Delivering early is beneficial.
- Struggle to deliver it cost-effectively.

What are you going to do about it?
Welcome to Bug Hunt
An online tutorial to instill good testing practices through lessons and challenges!

Engage students
- Hands-on testing challenges
- Immediate feedback for students
- Self-paced to accommodate students’ speeds

Deliver it cost-effectively
- Out-of-the-box solution for easy deployment
- Configurable to fit instructor requirements
- Automatic performance assessment
BugHunt: Student Perspective

Registration and Login → Guidelines → Lesson → Summary

Tutorial Results

Congratulations on completing the online testing tutorial 'Triangle' for Test Course. You discovered a total of 4, or 67% of the total bugs, throughout the 4 lessons.

- During lesson 1 you exposed 1 bug in 52 seconds.
- During lesson 2 you exposed 3 bugs in 1 minute, and 23 seconds.
- During lesson 3 you exposed 0 bugs in 1 second.
- During lesson 4 you exposed 0 bugs in 1 second.

Test Execution Log

<table>
<thead>
<tr>
<th>Lesson</th>
<th>Test Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
<th>Fault Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3 3 4</td>
<td>Isosceles</td>
<td>Isosceles</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3 3 3</td>
<td>Equilateral</td>
<td>Equilateral</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0 0 0</td>
<td>Not a Triangle</td>
<td>Equilateral</td>
<td>* findTriangleType() method: first condition in if statement should be &lt;=</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1 2 3</td>
<td>Not a Triangle</td>
<td>Scalene</td>
<td>* findTriangleType() method: first condition in if statement should be &lt;=</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4 3 5</td>
<td>Scalene</td>
<td>Isosceles</td>
<td>* smallest() method: second return should be side2 instead of side1.</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>100 100 100</td>
<td>Equilateral</td>
<td>Equilateral</td>
<td>* middle() method: return should be side1 instead of side2.</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>3 0 0</td>
<td>Invalid input(s)</td>
<td>Equilateral</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0 3 3</td>
<td>Isosceles</td>
<td>Isosceles</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>4 5 6</td>
<td>Scalene</td>
<td>Scalene</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>5 6 6</td>
<td>Scalene</td>
<td>Scalene</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>3 4 5</td>
<td>Scalene</td>
<td>Scalene</td>
<td></td>
</tr>
</tbody>
</table>

Thank you for trying out the BugHunt demo. If you are an instructor and you would like to use a BugHunt tutorial in your courses or labs, please contact Dr. Sebastian Elbaum (elbaum@ece.unl.edu)
BugHunt: Student Perspective

Lesson

Objective

Exercise

Results

Lesson Objectives

Lesson 1 - Basic Testing Concepts

Testing consists of operating a program and validating that it behaves as expected. A program is operated by executing it with different inputs. In your first lesson you must provide inputs that you consider important and check whether the outcome is what you expected.

At the end of this lesson you should be familiar with three important testing concepts:

1. Test Case: a set of inputs and expected outputs for a given program.
2. Test Suite: a group of related test cases.
3. Test Execution Log: this log is a record of the tests you have created.

In this lesson, you will create a test suite (i.e., a set of test cases) to locate the faults in the Java program provided by your instructor.
BugHunt: Student Perspective
BugHunt: Student Perspective

Lesson
Objective
Exercise
Results

Instructions
If this is your first time using BugHunt, look at the Hints right now. None are highlighted in the 'Requirements' box. The line containing the bug will be space-separated and the expected output is case sensitive. Any function names you enter should be in lowercase.

Test Execution Log
- Number
- Input
- Expected Output
- Actual Output

Current Message: Determine which side is the largest.
Bug Hunt: Student Perspective

Lesson

Objective

Exercise

Results

Lesson - Basic Testing Concepts

Instructions

If this is your first time using BugHunt, look at the Hints right now. Your job is to create a test case that will discover a given bug. The requirements affected by the bug are highlighted in the "Requirements" box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the "Source Code" box. Inputs should be space-separated and the expected output is case-sensitive. An example input would be '4 4 4' and expected output 'Equilateral'.

Requirements

1. The Triangle program accepts three integer values as input. Each value represents a side of the triangle.
2. If the inputs are invalid (sides smaller than 0, or not integers) or if fewer than three values are provided the program outputs the message "Invalid input value(s)".
3. If the length of the largest side is greater or equal to the sum of the lengths of the two smaller sides the program will output the message "Not a Triangle".
4. If all three sides of the triangle are of equal length the program will output the message "Equilateral".
5. If exactly two sides of the triangle are of equal length the program will output the message "Isosceles".
6. If all three sides of the triangle are of different lengths the program will output the message "Scalene".

Test Case Input

Input Value(s):

Expected Output:

Test Execution Log

Number | Input | Expected Output | Actual Output

Previous | Next
BugHunt: Student Perspective

Lesson

Objective

Exercise

Results

Instructions

If this is your first time using BugHunt, look at the hints right now. Now, in this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case sensitive. An example input would be '4 4' and expected output 'Equivalent'.

Source Code

```java
import java.io.*;
public class Triangle {

/* Declare side variables and set default values to 0 */
protected static int firstSide = 0;
protected static int secondSide = 0;
protected static int thirdSide = 0;

/* Determine which side is the largest */
public static int largest(int side1, int side2, int side3) {
    if (((side1 <= side2) && (side2 <= side3)) || ((side2 <= side1) || (side3 <= side1)) {
        return side3;
    } else if (((side1 <= side2) && (side3 <= side2)) || ((side3 <= side1) || (side2 <= side1)) {
        return side2;
    } else {
        return side1;
    }
}
```
BugHunt: Student Perspective

Lesson Objectives

Exercise

Results

Lesson 1 - Basic Testing Concepts

Instructions:

If this is your first time using BugHunt, look at the Hints right now. NOW In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug:

1. Where do I start?
   You start testing by selecting inputs worth testing. The 'Add Test' button will actually run the program with the inputs you provide and check to see if the actual output matches the output you expected.
   - If they match then the test case Passed (P).
   - If they do not match then two things may have happened:
     - Your expected output was wrong in which case the 'Invalid expected output' message will appear.
     - You found a fault in the program and the number of faults found is increased by one (Good Job!).

2. How many inputs do I specify?
   The choice and number of inputs is completely up to you. When you feel satisfied with your results (as per the execution log) move to the next lesson. In general, it is impossible to provide all of the possible inputs to a program. The tester's job is to select from the universe of inputs, the inputs that are most likely to expose a fault.

3. Why do I need to specify an expected output?
   Expected outputs are necessary to determine whether a test case passes or not. Sometimes, testers do this implicitly by checking the output of a test case. However, specifying the expected output is not only a good practice, but it can help you to develop a fully automated test suite.

4. How can I determine the expected values without running the program with that input?
   Expected values can be derived from the program's requirements or specifications. If these are not available, you may need to consult with the program's developers or documentation.
BugHunt: Student Perspective

Lesson 1 - Basic Testing Concepts

Instructions:
If this is your first time using BugHunt, look at the hints right now. Now your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case sensitive. An example input would be '4 4' and expected output 'Equal triangle'.

Test Case Input

Input Value(s):

Expected Output:

Add Test

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
</table>

Add Test
Bug Hunt: Student Perspective

Lesson
Objective
Exercise
Results

Instructions
If this is your first time using Bug Hunt, look at the hints right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case sensitive. An example input would be '3 4 4' and expected output 'Isosceles'.

Requirements
1. The Triangle program accepts three integer value input. Each value represents a side of the triangle.
2. If any of the sides is invalid (less than 0 or larger than 100), the program outputs: 'Invalid input value' and exit.
3. If the length of the longest side is greater or equal to the sum of the lengths of the two smallest sides, program will output the message "Not a Triangle".
4. If any of the sides of the triangle are of equal length, program will output the message "Isosceles".
5. If all three sides of the triangle are of equal length, the program will output the message "Equilateral".
6. If any two sides of the triangle are of equal length, the program will output the message "Isosceles".
7. If all three sides of the triangle are of different length, the program will output the message "Scalene".

Test Case Input
Input Value(s): 3 3 4
Expected Output: Isosceles

Test Execution Log
<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bug Hunt: Student Perspective

Lesson Objectives

Exercise

Results

Instructions:
If this is your first time using Bug Hunt, look at the hints right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case sensitive. An example input would be '4 4' and expected output 'Equilateral'.

Requirements:
1. The Triangle program accepts three integer-valued inputs. Each value represents a side of the triangle.
2. If the inputs are invalid (less than 0 or not integers) or if any three values are provided, the program outputs the message 'Invalid input values.'
3. If the length of the longest side is greater or equal to the sum of the lengths of the two smaller sides, the program will output the message 'Not a Triangle'.
4. If all three sides of the triangle are of equal length, the program will output the message 'Equilateral'.
5. If exactly two sides of the triangle are equal, the program will output the message 'Isosceles'.
6. If all three sides of the triangle are different, the program will output the message 'Scalene'.

Test Case Input:
Input Value(s):
Expected Output:

Add Test

Test Execution Log:

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 3 4</td>
<td>Isosceles</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>
BugHunt: Student Perspective

Lesson Objectives Results
Exercise

**Lesson 1 - Basic Testing Concepts**

**Instructions**
If this is your first time using BugHunt, look at the hints right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case sensitive. An example input would be '4 4 4' and expected output 'Equilateral'.

**Requirements**
1. The Triangle program accepts three integer-valued input. Each value represents a side of the triangle.
2. If the inputs are invalid (either smaller than 0 or integers or fewer than three values are provided), program output the message "Invalid Input value(s)"
3. If the length of the longest side is greater than the sum of the lengths of the two shorter sides, program will output the message "Not a Triangle"
4. If all sides of the triangle are equal, the program will output the message "Equilateral"
5. If exactly two sides of the triangle are equal, the program will output the message "Isosceles"
6. If all sides of the triangle are different, the program will output the message "Scalene"

**Test Case Input**

- **Input Value(s):**
- **Expected Output:**

**Test Execution Log**

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0 0 0</td>
<td>Not a Triangle</td>
<td>Equilateral</td>
</tr>
<tr>
<td>2</td>
<td>3 3 3</td>
<td>Equilateral</td>
<td>Equilateral</td>
</tr>
</tbody>
</table>

findTriangleType()
method: first condition in if statement should be <=.
BugHunt: Student Perspective

Lesson 1 - Basic Testing Concepts

Instructions

If this is your first time using BugHunt, look at the Hints right now. Now! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case sensitive. An example input would be '4 4' and expected output 'Equilateral'.

Test Case Input

Input Value(s):

Expected Output:

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3 3 3</td>
<td>Sealed</td>
<td>Equilateral</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0</td>
<td>Not a Triangle</td>
<td>Equilateral</td>
</tr>
<tr>
<td>2</td>
<td>3 3 3</td>
<td>Equilateral</td>
<td>Equilateral</td>
</tr>
<tr>
<td>1</td>
<td>3 3 4</td>
<td>Isosceles</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>
BugHunt: Student Perspective

Lesson

Objective

Exercise

Results

Instructions

If this is your first time using BugHunt, look at the Hint right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case sensitive. An example input would be '4 4 4' and expected output 'Equilateral'.

Requirements

1. The Triangle program accepts three integer value input. Each value represents a side of the triangle.
2. If the inputs are invalid (less than 0, or integers or if none of these values are provided) the program outputs the message "Invalid input value(s)"
3. If the length of the longest side is greater or equal to the sum of the lengths of the two smaller sides the program will output the message "Not a Triangle"
4. If all sides of the triangle are of equal length the program will output the message "Equilateral"
5. If exactly two sides of the triangle are of equal length the program will output the message "Isosceles"
6. If all three sides of the triangle are of different lengths the program will output the message "Triangle"

Test Case Input

Input Value(s):

Expected Output:

Add Test

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3 3 3</td>
<td>Scalene</td>
<td>Equilateral</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Equilateral</td>
<td>Equilateral</td>
</tr>
<tr>
<td>2</td>
<td>3 2</td>
<td>Equilateral</td>
<td>Equilateral</td>
</tr>
<tr>
<td>1</td>
<td>3 3 3</td>
<td>Equilateral</td>
<td>Equilateral</td>
</tr>
</tbody>
</table>
BugHunt: Student Perspective

Lesson

Objective

Exercise

Results

Lesson 1 - Basic Testing Concepts

Results

Next work, Demo User, you uncovered one bug in this lesson, which is 16% of the total bugs. Although this is less than the average number of bugs discovered on average during this lesson, you still have three more lessons to hunt down the remaining bugs and practice your software testing skills. Be sure to carefully read through the source and/or requirements and create test cases you think will reveal more bugs.

The Test Execution Log shown below details the results of your testing efforts so far. After you have reviewed the log, proceed to the next lesson for your next testing challenge. Good Luck!

<table>
<thead>
<tr>
<th>Lesson</th>
<th>Test Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
<th>Fault Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3.3.4</td>
<td>Isosceles</td>
<td>Isosceles</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3.3.3</td>
<td>Equilateral</td>
<td>Equilateral</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0.0.0</td>
<td>Not a Triangle</td>
<td>Equilateral</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3.3.3</td>
<td>Isosceles</td>
<td>Isosceles</td>
<td></td>
</tr>
</tbody>
</table>

Quit  | Hints  | Next Lesson  | Previous  | Next
Welcome to Bug Hunt
An online tutorial to instill good testing practices through lessons and challenges!

Engage students
- Hands-on testing challenges
- Immediate feedback for students
- Self-paced to accommodate students' speeds

Deliver it cost-effectively
- Out-of-the-box solution for easy deployment
- Configurable to fit instructor requirements
- Automatic performance assessment
Lessons’ Challenges

Lesson 1: Terminology → Find target fault
Lesson 2: Black-box → Exercise outputs
Lesson 3: White-box → Cover code blocks
Lesson 4: Automation & regression → Reduce test suite size
Lesson 5: Fixing → Choose right fixes
Feedback Mechanisms

Lesson 1 - Basic Testing Concepts

Instructions
If this is your first time using BugHunt, look at the Hints right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the "Requirements" box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the "Source Code" box. Inputs should be space-separated and the expected output is case sensitive. An example input would be "4 4 4" and expected output "Equivalent".

Test Case Input

Input Value(s): [ ] [ ]

Expected Output: [ ]

Add Test

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lesson Objectives

Exercise

Lesson 1 - Basic Testing Concepts

Instructions

If this is your first time using BugHunt, look at the Hints right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the "Requirements" box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the "Source Code" box. Inputs should be space-separated and the expected output is case sensitive. An example input would be "4 4 4" and expected output "Equilateral".

Requirements

1. The Triangle program accepts three integer values as input. Each value represents a side of theTriangle.
2. If the input values are invalid (less than 0 or non-integer) or if the sum of the input values is odd, the program outputs the message "Invalid Input Value(s)"
3. If the sum of the input values is even, the program calculates the sum of the lengths of the two smaller sides. If the sum is less than or equal to the length of the longest side, the program outputs the message "Isosceles"
4. If one of the input values is invalid, the program outputs the message "Invalid Input Value(s)"
5. If all three sides of the triangle are of equal length, the program outputs the message "Equilateral"
6. If any of the input values is invalid, the program outputs the message "Invalid Input Value(s)"

Test Case Input

Input Value(s): 3 3 4
Expected Output: Isosceles

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
</table>

Add Test
Feedback Mechanisms

If this is your first time using BugHunt, look at the Hills right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case-sensitive. An example input would be '4 4 4' and expected output 'Equilateral'.

Test Case Input

<table>
<thead>
<tr>
<th>Input Value(s):</th>
<th>Expected Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add Test

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 3 4</td>
<td>Isosceles</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>
Feedback Mechanisms

Lesson 1 - Basic Testing Concepts

Instructions

If this is your first time using Bug Hunt, look at the Hills right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the "Requirements" box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the "Source Code" box. Inputs should be space-separated and the expected output is case sensitive. An example input would be "4 4 4" and expected output "Equilateral".

Requirements

1. The Triangle program accepts three integer value inputs. Each value represents a side of the triangle.
2. If the inputs are invalid (less than 0 or greater than 999999999) or if there are only two provided, the program outputs the message "Invalid Input Value(s)."
3. The program will output the message "Not a Triangle" if the sum of the lengths of the two smaller sides equals the length of the largest side.
4. If all three sides of the triangle are of equal length, the program will output the message "Equilateral."
5. If the program outputs the message "Isosceles" or "Scalene", it implies that all three sides are distinct.
6. If the program outputs the message "Scalene" or "Equilateral" and an input value is less than 0, the program will output the message "Invalid Input Value(s)."

Test Case Input

Input Value(s):
Expected Output:

Add Test

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0 0 0</td>
<td>Not a Triangle</td>
<td>Equilateral</td>
</tr>
<tr>
<td>2</td>
<td>3 3 3</td>
<td>Isosceles</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

findTriangleType() method: first condition in if statement should be <=.
Feedback Mechanisms

Lesson 1 - Basic Testing Concepts

Instructions

If this is your first time using BugHunt, look at the Help now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the "Requirements" box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the "Source Code" box. Inputs should be space-separated and the expected output is case sensitive. An example input would be "4 4 4" and expected output "Equilateral".

Requirements

1. The Triangle program accepts three integer value input. Each value represents a side of the triangle.
2. If the inputs are invalid (less than 1 or integer) or fewer than three values are provided, the program outputs the message "invalid input".
3. If the length of the longest side of the triangle is greater than the sum of the lengths of the two smaller sides, the program will output the message "Not a Triangle".
4. If all three sides of the triangle are of equal length, the program will output the message "Equilateral".
5. If two of the triangle's sides are of equal length, the program will output the message "Isosceles".
6. If none of the triangle's sides are of equal length, the program will output the message "Scalene".
7. The program will output the message "scalars" if one of the triangle's sides is of different length.
8. The program will output the message "blank" if no input is given.

Test Case Input

Input Value(s):

Expected Output:

Add Test

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3 3 3</td>
<td>Scalene</td>
<td>Equilateral</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0</td>
<td>Not a Triangle</td>
<td>Equilateral</td>
</tr>
<tr>
<td>2</td>
<td>3 3 3</td>
<td>Equilateral</td>
<td>Equilateral</td>
</tr>
<tr>
<td>1</td>
<td>3 3 4</td>
<td>Isosceles</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>
Feedback Mechanisms

Lesson 1 - Basic Testing Concepts

Instructions
If this is your first time using BugHunt, look at the hints right now. NOW! In this lesson, your job is to create a test case that will uncover a given bug. The requirements affected by the bug are highlighted in the 'Requirements' box. The line containing the bug, or the line after the bug if the bug is caused by missing code, is highlighted in the 'Source Code' box. Inputs should be space-separated and the expected output is case sensitive. An example input would be '4 4 4' and expected output 'Equilateral'.

Requirements
1. The Triangle program accepts three integer-valued input. Each value represents a side of the triangle.
2. If the inputs are invalid (smaller than 1 or integer) or if values are not provided, the program outputs the message "Invalid input value(s)".
3. If the sum of the lengths of the two sides is less than or equal to the length of the third side, program will display the message "Not a Triangle".
4. If all three sides of the triangle are of equal length, program will output the message "Equilateral".
5. If the sum of lengths of two sides is less than the third side, program will output the message "Isosceles".
6. If any side of the triangle is of different length, program will output the message "Scalene".

Test Case Input
Input Value(s):
Expected Output:

Test Execution Log
<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3 3 3</td>
<td>Scalene</td>
<td>Equilateral</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>Equilateral</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>Equilateral</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

The Expected Output for this test case should be: Equilateral
Behind the scenes

Browser

Test Case Input

Test Execution Log

Server

Database

Inputs + expected outputs

Test results

BugHunt Backend
Behind the scenes

Faulty Output

Test Case Input

Input Value(s): 3 3 4
Expected Output: 18

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Oracle error

Yes

No
Behind the scenes

Test Case Input
- Input Value(s): 3 3 4
- Expected Output: isosceles

Test Execution Log

Correct

Output

Yes

No

Output

Output

Output
Feedback Mechanisms

Lesson 1 - Basic Test

Instructions
If this is your first time using BugHunt, look at the Hints right now. NOW! In this lesson, your job is to find the bug in the Triangle program. The line containing the bug, or the line after the bug if the bug spans to the next line, is highlighted in the Requirements box. The line containing the bug, or the line after the bug if the bug spans to the next line, is highlighted in the Source Code box. A line containing the expected output is case sensitive. An example input would be "4 4 4" and an example output would be "Equilateral".

Requirements
1. The Triangle program accepts three integer values as input. Each value represents a side of the triangle.
2. If the inputs are invalid (less than 0 or not integers) or fewer than three values are provided, the program outputs the message "invalid input values!".
3. If the sum of the lengths of the two smaller sides is less than the length of the longest side, the program outputs the message "This is a Triangle!"
4. If all three sides of the triangle are of equal length, the program will output the message "Equilateral!"
5. If one side is longer than the sum of the lengths of the other two sides, the program will output the message "Concave!"
6. If all sides of the triangle are of different length, the program will output the message "Subtype!"

Source Code
```java
public class Triangle {
    public static void main(String[] args) {
        int side1, side2, side3;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter three side values: ");
        side1 = sc.nextInt();
        side2 = sc.nextInt();
        side3 = sc.nextInt();
        if (side1 < 0 || side2 < 0 || side3 < 0) {
            System.out.println("Invalid input values!");
        } else {
            if (side1 + side2 > side3 && side2 + side3 > side1 && side1 + side3 > side2) {
                System.out.println("This is a Triangle!");
            } else {
                System.out.println("This is a Concave!");
            }
        }
    }
}
```

Bugs you found: 1
Class average: 3

Test Execution Log

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
On Lesson 2

Lesson 2 - Black Box Testing

Instructions
With Black-Box testing, you don't use the source code at all. Instead you must focus on what the program is supposed to do as defined in the Requirements. So, read the requirements carefully to validate the program.

Outputs Covered

- Scalene
- Isosceles
- Equilateral
- Not a Triangle
- Invalid input value(s)

Passed Test Cases / Total Executed (per output)
On Lesson 3

Source Code

```java
/* Determine which side is the largest */
public static int largest(int side1, int side2, int side3) {
    if (((side1<=side2) && (side2<=side3)) || ((side2<=side1)) || (side3<=side1))
        return side3;
    else if (((side1<=side3) && (side3<=side2)) || ((side3<=side1)) || (side2<=side1))
        return side2;
    else
        return side1;
}

/* Determine which side is the middle side */
public static int middle(int side1, int side2, int side3) {
    if (((side1<=side2) && (side2<=side3)) || ((side2<=side1) && (side3<=side1))
        return side2;
    else if (((side1<=side3) && (side3<=side2)) || ((side3<=side1)) || (side2<=side1))
        return side3;
    else
        return side2;
}

/* Determine which side is the smallest */
public static int smallest(int side1, int side2, int side3) {
    // Code snippet...
}
```

3 - White Box Testing

Code Covered

- Not covered
- Covered once
- Covered 2–5 times
- Covered 6–10 times
- Covered more than 10 times

You have reached 83% coverage.
On Lesson 4
On lesson 5 (evolving)
Lesson 5 - Bug Fixes

Instructions

This lesson will test your ability to find and apply fixes to the bugs that you have found. Validation of your fixes will be based on running tests you have chosen in lesson 4 of bug hunt. The textbox to the left is the code containing bugs. Each bug is highlighted and you can choose a fix to the bug by clicking on the highlighted region or the drop down arrow. You can click on the "run tests" button anytime to run your Junit test to validate your fix.

Bugs in Code

```java
import java.io.*;
public class Triangle {

  /* Declare side variables and set default values to 0 */
  protected static int firstSide = 0;
  protected static int secondSide = 0;
  protected static int thirdSide = 0;

  /* Determine which side is the largest */
  public static int largest(int side1, int side2, int side3) {
    if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3)))
      return side1;
    else if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3)))
      return side2;
    else if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3)))
      return side3;
    else
      return side2;
  }

  /* Determine which side is the smallest */
  public static int smallest(int side1, int side2, int side3) {
    /* Code */
  }
}
```

Test Suite

<table>
<thead>
<tr>
<th>Pass/Fail</th>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>1</td>
<td>15 3</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>2</td>
<td>2 2 4</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
<td>3 4 2</td>
<td>Scalene</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

Fixes Graph

Number of Fixes Attempted

Number of Tests: 1
Number of Failing Tests: 2
Number of Fixes Found: 3
Class average: 2
Bugs remaining: 3

Quit  Hints  Previous  Next
On lesson 5 (evolving)

Lesson 5 - Bug Fixes

Instructions

This lesson will test your ability to find and apply fixes to the bugs that you have found. Validation of your fixes will be based on running tests you have chosen in lesson 4 of bug hunt. The textbox to the left are the code containing bugs. Each bug is highlighted and you can choose a fix to the bug by clicking on the highlighted region or the drop down arrow. You can click on the “run tests” button anytime to run your Junit test to validate your fix.

Bugs in Code

```
import java.io.*;
public class Triangle {

    // Declare side variables and set default values to 0 */
    protected static int firstSide = 0;
    protected static int secondSide = 0;
    protected static int thirdSide = 0;

    // Determine which side is the largest */
    public static int largest(int side1, int side2, int side3) {
        if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3))
            System.out.println("Invalid input value(s)");
            return;
        }

        if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3))
            return side2;
        else if (((side1==side3) && (side3==side2)) || ((side3==side1) && (side1==side2))
            return side3;
        else
            return side2;
    }

    // Determine which side is the smallest */
    public static int smallest(int side1, int side2, int side3) {
        if (((side1==side2) && (side2==side3)) || ((side3==side2) && (side2==side3))
            return side1;
    }
```

Test Suite

<table>
<thead>
<tr>
<th>Pass/Fail</th>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>1</td>
<td>1 5 3</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>2</td>
<td>2 2 4</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
<td>3 4 2</td>
<td>Scalene</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

Fixes Graph

Number of
Failing Tests

1
Number of Fixes Attempted
On lesson 5 (evolving)

Lesson 5 - Bug Fixes

Instructions
This lesson will test your ability to find and apply fixes to the bugs that you have found. Validation of your fixes will be based on running tests you have chosen in lesson 4 of bug hunt. The textbox to the left are the code containing bugs. Each bug is highlighted and you can choose a fix to the bug by clicking on the highlighted region or the drop down arrow. You can click on the "run tests" button anytime to run your JUnit test to validate your fix.

Bugs in Code

```java
import java.io.*;
public class Triangle {

    /* Declare side variables and set default values to 0 */
    protected static int firstSide = 0;
    protected static int secondSide = 0;
    protected static int thirdSide = 0;

    /* Determine which side is the largest */
    public static int largest(int side1, int side2, int side3) {
        if (((side1<=side2) && (side2<=side3)) || ((side2<=side1) && (side1<=side2)))
            return side3;
        else if (((side1<=side3) && (side3<=side2)) || ((side3<=side1) && (side1<=side2)))
            return side2;
        else
            return side1;
    }

    /* Determine which side is the middle side */
    public static int middle(int side1, int side2, int side3) {
        if (((side1<=side2) && (side2<=side3)) || ((side2<=side1) && (side1<=side2)))
            return side2;
        else if (((side1<=side3) && (side3<=side2)) || ((side3<=side1) && (side1<=side2)))
            return side3;
        else
            return side2;
    }

    /* Determine which side is the smallest */
    public static int smallest(int side1, int side2, int side3) {
        if (((side1<=side2) && (side2<=side3)) || ((side2<=side1) && (side1<=side3)))
            return side1;
    }
}
```

Test Suite

<table>
<thead>
<tr>
<th>Pass/Fail</th>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>1</td>
<td>15 3</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>2</td>
<td>2 2 4</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
<td>3 4 2</td>
<td>Scalene</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

Fixes Graph

Number of Fixes Attempted: 3

Failing Tests: 2

Previous, Next
On lesson 5 (evolving)
On lesson 5 (evolving)

Instructions

This lesson will test your ability to find and apply fixes to the bugs that you have found. Validation of your fixes will be based on running tests you have chosen in lesson 4 of bug hunt. The textbox to the left are the code containing bugs. Each bug is highlighted and you can choose a fix to the bug by clicking on the highlighted region or the drop down arrow. You can click on the "run tests" button anytime to run your Junit test to validate your fix.

Bugs in Code

```java
import java.io.*;
public class Triangle {

    // Declare side variables and set default values to 0
    protected static int firstSide = 0;
    protected static int secondSide = 0;
    protected static int thirdSide = 0;

    // Determine which side is the largest
    public static int largest(int side1, int side2, int side3) {
        if ((side1 == side2) || (side2 == side3) || (side1 == side3))
            return (side1);
        else if ((side1 == side2) || (side2 == side3) || (side1 == side3))
            return (side2);
        else if ((side1 == side2) || (side2 == side3) || (side1 == side3))
            return (side3);
        else if ((side1 == side2) || (side2 == side3) || (side1 == side3))
            return (side1);
        else
            System.out.println("Invalid input value(s)");
            return;
    }

    // Determine which side is the smallest
    public static int smallest(int side1, int side2, int side3) {
        if ((side1 == side2) || (side2 == side3) || (side1 == side3))
            return side1;
    }
}
```

Test Suite

<table>
<thead>
<tr>
<th>Pass/Fail</th>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>1</td>
<td>15 3</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>2</td>
<td>22 4</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
<td>34 2</td>
<td>Scalene</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

Fixes Graph

- Number of Passing Tests: 2
- Number of Fixes Attempted: 3

Quit  Hints  Previous  Next
Lesson 5 - Bug Fixes

Instructions

This lesson will test your ability to find and apply fixes to the bugs that you have found. Validation of your fixes will be based on running tests you have chosen in lesson 4 of bug hunt. The textbox to the left are the code containing bugs. Each bug is highlighted and you can choose a fix to the bug by clicking on the highlighted region or the drop down arrow. You can click on the “run tests” button anytime to run your JUnit test to validate your fix.

Bugs in Code

```java
import java.io.*;
public class Triangle {

    /* Declare side variables and set default values to 0 */
    protected static int firstSide = 0;
    protected static int secondSide = 0;
    protected static int thirdSide = 0;

    /* Determine which side is the largest */
    public static int largest(int side1, int side2, int side3) {
        if ((side1==side2) & (side2==side3) || (side2==side1) & (side1==side3))
            return side1;
        if ((side1==side2) & (side2==side3) || (side2==side1) & (side1==side3))
            return side2;
        if ((side1==side2) & (side2==side3) || (side2==side1) & (side1==side3))
            return side3;
        else if (side1<side2 & side2<side3) {
            System.out.println("Invalid input value(s)");
            return;
        }
        if (side1<side2 & side2<side3) {
            return side1;
        } else if (side1<side3 & side3<side2) {
            return side1;
        } else if (side2<side1 & side1<side3) {
            return side2;
        } else if (side2<side1 & side1<side3) {
            return side2;
        } else if (side3<side1 & side1<side2) {
            return side3;
        } else if (side3<side1 & side1<side2) {
            return side3;
        } else {
            return side1;
        }
    }

    /* Determine which side is the smallest */
    public static int smallest(int side1, int side2, int side3) {
        if ((side1==side2) & (side2==side3) || (side2==side1) & (side1==side3))
            return side1;
        if ((side1==side2) & (side2==side3) || (side2==side1) & (side1==side3))
            return side2;
        if ((side1==side2) & (side2==side3) || (side2==side1) & (side1==side3))
            return side3;
        else if (side1<side2 & side2<side3) {
            System.out.println("Invalid input value(s)");
            return;
        }
        if (side1<side2 & side2<side3) {
            return side1;
        } else if (side1<side3 & side3<side2) {
            return side1;
        } else if (side2<side1 & side1<side3) {
            return side2;
        } else if (side2<side1 & side1<side3) {
            return side2;
        } else if (side3<side1 & side1<side2) {
            return side3;
        } else if (side3<side1 & side1<side2) {
            return side3;
        } else {
            return side1;
        }
    }
}
```

Test Suite

<table>
<thead>
<tr>
<th>Pass/Fail</th>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>1</td>
<td>1 5 3</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>2</td>
<td>2 2 4</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
<td>3 4 2</td>
<td>Scalene</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

Fixes Graph

Number of Fixes Attempted

Number of Fixing Tests: 3

Quit  Hints  Previous  Next
On lesson 5 (evolving)

Instructions

This lesson will test your ability to find and apply fixes to the bugs that you have found. Validation of your fixes will be based on running tests you have chosen in lesson 4 of bug hunt. The textbox to the left are the code containing bugs. Each bug is highlighted and you can choose a fix to the bug by clicking on the highlighted region or the drop down arrow. You can click on the “run tests” button anytime to run your Junit test to validate your fix.

Bugs in Code

```java
import java.io.*;
public class Triangle {

    /* Declare side variables and set default values to 0 */
    protected static int firstSide = 0;
    protected static int secondSide = 0;
    protected static int thirdSide = 0;

    /* Determine which side is the largest */
    public static int largest(int side1, int side2, int side3) {
        if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3)))
            return side3;
        else if (((side1==side3) && (side3==side2)) || ((side3==side1) && (side1==side2)))
            return side2;
        else
            return side1;
    }

    /* Determine which side is the middle side */
    public static int middle(int side1, int side2, int side3) {
        if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3))
            return side2;
        else if (((side1==side3) && (side3==side2)) || ((side3==side1) && (side1==side2))
            return side3;
        else
            return side2;
    }

    /* Determine which side is the smallest */
    public static int smallest(int side1, int side2, int side3) {
        if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3))
            return side1;
        else
            return side2;
    }
}
```

Test Suite

<table>
<thead>
<tr>
<th>Pass/Fail</th>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>1</td>
<td>1 5 3</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>2</td>
<td>2 2 4</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
<td>3 4 2</td>
<td>Scalene</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

Fixes Graph

Number of Fixing Tests: 3

Number of Fixing Tests: 2

Number of Fixing Tests: 1
On lesson 5 (evolving)

Lesson 5 - Bug Fixes

Instructions

This lesson will test your ability to find and apply fixes to the bugs that you have found. Validation of your fixes will be based on running tests you have chosen in lesson 4 of bug hunt. The textbox to the left are the code containing bugs. Each bug is highlighted and you can choose a fix to the bug by clicking on the highlighted region or the drop down arrow. You can click on the “run tests” button anytime to run your JUnit test to validate your fix.

Bugs in Code

```java
import java.io.*;

public class Triangle {

  // Declare side variables and set default values to 0 */
  protected static int firstSide = 0;
  protected static int secondSide = 0;
  protected static int thirdSide = 0;

  // Determine which side is the largest */
  public static int largest(int side1, int side2, int side3) {
    if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3)))
      return side3;
    else if (((side1==side3) && (side1==side2)) || ((side3==side1) && (side1==side2)))
      return side2;
    else
      return side1;
  }

  // Determine which side is the middle side */
  public static int middle(int side1, int side2, int side3) {
    if (((side1==side2) && (side2==side3)) || ((side2==side1) && (side1==side3)))
      return side2;
    else if (((side1==side3) && (side1==side2)) || ((side3==side1) && (side2==side1)))
      return side3;
    else
      return side2;
  }

  // Determine which side is the smallest */
  public static int smallest(int side1, int side2, int side3) {
    if (((side1==side2) && (side2==side3)) || ((side3==side2) && (side1==side3)))
      return side1;
  }
}
```

Test Suite

<table>
<thead>
<tr>
<th>Pass/Fail</th>
<th>Number</th>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>pass</td>
<td>1</td>
<td>1 5 3</td>
<td>Not a Triangle</td>
<td>Not a Triangle</td>
</tr>
<tr>
<td>Fail</td>
<td>2</td>
<td>2 2 4</td>
<td>Not a Triangle</td>
<td>Isosceles</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
<td>3 4 2</td>
<td>Scalene</td>
<td>Isosceles</td>
</tr>
</tbody>
</table>

Run Tests

Fixes Graph

Number of Fixes Attempted:

<table>
<thead>
<tr>
<th>Number of Failing Tests</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>
Welcome to Bug Hunt

An online tutorial to instill good testing practices through lessons and challenges!

Engage students

- Hands-on testing challenges
- Immediate feedback for students
- Self-paced to accommodate students’ speeds

Deliver it cost-effectively

- Out-of-the-box solution for easy deployment
- Configurable to fit instructor requirements
- Automatic performance assessment
Self-paced
Welcome to Bug Hunt

An online tutorial to instill good testing practices through lessons and challenges!

Engage students

✓ Hands-on testing challenges
✓ Immediate feedback for students
✓ Self-paced to accommodate students’ speeds

Deliver it cost-effectively

✓ Out-of-the-box solution for easy deployment
✓ Configurable to fit instructor requirements
✓ Automatic performance assessment
BugHunt for Instructors
Out of the Box

Find / try

Instructor
Bug Hunt for Instructors
Out of the Box

Find / try
Query (email)

Instructor

BugHunt Administrator

Instructor request Nov 7th 2010
"I teach software test principles & concepts at ... Classes are CE, for adults. I would like to add BugHunt to my teaching resources, please"
BugHunt for Instructors
Out of the Box

Instructor

Find / try

Query (email)

Information (email)

Configure

BugHunt Administrator
BugHunt for Instructors
Configurable
## BugHunt for Instructors

### Performance Assessment

<table>
<thead>
<tr>
<th>Lesson 1</th>
<th>Lesson 2</th>
<th>Lesson 3</th>
<th>Lesson 4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Cases</td>
<td>Faults</td>
<td>Duration</td>
<td>Test Cases</td>
<td>Faults</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>44:06</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>54</td>
<td>8</td>
<td>24:51</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>15:25</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>11:28</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>05:56</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>09:47</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>14:53</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>20:59</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>15:26</td>
<td>43</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>33:45</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>18:21</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>06:10</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>31</td>
<td>7</td>
<td>53:03</td>
<td>36</td>
<td>1</td>
</tr>
</tbody>
</table>

Averages: 12.14, 3.50, 19:35, 13.14, 2.50, 11:23, 14.21, 0.93, 12:40, 3.86, 0.50, 10:26

Overall Total: 43.36, 7.43, 54:05
Welcome to Bug Hunt
An online tutorial to instill good testing practices through lessons and challenges!

Engage students

 ✓ Hands-on testing challenges
 ✓ Immediate feedback for students
 ✓ Self-paced to accommodate students’ speeds

Deliver it cost-effectively

 ✓ Out-of-the-box solution for easy deployment
 ✓ Configurable to fit instructor requirements
 ✓ Automatic performance assessment
Taking Stock

Lack trained instructors ➔ Self-contained unit
Lack physical resources ➔ No lab space needed
Student low interest ➔ Challenges & feedback
Overloaded curriculum ➔ Students manage pace
Questions?
Exploring BugHunt Cost-Effectiveness

- Who is using it?
- Do students find it more engaging than a lab?
- Do we actually instill testing principles?
- What are the impressions from the instructors?
- What has worked and what hasn’t?
BugHunt Users

Since 2007

• 10,000+ hits
• 150 students per year (+50% at UNL)
Survey of Students

Feedback Survey

<table>
<thead>
<tr>
<th>The BugHunt tutorial gave me new knowledge about software testing.</th>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neutral</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>The BugHunt tutorial reinforced my previous knowledge about software testing.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The BugHunt tutorial system should be incorporated as a standard part of this course.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The BugHunt tutorial system should be incorporated as an optional part of this course.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web-based tutorials should replace weekly labs.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please rank the lessons according to their value with 1 being the most valuable.

<table>
<thead>
<tr>
<th>Lesson</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lesson 1 - General Testing Concepts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 2 - Black Box Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 3 - White Box Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 4 - JUnit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What aspects of BugHunt should be improved and how?


## Survey of Students (approx. 300)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neutral</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gave me new knowledge of software testing</td>
<td>16%</td>
<td>55%</td>
<td>18%</td>
<td>8%</td>
<td>3%</td>
</tr>
<tr>
<td>Reinforced my previous knowledge of testing</td>
<td>23%</td>
<td>53%</td>
<td>18%</td>
<td>4%</td>
<td>2%</td>
</tr>
<tr>
<td>BugHunt should be part of this course</td>
<td>28%</td>
<td>37%</td>
<td>25%</td>
<td>6%</td>
<td>4%</td>
</tr>
<tr>
<td>BugHunt should be an optional part of course</td>
<td>10%</td>
<td>31%</td>
<td>35%</td>
<td>20%</td>
<td>4%</td>
</tr>
<tr>
<td>Web-based tutorials should replace weekly labs</td>
<td>12%</td>
<td>20%</td>
<td>41%</td>
<td>19%</td>
<td>8%</td>
</tr>
</tbody>
</table>
Survey of Students (approx. 300)

Praise

"I really did enjoy my couple hours finding bugs!!"
"It was really fun way to learn"
"the program is great for teaching debugging skills"
"more lessons would be cool"
"it's quite interactive, I like it very much!"

Criticisms

"Tell me when I meet the lessons’ objectives so I know when to move on."
"Show how many total errors there are from the start."
"JUnit needs better instructions."
"Let me know where the bugs were after so you don’t have to wonder about it"
Do we instill testing principles?
Do we instill testing principles?
Assessment

Requirements
1. The program will take three arguments in order: two doubles and an operator (+, -, or ^)
2. If the arguments are wrong, the program will print, “Invalid Arguments”
3. If the arguments are correct, the program will print the result of the arithmetic expression

<table>
<thead>
<tr>
<th>Test</th>
<th>Input</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

main(String[] args) {
    if(args.length < 3) {
        print(“Invalid Arguments”);
        return;
    }
    try {
        solve(args[0], args[1], args[2]);
    } catch(Exception e) {
        System.out.println(“Error”);
    }
}
double arg1, arg2;
String op;
solve(String a, String b, String c) {
    arg1 = Double.parseDouble(a);
    arg2 = Integer.parseInt(b);
    op = c;
    if(op.equals(“-”)){
        System.out.println(arg1 - arg2);
    } else {
        System.out.println(arg1 + arg2);
    }
}
Assessment Measures

- Bugs Found
- Code Covered
- Tests Generated
- Valid Tests (correct behavior was specified)
Assessment: Bugs Found

Requirements
1. The program will take three arguments in order: two doubles and an operator (+, -, or ^)
2. If the arguments are wrong, the program will print, “Invalid Arguments”
3. If the arguments are correct, the program will print the result of the arithmetic expression

```
main(String[] args) {
    if(args.length < 3) {
        print(“Invalid Arguments”);
        return;
    }
    try {
        solve(args[0], args[1], args[2]);
    } catch(Exception e) {
        System.out.println(“Error”);
    }
}

double arg1, arg2;
String op;
solve(String a, String b, String c) {
    arg1 = Double.parseDouble(a);
    arg2 = Integer.parseInt(b);
    op = c;
    if(op.equals(“-“)){
        System.out.println(arg1 - arg2);
    } else {
        System.out.println(arg1 + arg2);
    }
}
```
Test: Bugs Found

Requirements
1. The program will take three arguments in order: two doubles and an operator (+, -, or ^)
2. If the arguments are wrong, the program will print, “Invalid Arguments”
3. If the arguments are correct, the program will print the result of the arithmetic expression

<table>
<thead>
<tr>
<th>Test</th>
<th>Input</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 3</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>+ 5 7</td>
<td>no ...</td>
</tr>
<tr>
<td>3</td>
<td>6 4.2 -</td>
<td>no ...</td>
</tr>
</tbody>
</table>

```java
main(String[] args) {  
if(args.length < 3) {  
  print(“Invalid Arguments”);  
  return;  
}  
try {  
solve(args[0], args[1], args[2]);  
} catch(Exception e) {  
  System.out.println(“Error”);  
}  
}

double arg1, arg2;
String op;
solve(String a, String b, String c) {  
  arg1 = Double.parseDouble(a);
  arg2 = Integer.parseInt(b);
  op = c;
  if(op.equals(“-”)){
    System.out.println(arg1 - arg2);
  } else {
    System.out.println(arg1 + arg2);
  }
```
Assessment: Bugs Found

Requirements
1. The program will take three arguments in order: two doubles and an operator (+, -, or ^)
2. If the arguments are wrong, the program will print, "Invalid Arguments"
3. If the arguments are correct, the program will print the result of the arithmetic expression

<table>
<thead>
<tr>
<th>Test</th>
<th>Input</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 3</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>+ 5 7</td>
<td>no ...</td>
</tr>
<tr>
<td>3</td>
<td>6 4.2 -</td>
<td>no ...</td>
</tr>
</tbody>
</table>
### Assessment: Code Covered

#### Requirements
1. The program will take three arguments in order: two doubles and an operator (+, -, or ^)
2. If the arguments are wrong, the program will print, “Invalid Arguments”
3. If the arguments are correct, the program will print the result of the arithmetic expression

```java
main(String[] args) {
    if(args.length < 3) {
        print("Invalid Arguments");
        return;
    }
    try {
        solve(args[0], args[1], args[2]);
    } catch(Exception e) {
        System.out.println("Error");
    }
}

double arg1, arg2;
String op;
solve(String a, String b, String c) {
    arg1 = Double.parseDouble(a);
    arg2 = Integer.parseInt(b);
    op = c;
    if(op.equals("-")){
        System.out.println(arg1 - arg2);
    } else {
        System.out.println(arg1 + arg2);
    }
}
```

<table>
<thead>
<tr>
<th>Test</th>
<th>Input</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 3</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>+ 5 7</td>
<td>no ...</td>
</tr>
<tr>
<td>3</td>
<td>6 4.2 -</td>
<td>no ...</td>
</tr>
</tbody>
</table>
main(String[] args) {
    if(args.length < 3) {
        print("Invalid Arguments");
        return;
    }
    try {
        solve(args[0], args[1], args[2]);
    } catch(Exception e) {
        System.out.println("Error");
    }
}

double arg1, arg2;
String op;
solve(String a, String b, String c) {
    arg1 = Double.parseDouble(a);
    arg2 = Integer.parseInt(b);
    op = c;
    if(op.equals("-")){
        System.out.println(arg1 - arg2);
    } else {
        System.out.println(arg1 + arg2);
    }
}
**Assessment: Code Covered**

**Requirements**
1. The program will take three arguments in order: two doubles and an operator (+, -, or ^)
2. If the arguments are wrong, the program will print, “Invalid Arguments”
3. If the arguments are correct, the program will print the result of the arithmetic expression

```java
main(String[] args) {
    if(args.length < 3) {
        print(“Invalid Arguments”);
        return;
    }
    try {
        solve(args[0], args[1], args[2]);
    } catch(Exception e) {
        System.out.println(“Error”);
    }
}
double arg1, arg2;
String op;
solve(String a, String b, String c) {
    arg1 = Double.parseDouble(a);
    arg2 = Integer.parseInt(b);
    op = c;
    if(op.equals(“-”)){
        System.out.println(arg1 - arg2);
    } else {
        System.out.println(arg1 + arg2);
    }
}
```

<table>
<thead>
<tr>
<th>Test</th>
<th>Input</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 3</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>+ 5 7</td>
<td>no ...</td>
</tr>
<tr>
<td>3</td>
<td>6 4.2</td>
<td>no ...</td>
</tr>
</tbody>
</table>
Assessment: Code Covered

Requirements
1. The program will take three arguments in order: two doubles and an operator (+, -, or ^)
2. If the arguments are wrong, the program will print, "Invalid Arguments"
3. If the arguments are correct, the program will print the result of the arithmetic expression

<table>
<thead>
<tr>
<th>Test</th>
<th>Input</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 3</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>+ 5 7</td>
<td>no ...</td>
</tr>
<tr>
<td>3</td>
<td>6 4.2 -</td>
<td>no ...</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Avg. Bugs</strong></td>
<td>4.0</td>
<td>4.5</td>
</tr>
<tr>
<td><strong>Avg. Paths</strong></td>
<td>6.4</td>
<td>5.5</td>
</tr>
<tr>
<td><strong>Avg. Tests</strong></td>
<td>8.4</td>
<td>9.0</td>
</tr>
<tr>
<td><strong>% Valid Tests</strong></td>
<td>79%</td>
<td>88%</td>
</tr>
</tbody>
</table>

**# of Bugs (max 8)**

Before: [Boxplot]

After: [Boxplot]
# Results

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Avg. Bugs</strong></td>
<td>4.0</td>
<td>4.5</td>
</tr>
<tr>
<td><strong>Avg. Paths</strong></td>
<td>6.4</td>
<td>5.5</td>
</tr>
<tr>
<td><strong>Avg. Tests</strong></td>
<td>8.4</td>
<td>9.0</td>
</tr>
<tr>
<td><strong>% Valid Tests</strong></td>
<td>79%</td>
<td>88%</td>
</tr>
</tbody>
</table>

![Box plot of number of tests before and after](chart.png)
Do we instill testing principles?

- Students in Software Engineering
- Found more faults
- Develop more tests
- More tests are correctly specified
- Next: CS2 December 2010 at UNL
Instructors feedback

From UNL - emails and informal conversations

"Lovely! Students learned testing and I do not have to cover it"

"I still need to run a lab"

"Needs to be part of syllabus and be graded"

I wish it could ...

Emails from other instructors

"My students and I absolutely loved BugHunt. They worked in pairs and the team that found most bugs won the competition"

“My students enjoyed the exercise and trying to cover 100% of the code”

“Thank you for making the tool available for academic use”
Observations from the trenches

Students that skip introductory material miss the big picture
Observations from the trenches

- Students really like to see code, not "just" test but also FIX!
- Feedback help students to converge quickly
- Not useful early in CS1. Getting code to compile makes it "correct enough"
- Instructors use BugHunt in a lab session, experience is better if TA provides context
- Successful use outside CS1 / CS2. Soft. Eng. courses and industry
BugHunt

take home message

- Teaches fundamental testing concepts
- It is easy to use for instructors
- Engages students
- It is actively being improved
- Used in ways and places we did not anticipate

Try it, use it your way, and help us improve it

http://esquared.unl.edu/BugHunt
Broader Lessons from BugHunt “Model”

- Correcting students' misconceptions much sooner, from days to seconds
- Continuous feedback anytime anywhere
- Identify tasks whose answers can be produced automatically and quickly
- Rich types of feedback we are not tapping yet
Broader Lessons from BugHunt “Model”

- Business case
  - Clients and competitors
  - Financial support
  - Role of community, and what community?
Broader Lessons from BugHunt “Model”

- Other SE areas suffer from same problems
- Lack of time, resources, expertise
- Important enough to have them early
BugHunt
Making Early Software Testing Lessons Engaging and Affordable

Sebastian Elbaum
K. Stolee, X. Guo, S. Trout, M. Jorde, S. Person, J. Dokulil
University of Nebraska - Lincoln

With support from NSF and UNL
With the patience of several generations of UNL students