GUI and Integration Testing

Milestone 5

George Dimitrov, Yasir al-Bender, Cory Ebner & Will Nguyen

ABSTRACT

Exploratory GUI Testing can reveal bugs that can be ignored by standard testing techniques. GUI Testing will be used to ensure proper functionality in this Java Calculator. This type of testing can be used as an integration test due to its ability to verify functional, performance, and reliability requirements of high-level components.

Table of Contents

Table of Figures

1. GUI Testing Approach and Results

1.1 Rationale

Graphical User Interface (GUI) testing can be used in order to determine if an application has met functional and non-functional requirements outlined in the specifications. GUI testing can also be used as an improvised way to conduct basic integration testing since the GUI makes use of several of components in unison to perform tasks.

The approach in GUI testing for this application can be described as a focused, exploratory method. The team chose to do exploratory testing it can uncover more bugs that normal standard ways of testing may ignore. It also utilizes the team's experience and prior knowledge to adjust to test cases to cover more cases and scenarios; the ability to adjust accordingly during testing is one advantage that exploratory testing has over automated testing.

1.2 Approach

GUI testing and preparations were primarily based on Session Based Test Management Cycle (SBTM): Bug Classification, Test Objectives, Time Restrictions, Review and Debriefing.

1.2.1 Define Bug Classification

The team categorized bugs in four categories of severity/priority: critical, normal, minor or cosmetic. Critical bugs are ones that compromises and impedes the usability of application. Such types of bugs are: data loss, corruption, application crashes, and inability to save work. All other bugs are classified as major or minor according to tester experience and how severe they impede the user's ability to make use of the application. Cosmetic bugs only affect graphical appearance and do not affect the logic behind the program at all.

1.2.2 Test Objectives

The prime objective is very simple for this type of application: Determine if, under varying real-world environments, the components of the applications are working correctly underneath the GUI.

Test ideas should be the starting point for exploratory testing and should be based of prior black-box testing results.

1.2.3 Time Restrictions

Restrictions on test sessions encourage testers to react on response events from the system and prepare for the correct outcome. Testing was done in 90 minutes sessions with the following restrictions:

- There should be no interruptions in the 90 minutes session
- The sessions can be extended or reduced by 45 minutes

1.2.4 Review

After testing, the team will reconvene to review all outcomes. This review entails that the testing team will evaluate all bugs and learn where they are locate so that they can be fixed. A small analysis of coverage will be concluded as well.

1.2.5 Debrief

Once the review session is over, a compilation of all data obtained will happen. This compilation will be examined to see if all testing objectives are covered. Based off that, additional tests may be created in order to achieve any objectives that were not fulfilled.

1.2.6 Other Testing Specifications

Testing is to be conducted in pairs since one person may miss bugs that occur or paths to certain states that need to be traverse. Notes and test execution logs are required to track progression and program states. An example of this log can be seen in Appendix C.

1.3 Testing Results

For a more detailed account of results, please refer to Appendix A.

1.3.1 Team A Results Summary

Tests Attempted	33
Tests Passed	30
Bugs Found	3

1.3.2 Team B Results Summary

Tests Attempted	9
Tests Passed	8
Bugs Found	1

2. Integration Testing

2.1 Rationale

Integration testing is the testing of one or more components within a system. A component consists of more than one unit, which was tested during black- and white-box testing. This type of testing checks to see how units interact with each other in a typical environment and if the interfaces used to govern these units are working properly.

Integration testing was not done by the team due to the structure of the program. All units that were tested during black- and white-box testing are subclasses of abstract classes. These abstract classes cannot be tested due their inability to be instantiated. The class (Parser) that uses those abstract super classes (GObject/OObject) could have been tested but after reviewing the program structure, the team decided to test the perform GUI testing instead. The diagram bellows this structure in detail:

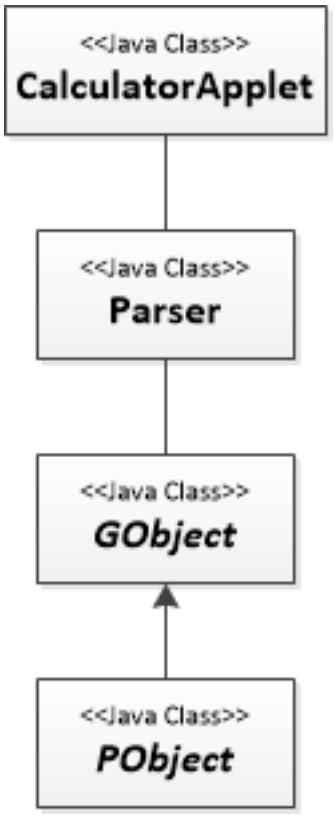


Figure 1: Diagram of Program Structure

By performing GUI testing (on CalculatorApplet), the team, in a way, has executed a form of integration testing. The GUI uses the Parser class, which uses those abstract super classes. By testing the GUI, the team can see if those units are working together properly.

3. Bug Reporting

3.1 Formatting

Bug reports will have the following format:

- **Bug Name** – one line summary of bug
- **Bug ID** – identifying number assigned by bug tracking tool
- **Area Path** – Buttons/Commands needed to get to state

- **Build Number**
- **Severity** – critical, major, minor or cosmetic
- **Priority** – critical, major, minor or cosmetic
- **Reported By**
- **Reported On**
- **Reason**
- **Status:** New/Open/Active
- **Environment** – OS Environment
- **Description** – More detailed summary of bug
- **Steps To Reproduce**
- **Expected Result**

3.2 Bugs Found during GUI Testing

Bugs uncovered during testing were similar to the ones uncovered during black-box testing. These bugs are mostly precision based and were deem minor. They were not deemed cosmetic because it may affect results in sequential calculations. No major or critical bugs were found during the exploratory testing.
The raw results are in Appendix A as well as summaries in Appendix B.

4. Conclusions and Recommendations

Exploratory GUI Testing as a methodology for find bugs is somewhat effective. It can be effective in finding typical human prone errors since testing environments are similar to real-world applications. It can also be effective, if the testers have a lot of experience with the application's background, since testers can easily see where potential problem spots are.

However, even with the many advantages that exploratory GUI testing does provide, there are many disadvantages as well. Although automation does exclude the ability to adjust tests accordingly to application behavior, it is more effective due to being repeated and faster. Exploratory testing requires a large amount of time in comparison. This is not to say that GUI testing should be fully automated, but it should be somewhere in between.

Although in-depth integration testing was done not performed for this application, the testing team can see the importance to do some integration testing within the testing cycle. Units that work correctly independent of each other **may** be defective when interacting with one another. That being said, the testing focus should primarily focus on black- and white-box testing to find the majority of the more probable bugs that can occur. Integration testing should be given more priority more than mutation testing but should be done only if time permits.

Appendix A. – GUI Testing Results

A.1 Team A Results

Test Description	Mac OSX Results	Windows Results
Adding	Pass	Pass
Subbing	Pass	Pass
Multiplying	Pass	Pass
Dividing	Pass	Pass
Clearing Values	Pass	Pass
Deleting values	Pass	Pass
Powering Calculation On/Off	Pass	Pass
x^{-1}	Pass	Pass
x^2	Pass	Pass
Root	Pass	Pass
Sin	Pass	Pass
Cos	Fail	Fail
Tan	Fail	Fail
Log	Pass	Pass
Ln	Pass	Pass
^	Pass	Pass
nCr	Pass	Pass
i	Fail	Fail
Pi	Pass	Pass
Mode	Pass	Pass
Shift	Pass	Pass
STO	Pass	Pass
RCL	Pass	Pass
M+	Pass	Pass
Braces	Pass	Pass
Up arrow	Pass	Pass
Down Arrow	Pass	Pass
Left Arrow	Pass	Pass
Right Arrow	Pass	Pass
?	Pass	Pass
Numbers	Pass	Pass
.	Pass	Pass
e	Pass	Pass

A.2 Team B Results

Test Description	Linux Results
x^3	Pass
$\sqrt[3]{n}$	Pass
$\sin^{-1}$	Fail
$\cos^{-1}$	Pass
tan-1	Pass
10x	Pass
exp	Pass
nCr	Pass

Appendix B. – Partial Bug Report

B.1 Cosine Bug

Bug Name	Cosine of 90 - Incorrect Answer
Bug ID	-
Area Path	-
Build Number	2:0.5
Severity	Minor
Priority	Minor
Reported By	George
Reported On	March 30, 2014
Reason	Unexpected result
State	Active
Environment	Mac OSX Mavericks 10.9.2/Windows 8.1
Description	Cos(90) ~=0 on the calculator, it shows long decimal value: 0.0000000061; should just be 0
Steps to Reproduce	cos →9→0→=
Expected Result	0

B.2 Tangent Bug

Bug Name	Tangent of 180 - Incorrect Answer
Bug ID	-
Area Path	-
Build Number	2:0.5
Severity	Minor
Priority	Minor
Reported By	George/Yasir
Reported On	March 30, 2014
Reason	Unexpected result
State	Active
Environment	Mac OSX Mavericks 10.9.2/Windows 8.1
Description	Tan returns small negative values on Tan(180) & Tan(360), they should be just 0
Steps to Reproduce	tan →1→8→0→=
Expected Result	0

B.3 Combination Bug

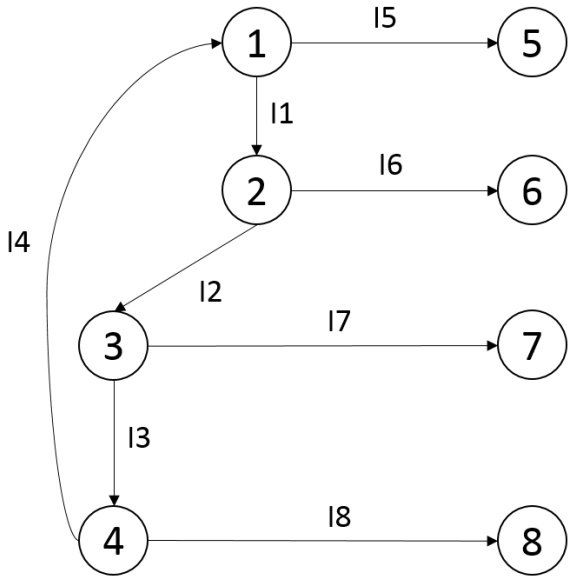
Bug Name	Sequential use of Combination and i produces Incorrect Answer Error
Bug ID	-
Area Path	-
Build Number	2:0.5
Severity	Minor
Priority	Minor
Reported By	George
Reported On	March 30, 2014
Reason	Unexpected behaviour
State	Active
Environment	Mac OSX Mavericks 10.9.2/Windows 8.1
Description	5C-3 produces an error, and the usage of i sequentially produced the wrong answer (25i)
Steps to Reproduce	5→ nCr→-3→=→5→i→=
Expected Result	Error; 5i

B.4 Inverse Sine Bug

Bug Name	Inverse Sine Error Answer
Bug ID	-
Area Path	SHIFT
Build Number	2:0.5
Severity	Minor
Priority	Minor
Reported By	Cory
Reported On	March 30, 2014
Reason	Unexpected behavior and result
State	Active
Environment	Linux Mint 16
Description	$Sin^{-1}(30)$ produces an error for an answer. If the equal key is pressed again, the answer of 30 will be produced.
Steps to Reproduce	SHIFT → sin^{-1}→3→0→=→=
Expected Result	NaN

Appendix C. – Example of Test Execution Log

C.1 Combination State Diagram



Where I1 – I8 are inputs, 1-8 are states of the GUI

Input #	Input	Expected Output	Actual Output	Notes
I1	Click 3 button	3 should be displayed in calculator output	3 is displayed in calculator output	
I2	Click **nCr** button	3C Should be displayed	3C Is displayed	
I3	Click **2** button	3C2 should be displayed	3C2 is displayed	
I4	Click **=** button	3 should be displayed as the answer	3 is displayed as the answer	
I5	Click = button when no new input has been entered	Displays previous answer	Displays previous answer	
I6	Click = button without adding nCr	Display 3 as answer	Display 3 as answer	
I7	Click a button that is not a valid input for the	Error message is displayed	Error message is displayed	

	combination function			
I8	Add additional inputs	Displays additional outputs	Displays additional outputs	You can create equations in the calculator with an unlimited amount of possibilities