SEGR 5210 Software Testing
Fall 2014

Project

In this project, you are to write and implement a test plan for an open-source Android software app. If you are familiar with Apple iOS development and would rather test an iOS app, please see the instructor.

Preliminary Steps

1. If necessary, download and install an Android development environment from http://developer.android.com/sdk/index.html.
   - Note that the Android development environment is not installed on campus computers. If you were planning to use campus computers, please see the instructor.
   - It is not necessary to have an Android device to test the app. Using the emulator that is provided with the Android development environment is sufficient (though you are welcome to use own Android device).
2. Choose an open-source software app. Here are some guidelines when selecting projects:
   - One place to find open-source Android apps is: https://f-droid.org/ (Select the 'Browse' tab).
   - The app should not be too complex given the time constraints and the lack of familiarity with the source code.
   - The app should be working (for the most part). It should either have a well-written specification or be intuitively obvious on what the app is supposed to do.
   - The app must be written in Java.
   - The app should not have unit tests already written.
   - Testing will be focused on the app running on the device. Avoid apps that offload significant portions of the functionality to a backend server (such as a weather app that is grabbing weather information and displaying it on the screen). Apps can have some network connectivity (such as an online leaderboard for a game).
   - Apps must have a GUI with touch input.
   - You may use an app you have personally written provided it was not written for a previous class (these programs are often not sufficiently real-world) and it meets the criteria listed above. In particular, it must be reasonably sized (not too small or too large), open-source (anyone can download it), has a good specification or is intuitively obvious, and mostly working.
3. Once an app has been selected, choose a unit to apply unit testing to. The unit should not be too complex but not too trivial either.
4. Once you have selected an app and unit, they need to be approved. Please send me an email that describes the program, the selected unit, and a link to the website containing the project. This approval process must be completed before Part I is due and ideally should be done during the first week.

Project Overview

For this project, you will be carrying out the following testing:

- ***Functional unit testing***: Write and execute unit tests that exercise the selected unit. The unit tests should utilize functional testing techniques as described in class.
- ***Structural unit testing***: For the selected unit, write and execute additional tests to get as close to 100% statement coverage as possible.
- ***Integration testing***: Develop tests that test the integration between the selected unit and other units it interacts with in the program. It is not necessary to execute these tests.
- ***Exploratory system testing***: Carry out tours of the app based on exploratory software testing techniques described in class.
- ***Scenario testing***: Write and carry out tests based on use case scenarios.

The project is divided into four parts, each with their own due date. For each submission, you will submit a project report (the contents are described in the subsequent sections). The project report is a cumulative document and the entire document must be submitted at each due date. For instance, the Part 2 submission will contain both Parts 1 and 2 (with part 2 coming at the end). When parts 1-3 are graded, it may contain comments that need to be addressed in the subsequent submission. Failing to address these comments will result in a lower grade. You will also need to submit various source code artifacts (more later).

Part 1 Software Description

Due Thursday, October 9 (beginning of class)

In this part of the project, you will be describing the software app you selected with an eye to testing. The deliverable for this part of the project is a document containing the following parts:

System Description: Describe the app that you are testing. Your description should contain answers to the following questions:

- What does the app do?
- What are the inputs?
- What are the outputs?
- What non-functional characteristics does your app have? For example, consider performance of the code, interaction with the environment, security, reliability, etc.
- What kind of faults could your app have?
- What are the highest risks for the app?

Unit Description: Describe the selected unit from your system. Your description should contain answers to the following questions:

- What does the unit do?
- What are the inputs and outputs?
- What kind of faults can the unit have?

Part 2 Functional Unit Testing

Due Monday, November 3 (8:00am)

In this part of the project, you will be focusing on testing the selected unit. You will start by creating unit tests that utilize the functional testing strategies discussed in class. If appropriate, model-based testing should also be employed. You can also add other interesting tests that are not necessarily derived from a particular testing technique. However, your test plan must show evidence of testing the unit systematically. Add the following sections to the report:

Approach for Functional Unit Testing: Describe how you tested the unit. In particular, you should describe what functional test techniques you used and how they were applied.

Functional Test Cases: A table of test cases. Each test case should have the following:

- A unique identifier (a number is sufficient).
- A brief description of the test case that describes what test case specification is being satisfied.
- The inputs used.
- The expected output.
- The status of the test: not executed, passed, failed

The table itself should be organized in a manner so that it is clear that the suite of tests exhibits systematic and thorough functional testing. You may find it easier to create separate table(s) of test case specifications and have the table of test cases cross reference that table.

Functional Unit Testing Results: Provide the following information:

- If necessary, an explanation on why certain tests were not executed.
- A list of bugs detected. It is not necessary to debug the failures.

Part 3 Structural Unit Testing and Integration Testing

Due Monday, November 17 (8:00am)

In this part of the project, you will apply structural testing to the selected unit and develop (but not execute) integration tests. To apply structural testing, first measure the code coverage (statement coverage) of your functional test suite. Then write additional tests to obtain 100% statement coverage (if possible). Add the following sections to the report:

Structural Unit Testing Results: Provide the following information:

- How much statement coverage was obtained (before writing structural tests).
- A description of what was missed during functional testing. If there were deficiencies during functional testing, describe the deficiencies and make the appropriate modifications to the unit functional testing sections in this document.
- A final measure of statement coverage. If less than 100%, explain why.

Integration Testing: Describe how your unit interacts with other units in the program. Your description should contain answers to the following questions:

- Briefly describe what each unit it interacts with.
- How do the modules interact? Do they call each other's functions? What data do they exchange?
- What kind of faults can there be in the integration of those modules?
- Describe some specific integration tests that can be carried out. (It is not necessary to execute these tests).

Part 4 System Testing and Conclusions

Due Monday, December 8 (8:00am)

NOTE: This section is vastly different from the original version.

Exploratory System Testing: At the minimum, you must carry out these exploratory testing tours:

- Back Alley tour: Focus on the least likely used features.
- Lonely Businessman tour: Test the feature(s) furthest away from the starting point.
- Garbage Collection Tour: Check the interface in a methodical fashion.
- Supermodel tour: Check the interface to see if it looks good.
- Couch Potato tour: Do as little work as possible, checking for default values.
- Antisocial tour: Enter the least likely or bad inputs.

Additional notes and requirements:

- Feel free to include additional tours if appropriate.
- It is sufficient to carry out this testing on a single device.
- The supermodel tour must use screen rotations. To do this in the emulator, press the left Ctrl and F12 together.
- This testing can be done on an actual Android device if desired. It is likely faster than using the emulator.

Add the following sections to the report:

- Mention the (possibly emulated) device you carried out exploratory testing. Please note whether the device is a phone or tablet and whether you carried out the testing on the emulator or the actual device.
- Then, for each "tour", describe the following:
  - How you carried out the tour. For the Back Alley and Lonely Businessman tours, the descriptions should indicate the features you tested. For the Antisocial tour, the description should describe the bad inputs and where they were applied.
  - How long it took to carry out the tour.
  - Any bugs found during the tour.
  - Any additional test cases that came to your head while applying the tour (if appropriate).

Scenario Testing Using Robotium (see tutorial on the course website), create three scenario tests. Each test should represent a use case of the software. In the report, describe the following:

- Describe the three tests.
- Note if you found any bugs with the tests.

Conclusion Answer the following questions:

- What did you learn from this project?
- What were the biggest challenges in completing this project? How did you overcome these challenges?
- What would you do differently if you were to test a similar app?

Optional Part Random Testing

Due Monday, December 8 (8:00am)

Monkey Testing Using the monkey tool (see tutorial on the course website), send 2000 events to the app and watch it execute. In the reports, describe the following:

- Describe some of the events that occurred when running the tests. I'm not expecting a complete list – a list of highlights.
- Note if you found any bugs with the tests.
- Your personal thoughts on using this form of random testing.

Project Deliverables

NOTE: The deliverables have been updated to reflect what source code files need to be submitted with each part.

All submissions will be done electronically via Canvas. For each part, submit the cumulative project report electronically (either in Word (.docx) or PDF format).

Also submit the following source code file(s):

- Part 2: The source code for the unit (please send the entire file even if you are not testing everything in the file) AND the source code for the unit tests.
- Part 3: The source code for the unit tests (includes the tests from Part 2 and any additional tests from structural testing).
- Part 4: All tests (the unit tests from Part 2 and 3 and the scenario tests from Part 4).

Grading

The project will be scored using 200 points, the breakdown will be as follows:

Part 1 deliverables	20 points (10%)
Part 2 deliverables	50 points (25%)
Part 3 deliverables	50 points (25%)
Part 4 deliverables	50 points (25%)
Overall quality (assessed at the end along with Part 4)	30 points (15%)

By completing the optional Monkey testing part, you can earn an additional 15 points. This allows the student to achieve up to 215 points but the overall percentage will be calculated using 200 points. However, you cannot earn more than 100% on the project – if you receive 200-215 points on this assignment, you will receive exactly 100% on the project.

The grade can be impacted negatively for the following:

- Not completing the preliminary steps in a timely manner
- Incomplete project report.
- Poorly written or uninspired project report.
- Inadequate functional unit testing[*].
- Inadequate structural unit testing[*].
- Inadequate exploratory system testing[*].

[*]If you are having trouble with these items, mention this in the project report. Perfection is commonly an unattainable goal. Before writing these off, it would be wise to see your instructor.