Glucosio Testing Framework

Final Report

Jake Marotta, Baylee Sims, Tyler Montgomery

College of Charleston

CSCI 362-01

TABLE OF CONTENTS

TITLE **PAGE NO.**

1. Introduction

Our goal was to create a testing framework for an existing project which reads several test cases, taking the form of .txt files. The framework should be able to be run via a single command, and a report should be automatically generated and displayed.

The development process of this testing framework consisted of project selection, class/method evaluation, and script construction. Along this process, progress proved to be hindered and difficult in some areas, but we succeeded one step at a time. This report documents this development process and reflects on the efforts put into this endeavor.

2. Candidate Selection

Three H/FOSS projects were considered for testing:

Tanaguru is an open-source website assessment tool. It is supposed to function as a tool that reliably automates websites to determine their level of accessibility. Things like color contrast and language specification are covered in the tests provided by the software.

Empathy has been the default messaging program in Ubuntu since version 9.10, when it replaced Ekiga. It supports text, voice, and video chat and file transfers over many different protocols. You can tell it about your accounts on all those services and do all your chatting within one application.

OpenPetra.org is still in development, and is an administrative control software for non-profit organizations. Its main goal is to ease the burden on non-profits by reducing software costs and training time.

3. Deliverable 1: First Experiences

Tanaguru was our first choice for this endeavor. However, this project soon became troublesome.

The complications discovered when working with Tanaguru Contrast-Finder were:

1. The last commit was 3 months before. It would seem that this portion of the project had probably been deemed outdated or incomplete.
2. It had been roughly 2 years since the last edit to the page with the project's build instructions, and we had not been able to build it without changes.
3. Even with as many changes as we made, there was a LifecycleException caused by a ClassNotFoundException when attempting to load the build into Tomcat. There are either dependencies missing or a class name was changed without reflecting the change in the build file.

After deciding that working out these problems would take too much time, the project was changed to Glucosio, which we were able to compile and run.

Glucosio is a collection of free and open source Android apps for tracking glucose levels for personal use or for diabetes research. There were a few platforms to choose from, but we focused on the mobile Android app. As it is an Android app, the bulk of the source code is written in Java. We used Android Studio to run the app and

its associated tests on an Android Virtual Device.

The project uses Gradle as a build tool, so we had thought it would be much simpler to put together than it was. Although we had only forked the repository and not made any changes, the Gradle build kept failing due to a set of "debug" tests in the project. An hour later, we successfully imported the code into Android Studio, after which we had to figure out where Android Studio had put the Android SDK, so we could set the ANDROID_HOME environment variable. Then came the setup of the Android Virtual Device, which was its own animal. Only two of our systems were able to get an Android Virtual Device set up; the other was deemed incompatible for emulation through Android Studio. Android Studio told us how we could presumably emulate a device anyway, but when we tried, we were greeted with the same error message. It seems that despite being built on top of IntelliJ, which two group members are very familiar with, Android Studio has gone out of its way to make its suggestions remarkably unhelpful.

On the systems for which the Android Virtual Device was supported, the rest of the way was very simple, and the emulation surprisingly fluid. Glucosio had a multitude of (mostly JUnit) tests that we were able to explore. Obviously, all of them pass when the code is unchanged. We played around with some of the Glucose Range tests, changing the values to make them fail.

4. Deliverable 2: Test Plan

Testing Process: The bulk of our testing occurs inside of the org.glucosio.android.tools package. The classes therein act as tools and utilities that assist the other, larger sections of the product. Only about a third of the tools are comprehensively tested, so we had our work cut out for us. Our first step had been to identify the tools which do not have tests written for them. They are as follows:

1. AlgorithmUtil
2. AnimationTools
3. FormatDateTime
4. GlucosioAlarmManager
5. GlucosioConverter
6. GlucosioNotificationManager
7. LabelledSpinner
8. NotDismissableEditText
9. RealmBackupRestore
10. SplitDateTime
11. TipsManager

Some of these, such as LabelledSpinner, are composed of primarily getters and setters; not exactly things worth testing, within the bounds of this project. Others, like FormatDateTime, contain methods that return formatted results, which would require tests to ensure valid output. Those were our focus.

Tested Items: By the end of the testing period, we had developed a total of twenty-five test cases. These test cases are presented in Chapter 6.

Test Recorded Procedures: A test report is generated and automatically displayed in the system default browser upon the completion of "runAllTests".

Hardware and Software Requirements: The system runs on a UNIX-like operating system, and needs to have Java installed. The app itself requires the Android platform of at least version 4.1. The app was edited and tested in Android Studio.

Constraints: Jake was unavailable from October 24th through the 28th, conflicting

with the period of time leading up to Deliverable #3, which means we had to put in most of the group work required for that deliverable prior to October 24th. Immediately following Deliverable #3, Tyler was unavailable from November 1st through the 5th, limiting the amount of time we had to meet concerning Deliverable #4 (due November 14th).

System Tests: System tests were out of the scope of this plan, as it was restricted to unit testing.

5. Deliverable 3: First Draft

The building of the script was pretty straightforward. The script that was written to list the current script's directory previously served as a good basis for our runAllTests script. Some slight modifications needed to be made to the script, which included compiling and executing the Java file/Driver. The goal of the runAllTests script was to compile and run the Driver class and output the result into an html file.

The Driver itself handled most of the work and is the bulk of the framework. The script would send the correct directory to the driver (the /testCase/ directory for our purposes), and would put this into the args[0] variable. From there, the driver builds an array to store all of the files that need to be analyzed. Then, the framework splits the text document up into separate tokens, and compares the result to the expected output in order to determine if the test passed or failed.

HOW TO RUN:

1. Installing Java
    - Check if the Java JDK is installed: java -version
    - If not, Update libraries with sudo apt-get update
    - After libraries are updated, run sudo apt-get install -y default-jdk
2. Cloning the TBD Repository
    - Create the folder to put the repository touch <your_foldername_here>
    - Navigate to the folder you created with cd <your_foldername_here>
    - Initialize the git folder git init
    - Clone the repository to your system: run git clone https://github.com/CSCI-362-01-2017/TBD
3. Running the Script
    - Navigate to the script's directory: cd /TBD/scripts
    - Run the command ./runAllTests

If done correctly, the script executes and opens a html page with the test cases displayed on the screen.

Upon presenting our work on Deliverable 3 to the class, we were informed that our single-driver structure was incorrect, and that file handling should be done by the Bash script, instead. Using multiple, tailored Java drivers is a simpler process than trying to handle all possible cases using reflection, anyway, but, time had to be spent re-structuring the testing framework.

6. Deliverable 4: Test Cases

Our test specification template shows what goes in each of the rows of a test case. Only the desired value is present in each row.

Test Case Specification Template

- TestID - An identifier for the test.
- Requirement - A description of the method being tested, as well as the condition being tested.
- Component - A class within the application being tested.
- Method - A method within the component being tested.
- Specified Driver - The name of the Java driver required to run the test (no file extension).
- Test Input - This will include the parameters necessary to test the specified requirement.
- Expected Outcome - This will specify the output which would be returned if the method executed as expected.

We developed a total of twenty-five test cases. For each method chosen for testing, we developed a driver. Each driver corresponds to a single method from some class within Glucosio's "tools" package. When a driver is executed, it parses the available arguments and sets up what it needs to in order to run said method. The output from the method is collected, then given back to the Bash script via standard outupt.

7. Deliverable 5: Faults

The following faults were introduced into the Glucosio project code in order to cause some, but not all, of our tests to fail:

Fault #1:

ReadingTools.hourToSpinnerType()

A single operator is changed from '>' to '>=', causing the test case hourToSpinner1 to fail.

Fault #2: ReadingTools.parseReading()

(We didn't actually introduce this fault, it was already there.) Method attempts to parse a Number from a String that contains letters, truncating the String at the first invalid character (e.g. "1.a23" to "1."), causing parseReading5 to fail. In addition, Strings that may be in a format for a different Locale are read in as normal for the current Locale (e.g. In English, "1,23" to 123.0), which causes parseReading2 to fail.

Fault #3: SplitDateTime.getMonth()

The pattern for getMonth() is changed from "MM" to "mm", which is the pattern for minutes, causing splitDateTime4 to fail.

Fault #4: GlucosioConverter.round()

The RoundingMode is changed from HALF_UP to FLOOR, which causes converterRound1 to fail.

Fault #5:

GlucosioConverter.a1cToGlucose()

The number of places to round to is changed from 2 to 3, causing a1cToGlucose1, a1cToGlucose2, and a1cToGlucose3 to fail.

8. Final Product

Our testing framework can be successfully run and applied by executing a single command from a terminal. It allows for anyone to add new test cases without changing the main Bash script. The report generated provides information used in each test case, including input, expected output, and actual output, to make troubleshooting an easier process.

Our testing framework uses a main Bash

script, “runAllTests.sh”, to find and execute test cases. Upon execution, the script searches our repository’s “testCases” directory for applicable .txt files. On each line of these .txt files is a piece of information that the script needs to run a test. The layout of the test cases is defined in our Test Case Specification.

Each test case specifies the name of a Java driver which is used to set up and run each test. The drivers are contained in the “drivers” directory. Output is collected from the driver and compared to the expected output to determine whether the test passed or failed. The information for the test is then appended onto a report to be displayed in the system’s default browser once all tests have been run.

9. Self-Evaluation

Our team proved particularly good at distributing tasks to improve efficiency. During each meeting, tasks which did not require everyone to be present where assigned to each group member to work on individually. This way, meeting time could be better spent on tasks which required the entire group’s attention.

By contrast, actually finding those meeting times was more difficult. Our team members have differing schedules, and they are very spread out; one in downtown Charleston, and two in different Charleston suburbs.

In addition, careless interpretation of project instructions contributed to wasted time. A fair amount of time prior to Deliverable 3 was spent on a monolithic Java driver which handled the .txt files directly, and attempted to run project methods via reflection. We had to re-structure our framework to accommodate the project requirements before continuing.

10. Possible Improvements

Given the time wasted on an incorrect structure prior to Deliverable 3, future projects should have clear instructions or be questioned thoroughly to ensure correct proceedings.

Concerning time, the decision to switch from Tanaguru to Glucosio put the team behind schedule. The selection of projects to test would be a better source if the projects were frequently updated with only applications of working order.

In addition, the framework assumes that an internet browser is the default program for opening .html files. This is listed as a requirement for now, but it may be possible in the future to circumvent that issue.