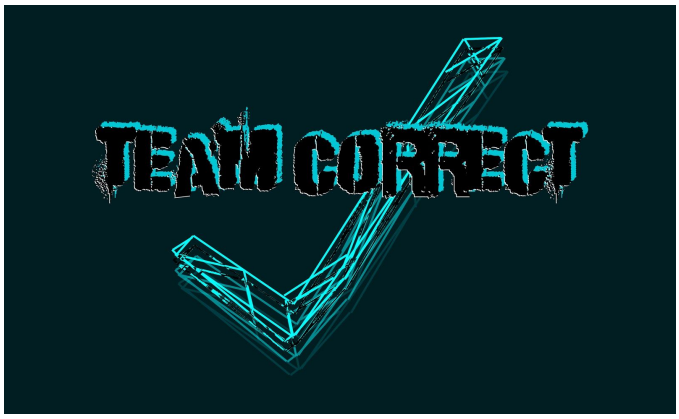
Glucosio Automated Testing Framework

CSCI 362 Final Report



Team Correct

Team Members: Hannah Posch, Alex Thropp,

Daniel Lee, Daniel Baczmaga

An automated framework presented for the testing of the Glucosio open-source project

College of Charleston

CSCI 362: Software Engineering

Dr. Leclerc

Table of Contents

0 Introduction

0.0 Project Description

Glucosio is an open-source software project that allows users to record and manage their diabetes data. The Glucosio project includes an android and iOS application for users to keep their data on. The project included an experimental web app, but was an experimental project that never reached completion. The software provides users with feedback and tips based on their data to ensure they are managing their diabetes. Glucosio allows users to share their diabetes data anonymously with researchers and contribute to diabetic research. The Glucosio project is focused on being user centered with a design that meets the needs of diabetic users. The code of the project is divided among Glucosio-API, Glucosio-iOS, Glucosio-android, Glucosio-tizen, brand-assets, Glucosio web app, project tools, cla-enforcer. As a team, we focused our testing effort on the Glucosio-android portion as this application could be run with Ubuntu which was a requirement for the 362 project.
Glucosio: https://github.com/Glucosio

0.1 Why we chose it

A few of our team members have family members personally affected by diabetes, so this project was one that we wanted to tackle. We wanted to learn more about ways to help and manage diabetes and try our best to help by testing and applying our knowledge to an existing application.

0.2 Testing Framework

For our project, we aimed to create a testing framework that would thoroughly test the main components of Glucosio, which include Activity, Adapter, Db, Fragment, Presenter, and Tools. We also wanted to make sure our framework tested as many requirements for the project as possible to ensure maximum functionality. The document includes our full process for creating this framework from start to finish. In Section 1, we describe the existing tests within Glucosio and any problems that may have arose while testing. In Section 2, we begin our testing process

and begin writing test cases for our framework. Section 3 hold our automated testing framework and 5 of the 25 test cases. In Section 4, we build the full list of 25 test cases and in section 5, we inject faults into 5 of them. Finally, in section 6, we discuss our overall experiences and team evaluation. These sections illustrate our process in the creation of our testing framework and show what we focused on and decided to change throughout the process.

1 Deliverable 1

1.0 Running project

To begin we first cloned the Glucosio Github Repository using the command, git https://github.com/Glucosio. After the project was cloned, we needed to install Android Studio and required android libraries through the terminal. Once Android Studio was installed, we were able to access the current tests created for Glucosio.

1.1 Existing Tests

Tests Completed within Android Studio:

- GlucosioApplicationTest
  - 3 tests were included within this java file
    - ShouldNotClearLanguage_WhenAlreadyDone
    - ShouldClearLanguage_WhenSetFromHelloActivityAndNotFixedYet
    - ShouldSaveLanguageClearedToPreferences_WhenItIsDone
  - All tests were passed and completed in 887 ms
- RobolectricTest
  - Included a variety of test builds, however, if the option “All Tests” was chosen, the return stated that the test suite was empty
    - If each individual test was run, then the code worked correctly
  - A1CCalculatorActivityTest
    - ReturnTrue_WhenKeyboardActionDone

- Notify Presenter_WhenGlucoseValueChanged
- ShouldAskPresenterToCheckUnit_When Created
- ShouldBindViews_WhenCreated
- ShouldSetMmolToMolAsUnit_WhenUserSettingsIsPercentage
- ReturnFalse_WhenKeyboardActionOther
- All tests were passed in 1m 7s

➢ AssistantAdapterTest

- CallPresenter_WhenExportAsked
- CallPresenter_WhenFeedbackAsked
- CallPresenter_WhenA1CalculatorAsked
- CallPresenter_WhenAddReadingAsked
- CorrectlyBindRows
- All tests passed 42s

➢ BackupActivityTest

- ShouldDelegateToBackup_WhenDisconnectIsCalled
- ShouldDelegateToBackup_WhenConnectIsCalled
- ShouldDelegateToBackup_WhenActivityResultRecieved
- ShouldInitBack_WhenCreated
- All tests passed 41s

➢ HelloActivityTest

- ShouldReportAnalytics_WhenCreated
- ShouldInitLanguageSpinner_WhenCreated
- ShouldSelectLanguage_WhenCreated
- ShouldPassNLAsLocale_WhenNLSelected
- ShouldBindView_WhenCreated
- All tests passed 59s

➢ MainActivityTest

- ShouldReportAnalytics_WhenCreated
- Test passed 37s

- PreferencesActivityTest
  - ShouldReportAnalytics_WhenCreated
  - Test passed 38s
- AssistantAdapterTest
  - CallPresenter_WhenExportAsked
  - CallPresenter_WhenFeedbackAsked
  - CallPresenter_WhenA1CCalculatorAsked
  - CallPresenter_WhenAddReadingAsked
  - CorrectlyBindRows
  - All tests passed 39s
- BackupAdapterTest
  - ReturnViewWithSizeFormatted_WhenAsked
  - Test passed 30s
- MigrationTest
  - ProperlyMigrateUserFromVersion4
  - ProperlyMigrateGlucoseFromVersion4
  - ProperlyMigratePressureFromVersion4
  - ProperlyMigrateWeightFromVersion4
  - ProperlymigrateCholesterolFromVersion4
  - All tests passed 314 ms
- UserBuilderTest
  - Builds user account
  - All tests passed 1s
- AssistantFragmentTest
  - ShouldUnBindViews_WhenDestroyed
  - ShouldBindViews_WhenCreated
  - Tests passed 36s
- A1CCalculatorPresenterTest
  - 9 tests passed 433ms

- ➢ AssistantPresenterTest
  - CallFragment_WhenUserAskedExport
  - CallFragment_WhenUserAskedSupport
  - CallFragment_WhenUserAskedToAddReading
  - CallFragment_WhenUserAskedToShowCalculator
  - All passed 92s
- ➢ ExternalViewPresenterTest
  - shouldLoadOpenSourceLicenses_WhenLicenseParameters
  - shouldThrowException_WhenNoParameters
  - shouldInvokeShowNoConnectionWarning_WhenNetworkIsNotConnected
  - All passed 28s
- ➢ HelloPresenterTest
  - 9 tests passed 28s
- ➢ OverviewPresenterTest
  - ShouldAddZerosBetweenReadings_WhenAsked
  - ShouldSortReadingsChronologically_WhenAsked
  - Tests passed 402ms

❖ AndroidPhoneAppTests

- ➢ Required running emulators for test;
  - We are using an android emulator
- ➢ HelloActivityTest
- ➢ MainActivity Test

Test Speeds: We noticed that as the test were run across multiple systems, the speeds were inconsistent. We have determined that there is nothing significant to be worried about and that the difference in testing speeds is most likely a result of being run on a specific machine.
Organization: The project is organized in the following categories for which these tests were focused on.

- Activity

- Adapter
- Db
- Fragment
- Presenter
- Tools

1.2 Experiences

Overall, the project is fairly straightforward and easy to follow, Glucosio included plenty of documentation on how to set up and test/run their source code. The tests were thorough and easy to run, and we appreciated the pass/fail description for each subunit test. The ability to run all of the tests will need to be fixed for future runs, as that will allow testing to be more efficient, but we did not mind running the tests on their own as they are organized and easy to find in the directory. Of our group, Daniel L. owns a phone running an Android OS, so we hope to test on there.

Project:

The project is fairly robust and well organized. The developer made the code and terminology easy to follow with great documentation. We are personally excited to continue to work on this project and contribute to make the overall application better. We were able to run the app on an Android emulator, and hope to test on a physical Android device in the future.

2 Deliverable 2

2.0 Testing Process

Our testing process for this project is very closely related to the Agile process, specifically Scrum practices. Our current Scrum process has included short sprint planning, sprints, and sprint review meetings over skype. We have been following this process throughout the development of this project and will continue to do so to finish. Our scrum master is Professor Leclerc, who is responsible for setting up our team and guiding us in the process. This process is

currently working well for us, as we are able to meet our delivery deadlines. The scrum process also works well with our Glucosio project as we continue to build and test our project this semester.

- ➢ Our major phases of our testing process so far have included each deliverable, which are focused on a specific portion of the final process.
    - Phase 0: Select 3 H/FOSS projects to present to class and ultimately discuss with Professor Leclerc to decide which one we would work on.
    - Phase 1: Clone project and work on existing tests within the system. Report on experience.
    - Phase 2: Produce a test plan and create 5 of the eventual 25 test cases.
    - Phase 3: Build an automated testing framework.
    - Phase 4: Finish the 25 test cases which will be tested using the automated testing framework previously built.
    - Phase 5: Insert 5 faults into code to ensure 5 tests fail and discuss experience.
    - Phase 6: Compile our final report and present our project.

2.1 Requirements Traceability

- ➢ The main requirements of Glucosio include:
    - User Input
    - Log of input
    - Graphics
    - Export of data
    - Ability to connect to other devices
    - Provide diabetes information
    - Reminders/Schedule
- ➢ We will be focusing our testing on more specific components of these requirement categories.

2.2 Tested Items

- The items we will be testing include:
    - Glucosio GitHub repository code
        - Glucosio Android App Folder
        - Via Android Studio
    - Glucosio Android App
        - Via Android emulator
    - Glucosio iOS App
        - Via iPhone

2.3 Testing Schedule

Test cases 1-5 date: 10/13

Deliverable 2 date: 10/15

Automated Testing Framework: 11/7

Deliverable 3 date: 11/9

Test cases 6-25: 11/17

Deliverable 4 date: 11/19

Deliverable 5 and Final Report: 11/28

2.4 Test Recording Procedures

- Our tests will be recorded through shared Google Drive Documents to ensure all team members have access and each member is using the most up-to-date document.
- Our tests will also be uploaded to our Team Correct Github Repository.

2.5 Hardware and Software Requirements

- Operating System: Linux Ubuntu
- Software Requirements: Android Studio, VirtualBox
- Hardware Requirements: PC, iOS phone

2.6 Constraints

- ➢ Time: For this project, we have a limited timetable of a semester. We have explicit deliverables that will need to be completed in a timely manner.
- ➢ Staff: We have a team of 4 members which is small for a normal software team, however, for the scope of our project, we have an appropriate team size.
- ➢ Cost: We do not currently have an associated cost with our project.
- ➢ Risk: We will be using VirtualBox for our system testing and project, so if there were to be a problem with a virtual machine, then we may need to start the process over. This is not a huge risk due to our team backing up files to Google Drive and uploading work to GitHub.
- ➢ Resources: Our resources include our personal computers and the lab, which means this should not be a constraint in our current project scope.

2.7 System Tests

Test Case 01

Test Suite ID	TS001
Test Case ID	TC001
Test Case Summary	To verify that Share Anonymous Data For Research checkbox allows user to share their glucosio data with researchers
Related Requirement	RS005
Prerequisites	1) User has downloaded Glucosio app 2) User has entered data into app
Test Procedure	1) Open Settings within Glucosio app 2) Enter user data 3) Select ‘Share anonymous data for research’
Test Data	N/a
Expected Result	1) ‘Share anonymous data for research’ should have a check next to it

Actual Result	1) If the checkbox is working correctly, a check will display 2) If the checkbox is not working correctly, an error message will display
Status	Pass
Remarks	N/a
Created By	Team Correct (Hannah P.)
Date of Creation	10/11/18
Executed By	Team Correct (Hannah P.)
Date of Execution	10/11/18
Test Environment	OS: iOS 12.0.1 App: Glucosio Application on iPhone

---

Test Case 02

Test Suite ID	TS001
Test Case ID	TC002
Test Case Summary	To verify that the user data is properly displayed graphically
Related Requirement	RS004
Prerequisites	1) User has downloaded Glucosio app 2) User has entered readings into app
Test Procedure	1) Open Overview within Glucosio app 2) View day, week, month graphs
Test Data	At least one input of each type (Glucose, HbA1c, Ketones, Body Weight, Blood Pressure, and Cholesterol)
Expected Result	1) Readings are displayed correctly on graphs

Actual Result	1) Readings are displayed correctly 2) If there are no readings for a certain type, the message “No chart data available” will be displayed
Status	Pass
Remarks	How to automate?
Created By	Team Correct (Daniel L.)
Date of Creation	10/12/18
Executed By	Team Correct (Daniel L.)
Date of Execution	10/12/18
Test Environment	Emulator: Pixel API 28 OS: Android Version 9 (Pie) App: Glucosio Application on Android

---

Test Case 03

Test Suite ID	TS001
Test Case ID	TC003
Test Case Summary	To verify that user data can be input properly
Related Requirement	RS001
Prerequisites	1) User has downloaded Glucosio app
Test Procedure	1) Open Overview within Glucosio app 2) Press + in lower right hand corner 3) Select input type 4) Complete input form
Test Data	Any kind of input

Expected Result	1) Readings are displayed in ‘History’
Actual Result	1) Readings are displayed correctly
Status	Pass
Remarks	Can you delete readings?
Created By	Team Correct (Daniel L.)
Date of Creation	10/12/18
Executed By	Team Correct (Daniel L.)
Date of Execution	10/12/18
Test Environment	Emulator: Pixel API 28 OS: Android Version 9 (Pie) App: Glucosio Application on Android

---

Test Case 04

Test Suite ID	TS001
Test Case ID	TC004
Test Case Summary	To verify that the app successfully logs users past entries
Related Requirement	RS001
Prerequisites	1) User has downloaded Glucosio app 2) Users have previously entered data
Test Procedure	1) Open Overview tab within Glucosio app 2) Press the History Tab located next to the Overview tab 3) Verify past entries to be correct
Test Data	Past Data Points

Expected Result	1) Readings to be displayed in ‘History’
Actual Result	1) Readings displayed correctly
Status	Pass
Remarks	Can you delete readings?
Created By	Team Correct (Alex T)
Date of Creation	10/12/18
Executed By	Team Correct (Alex T)
Date of Execution	10/12/18
Test Environment	Emulator: Pixel API 28 OS: Android Version 9 (Pie) App: Glucosio Application on Android

---

Test Case 05

Test Suite ID	TS001
Test Case ID	TC005
Test Case Summary	To verify that the app exports spreadsheet data effectively
Related Requirement	RS001
Prerequisites	1) User has downloaded Glucosio app 2) User has entered readings into app
Test Procedure	1) Open Settings within Glucosio app 2) Select “Export to CSV”
Test Data	Any kind of input
Expected Result	1) User inputs and calculated readings are converted to a .csv file

	properly
Actual Result	1) Charts are converted to zip file 2) 11 individual .csv files for each reading are produced within the zip file
Status	Pass
Remarks	Composite csv containing all chart readings
Created By	Team Correct (Daniel Baczmaga)
Date of Creation	10/12/18
Executed By	Team Correct (Daniel Baczmaga)
Date of Execution	10/12/18
Test Environment	OS: iOS 12.0.1 App: Glucosio Application on iPhone

2.8 Evaluation and Experiences

We had a good experience when creating deliverable #2 due to Glucosio's organization and ability to work with the app via the Android emulator and iOS app. We were able to create the test cases relatively easily, due to organization the project's requirements and focusing our tests on these main categories.

3 Deliverable 3

3.0 Architectural Description

We organized our testing framework into the following sections:
/TestAutomation

Joda-time-2.10.1.jar (Required Library)

Junit-4.12.jar (Required Library)

/project (Holds Glucosio repository)

/glucosio-android-develop

/scripts

runAllTests.sh

/testCases

testCase01.txt

testCase02.txt

...

/testCasesExecutables

/TeamCorrectTestExecutables

testCase01.java

testCase02.java

….

TestCasesInput.txt (All input for test cases)

Relevant Glucosio java files needed for compiling

/TeamCorrectFaultInjection

testCaseXX.java (test cases with faults injected)

….

/test

/temp

/oracles

/docs

README.md

/reports

testResults.html

3.1 Documentation

We are using the format specified on the class web page to document our test cases.

- test number or ID
- requirement being tested
- component being tested
- method being tested
- test input(s) including command-line argument(s)
- expected outcome(s)

3.2 Requirement List made by Team Correct

RS001: Record user input
Glucose
HbA1C reading
A1C reading
Cholesterol level
Blood pressure
Ketones
Body weight
RS002: Display last check
RS003: Provide a log of user inputs for the past days
RS004: Display graphically
RS005: Opt in option of sharing diabetes data with researchers (API)
RS006: Connect to other services (Google Fit, My Fitness Pal)
RS007: Export data to various formats and services
RS008: Sleep
RS009: Treatments
RS010: Medication
RS011: Exercise
RS012: Food and carbs
RS013: Set reminders
RS014: Export pictures of graphs to camera roll
RS015: Log diabetes type
RS016: Provide tips
RS017: Build User

RS018: Provide correct Glucose calculations based on User's data
RS019: Provide alerts

3.3 Instructions to run tests

1. Open Terminal
2. git clone https://github.com/csci-362-fall-2018-01/TeamCorrect
3. cd TeamCorrect/TestAutomation/scripts (May need additional navigation, depending on where the repository was cloned to)
4. ./runAllTests.sh
5. Output will be stored in an html file which should open automatically in the default browser.

*If any of the above commands return permission denied, use 'sudo' in front of the command.

*If the ./runAllTests.sh returns command not found, use 'sudo chmod 755 runAllTests.sh', then './runAllTests.sh'

3.4 Test Case Specifications

Test Number: 01
Requirement being tested: RS013 Set Reminders

Component being tested: Reminder.java

Method being tested: setMetric

Test input(s) including command-line argument(s):

0ml, 9ml, 10ml, 20ml

Expected outcome(s):

Metric returned = 0ml

Metric returned = 9ml

Metric returned = 10ml

Metric returned = 20ml

Test Number: 02

Requirement being tested: RS001 Record user input - Glucose

Component being tested: Glucose Converter

Method being tested: glucoseToA1C

Test input(s) including command-line argument(s):

10.0, 25.0, 50.0, 0.0

Expected outcome(s):

10.0 glucose level converted to 1.98

25.0 glucose level converted to 2.5

50.0 glucose level converted to 3.37

0.0 glucose level converted to 1.63

Test Number: 03

Requirement being tested: RS001 Record user input - Ketone

Component being tested: KetoneReading.java

Method being tested: setReading

Test input(s) including command-line argument(s):

2.0, 10.0, 50.0, 100.0

Expected outcome(s):

Reading returned = 2.0

Reading returned = 10.0

Reading returned = 50.0

Reading returned = 100.0

Test Number: 04

Requirement being tested: RS001 Record user input - Body Weight

Component being tested: WeightReading.java

Method being tested: setReading

Test input(s) including command-line argument(s):

100.0, 50.0, 10.0 0.0

Expected outcome(s):

Reading returned = 100.0

Reading returned = 50.0

Reading returned = 10.0

Reading returned = 0.0

Test Number: 05

Requirement being tested: RS001 Record user input - Blood Pressure

Component being tested: PressureReading.java

Method being tested: setMinReading

Test input(s) including command-line argument(s):

110.0, 30.0, 15.0, 2.0

Expected outcome(s):

Min Reading returned = 110.0

Min Reading returned = 30.0

Min Reading returned = 15.0

Min Reading returned = 2.0

3.5 Evaluation and Experiences

Deliverable 3's production was probably the most tricky as we were required to navigate between a shell script and java files, and ensure the syntax between them worked seamlessly. We were required to create a shell script in bash that would automate our java test files for our project, Glucosio. This required that our shell could navigate the correct directories and automatically run through our test cases. We also needed to ensure that our java test files could compile and execute from terminal as that is how the framework would run each file. These requirements required time and collaboration to complete and overall we are happy with the progress we made. The syntax of the script may have required a bit of trial and error, but once it was working, it made the testing of our project Glucosio much more efficient.

4 Deliverable 4

4.0 Documentation

During this deliverable, we utilized the same format for our test cases as mentioned in a previous section. We also added an input file for each test case to use. This input file can be edited to add additional inputs for one or all test cases and the output will automatically be updated.

4.1 Test Case Specifications

Test Number: 01
Requirement being tested: RS013 Set Reminders

Component being tested: Reminder.java

Method being tested: setMetric

Test input(s) including command-line argument(s):

0ml, 9ml, 10ml, 20ml

Expected outcome(s):

Metric returned = 0ml

Metric returned = 9ml

Metric returned = 10ml

Metric returned = 20ml

Test Number: 02

Requirement being tested: RS001 Record user input - Glucose

Component being tested: Glucose Converter

Method being tested: glucoseToA1C

Test input(s) including command-line argument(s):

10.0, 25.0, 50.0, 0.0

Expected outcome(s):

10.0 glucose level converted to 1.98

25.0 glucose level converted to 2.5

50.0 glucose level converted to 3.37

0.0 glucose level converted to 1.63

Test Number: 03

Requirement being tested: RS001 Record user input - Ketone

Component being tested: KetoneReading.java

Method being tested: setReading

Test input(s) including command-line argument(s):

2.0, 10.0, 50.0, 100.0

Expected outcome(s):

Reading returned = 2.0

Reading returned = 10.0

Reading returned = 50.0

Reading returned = 100.0

Test Number: 04

Requirement being tested: RS001 Record user input - Body Weight

Component being tested: WeightReading.java

Method being tested: setReading

Test input(s) including command-line argument(s):

100.0, 50.0, 10.0 0.0

Expected outcome(s):

Reading returned = 100.0

Reading returned = 50.0

Reading returned = 10.0

Reading returned = 0.0

Test Number: 05

Requirement being tested: RS001 Record user input - Blood Pressure

Component being tested: PressureReading.java

Method being tested: setMinReading

Test input(s) including command-line argument(s):

110.0, 30.0, 15.0, 2.0

Expected outcome(s):

Min Reading returned = 110.0

Min Reading returned = 30.0

Min Reading returned = 15.0

Min Reading returned = 2.0

Test Number: 06

Requirement being tested: RS001 Record User Input - Cholesterol Level

Component being tested: CholesterolReading.java

Method being tested: setLDLReading

Test input(s) including command-line argument(s):

190.0, 50.0, 1.0, 0.0

Expected outcome(s):

LDL reading set to 190.0

LDL reading set to 50.0

LDL reading set to 1.0

LDL reading set to 0.0

Test Number: 07

Requirement being tested: RS017 Build User

Component being tested: User.java

Method being tested: setD_Type

Test input(s) including command-line argument(s):

1, 2

Expected outcome(s):

User’s diabetes type set to Type 1

User’s diabetes type set to Type 2

Test Number: 08

Requirement being tested: RS016 Provide Tips

Component being tested: ActionTip.java

Method being tested: set/getTipDescritpion

Test input(s) including command-line argument(s):

Use whole foods in cooking like brown rice and whole wheat pasta.

Try to eat at least 8 grams of fiber per meal.

Expected outcome(s):

Tip: Use whole foods in cooking like brown rice and whole wheat pasta.

Tip: Try to eat at least 8 grams of fiber per meal.

Test Number: 09

Requirement being tested: RS018 Provide correct Glucose calculations based on User’s data

Component being tested: GlucoseData

Method being tested: glucose

Test input(s) including command-line argument(s):

10, 50, 100, 0

Expected outcome(s):

mgdl: 10, mmol: true, Glucose: .6

mgdl: 50, mmol: true, Glucose: 2.8

mgdl: 100, mmol: true, Glucose: 5.6

mgdl: 0, mmol: true, Glucose: .0

Test Number: 10

Requirement being tested: RS007: Export data to various formats and services

Component being tested: ReadingTools

Method being tested: testHourToSpinnerType

Test input(s) including command-line argument(s):

24, 8 , 4, 19

Expected outcome(s):

Hour: 24, SpinnerType: 8, Time of day: night

Hour: 8, SpinnerType: 1, Time of day: after breakfast

Hour: 4, SpinnerType: 8, Time of day: night

Hour: 19, SpinnerType: 4, Time of day: before dinner

Test Number: 11

Requirement being tested: RS018: Provide correct Glucose calculations based on User's data

Component being tested: InputFilterMinMax

Method being tested: isInRange

Test input(s) including command-line argument(s):

[5.0,8.0,10.0]

[1.0,0.0,100.0]

[4.0,5.0,6.0]

Expected outcome(s):

Range Tester [5.0,8.0,10.0] = false

Range Tester [1.0,0.0,100.0] = false

Range Tester [4.0,5.0,6.0] = false

Test Number: 12

Requirement being tested: RS018: Provide correct Glucose calculations based on User's data

Component being tested: GlucoseRanges

Method being tested: colorFromReading

Test input(s) including command-line argument(s):

69, 201, 75, 195, 100

Expected outcome(s):

reading: 69, color: "purple"

reading: 201, color: "red"

reading: 75, color: "orange"

reading: 195, color: "orange"

reading: 100, color: "orange"

Test Number: 13

Requirement being tested: RS019: Provide alerts

Component being tested: GlucosioNotificationManager

Method being tested: generateRandomNumber

Test input(s) including command-line argument(s):

[10,100], [50,500], [1,11], [0,90]

Expected outcome(s):

Number between the 10 and 100

Number between the 50 and 500

Number between the 1 and 11

Number between the 0 and 90

Test Number: 14

Requirement being tested: RS001: Record user input - Blood Pressure

Component being tested: PressureReading.java

Method being tested: setMaxReading

Test input(s) including command-line argument(s):

180.0, 100.0, 10.0

Expected outcome(s):

Max Reading returned = 180.0

Max Reading returned = 100.0

Max Reading returned = 10.0

Test Number: 15

Requirement being tested: RS001: Record user input - Blood Pressure

Component being tested: PressureReading.java

Method being tested: setId

Test input(s) including command-line argument(s):

1234, 146789

Expected outcome(s):

Id returned = 1234

Id returned = 146789

Test Number: 16

Requirement being tested: RS001: Record user input - HB1ACReading

Component being tested: HB1ACReading.java

Method being tested: setReading

Test input(s) including command-line argument(s):

2.0, 10.0, 15.0, 100.0

Expected outcome(s):

Reading Returned = 2.0

Reading Returned = 10.0

Reading Returned = 15.0

Reading Returned = 100.0

Test Number: 17

Requirement being tested: RS001: Record user input - Glucose

Component being tested: GlucoseReading.java

Method being tested: setReading, getReading

Test input(s) including command-line argument(s):

2.0, 10.0, 15.0, 100.0

Expected outcome(s):

Reading Returned = 2.0

Reading Returned = 10.0

Reading Returned = 15.0

Reading Returned = 100.0

Test Number: 18

Requirement being tested: RS001: Record user input - A1c reading

Component being tested: A1cEstimate.java

Method being tested: setMonth, getMonth

Test input(s) including command-line argument(s):

June, may, july, november

Expected outcome(s):

month returned = june

month returned = may

month returned = july

month returned = november

Test Number: 19

Requirement being tested: RS018: Provide correct Glucose calculations based on User's data

Component being tested: GlucoseConverter.java

Method being tested: glucoseToMmolL

Test input(s) including command-line argument(s):

60.0, 2.0, 10.0, 11.0

Expected outcome(s):

glucoseToMmolL: 3.33

glucoseToMmolL: 0.11

glucoseToMmolL: 0.56

glucoseToMmolL: 0.61

Test Number: 20

Requirement being tested: RS007: Export data to various formats and services

Component being tested: ReadingTools.java

Method being tested: parseReading

Test input(s) including command-line argument(s):

6, 1, 2.3, 8.1

Expected outcome(s):

str to number "6" = 6

str to number "1" = 1

str to number "2.3" = 2.3

str to number "8.1" = 8.1

Test Number: 21

Requirement being tested: RS004 - Display Graphically

Component being tested: DoubleGraphObject.java

Method being tested: setReading, get/setCreated

Test input(s) including command-line argument(s):

n/a

Expected outcome(s):

2018-11-28T11:05:27.951-05:00 10.0

Test Number: 22

Requirement being tested: RS004 - Display Graphically

Component being tested: IntGraphObject.java

Method being tested: setReading, get/setCreated

Test input(s) including command-line argument(s):

n/a

Expected outcome(s):

2018-11-28T11:05:28.222-05:00 10

Test Number: 23

Requirement being tested: RS003 - Provide a log of user inputs for the past days

Component being tested: SplitDateTime.java

Method being tested: SplitDateTime; DateFormat.getDateInstance

Test input(s) including command-line argument(s):

getMinute()

getHour()

getDay()

getMonth()

getYear()

Expected outcome(s):

Current DateTime

(05 11 28 11 2018 output when last tested)

Test Number: 24

Requirement being tested: RS007: Export data to various formats and services

Component being tested: NumberFormatUtils.java

Method being tested: getMinimumFractionDigits; getMaximumFractionDigits

Test input(s) including command-line argument(s):

n/a

Expected outcome(s):

0 3

Test Number: 25

Requirement being tested: RS017 Build User

Component being tested: User.java

Method being tested: setPreferred_unit_a1c

Test input(s) including command-line argument(s):

"mg/dL", "mmol/L"

Expected outcome(s):

User's A1c unit set to mg/dL

User's A1c unit set to mmol/L

4.2 Evaluation and Experiences

Deliverable 4's production was difficult as we needed to revise our script to be more universal and modifiable. We wanted our script to run all of the tests without requiring hard coding the commands to compile our test java files. However, most of the java files in Glucosio must be compiled simultaneously and executed in a specific order which required us to build a creative solution to our script problem. After solving this, we had a good experience with building the test cases and using our script to run them. We split up the test case creation among us to ensure that we were all contributing to the testing and also testing different portions of Glucosio's requirements. Overall, we believe that our Glucosio testing framework provides a buildable

solution to testing the functionality and features of the Glucosio application and can be enhanced with further tests in the future.

5 Deliverable 5

5.0 Description

For our fault injection, we chose to create a second folder called TeamCorrectFaultInjections that our testing script could loop through and run each faulted tested file. We wanted to create a second folder for ease of use and easy replication.

5.1 How to Run Faults

1. Open Terminal
2. git clone https://github.com/csci-362-fall-2018-01/TeamCorrect
3. cd TeamCorrect/TestAutomation/scripts (May need additional navigation, depending on where the repository was cloned to)
4. ./runAllTests.sh
5. Output will be stored in an html file which should open automatically in the default browser with the fault injection output on the bottom.

*If any of the above commands return permission denied, use 'sudo' in front of the command.

*If the ./runAllTests.sh returns command not found, use 'sudo chmod 755 runAllTests.sh', then './runAllTests.sh'

5.2 Faults Injects

1) Test Case 06 Fault Injection:
    a) File/Method: CholesterolReading.setLDLReading
    b) Fault: LDL Reading was changed to HDL Reading
        i) LDL Reading is not correctly set with input of 6.4
    c) Output: LDL reading set to 0.0
2) Test Case 10 Fault Injection:
    a) File/Method: ReadingTools.hourToSpinnerType
    b) Fault: changed hour>20 to hour<20

c) Fault: changed hour>17 to hour<17
d) Fault: changed hour>13 to hour<13
e) Fault: changed hour>7 to hour<7
f) Fault: changed hour>4 to hour<4
    i) Logic is now incorrect for the expression
g) Output:
    i) Hour: 24, SpinnerType: 8, Time of day: night
    ii) Hour: 8, SpinnerType: 5, Time of day: after dinner
    iii) Hour: 4, SpinnerType: 5, Time of day: after dinner
    iv) Hour: 19, SpinnerType: 5, Time of day: after dinner

3) Test Case 11 Fault Injection:
    a) File/Method: InputFilterMinMax.isInRange
    b) Fault: Changed operation signs in expression
        i) (Original: b > a ? c >= a && c <= b : c >= b && c <= a)
        ii) (Current: b < a ? c <= a && c >= b : c <= b && c >= a)
        iii) Logic is now incorrect.
    c) Output: Range Tester [12.0,1.5,8.2] = true
4) Test Case 14 Fault Injection:
    a) File/Method: PressureReading.setMaxReading
    b) Fault: Changed min to max and vice versa
        i) Incorrect value returned for set max
    c) Output: Max Reading returned = 0.0 Max Reading returned = 0.0
5) Test Case 20 Fault Injection:
    a) File/Method: ReadingTools.parseReading
    b) Fault: Set all returns to null.
        i) All values return as null.
    c) Output: str to number 6 = null

5.3 Evaluation and Experiences

Fault injection was a fun experience because it meant breaking the code that we have been thoroughly testing. For our application, modifying the logic and expressions for the Glucose application made the most sense to ensure that the application was not comparing or calculating numbers correctly. We are happy with our fault injection as they test a variety of the functionality of the system. All 5 fault injections broke their respective functions and did not break additional functionality of the code. This illustrates how well the Glucosio application is low coupled and ensures the maximum functionality of the system overall.

6 Deliverable 6

6.0 Experiences and Lessons Learned

For this final deliverable, our team needed to compile the previous documents into a coherent document and finalize our presentation. We had a chance to review everything and revise any previous documents if necessary.

This project was challenging due to the complexity of the Glucosio project and the testing framework required. It was even more challenging due to the team aspect which required consistent communication and teamwork. We have learned many lessons from the project including new skills in Bash, Java, and testing. We have also learned about effective team dynamics and how to maintain a cohesive team when the challenges from other school projects come up.

6.1 Team Evaluation

Overall, our team worked well, and we were able to keep up with communication for each deliverable. We did struggle with participation at some points, but some team members filled in the gaps and made up any work that was not completed. We did not want any deliverables to not be submitted on time, so this required extra effort at some points from some team members.

After completing the project, we are happy with the automated testing framework that we wrote and the ability for it to be applied to even more test cases than 25. Glucosio is a huge project and we could have continued building test cases for another semester, but we feel we captured some of the most important ones in terms of the application's functionality.

6.2 Evaluation of Project Assignments

These project assignments were challenging and rewarding. It was difficult at first jumping into bash and immediately starting on a framework even with tutorials. However, after working on this project, as a team, we are much more comfortable with terminal and using bash to create an automated script. We believe the assignments were spaced out appropriately and allowed enough

time for completion. Overall, the project was a great challenge and we feel that we learned a lot throughout the whole process.