Team TBD

October 25th, 2016

Chapter 3 - Automated Testing Framework

Foreword - our experience with pidgin

With Pidgin we took on a project that was far beyond our abilities and the amount of time we have to put into it, but we did learn some interesting things from our attempt to create a working test driver for it.

- Big projects require a more complex framework to pull it all together
- Reinforced the idea that documentation is very important (pidgin had basically none)
- Computer languages have a lot of useful built in tools that are sometimes passed by that should be more thoroughly studied

The Testing Process

The ultimate goal of our team term project is to develop an automated testing framework for the ph7-Simple-Java-Calculator and implement the framework with twenty-five test cases, five injection faults, and plenty of documentation. We switched to this simple Java calculator from Pidgin, and recorded our experience with Pidgin in Chapter 2.1.

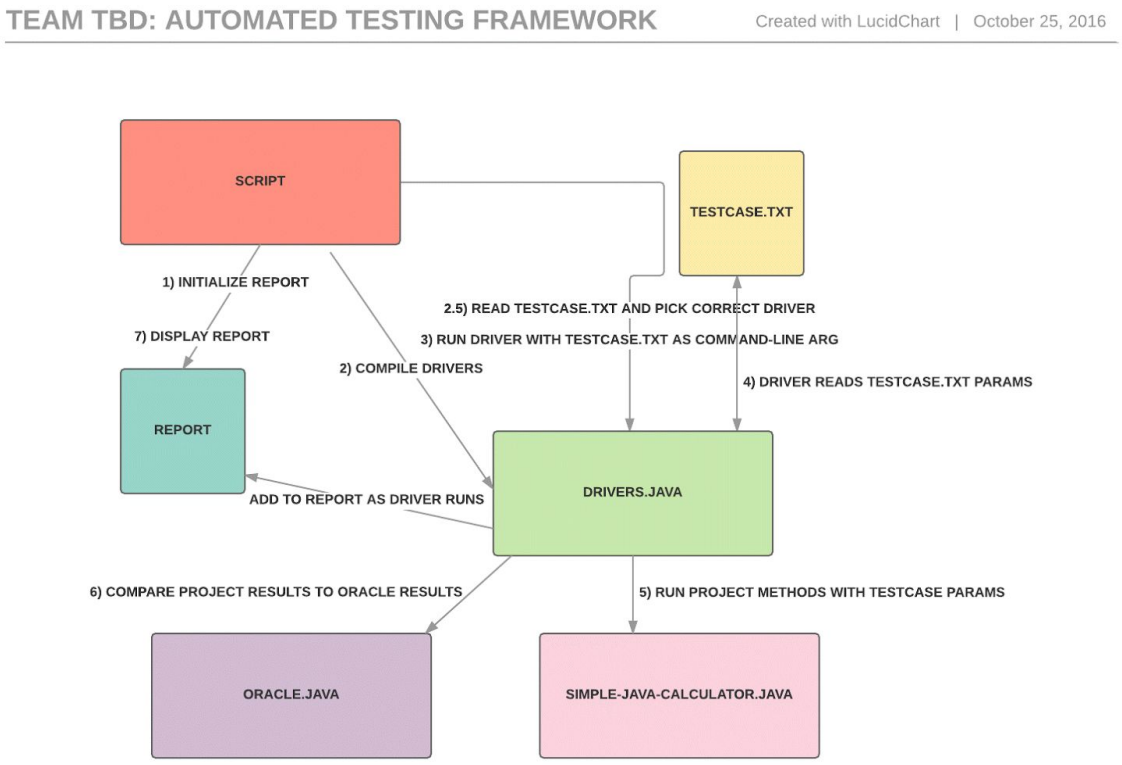
The calculator has just three Java files, one that defines the GUI, a second that defines the underlying Calculator, and a third that instantiates the GUI calculator. For this project we are ignoring the UI and simply testing the underlying Calculator class that contains computational methods.

Because we recently switched projects, we are planning on the modified schedule below:

1. Deliverable #3:
    a. Chapter 2 Revised - Deliverable #2 regarding our experience with Pidgin.
    b. Chapter 3 - New project test plan and testing framework with 12 test cases .
2. Deliverable #4:
    a. Completed and revised automated testing framework with 25 test cases and all necessary documentation on how it works and how to use it.
3. Deliverable #5:

    a. Fault injection - 5 faults that cause some but not all of our 25 test cases to fail.
4. Deliverable #6:
    a. Final Report

Testing Framework



- scripts/runAllTests.sh is a shell script that compiles and executes each testCase.java driver, passing the appropriate testCase.txt file as a command line argument.

- Each testCase.java driver takes a testCase.txt file name as a command line arg and reads the inputs specified in the file. The driver then executes the specified Calculator method, passing the specified inputs as parameters. Each driver can run multiple testCase.txt test cases, because the method that it chooses to run is specified in each test case file. The driver also runs an oracle.java file that calculates and returns expected results that are compared to the actual results produced by the Simple-Java-Calculator.

- testCase.txt file contains the details of each test case.

Instructions

- To actually run the calculator GUI, clone the whole repo and run in the terminal.

- To run our testing framework, run scripts/runAllScripts.sh at the command line.

Requirements traceability

List requirements here, see format

- *Take in numerical inputs*
- *Calculate squares*
- *Calculate square roots*
- *Calculate division*
- *Calculate addition*
- *Calculate subtraction*
- *Calculate multiplication*
- *Calculate trig functions*
- *Be able to clear the calculator inputs*
- *Display the human readable results*

Tested items

List of testcases

ID: 01
Requirement: Calculate squares
Component: Calculator.java
Method: calculateMono(MonoOperatorModes square, Double num)
Inputs: 2.0
Expected Outcome: 4.0

ID: 02
Requirement: Calculate squares
Component: Calculator.java
Method: calculateMono(MonoOperatorModes square, Double num)
Inputs: 9.0
Expected Outcome: 18.0

ID: 03
Requirement: Calculate square roots
Component: Calculator.java
Method: calculateMono(MonoOperatorModes squareRoot, Double num)
Inputs: 9.0
Expected Outcome: 3.0

ID: 04
Requirement: Calculate square roots
Component: Calculator.java
Method: calculateMono(MonoOperatorModes squareRoot, Double num)
Inputs: 0.0
Expected Outcome: 0.0

ID: 05
Requirement: Calculate division
Component: Calculator.java
Method: calculateMono(MonoOperatorModes onedividedby, Double num)
Inputs: 1.0
Expected Outcome: 1.0

ID: 06
Requirement: Calculate division
Component: Calculator.java
Method: calculateMono(MonoOperatorModes onedividedby, Double num)
Inputs: 10.0
Expected Outcome: 0.1

ID: 07
Requirement: Calculate trig functions(cos)
Component: Calculator.java
Method: calculateMono(MonoOperatorModes cos, Double num)
Inputs: 1.0
Expected Outcome: 0.540302305868

ID: 08
Requirement: Calculate trig functions(cos)
Component: Calculator.java
Method: calculateMono(MonoOperatorModes cos, Double num)
Inputs: 0.0
Expected Outcome: 1.0

ID: 09
Requirement: Calculate trig functions(sin)
Component: Calculator.java
Method: calculateMono(MonoOperatorModes sin, Double num)
Inputs: 1.0
Expected Outcome: 0.841470984808

ID: 10
Requirement: Calculate trig functions(sin)
Component: Calculator.java
Method: calculateMono(MonoOperatorModes sin, Double num)
Inputs: 0.0
Expected Outcome: 0.0

ID: 11
Requirement: Calculate trig functions(tan)
Component: Calculator.java
Method: calculateMono(MonoOperatorModes tan, Double num)
Inputs: 1.0
Expected Outcome: 1.557407724655

ID: 12
Requirement: Calculate trig functions(tan)
Component: Calculator.java
Method: calculateMono(MonoOperatorModes tan, Double num)
Inputs: 0.0
Expected Outcome: 0.0

Testing schedule

Deadline	Deliverable	Resources
October 25th	Testing framework with 12 test-cases	ph7-Simple-Java-Calculator
November 17th	Revised framework with 25 test-cases	TBD
November 29th	Fault Injection	TBD
December 6th	Final Report	TBD

Test recording procedures

Tests are recorded as date stamped html files called record$DATE.html and include the $USER who ran the test within the file. Tests are stored in the records/ directory.