Team Teamwork – CSStructural

Milestone 2 Report

Team Member	Role	Tasks Assigned	Completed
Joshua Feltman	Contact, Developer	Halstead Checks Unit and Integration Test	Yes
Matthew Johnson	Developer	Super Checks Unit and Integration Test	Yes
Andrew Fallin	Developer	Maintainability Index Unit and Integration Tests	Yes
Benjamin Hamlin	Developer	Helped plan out unit and integration tests	Yes

Testing Frameworks and Strategies:

The testing frameworks that were used for this deliverable was JUnit 4 and Mockito. We decided to use these two frameworks since these were the ones explained the most in lecture and were easy to import into Eclipse.

The integration test strategy that we adopted was the top-down approach. We believed this to be the most straightforward strategy since most of our classes had very small call graphs, which led us to only using stubbing instead of using both stubbing and drivers, which made our tests a lot easier to write.

Major Project Activities:

- Meetings:
  - October 8, 2018
    - First team meeting for this deliverable
    - Decided on how to split up the work for testing
  - October 15, 2018
    - Brief meeting making sure everyone was thinking about/working on unit tests
  - October 22, 2018
    - Another brief meeting making sure unit tests were being written.
    - Team is struggling with integration tests
  - October 24, 2018
    - Meeting after getting help with integration tests from professor

- Making sure everyone is caught up with how we are going to go about writing integration tests

Test Outcomes:

We did not find many bugs after writing unit tests for our structural metric checks, however we did realize that throwing all our logic into one function for a variety of checks made it extremely difficult to write unit tests for small defined components. This led us to refactoring a lot of the SuperCheck and Maintainability Index classes, so each check was being defined in its own separate decoupled function, that allowed for easier unit testing and integration testing.

After talking with the professor about integration tests, we ended up using the command line way of writing integration tests, which worked a lot simpler, but we ended up with integration tests that were not automated at all (which was said to be okay after talking to the professor). Since the integration tests take a while to manually run since we have to plug in each component for each class one at a time, we found it very tedious and inefficient to write integration tests like this.

The unit testing however was very simple to write and even easier to run as a whole with Junit and Eclipse.

How to Run Unit and Integration Tests:

<u>Prerequisites:</u>

1. Clone Team-Teamwork-CSStructural
2. Import net.sf.eclipsecs.sample from Team-Teamwork-CSStructual in Eclipse (see software documentation on how to)
3. Add JUnit 4 to the projects build path.
    a. Right click net.sf.eclipsecs.sample -> Build Path -> Configure Build Path
    b. Click on Libraries Tab
    c. Click on Add Library -> JUnit -> Next -> JUnit 4 -> Finish

<u>Unit Tests:</u>

1. In Eclipse, click on the src directory
2. Right click net.sf.eclipsecs.sample.checks -> Run As -> JUnit Test
3. Make sure the Window -> Show View -> JUnit is enabled

<u>Integration Tests:</u>

NOTE: To run the integration tests, you have to do it manually, adding in one component at a time, so bear with it.

1. Must have access to terminal (Mac OSX or Linux system)
2. Enter this command to get into the right directory
    cd <path>/Team-Teamwork-CSStructural/net.sf.eclipsecs.sample
    Where <path> is the directory you cloned the repo
3. Once inside, open up “sample_checks.xml” with a text editor

4. In the xml comment all modules out everything but the one:

   <module name="net.sf.eclipsecs.sample.checks.HalsteadChecksIntegration">

   </module>

   a. Save and exit the XML file
5. Now enter the command:

   bash integration.sh

   a. This will display the results of stubbed call graph for the Halstead Checks Class
6. Reopen the XML file and comment over every module again besides:

   <module name="net.sf.eclipsecs.sample.checks.HalsteadChecks">

   </module>

7. Renter the command from Step 5.
   a. This will display the actual Halstead Check results on the example.java file without any stubbed methods.
8. Repeat step 3 – 7 with the SuperCheck and MaintainabilityIndexCheck classes by commenting and uncommenting their modules in the “sample_checks.xml” file