SOEN 6011: Project Deliverable 3

Team C, Student N4: Kenlo Dinh Van, 26641652

August 2nd, 2019

1 Source Code Review of F5: $\Gamma(x)$

1.1 Repositories

The following report and Deliverable 3 artifacts are available on the following GitHub private repository. [**1**] Following Deliverable 2, Team members N5 and N8 (as N6 and N7 are no longer registered) shared their compressed source code and executable by email for Peer Review and Testing, respectively. In order to conduct a Code Review using GitHub features, F5 was then uploaded to its own repository [**2**], in a conventional team collaboration environment we simply would have created a new branch.

1.2 Programming Style

As previously mentioned in D2, the team aims to conform to Google's Java Style Guide [**3**] for our respective functions implementation.

1.3 Tools

The **CheckStyle plugin** [**4**] with embedded Google Checks, previously used during F4 development has also been used during the code review process to ensure that F5 did adopt the same coding standard.

Additionally, the native IntelliJ **Code Inspection** [**5**] feature was also used to highlight inconsistencies and make recommendations.

For code review purpose we've also used the tool **Reviewable** [**6**] which is an online code review that integrates within GitHub and allow to visualize the changes of a Pull Request.

1.4 Code Review Guidelines

Throughout this code review, We've aimed to follow the simple code review guidelines written on Philipp Hauer's Blog [**7**], which encourages a reviewer to express his point of view, ask questions, criticize or compliment the written code. The ensuing discussion and resolution should result high quality code. The review process was also heavily influenced by GitHub's

recommended code review [**8**] feature, which involves creating a Pull Request, inserting changes, and discussing the code.

1.5 Detailed Approach

The source code review process involved the following step:

Step 1: Run the java .jar file inside a command line window to ensure the program executes as expected, then try the Gamma function for a random value and compare the given result with Casio's Keisan Online Calculator [**9**].

Step 2: Import the decompressed Eclipse project into IntelliJ IDE, upload it to it GitHub (master) [**]2**].

Step 3: Run a CheckStyle Scan to ensure the programming style is conform.

Step 4: Create a new branch called 'review', and start editing/commenting the code based on the Code Inspection warnings.

Step 5: Create a Pull Request [**10**] on GitHub between the master and review branch, which includes all the comments and changes.

Step 6: Visualize the propositions of the pull request with Reviewable.[**11**]
(A further step would be for the Author to discuss the propositions, accept or reject them, then merge the branch into the master).

1.6 Results

F5 Gamma function implementation required 376 lines of code. From our observation the function seem to offer correct results with a precision of up to 1.0 x 10^{-8}.The Google's Java Coding Style Guide was respected with no errors nor warnings reported by CheckStyle (automated). Additional minor modification were then recommended by IntelliJ Code Inspection (automated), some of which we deemed interesting to consider and integrate. The detailed manual code review is available on:
https://reviewable.io/reviews/k3nlo/calculator/1

2 Testing F8: $B(x, y)$

2.1 Testing Approach

For Testing F8 a simpler approach was followed:

Step 1: Import the uncompressed source code and tests in IntelliJ and execute the main function.

Step 2: Try for random correct values (x,y) and compare the results to Casio's Keisan Online Calculator [**9**]. Precision was comparable up to 1.0 x 10^{-10}.

Step 3: Try for random incorrect values (x,y) and read the error messages. (x and y are supposed to be real positive numbers)

Step 4: Run the JUnit test cases. The 25 test cases successfully passed.

Step 5: Verify that the defined test cases do cover the previously elicited functional requirements.

2.2 F8 Requirements

FRQ1: Validation of input values
When x and y are received, the system should validate that x and y are real numbers greater than 0 2
Test cases 20 to 23: Test Gamma for valid and invalid inputs

FRQ2: Beta Function Calculation
After x and y are received and validated, the system should compute and store the Beta function value for x and y.
Test cases 1 to 21: Test all the implemented math "utility" functions (power, square root, ln..) and the Gamma calculation

FRQ3: Displaying the output value
After the beta function calculation is completed, the system shall display the output value of the beta function
Test cases 20 to 21: The tests testBetaFunctionForIntegers and testBetaFunctionForRealNumbers are re-used to assert an equality in values, but does it test the display of value ?

FRQ4: Interacting with user
The system shall provide a textual User Interface for the user to enter x and y and to display the result.
Test cases 20 to 21: The tests testBetaFunctionForIntegers and testBetaFunctionForRealNumbers are again re-used to assert an equality in the calculated values, but does this test the display of user interface ?

FRQ5: Displaying previous results
The system shall display all previous Beta Function calculations when prompted by the user.
Test case 24: Specific test case to ensure proper functionality of the user history.

FRQ6: Deleting history The system shall delete all previous Beta Function results when prompted by the user. **Test case 25**: Specific test case to ensure proper functionality of the user history.

Recommendations:
It seems that the test cases 20 and 21 are covering a range of requirements from FRQ1 to FRQ4.

FRQ3 refers to the system ability to display the results, wouldn't two dedicated test cases that assert the output of a calculated result and the output of an error message if the program does not have a result to display match that requirement?

Similarly, if FRQ4 refers to the user interface and interaction, maybe two test cases asserting that the program displays a user input prompt with message and an error message in case of wrong input be adequate?

The Test Class and accompanying report were thoroughly detailed and made testing the program a breeze. Thank you.

References

[**1**] https://github.com/k3nlo/SOEN6011_TeamC_N4/26641652_D3

[**2**] https://github.com/k3nlo/Calculator/blob/master/src/Calculator.java

[**3**] https://google.github.io/styleguide/javaguide.html

[**4**] https://plugins.jetbrains.com/plugin/1065-checkstyle-idea

[**5**] https://www.jetbrains.com/help/idea/code-inspection.html

[**6**] https://reviewable.io/

[**7**] https://phauer.com/2018/code-review-guidelines/code-reviews-guidelines-for-the-reviewer

[**8**] https://github.com/features/code-review/

[**9**] https://keisan.casio.com/exec/system/1180573444

10] https://github.com/k3nlo/Calculator/pull/1

[**11**] https://reviewable.io/reviews/k3nlo/calculator/1