CISC / CMPE 327 - Fall 2019

Assignment #3 - Front End Requirements Testing

In this assignment, you will test the prototype Front End you built in Assignment 2 using the acceptance test cases you created in Assignment 1. You should proceed as follows:

1. According to your chosen programing language, pick one of the example templates provided on our GitHub group: https://github.com/CISC-CMPE-327

2. The provided template has an example program and showcases a typical approach to conduct requirement testing. The example program is different to our frontend requirement! It takes a program argument, versus our frontend program which takes two arguments. ***Thoroughly understand the testing code as well as how things are structured referring to the code comments in the template.***

    a. For Python, we will use the ***pytest*** framework. We wrote a ***run_app*** helper function to capture, redirect, and compare different output signals. The helper function can compare the tail section of the terminal output, so we don't need to include welcoming messages (e.g. 'hello world!' in the example) as the expected output.

    b. For Java, we will use the Junit framework (well, the name is a bit misleading, it is also used for other tests rather than just unit testing). Like the Python template, we also create a helper function to capture, redirect, and compare different output signals.

    c. For C/C++, we will use the script-based approach. The template also provides an example script for the program that takes one argument.

3. **Adopt the chosen template for your frontend.** Again, the provided example only takes one argument, but here our frontend takes two. **Implement your own helper function or script for testing.** *You can propose a different testing approach than the ones provided in the templates.*

4. Implement and prepare the expected outputs for all the test cases you listed in assignment #1. This sounds a lot, but it actually only involves copy, paste, and slight modification once you set up the helper function or script.

5. Run all the testcases using the testing framework (e.g. pytest or mvn test) or using your script (e.g. make test) locally or through GitHub Action.

6. Compile a report of each failure observed, noting which test failed, what it was testing, and what was wrong about its output.

7. Fix each of the observed failures, note in the failure report what actions or changes were made to address it. For example, if the test input itself was wrong, you would write in the report that you removed or fixed the test input. If your program was wrong, you would write in the report what the error in your code was and how you changed the program to fix it.

8. After you've fixed all the observed failures, go back to step 5 and run ALL your tests over again to see if the problems are fixed (or if you've created any new ones). Keep repeating this loop until all your tests work properly.

What to Hand In

In this assignment, you will hand in, as a PDF in OnQ:

1. Descriptions on how you modified the template to conduct the testing. If you propose a different approaches than the template, explain how you did that and justify why.

2. The detailed failure report for your test runs, with a row for each observed test failure and columns showing the test name, what it was testing, how its output was wrong, what the error in your code was, and how you changed the program (or test input) to fix it. You may wish to make this a spreadsheet.

3. Your GitHub repository as well as the tag for this assignment. Like we discuss before, your assignment is submitted through tags on commits (snapshot). So please feel free to delete or re-organize your repository folder structure. We can still access the original past assignment submission through the snapshots with tag.

Marking Scheme (SUBJECT TO CHANGE)

Marking is out of ten, according to the following criteria. In each category, marks are assigned between zero and the number of marks shown, to a resolution of 1/2 mark.

Scripts or Programs to Run Your Tests — 4 marks

Organization
- clear, readable scripts or testing code
- understandable variable names
- clear internal comments or documentation explaining the script or testing process

Automation
- script or program does vast majority of test work, minimal hand work to be done

Failure Report — 4 marks

Organization
- failure report in clear table form, listing test name or number, what was being tested, the nature of the failure, what the error in the code was, and what actions were taken to fix it

Completeness
- all tests have been run, and all failures addressed

New Source Code
- listing of final source code that passes all tests

Overall Presentation & Quality — 2 marks

=======

10 marks