TDA/DIT 594 - Assignment 5 - Feature-Based Testing

Due Date: Sunday, January 10, 11:59 PM
Submission: Via Gitlab and Canvas

Overview

Your company has established a flexible framework for modularizing features (to the extent possible) and enabling/disabling them via configuration and parameters. Now that it has a working implementation and solved various technical issues, it wants to get a proper quality assurance through testing, to sustain the platform in the long term, since many more features are planned.

You know that two types of testing are essential to ensuring that features work well and are well-modularized:

- Unit tests are used to verify the correctness of individual features.
- System-level, or integration, tests are used to examine combinations of features.

You have decided to create a small number of tests for both scenarios as a way to establish your testing pipeline. You will use Continuous Integration to automatically execute these tests when code is checked in, as a way to immediately see if code changes have broken working functionality.

Following this assignment, you will be able to create test cases (using the JUnit framework) that test both individual features (unit tests) and combinations of features (integration tests). You will also understand the use of combinatorial interaction testing to choose interesting feature combinations for integration testing.

Your Tasks

1. If you have not yet, enable Continuous Integration. Once set up, it will execute all JUnit test cases in your project automatically. The following tutorial will help[1].
2. Create unit tests (using either JUnit 4 or 5) manually for two of your individual features implemented in Assignment 3 and/or 4.
    a. These should not be too trivial features, but don't need to be too difficult either. The features should have complex conditional behavior in their code (multiple if-statements or switch statements, especially nested if-statements), indicating multiple possible outcomes when it is executed. If unsure of what to focus on, discuss your choices with your supervisor.
    b. For each feature, create unit tests that cover each of the major outcomes of that feature (i.e., if there is conditional behavior, try to cover each condition that affects the outcome). If there is exception handling, try to ensure that

[1] https://chalmers.instructure.com/courses/7439/pages/continuous-integration-with-gitlab-ci

exception-triggering input is handled. We do not expect exhaustive testing, but try to write enough test cases to ensure thorough testing.
    c. There is not a correct "number" of tests, but try to aim for 5+ per feature.
3. Perform the process for Combinatorial Interaction Testing (as discussed in Lecture 12) to identify a set of feature configurations that would cover all 2-way feature interactions.
    a. Create a table where each column represents a variation point, and each value is a feature you can choose for that variation point.
    b. List the total number of test cases that would be required to cover the full set of possible feature interactions.
    c. Identify a set of feature configurations that covers all 2-way feature interactions. State how many total configurations are in this set.[2]
4. Choose two features and write tests using JUnit that test the combination and interaction of those features.
    a. Often this type of testing is done through a high-level interface, but for this assignment, you can call methods of individual classes. The goal here is to test an identified pair of features in some interesting way.
    b. Similar to the unit testing task above, try to write tests that cover multiple ways these features can interact with each other. You do not need to be exhaustive, but try to cover a range of major execution outcomes that could emerge from this combination of feature choices. Again, try to aim for 5+ tests.
    c. These do not need to be the same two features you tested individually earlier, but choosing features to test individually that would naturally be used in combination may help in testing their interaction.

Hints

If you have not used JUnit before, there are several excellent tutorials. Some options:

- https://www.vogella.com/tutorials/JUnit/article.html
- https://www.tutorialspoint.com/junit/index.htm
- https://www.guru99.com/junit-tutorial.html

A quick JUnit reference can be found at:
https://docs.google.com/document/d/1_rrUlDcA7E9UcVY7mZLa64ZlWtDDEfgpsfs6N51QBto/edit?usp=sharing

[2] You do not need to create actual test cases for this task, just the set of feature selections that would cover all of the 2-way interactions between the features chosen for variation points.

Deliverable

Submit, via Canvas, the following (one submission per team):

- A document in PDF format, containing:
    - A link to a “release” of your GitLab repository containing the source code of this assignment[3].
    - A list of the test cases designed and brief explanations of each:
        - Note whether it is a unit or integration test.
            - If it is a unit test, state which feature it tests.
            - If it is an integration test, state which combination of features it exercises.
        - List the file where this test is located in the source code.
        - Explain the purpose of the test (i.e., what functional outcomes are covered, what types of errors it could identify).
        - Document why the specific input was chosen that you applied in the test case.
        - Document why the assertions/fail statements used in the test are sufficient to demonstrate the correct behavior of that feature.
    - The full matrix of feature choices for variation points you created for performing combinatorial interaction testing.
    - The set of feature configurations that you identified that covers all 2-way feature interactions.
        - List the total number of tests that would be needed to cover all feature interactions from this table once.
        - Note the total number of tests needed to cover all 2-way feature interactions once.

Grading Guidelines

Note, these guidelines are intended to give some guidance, but are not exhaustive. Each supervisor will assign a grade based on the correctness and quality of your work.

Grade (Chalmers)	**Grade (GU)**	**Guidelines**
5	VG	● Each feature and feature combination is thoroughly tested, covering the full range of outcomes and exception handling. ● All tests are well explained, covering purpose, input choices, and assertion choices in detail and with included rationale. Why these tests are sufficient to show correctness of a feature or feature combination is clearly explained. ● Test code is commented and can be understood

[3] For information on creating a release, see https://docs.gitlab.com/ee/user/project/releases/

		without having written the code. ● The Continuous Integration process passes for all test cases. ● Full table of configurations for Combinatorial Interaction Testing is included, as well as the identified set of 2-way feature interactions that could cover all pairs of features. ● The identified set of 2-way interactions is compact, and further reductions in the number of test cases are not possible without significant effort (i.e., if you cover something in 50 tests and we immediately see a way to cover it in 24 tests, then this condition is not met).
4	G	● Each feature and feature combination is well-tested, covering a wide range of outcomes and exception handling. ● All tests are explained, covering purpose, input choices, and assertion choices with included rationale. Some attempt is made to show why these tests are sufficient to show correctness of a feature or feature combination. ● Test code is commented. ● The Continuous Integration process passes for all test cases. ● Full table of configurations for Combinatorial Interaction Testing is included, as well as the identified set of 2-way feature interactions that could cover all pairs of features. ● The identified set of 2-way interactions is compact, and further reductions in the number of test cases are not possible without significant effort.
3		● Each feature and feature combination is tested, covering a range of outcomes and exception handling. ● All tests are explained, covering purpose, input choices, and assertion choices with included rationale. ● The Continuous Integration process passes for all test cases. ● Full table of configurations for Combinatorial Interaction Testing is included, as well as the identified set of 2-way feature interactions that could cover all pairs of features. ● The identified set of 2-way interactions is compact.
U	U	● Testing effort is inadequate, and multiple tests

		are not present for one or more features or the combination. ● Tests are not explained or poorly explained. No or little rationale for choices made. ● Continuous integration not used or tests do not pass (tell us ahead of time if there is an issue). ● Set of 2-way interactions is not identified or major mistakes are made in their identification.