TDA 594/DIT 593 - Assignment 5 - Feature-Based Testing

Due Date: Sunday, December 19, 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 explore 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 individual features (unit 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 a set of unit tests (using either JUnit 4 or 5) for one of the individual features implemented in Assignments 3 and 4.
    a. These should not be too trivial features. 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 a set of 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 exception-triggering input is handled. We do not expect exhaustive testing, but try

[1] https://chalmers.instructure.com/courses/16077/pages/continuous-integration-with-gitlab-ci

to write enough test cases to ensure thorough testing. There is not a correct "number" of tests, but try to aim for 5 or more.

   c. Use Continuous Integration to execute all test cases.

3. Perform the process to identify test specifications for Combinatorial Interaction Testing (as discussed in Lecture 11) to identify a set of feature selections that would cover all 2-way feature interactions. This set is a starting point for designing integration/system-level tests.
   a. For this task, use the feature model you derived in Assignment 2 (Domain Analysis).
   b. Create a table[2] where each column represents a variation point, and each value is a feature you can choose for that variation point.
   c. List the total number of test cases that would be required to cover the full set of possible feature interactions (i.e., if you did not perform combinatorial interaction testing to identify a smaller subset).
   d. Identify a set of feature selections that covers all 2-way feature interactions. State how many total feature selections are in this set.[3]

Note that, for task 3, you do NOT need to write JUnit test cases. You simply need to identify the test cases that you would go on to create following the identification process.

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_rrUIDcA7E9UcVY7mZLa64ZIWtDDEfgpsfs6N51QBto/edit?usp=sharing

The class RobotTestBed can be used to help with testing RoboCode bots. A tutorial on testing in RoboCode can be found at https://www.youtube.com/watch?v=dHrupYhq7sI.

Deliverable

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

- A document in PDF format, containing:

[2] See Lecture 11 for examples.
[3] You do not need to create actual test cases for this task, just identify the set of feature selections that would cover all of the 2-way interactions between the features. **You may ignore all cross-tree constraints when performing this task.**

- A link to a “release” of your GitLab repository containing the source code of this assignment[4].
- A list of the test cases designed and brief explanations of each:
  - State which feature is being tested.
  - List the file where this test is located in the source code.
  - Explain the purpose of the test (e.g., 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 show that the feature’s behavior is correct or incorrect.
- The set of feature selections that you identified that covers all 2-way feature interactions, in table format.
  - List all choices and their possible values.
  - List the total number of tests that would be needed to cover all feature interactions.
  - List the total number of tests needed to cover all 2-way feature interactions.

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	Guidelines
5	● The chosen feature 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 is clearly explained. ● Test code is commented and can be understood without having written the code. ● The Continuous Integration process passes for all test cases. ● Full table of selections for Combinatorial Interaction Testing is included. ● The identified set of 2-way interactions is compact, and further reductions in the number of test cases are not possible without significant effort (e.g., 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	● Each feature is well-tested, covering a wide range of outcomes and exception handling. ● All tests are explained, covering purpose, input choices, and assertion

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

	choices with included rationale. Some attempt is made to show why these tests are sufficient to show correctness of a feature. ● Test code is commented. ● The Continuous Integration process passes for all test cases. ● Full table of selections for Combinatorial Interaction Testing is included. ● 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 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 selections for Combinatorial Interaction Testing is included, and is compact.
U	● Testing effort is inadequate. ● 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.