COMP 3350 Project Iteration 2

Group 6 (R.O.S.E.)

Planning and process (4/4)

- GIT (1/1)
    - version control is being used properly for example, has more than one committer and commits are reasonable size and frequency. They are not only big commits at the end. (1/1)

  Comments: No comments

- Updated plan (2/2)
    - Plan should be up-to-date (if there is any change to the previous plan for Iteration 2 it should be explicit and justified) (0.5/0.5)
    - Big user stories for iteration 3, if it was not already in plan (0.5/0.5)
    - Development tasks assigned in iteration 2 (what exactly has been done by developers) (0.5/0.5)
    - The time planned for the development tasks and detailed user stories and the actual time it took, in iteration 1 (-/0.5)

  Comments: No comments

- Wiki (1/1)
    - Should include description of the content of the submission. Can include other things as well. (1/1)

Comments: No comments

Functionality (7.5/8)

- DB support including the actual DB and stubbed version (2/2)
- System performs end-to-end (including GUI) processing for all stories (2/2)
- Works on both emulator and tablet device. (2/2)
- The developed program conforms the updated plan (the stories that are claimed to be implemented, are indeed there) (1/1)
- No easy bug (No crashes or unexpected behavior while trying normal scenarios) (0.5/1)

Comments:

1. Bug: SORT (no matter what criteria, for any field), all searches showing the same set of recipes. -0.5

Implementation (5.5/6)

- No obvious code/design smells (1.5/2)
    - Classes are in the wrong package (e.g., logic is developed in the UI layer)
    - Big classes: Classes are taking too much responsibility (SRP)
    - Very long methods (over 20 lines)
    - Wrong usage of inheritance
- Proper dependency injection for DB (2/2)
- Good standard coding style (2/2)
    - Informative naming
    - Comments explain "why" and not "What"
    - No to-do
    - Too much code duplication (copy-paste)

Comments:

1. Very long methods: All functionality of "Edit" & "SortRecipes" in just 2 methods, for each class. -0.5
2. Consistent naming should be used. For instance, "mealTypes" in class "SortRecipes" is camel case but another variable just below is lower case.

Unit tests (5/7)

Automated JUnit test cases for both new features and old ones (2/2)

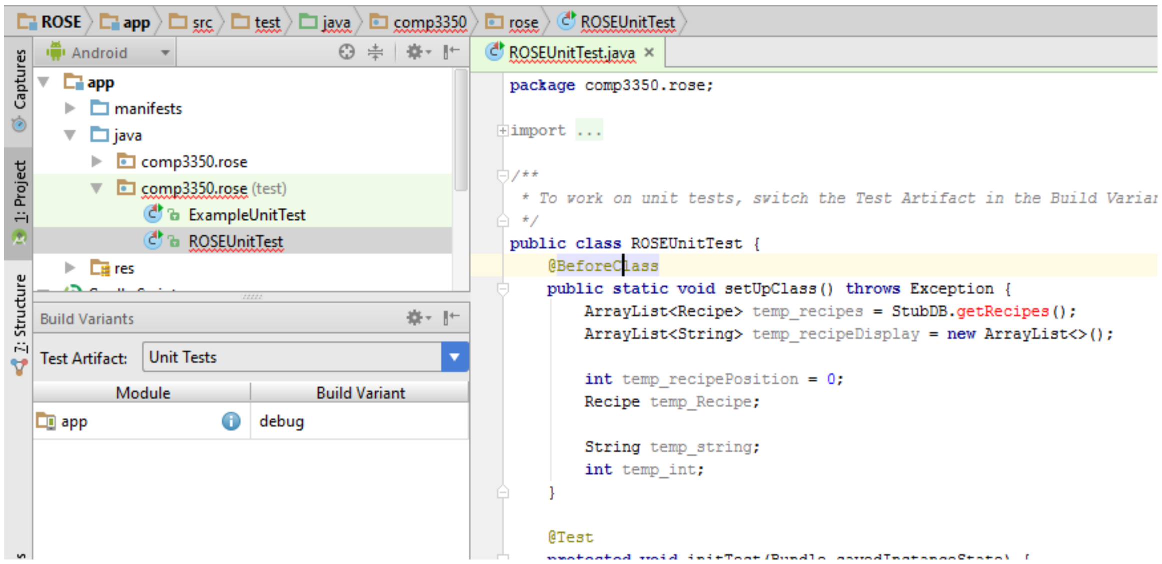
Passes all unit tests for domain objects and business logic (0/2)

Reasonable test coverage of normal and corner cases (1/1)

Integration tests (actual DB in the loop) (2/2)

Comments:

1. Unit tests could not be compiled and run. Screenshot attached below. -2



Test Artifact: Unit Tests

```java
package comp3350.rose;

import ...

/**
 * To work on unit tests, switch the Test Artifact in the Build Varia
 */
public class ROSEUnitTest {
    @BeforeClass
    public static void setUpClass() throws Exception {
        ArrayList<Recipe> temp_recipes = StubDB.getRecipes();
        ArrayList<String> temp_recipeDisplay = new ArrayList<>();

        int temp_recipePosition = 0;
        Recipe temp_Recipe;

        String temp_string;
        int temp_int;
    }

    @Test
```

Penalties ()

- Log file (up to -2 if missing or incomplete)
- Missing libraries. Unspecified dependencies. (up to -2)
- Previous unresolved issues (up to -5)

Comments:

Total (22/25)