**Report**

For this project, along with mutation testing we have to perform refactoring of code to improved code quality, write acceptance tests and perform unit testing and mutation tesing using Pitclipse tool.

Used Google Code Pro to calculate Cyclomatic Complexity for the Coffee Maker project. Cyclomatic Complexity indicates complexity of a program and higher the Cyclomatic Complexity, it becomes difficult to test the program. Google Code Pro tool suggests to modify the methods with complexity of 7 and higher. In our projects, some methods in CoffeeMaker, Recipe and Inventory have complexity of more than 7 . For example addRecipe() in CoffeeMaker.java class check for : if recipe already exists and search for empty slot to add recipe and then adds recipe. Hence extracted method isRecipeExist() and returnEmptySlot() from addRecipe(). This improved readability of the code thus its quality.

Writing acceptance tests from user stories was good experience. We have to study user stories carefully and consider all the details in user stories. We need to test whether all requirements are satisfied or not. For example only three recipes can be added in coffee maker and names of each recipe’s name should be unique. Thus in addition to test case to add recipe we need to write test case to add already existing recipe and to add more than three recipes. Similarly for other methods we have to examine User Stories to derive requirements to write acceptance tests.

To perform testing, implemented acceptance tests as unit test using JUnit test cases and run the test cases with EclEmma tool to check for code coverage. To improve the code coverage, added test cases to test Inventory and Recipe classes.

Finally used Pitclipse tool to perform mutation testing. Pitclipse tool is a plugin for Eclipse IDE for mutation testing. Pitclipse evaluates unit test cases. Pitclipse mutates the code and checks whether test case fails or not.

Some mutants are:

1. Conditional boundary mutator: Replaces relational operators (<, <=, >, >=). Example if (num1 <= num2) it mutates if (num1 < num2).
2. negate conditionals mutator: for example, if(num1 == num2), it replaces if (num1 != num2)
3. Math mutator: replaces arithmetic operators. Example a + b will be mutated as a-b.
4. Increments mutator: Replaces increments with decrements and vice versa.

Pitclipse evaluates test cases against all the possible mutations and demonstrates which mutants survived or which are killed by our test cases. It tests for all the situations which can break the code.

**public** **boolean** enoughIngredients(Recipe r) {

**if** ((Inventory.*coffee* < r.getAmtCoffee())

|| (Inventory.*milk* < r.getAmtMilk())

|| (Inventory.*sugar* < r.getAmtSugar())

|| (Inventory.*chocolate* < r.getAmtChocolate())) {

**return** **false**;

}

**return** **true**;

}

If above program is change to Inventory.*coffee* <= r.getAmtCoffee()thenit may return wrong results which is risky. Pitclipse revealed that this situation was not handled. Hence added test case in InventoryTest class to handle it.

Overall this project enhanced my understanding and confidence in testing. It provided experience of using various useful tools such as Pitclipse, Google Code Pro and Eclemma. I learned to write acceptance tests from User Stories and unit test cases. Through this project I learned how to test the code effectively.