

The code smell under test being evaluated is “**Duplicate Code**” submitted by *David Kelly*
<https://github.com/subukandaswamy/582Project/tree/David>

Based on the unit test, there are two test cases that have been implemented to test the **hasDuples()** method in the **DuplicateCodeCheck** class.

The first test case, **testHasDuples_ReturnsTrue()**, checks whether the method correctly identifies duplicated lines in a list of code lines. The second test case, **testHasDuples_ReturnFalse()**, checks whether the method correctly identifies non-duplicated lines in a list of code lines.

However, the first test case doesn't seem to be testing a realistic scenario, as it compares the same line of code twice, which would always result in duplication. A more realistic test case would be to have two different lines of code that are similar or identical.

Additionally, there is a third test case **testDuplicateCodeCheck()** that seems to be trying to read from a file, but it is not clear how it relates to the testing of the **DuplicateCodeCheck** class, and it doesn't seem to be checking anything or asserting any expected behavior.

Overall, while the two implemented test cases appear to be testing the intended behavior of the **hasDuples()** method, they could be improved by testing more realistic scenarios and by providing more extensive coverage of the method's behavior, such as testing edge cases or different inputs. Additionally, the third test case **testDuplicateCodeCheck()** should be revised or removed to better align with the testing of the **DuplicateCodeCheck** class.

To improve the test cases, we can use a combination of techniques and tools. One strategy is to generate test cases automatically using a code coverage tool. This tool can analyze the code and generate test cases that cover all possible code paths. Another strategy is to use mutation testing to evaluate the quality of the test suite. Mutation testing involves modifying the code slightly to create faulty versions of the program. If the test suite can detect these faults, then it is considered effective.

Another technique to improve the test cases is to use mocks and stubs to isolate the code being tested from its dependencies. This technique can help to simplify the test setup and make the tests more focused. Tools such as JUnit and Mockito can be used to implement these techniques and improve the test cases.

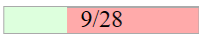
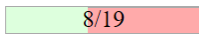
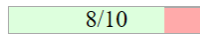
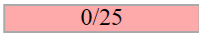
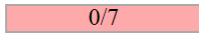
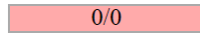
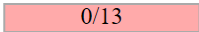
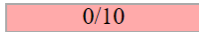
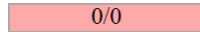
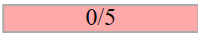
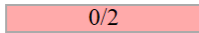
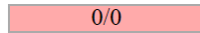
Link for the Test cases: <https://github.com/CPTS-582-Group-Work/Project-Deliverable-3>

The new (modified) test - **CodeSmell.DuplicateCodeCheck**: on the other hand /*This unit test creates an instance of myClass and calls the AA() and AB() methods to populate an ArrayList with their return values. The hasDuples() method is then called to check for any duplicate entries in the ArrayList, and the test asserts that the result is true, which indicates that there is indeed duplicate code in myClass.

Below is a comparison of the mutation run:

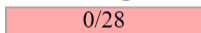
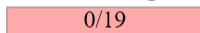
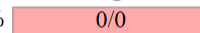
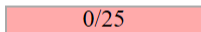
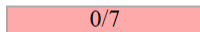
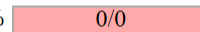
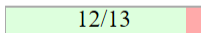
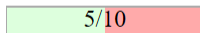
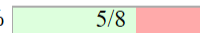
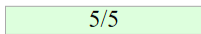
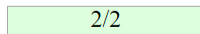
The old: **DuplicateCodeCheck**

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
DuplicateCodeCheck.java	32% 	42% 	80% 
TestDuplicateCode.java	0% 	0% 	0% 
WhiteBoxTestDuplicateCode.java	0% 	0% 	0% 
myClass.java	0% 	0% 	0% 

The new **CodeSmell.DuplicateCodeCheck**:

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
DuplicateCodeCheck.java	0% 	0% 	0% 
TestDuplicateCode.java	0% 	0% 	0% 
WhiteBoxTestDuplicateCode.java	92% 	50% 	63% 
myClass.java	100% 	100% 	100% 