Smell Information and Refactoring

What is code smell?

Code smells as "indications that there is trouble" in the code, which can be solved by refactoring. Code smells indicate areas, which potentially house the root of bug or error-causing code, as they are "symptoms of poor design and implementation choices". Code smells have many implications and consequences. The smell Duplicated Code denotes a code portion that can be found in more than one place. This duplication makes the code harder to maintain and debug. Changes to the code in one-part draw changes in the duplicated version with it. This impediment can easily introduce bugs into the code leading to errors.

What is refactoring?

The method to remove smells and to thereby improve the code is performed by refactoring. So, can duplicated code in two methods be refactored by extracting the duplicated code into a method, which will be invoked from both previous places. Refactoring will ease the future development and present maintainability, as a combination of smells significantly reduces the comprehension of the code.

Code Smells and Refactoring by Martin Fowler

There are various code smells and refactoring described by Fowler in his book. If you would like to go through all the smells he has described, follow the below mentioned link.

https://www.csie.ntu.edu.tw/~r95004/Refactoring_improving_the_design_of_existing_code.pdf

List of code smells that we have considered in our study described by Fowler, is mentioned below:

ID	Smell Name	Description
F2	Long Method	A Long Method occurs if a function does more than one thing.
F3	Large Class	The class is doing too much. A Large Class often leads to duplicated code and chaos.
F4	Long Parameter List	A Long Parameter List smell is a method that requires too many parameters. A long list of parameters is hard to understand, may becomes inconsistent, difficult to use and is suspect to constant changes as you need more data.

Fowler has further defined the ways to eliminate the smells. Fowler first defined several refactoring methods. He then described ways to remove smells by applying those refactoring on the smelly code. There are many instances and scenarios where smells can take place.

List of code smells refactoring that can be applied to remove the smells affecting the test cases in our study, as described by Fowler:

ID	Smell Name	Refactoring
F2	Long Method	Find parts of the method that go together and extract them into a new method.
F3	Large Class	Bundle instance variables together and extract those into new classes.
F4	Long Parameter List	Replace the parameter with a new class that holds all requested data, or if they already belong to one object, pass the whole object.

Code Smells and Refactoring by Van Deursen

There are various code smells and refactoring described by Van Deursen in this paper. The code smells indicate trouble in test code, i.e., test smells. Code smells are not specific to production code and can also be applied to test code, but van Deursen et al. have acknowledged that refactoring test code requires additional test-specific refactoring.

If you would like to go through all the smells he has described, follow the below mentioned link.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.486&rep=rep1&type=pdf

List of code smells that we have considered in our study described by Van Deursen, is mentioned below:

ID	Smell Name	Description
D4	General Fixture	The fixture is too general. Individual test cases only access and require part of the provided fixture.
D5	Eager Test	The test checks too much functionality/methods of the object under test in a single test case.
D6	Lazy Test	Several test methods check the same method using the same fixture. The tests only have meaning when they are considered together.

List of code smells refactoring that can be applied to remove the smells affecting the test cases in our study, as described by Van Deursen:

ID	Smell Name	Refactoring
D4	General Fixture	Extract the parts of the fixture that are not required by all methods into the methods that require it.
D5	Eager Test	Separate the test case into methods that test only one method of the class under test. Additionally, assign meaningful names to the methods describing the goal of the test cases.
D6	Lazy Test	Combine the individual test cases into one test method.