## 3.1:

Code Smell: Code duplication. There are serious problems with repetition rates. And it is so hard to maintain. It needs to be changed frequently. There are lots of risk that faults are introduced. It may easily affect other software behaviors. Code duplication is almost the most common smell. Code duplication often comes from the copy-and-paste programming style. Due to high code repetition rate, I chose the code duplication to explain and discuss.

## 3.2:

For some problem about code smell, we can use extract method to solve code clones. Organize a piece of code together and separate it into a separate function, and let the function name resolve the purpose of the function. When we need to handle complex functions, long functions we can use extract method. Fine-grained functions are easy to reuse, easy to comment, and easy to overwrite. Because the smeller the module, the greater the reusability, we can break down code modules. If the function is too large, it means that this function needs to be refactored. Because the function is too large, maintainability and intelligibility will deteriorate. And then when we implement similar functions, it is easy to generate duplicate code. So when the function is too large, we need to subdivide it and split the original function into several functions. There are three ways to extract method. First of all, no local variables. And then there is a local variable, but the target method only reads it. Finally, There are local variables and the target method has to assign them. In summary, there are three benefits to choosing a refactoring approach. Firstly, each method is fine-grained enough to facilitate code reuse. Secondly, High-level methods are as clear and clear as comments. Finally, if the methods are fine-grained, method replication will become easier.

**Reference:**

1: Liu W, Liu H. Major motivations for extract method refactorings: analysis based on interviews and change histories[J]. Frontiers of Computer Science, 2016, 10(4): 644-656.

2: Tsantalis N, Chatzigeorgiou A. Identification of extract method refactoring opportunities for the decomposition of methods[J]. Journal of Systems and Software, 2011, 84(10): 1757-1782.

3: Marticorena R, López C, Crespo Y, et al. Refactoring generics in JAVA: a case study on Extract Method[C]//2010 14th European Conference on Software Maintenance and Reengineering. IEEE, 2010: 212-221.

## 3.3:

**a:** The test refactoring class makes it easy to verify the correctness of the refactoring when we refactor, because the test case is executed after each refactoring to ensure that our refactoring is correct.

**b:** For the aspects involved in refactoring. Cover all pages to see all fields and functions, we need to ensure that every feature is covered as much as possible.

**c:** Run tests frequently. Every time we compile, we need to take the test into consideration, and execute each test at least once a day.

**d:**For quality standard, we can use TDD or unit test introduction. And for efficiency standards, we can use comparison test or case-statement approach. We only need to define the input, we do not need to define the output, and we can use the output of the old system as the output. At the same time, the validation rules have also been simplified a lot. The data usually only needs to be consistent with the old system, and a small number of data that is not completely consistent can be verified only be meeting certain rules.