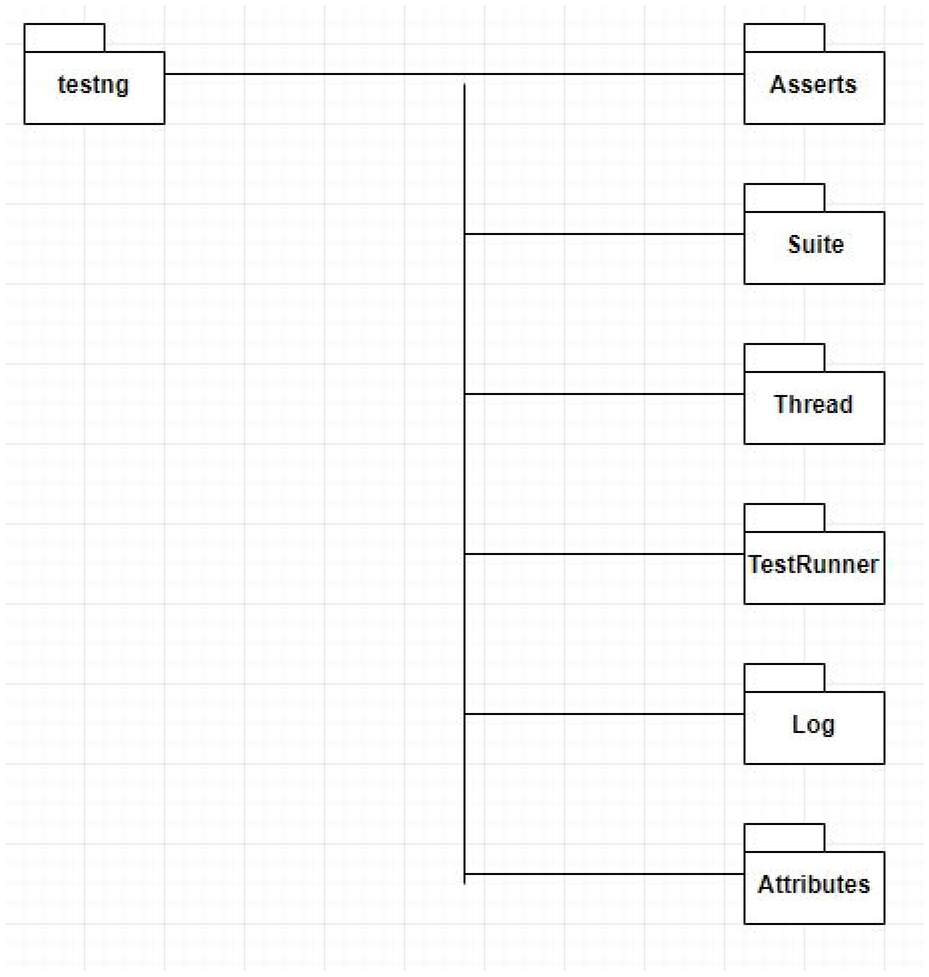
Team Silver
Benjamin Trettin
Christian Shinkle
Kendall Berner
William Fuhrmann

Structural Diagram



Code Smell Report

PMD:

After running the PMD test, we have been able to identify a total of 936 problems found. Looking through the report, we found many of the problems were similar but in different locations throughout the code. Some minor problems that the PMD found that we did not see as a serious or necessary issue to address was that of java code conventions such as using proper if()....else() blocks. Another minor error that came up dozens of times was

We have come up with a list of more serious code smells that we would like to further address and see if they need to be fixed.

Code Smell: Empty Method Body
Code Involved: Many methods in testng2\src\main\java\org\testng\TestNG.java and Users\cshin\git\testng2\src\main\java\org\testng\internal\ClonedMethod.java
Why it is a problem: We still need to look into the reasons why these empty methods are being created and what purpose they hold given that they are empty. The problem is it seems to be redundant and if these methods are there for a purpose, then there should be a comment at the beginning of the class to better understand their purpose.

Code Smell: Avoid reassigning parameters
Code Involved:testng2\src\main\java\org\testng\AssertJUnit.java and testng2\src\main\java\org\testng\internal\annotations\JDK15TagFactory.java as well as spread throughout many other classes individually.
Why it is a problem: Reassigning values to incoming parameters is not recommended. Using temporary local variables instead would help with memory leak. The problem has enough instances(approximately 21) that the issue should at least be looked at.

Code Smell:A class which only has private constructors should be final
Code Involved: testng2\src\main\java\org\testng\log4testng\Logger.java and testng2\src\main\java\org\testng\internal\PackageUtils.java as well as a few others.
Why it is a problem: A class with only private constructors should be final, unless the private constructor is invoked by a inner class.

Code Smell: Overridable method 'initTestClassesAndInstances' called during **object construction**

Code Involved:testng2\src\main\java\org\testng\reporters\EmailableReporter2.java and most of the problems come from this particular class.
Why it is a problem: Calling overridable methods during construction poses a risk of invoking methods on an incompletely constructed object and can be difficult to debug. It may leave the sub-class unable to construct its superclass or forced to replicate the construction process completely within itself, losing the ability to call super(). If the default constructor contains a call to an overridable method, the subclass may be completely uninstantiable.

CPD:

The CPD report we used had a token minimum of 100 in order to really see the duplicated code that was more seriously needing fixed. After our CPD report, we found very few cases that indeed actually had duplicated code with more than 100 tokens.

```
Found a 21 line (171 tokens) duplication in the following files:
Starting at line 1090 of
C:\Users\cshin\git\testng2\src\main\java\org\testng\Assert.java
Starting at line 1121 of
C:\Users\cshin\git\testng2\src\main\java\org\testng\Assert.java
```

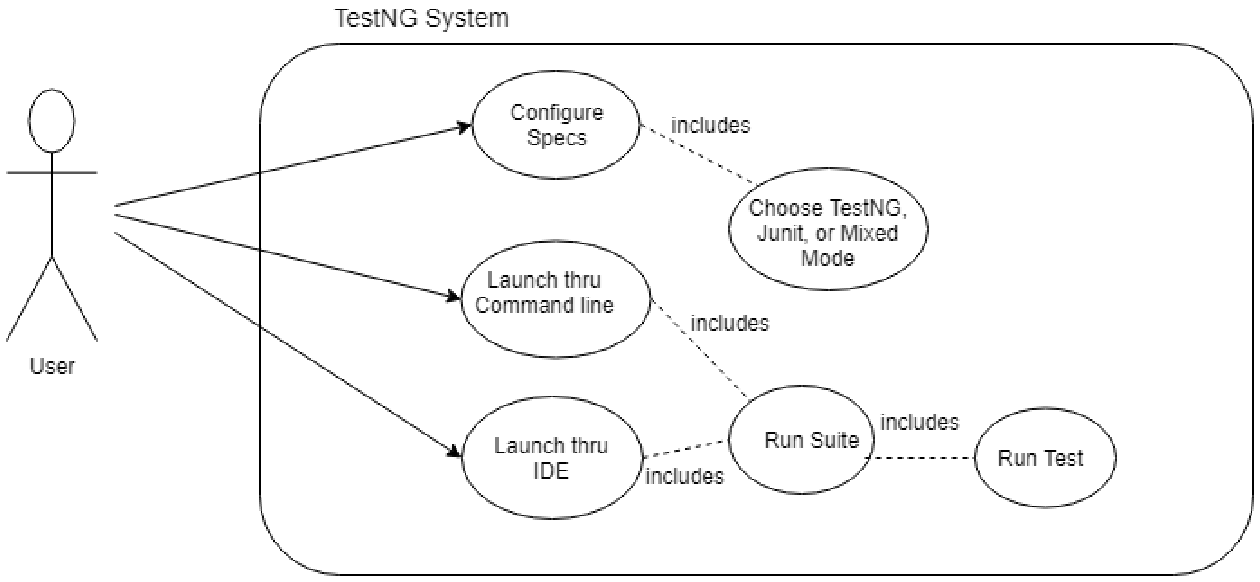
The 171 tokens that he is duplicating is the first main chunk of two assertEqual methods. The only difference between the two is the second method is an assertEquals Deep. What he could do in order to fix this issue is create a helper method that runs these 171 tokens and has both his two methods call that helper method at the beginning.

The second and last duplicated code that the CPD program found was a little under 20 lines.

```
Found a 17 line (108 tokens) duplication in the following files:
Starting at line 70 of
C:\Users\cshin\git\testng2\src\main\java\org\testng\reporters\jq\SuitePane
l.java
Starting at line 200 of
C:\Users\cshin\git\testng2\src\main\java\org\testng\reporters\JqReporter.j
ava
```

This code appears to be in a similar boat as the first CPD, in that there is some duplicated code within two methods that could potentially be reduced down to a helper method. The one glaring issue with doing that with this code though, is that the two method are not in the same class or even in the same package. With this little of duplicated code, in may not be necessary to create a helper method or even viable with how the packages are set up.

Use Case Diagram



Detailed Level Scenarios

Writing a test is typically a three-step process:

1. Write the business logic of a test and insert TestNG annotations in your code.
2. Add the information about a test (e.g. the class name, the groups you wish to run, etc...) in a testng.xml file or in build.xml.
3. Run TestNG.

1. User runs a testing suite through specifications in a xml file.The user can specify the package name instead of the class names. The user can also specify groups and methods to be included or excluded. The user can also define new groups inside testng.xml and specify additional details in attributes, such as whether to run the tests in parallel, how many threads to use, whether you are running JUnit tests, etc…

2. User runs a test file through ant. In this scenario, the user runs a test always through forked JVM. Many attributes can be assigned, but ant requires one of classpath, classpathref or nested <classpath> to be used to run a test. The user can specify the package name instead of the class names. The user can also specify groups and methods to be included or excluded. New groups and attributes can also be specified.

3. User runs a test through the command line. The user uses the command java org.testng.TestNG testng1.xml [testng2.xml testng3.xml ...] to invoke testng. The user needs to specify at least one XML file describing the TestNG suite you are trying to run. Additionally many command-line switches are available, but the command line is the simplest way to invoke a test, because you are unable to specify parameters.

4. User runs a test through eclipse. The user creates a TestNG class with TestNG annotations and/or one or more testng.xml files and creates a TestNG Launch Configuration. The user then Selects the Run menu and create a new TestNG configuration. The user then launches the test in 3 different ways: from a class file, groups, definition file, or a method.

5. TestNG mode. The TestNG mode gets applied when tests are passed to TestNG using classfilesetref, methods or nested <classfileset> and tells TestNG what kind of tests it should look for and run:
    - "testng": find and run TestNG tests.
    - "junit": find and run JUnit tests.
    - "mixed": run both TestNG and JUnit tests.

6. User converts Junit tests into TestNG tests. The users uses quick fix function to have TestNG automatically convert Junit tests to TestNG.

7. User uses Conversion refactoring. The refactoring wizard contains several pages that let the user generate a testng.xml automatically. The user can specify by packages or individual classes as well as decide to exclude certain files from the refactoring.

8. User runs a test suite. The user creates a series of test files files and groups the files together into a suite. The user then runs the suite and receives that output grouped together.

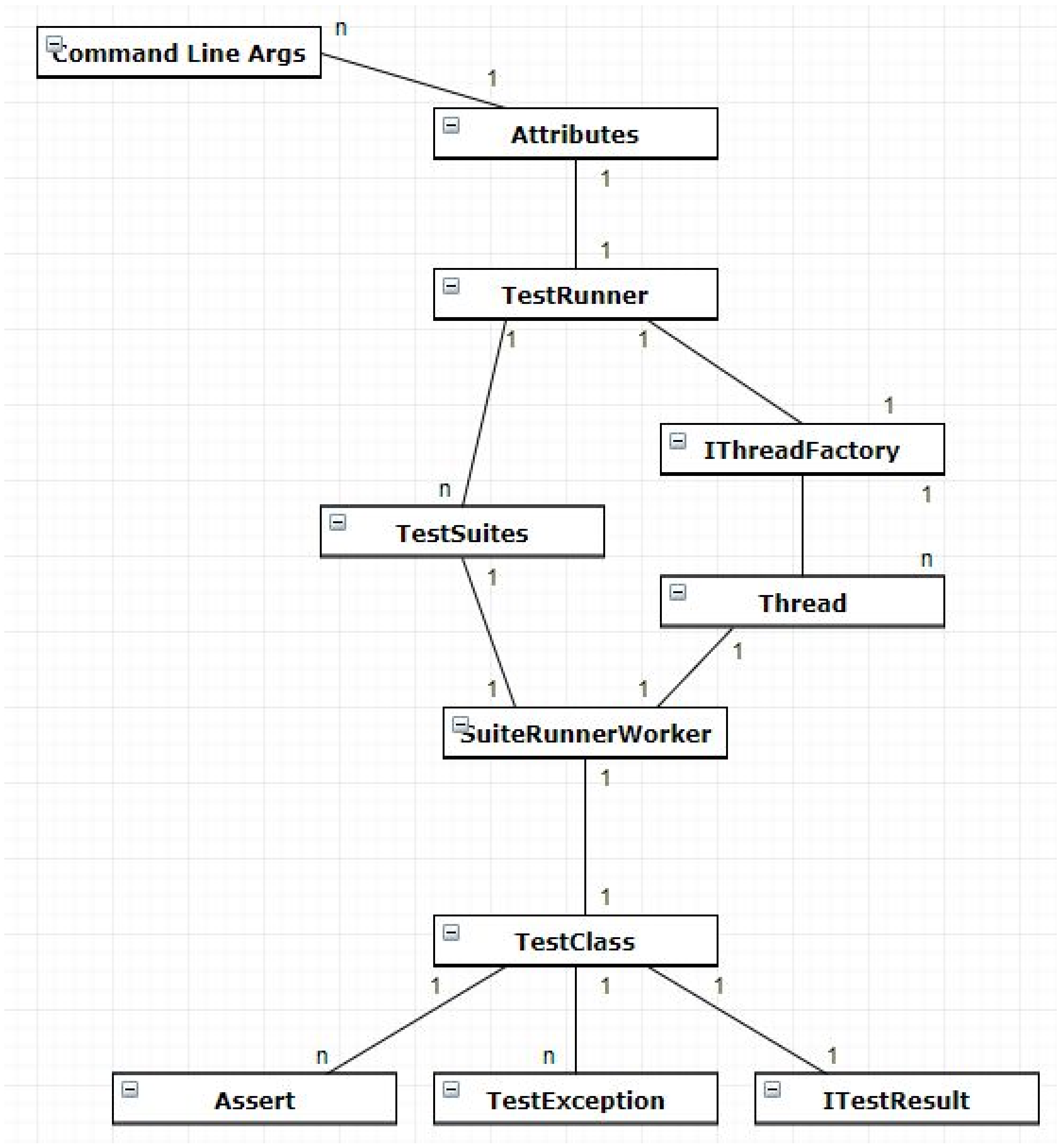
9. User writes a method using the @Before annotation that runs before each test in their suite. The user then supplies multiple tests with the @Test annotation.

10. User runs a single test that calls a function and compares its output to the desired output using methods with the @Test annotation.

11. User organizes tests with a group. The user defines a set of tests to belong to the same group. The user then can run the all the tests in a group.

12. User defines parameters. The user defines parameters to be passes into a method in xml file. The user then uses @Parameter annotations to enforce parameters of specified methods.

Domain Model



Tabular Summary

- Coverage: 68%
- Lines: 17,813
- Classes: 431
- Packages: 24
- Test Classes: 319
- Test Cases: 1641
- Cyclomatic Peak: 232 (Invoker)