Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Pooja Rani

Solution
Assignment 08 — 04.11.2020 – v1.0
Code and Test Smells

Please submit this exercise by email to pascal.gadient@inf.unibe.ch before 11 November 2020, 10:15am.

You must submit your code as editable text, i.e., use plain text file(s).

For the second part of this exercise, we use the latest GT release from here. If you already imported some Moose (MSE) models from the previous assignments into your GT environment, please start with a fresh copy of GT to reduce the strains on your computer's random access memory.

First, we have to download and extract the *Weka* dataset, and second, we need to import it into GT. We can perform both tasks using GT's Playground. Be warned: this process will take several minutes depending on your device's CPU and internet connection. We strongly advise you to save the image when the process succeeded to avoid redoing these steps.

The datasets can be downloaded and extracted with the following script:

```
targetFolder := (FileLocator imageDirectory asFileReference / 'models')
ensureCreateDirectory.
archiveFileName := 'weka-3-8.zip'.
archiveUrl := 'https://dl.feenk.com/moose-tutorial/weka/'.
ZnClient new
  url:  archiveUrl, archiveFileName;
  signalProgress:  true;
  downloadTo:  targetFolder.
(ZipArchive new
  readFrom:  targetFolder / archiveFileName)
  extractAllTo:  targetFolder.
```

The sample dataset can be imported with the following script:

```
modelFile := (FileLocator imageDirectory asFileReference / 'models')
  / 'weka-3-8'
  / 'weka-3-8.mse'.
modelWeka := MooseModel new
  importMSEFromFile:  modelFile.
```

Exercise 1: Code smells (4 pts)

a) Choose two different code smells and answer the following questions for each of the smells: Which code smell did you chose? What are its characteristics? What is the resulting problem? (2 pts) **Answer:**

There exist numerous code smells and all of them eventually lead to high technical debt. We list four of them:

- ***Class Data Should Be Private**: A class exposing its attributes, violating the information hiding principle.* Problem: *When classes expose more attributes than necessary, information leaks or inappropriate value changes might occur. In addition, the technical complexity increases, e.g., every attribute can become referenced at any location in the code.*
- ***Complex Class**: A class having high cyclomatic complexity.* Problem: *The higher the complexity, the more code paths exist through the class. The more code paths exist, the harder it is for developers to fully understand (and test!) the class.*
- ***Functional Decomposition**: A class where inheritance and polymorphism are poorly used, declaring many fields and implementing few methods.* Problem: *The domain model does not adequately reflect the real world model. Hence, it is cumbersome for developers to maintain the code, because they have to work around many dependencies and think in a counter-intuitive way.*
- ***Spaghetti Code**: A class without a structure that declares long methods without parameters.* Problem: The lack of parameters unevitable leads to shared state (variables) within a class. If methods become longer, they usually use more of those shared variables without clear boundaries. As a result, it is particularly hard to make minor adjustments in such classes, because they might impact the whole class.

b) What is the fundamental problem in developers' code smell perception? (1 pt) **Answer:**

The perception is a subjective matter. Hence, it is very difficult for developers to reach an agreement regarding the "smelliness" of code. The perception is even highly context dependent, e.g., *an internal application for cleaning log files is in general less exposed to security threats than web endpoints, thus some smells might become neglected. Moreover, different programmers have different programming habits shaped by their culture, and previous experiences. Hence, a code smell in one project might not be considered as a code smell in another project. Anaïs Nin sums it up quite nicely: "We don't see things as they are, we see things as we are."*

c) What is *association rule mining* in the context of the HIST code smell paper you can find here? (1 pt) **Answer:**

More general, association rules are the correlations between elements that co-occur frequently within a dataset consisting of multiple independent selections of elements. In the context of the HIST paper, the association rules represent correlations between subsets of methods in the same class that frequently change together. The discovery of all association rules within one or more projects is then called "association rule mining".

Exercise 2: Test code smells (3 pts)

a) Choose one test code smell (except "Eager Test") and answer the following questions:
Which test code smell did you chose? What are its characteristics? What is the resulting problem?
(1 pt) **Answer:**

There exist numerous test code smells and all of them eventually lead to high technical debt. We list three of them:

- ***Empty Test**: A test without any code.* Problem: *Such tests clutter the results and do not have any purpose. Hence, they should be removed.*
- ***Ignored Test**: A test that is ignored either during or after execution time.* Problem: *Such tests should be revised or removed, because they only add dead code to the project.*
- ***Sleepy Test**: A test that lets the testing thread sleep, e.g., invoking `Thread.sleep()`.* Problem: *A delayed testing thread can cause several problems. Most prominently, it delays the execution of all remaining tests making it unfeasible to execute them in a timely manner. In addition, such tests that rely on platform-dependent features might yield different results on different platforms.*

b) Considering the code below, report in which line you can find the "Eager Test" code smell and explain why it represents a problem. (2 pts)

```
01:  public void testDataIsVariable() throws Throwable {
02:    JSTerm term = new JSTerm();
03:    term.makeVariable();
04:    term.add((Object) "");
05:    jSTerm0.matches(jSTerm0);
06:    assertEquals(false, term.isGround());
07:    assertEquals(true, term.isVariable());
08:  }
```

Answer:

The "Eager Test" code smell can be found in line 07. This test uses two different assert statements each testing a different method within the same test case. This complicates not only the implementation of the test, it also misleads developers when the test breaks due to the inappropriate test name, and finally, it makes it harder to fix code that breaks such a test because it is not obvious what the cause of the failure is. Consequently, a single test should use a single `assert` statement on one specific method call of the class under test.

Exercise 3: Detection of eager tests (3 pts)

Your task is to extract all JUnit 3 tests from `modelWeka` that suffer from the "Eager Test" code smell. That is, you have to find every method with `#isJUnit3Test` set to `true` that contains an assertion statement at least two times. **Answer:**

```
tests := modelWeka allModelMethods select:  #isJUnit3Test.
eagerTests := tests select:  [ :  m |
  | asserts astNode |
  asserts := OrderedCollection new.
  astNode := m gtASTNode.
  astNode
    ifNotNil:  [ astNode
      allNodesOfType:  JavaMethodInvocationNode
      do:  [ :node |
        (node name value beginsWith:  'assert')
          ifTrue:  [ asserts add:  node ] ] ].
  asserts size > 1 ].
```