CS 471: Software Engineering
Spring 2018
Homework 5 – Testing

Due date: Wednesday, April 25, 2018 (at the end of day)

1. Objectives

Part 1: This homework assignment introduces the use of `maven` for building a large, existing project, `ical4j`, and other software tools to evaluate the effectiveness of `ical4j`'s test suite.

Part 2: You will also create two defect removal model to estimate the delivered defect density based on some provided assumptions.

Part 3: You will improve a defect removal model by including a different defect removal activity.

All three parts of the assignment are completely independent of one another.

Part 1

2. Resources

Some useful resources:

- `maven` (http://maven.apache.org) is a widely used builder for server-side development in industry
- Code Coverage (`EMMA`) (http://emma.sourceforge.net/) is an advanced topic, beyond the current practice of some industry teams. It provides an approach to estimate the coverage of your test suite.
- Mutant Analysis (`PIT`) (http://pitest.org/quickstart/maven) is well beyond current industry practice. I want you to be prepared for a more competitive future. In this exercise, you will use PIT to determine the effectiveness of `ical4j`'s test suite by injecting defects into the `ical4j` bytecode and measuring the test suite's ability to find those injected defects. You will discover the painful weakness of a real-world test suite.
- `cloc` (http://cloc.sourceforge.net/) is a tool for counting lines of code

3. Procedure

Carefully follow the procedure below! Building and modifying a large, existing program can be challenging/frustrating/time-consuming if you stray from the known-working path. The steps below have been optimized/simplified to reduce as much as possible many of the pitfalls you are likely to encounter. You will not find help in the textbook, and will have to rely upon the lecture, google searches, Piazza posts and the references above to overcome difficulties. This procedure was tested on onyx, OSX and Windows 10-64 bit (your experience may vary on other platforms).

4. Familiarize with Report file

Download the report file `CS471_S18_HW5_Report_Template` file from Piazza and use it throughout this homework assignment to record values and statistics.

5. Download ical4j

Download the `ical4j.zip` file from Piazza and unzip it to a folder `<ical4jFolder>`. (Your build experience may vary significantly if you download a different version and you will get points deducted during grading)

6. Ensure Java 8 and maven is installed

Install `maven` if it is not already installed on the system.

- You can use the following console command to see if `maven` is properly installed

```
mvn -v
```

- Assignment was tested using Java 8, `maven` 3.1.1 and 3.3.9
- NOTE: Java 9 does not work for this assignment

7. **Perform a clean build**

Using a terminal command window, change to your `<ical4jFolder>` (containing `ical4j`'s `pom.xml` file) and generate a "clean" build of `ical4j` with the command:

```
mvn clean install
```

The build will likely emit terrifying warnings that you can ignore if `maven` successfully created `<ical4jFolder>/target/ical4j-1.0.6.jar` with a size of 933,453 bytes (on onyx and Windows 10).

8. **Run EMMA test coverage**

Only after you have a "clean" build, collect the initial code coverage data using the command:

```
mvn emma:emma
```

Examine the generated EMMA test coverage report in

```
<ical4jFolder>/target/site/emma/index.html
```

Record in the report file the overall coverage of all code blocks. Note the test suite exercised less than half of the product's code blocks!

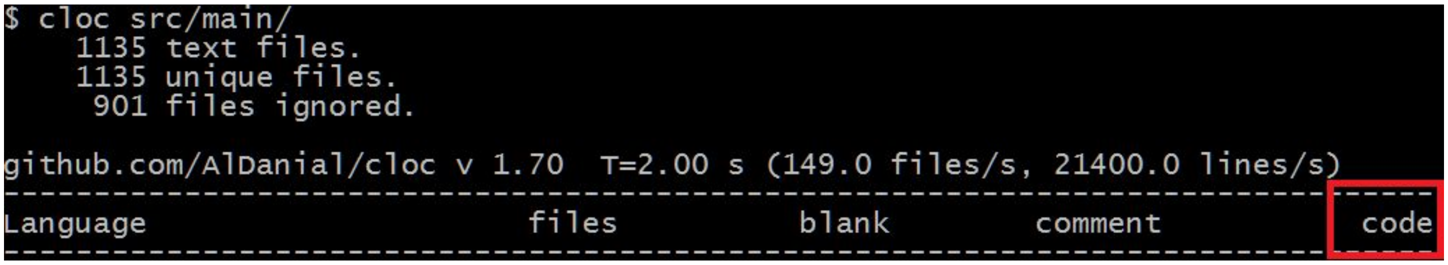
Take a screenshot of your browser displaying the EMMA results and record it in the report file.

9. **Compute ratio of test to product lines of code using cloc**

Use `cloc` (http://cloc.sourceforge.net/) to count the number of lines of **product code** and **test code** in `ical4j`. Here are a few things to keep in mind:

- Maven projects are organized with
  - o the **product code** under the `src/main` folder and
  - o the **test code** under the `src/test` folder
- The `ical4j` project uses two programming languages, namely Java and Groovy (i.e., a scripting language based on Java). You will need to count the number of lines of code in the `*.groovy` and in the `*.java` files, and `cloc` can help with that.
- Use the following commands from inside `<ical4jFolder>` and examine the right-most column name called "code" (see red rectangle in the image below):

```
cloc src/main
```
and
```
cloc src/test
```



```
$ cloc src/main/
    1135 text files.
    1135 unique files.
     901 files ignored.

github.com/AlDanial/cloc v 1.70  T=2.00 s (149.0 files/s, 21400.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
```

Calculate the ratio of **test code**/**product code** and record in the report file. Note that the ratio is nearly 50%, a value often suggested as being the minimum required.

10. **Run the PIT mutator**

Run the PIT mutator using the following command, which (i) **builds**, (ii) **mutates the bytecode** and (iii) **executes the tests**:

```
mvn clean install org.pitest:pitest-maven:mutationCoverage
```

It could take 15-30 minutes to run, but the screen should be constantly updated with "cryptic" messages while running.

Unlike many mutators, PIT mutates the bytecode rather than the source code. This means that PIT can mutate the binaries from both Java and Groovy as both compile into bytecode.

PIT will place the results of the mutated tests in a time-stamped folder located in `<ical4jFolder>/target/pit-reports/<TimeStampedFolder>/index.html`. Record the overall "Mutation Coverage" percentage for the `ical4j` project in the report file.

- This value represents the fraction of mutations injected into the bytecode by PIT that were

discovered by the test suite.
    - Fun fact: in the mutant world, a "killed mutant" refers to an injected defect that was found by the test suite.
- Note that, although the test suite executed a significant fraction of the product code, it failed to discover a majority of the mutations.
- Running the PIT analysis multiple times may generate different results, because it is a stochastic process.

Take a screenshot of your browser displaying the results of your mutation testing and record it in the report file.

Troubleshooting: If the previous PIT command was not successful, you can try to edit `ical4j`'s `pom.xml` file before running the PIT mutator command again. The following `xml` snippet should be added under the `<plugins>` element in the `pom.xml` (refer to http://pitest.org/quickstart/maven for more information).

- The `<targetClasses>` element specifies which classes should be mutated. In `ical4j`, these classes are `net.fortuna.ical4j.*`

```
<plugin>
    <groupId>org.pitest</groupId>
    <artifactId>pitest-maven</artifactId>
    <version>1.2.0</version>
    <configuration>
        <targetClasses>
            <param>net.fortuna.ical4j.*</param>
        </targetClasses>
    </configuration>
</plugin>
```

11. Comparison of three instruments of the effectiveness of the test suite

Note that you now have the following three instruments of the effectiveness of this test suite:

- `EMMA` test coverage (Section 8)
- Ratio of test to product lines of code (Section 9)
- `PIT` mutation coverage (Section 10)

Discuss in the report file which of these three instruments you think is the most accurate in determining the quality of the test suite. Justify your answer.

Part 2

12. Generate a Defect Removal Models: DRM1 (Unit Tests) and DRM2 (Unit Tests + Beta Testing)

NOTE: Part 2 is independent of the `ical4j` project discussed in Part 1.

Generate a Defect Removal Model, called `DRM1`, and discuss its characteristics in the report file. `DRM1` should estimate the delivered defect density using the following assumptions:

- Assume that development activities injected 35 defects/KLOC into the product source code.
- Assume that no other defect removal activities were performed within the project, other than the unit tests appearing in its test suite, which have an effectiveness of 30%.
    - You will lose points if you include additional defect removal activities in DRM1

Generate a Defect Removal Model, called `DRM2`, and discuss it in the report file. `DRM2` should estimate the delivered defect density using the following assumptions:

- Assume that development activities injected 35 defects/KLOC into the product source code.

- Assume that the development used the unit tests appearing in the project test suite to remove defects which have an effectiveness of 30%.
- In addition to the unit test, the project was also beta tested with a defect removal effectiveness of 60% (this is not unreasonable for an open source product that has been released for several years).
    - You will lose points if you include additional defect removal activities in DRM2

Part 3

13. Improve a given Defect Removal Model

NOTE: Part 3 is independent of Part 2.

Assume that you are a developer working on a project. Your manager has assigned you the responsibility for creating a plan to improve the quality of the project. You may assume that the existing defect removal model of the project includes a combination of:

- unit-level,
- integration-level and
- system-level (e.g., Acceptance Tests) functional tests.

Identify the most cost-effective activity you plan to use for the removal of defects from the existing code, and explain why you chose this activity.

14. Homework submission

Submit via Blackboard (see `HW5Testing` assignment) a single pdf file named `CS471_S18_HW5_[LastName].pdf`, based on the `CS471_S18_HW5_Report_Template` template.

15. Grading Rubric

The maximum points for this homework representing 5% of the final grade is 100, and the points are distributed as follows:

Item (see CS471_S18_HW5_Report_Template file)	Points
Part 1	
Overall coverage of all code blocks after running `EMMA`	5
Screenshot of "`EMMA` Coverage Report"	5
Ratio of (Number of test lines of code) / (Number of product lines of code)	10
Overall "Mutation Coverage"	10
Screenshot of "Pit Test Coverage Report"	5
Comparison of three instruments of the effectiveness of the test suite	10
Part 2	
Discuss the Defect Removal Model DRM1	13
Estimate the delivered defect density using DRM1	5
Discuss the Defect Removal Model DRM2	13
Estimate the delivered defect density using DRM2	5
How does DRM1 compare with DRM2?	4
Part 3	
Discuss your plan for improving a given Defect Removal Model	15