**CS4218 SOFTWARE TESTING AND DEBUGGING Spring 2015**

**QA Report**

Team: TeamMix

Member: Nguyen Trong Son A0088441B  
 Fam Wei Jian A0097935L

Melvin Lee A0097719L

**Topics**: Comparison of testing strategies, measures of confidence, Orthogonal

Defect Classification (ODC).

**Analysis across project artifacts and milestones**

**Project artifacts:**

1. Unit test cases.

2. TDD test cases.

3. Integration test cases + Randoop test cases + 100% method coverage.

4. Hackathlon test cases + bug reports + incremental coverage.

5. Unit test coverage vs. integration test coverage. Incremental coverage.

*Please answer the fol lowing questions. Please provide numbers, plots and expla- nations 2-3 lines of text for each question.*

**Report**

1. How much source and test code have you written?

Test code (LOC) vs. Source code (LOC).

Test code (6840 LOC) vs Source (2950 LOC)

2. Analyze distribution of fault types versus project activities:

2.1. Plot diagrams with the distribution of faults over project activities.

*Types of faults*: unit fault (algorithmic fault), integration fault (interface mismatch), missing functionality.

*Activity*: requirements review, unit testing, integration testing, hackathlon, coverage analysis.

Each diagram will have a number of faults for a given fault type vs. different activities.

**Discuss what activities discovered the most faults.**

Hackathlon discovered the most faults. Mostly the faults are found in Unit and integration testing as we have missed out a few corner cases which required special implementation to handle.

**Discuss whether the distribution of fault types matches your expectations.**

The distribution of faults matches are within our expectations. For example, in Integration faults, there no Integration faults found in the Unit testing activity because Unit testing tests the basic functionality of each classes without integrating with other components.

It is also observe in Unit faults that there are more faults found in unit testing than in integration testing because faults found in Unit testing would have been fix before integration testing.

2.2. Plot a diagram for distribution of faults found in basic functionality (old code) during activities on adding extended functionality (new code):

*Activity*: integration testing, hackathlon, coverage analysis.

**Discuss whether the distribution of fault types matches your expectations.**

The distribution of faults matches our expectations. Hackathlon and Integration Testing had the most faults found during the implement of extended functionality as more classes and functionalities work together. This meant more branches in code and also chances of fault to occur.

2.3. Analyze bugs found in your project according to their type.

Analyze and plot a distribution of *causes* for the faults discovered by

Hackathlon activity.

There are more unit faults found than Integration Faults found in Hackathlon.

Analyze and plot a distribution of *causes* for the faults discovered by

Randoop.

*Causes*: Error in constants; Error in identifiers; Error in arithmetic (+,-), or relational operators (<, >); Error in logical operators; Localized error in control flow (for instance, mixing up the logic of if- then-else nesting); Major errors (for instance, ‘unhandled exceptions that cause application to stop’).

**Is it true that faults tend to accumulate in a few modules?**

No. The faults that were found in Hackathlon were distributed evenly among the affected modules.

**Is it true that some classes of faults predominate? Which ones?**

Yes, the shell or rather the front end classes. As these classes are being called more than other classes.

3. Provide estimates on the time that you spent on different activities (per- centage of total project time):

Requirement analysis: 20%

Coding: 50%

Test Development: 10%

Test Execution: 20%

4. TDD vs. Specification-Based Testing.

What are advantages and disadvantages of both based on your project experience?

TDD allows implementation phase to be carried out more smoothly as it helps the programmer to ensure that the requirements are met. However, it does not help the programmer to find out corner cases which may cause a fault in integration testing.

Specification-Based Testing helps to find faults that may have been missed out by TDD. However, Specification-Based Testing has so many combinations it is hard to keep track of what is tested and what has not been tested.

5. Do coverage metrics correspond to your estimations of code quality?

In particular, what 10% of classes achieved the most branch coverage? How do they compare to the 10% least covered classes?

Provide your opinion on whether the most covered classes are of the highest quality. If not, why?

The 10% of most branch coverage class did have lesser fault found compared to the other 10% of least branch coverage. In overall, it is expected that the most covered classes are of the highest quality as test cases covered most of the paths. If there is a fault that occurs in those classes it is most likely due to missing functionalities, whereas those classes that are not so covered are likely to be missing functionalities and logical faults.

6. What testing activities triggered you to change the design of your code?

Did integration testing help you to discover design problems?

Integration testing triggered a few changes in design of code. It is because special characters such as "\" and "\*" has be specially taken care of so as to pass it properly to the Find and Sed applications.

7. Automated test case generation: did automatically generated test cases

(using Randoop) help you to find new bugs?

Compare manual effort for writing a single unit test case vs. generat- ing and analyzing results of an automatically generated one(s).

8. Hackathlon experience: did test cases generated by the other team for your project helped you to improve its quality?

Yes, the other team help to discover faults that our team have missed out.And thus allows us to improve on our code.

9. Debugging experience: What kind of automation would be most use- ful over and above the Eclipse debugger you used - specifically for the bugs/debugging you encountered in the CS4218 project?

Would you change any coding or testing practices based on the bugs/debugging you encountered in the CS4218 project?

We believe the coverage path automation is most useful as it shows us clearly which paths have not been tested.

Yes, we would definitely use TDD more in the future.

10. Did you find static analysis tool (PMD) useful to support the quality of your project?

Did it help you to avoid errors or did it distract you from coding well?

PMD did help to improve our coding standards but it is very distracting as code are constantly been flagged and refactoring needs to be done which may result in breaking the code.

11. How would you check the quality of your project without test cases?

By running the application trying out different inputs manually.

12. What gives you with the most confidence in the quality of your project?

Passing test cases that we generated as it assure us that we did not have regression errors.

13. Describe the one most important thing on testing that you have learned/discovered.

Passing many test cases does not mean bugs free, it only means bugs not yet found.

14. What answers/results in the questions above are counterintuitive to you?

Results in question 11. As it is too slow to find bugs and not feasible in a big project.

15. We have designed the CS4218 project so that you are exposed to industrial practices such as personnel leaving a company, taking ownership of other’s code, geographically distributed software development, and so on. Can you try to suggest an improvement to the project structure which can help us relate the project to industrial practices more tightly?