**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 following questions. Please provide numbers, plots and explanations 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 most faults. The faults are mainly found in Unit and integration testing as we have missed out a few corner cases, which required special conditions to handle them.

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

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

It is also observed 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 fronts end classes. As these classes are being called more than other classes.

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

Requirement analysis: 20%

Coding: 40%

Test Development: 20%

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, unless corner cases are also crafted for the programmer.

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?

The higher the coverage metrics, the lesser the faults in the program. This can be observed from our testing where the faults have been taken care of in areas with high coverage metrics.

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

The Shell, where all the user inputs have to be process there, has the most branch coverage. Some of the applications have lesser branch coverage because some branches can only be reached during a system exception. As paths of the branches have been covered at the Shell, it contains lesser faults during the hackathlon, as compared to those with lesser branch coverage at some of the applications.

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 be 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?

Both unit and integration testing have triggered our team to change some of the design, such as adding more conditions to satisfy certain corner and boundary test cases.

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?

Our team did not use Randoop to generate test cases automatically.

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

This can be subjective because experience tester can write effective and efficient test case to check the integrity of the program, while generating and analyzing results of an automatically generated one(s) can be exhaustive and may not yield the same result, based on randomization. However, the automatically generated test cases can be useful to check for areas that tester may miss out.

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

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

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

Randoop. It will automatically generate test cases to check the integrity of the program where tester like us may miss out. This can also help us to gain more experience of understanding what areas (that we have missed) to watch out for in the future.

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?

Yes. All of us have different coding style. By using PMD, it helps to standardized our coding style throughout the whole project, so if other teammates were to analyst our codes, it would be more readable and they are able to jump in quickly in the development and debugging process.

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. However, we have adopted the coding style and standards of PMD and we are more comfortable from then on.

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

By running the application and trying out different inputs manually with a text file which contains all the test cases. This can be very time consuming and prone to human errors.

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

Passing all test cases that our team and other team have generated as it assured us that we have meet the system requirements and did not have regression errors.

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

Passing all test cases does not mean bugs free, it only means bugs not yet found. This is also mentioned before by Edsger Dijkstra , “Testing can show the presence of errors, but not their absence.”

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

Result in question 11. It is very time consuming and prone to human errors 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?

During one of the lecture, Prof. Abhik Roychoudhury has mentioned that the first committed bug fixed from a fresh developer is usually the hardest. We suggest that there will be a set of test cases to be satisfied before the team is accepting the committed work.

Furthermore, User Acceptance Testing (UAT) can be included in the project as well as it plays a major role in the industry. This will require a user who understand the basic shell functionalities but not involve in this module to verify that the solution (shell system) works for the user.