QA Report

1. **How much source and test code have you written? Test code (LOC) vs. Source code (LOC).**

   Source: 3793 LOC

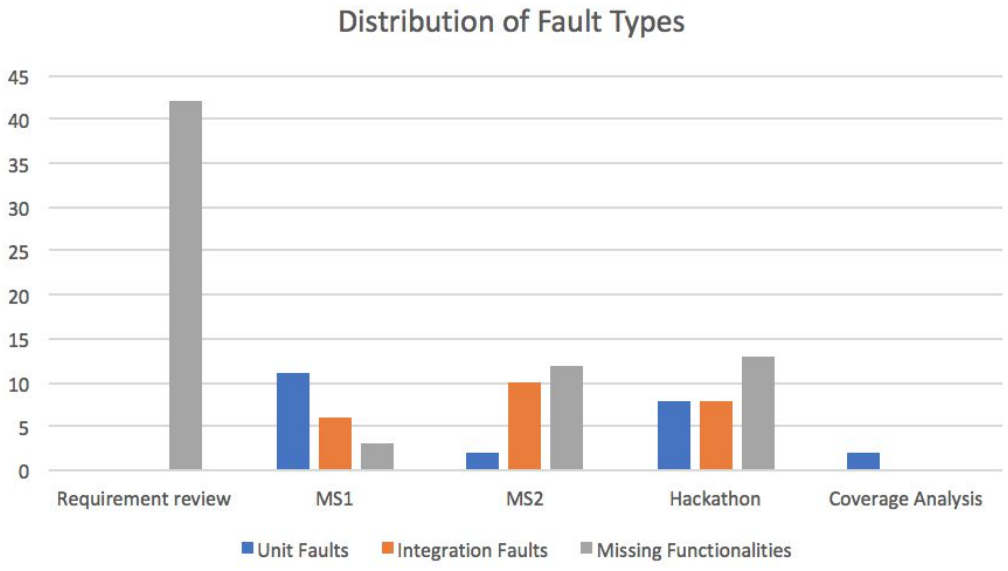
   Test: 4924 LOC

2. **Analyze distribution of fault types versus project activities:**

   Instead of using unit testing and integration testing phase, we will use milestone 1 (MS1) and milestone 2 (MS2) instead where most unit testing is done in MS1 and integration testing is done in MS2.

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

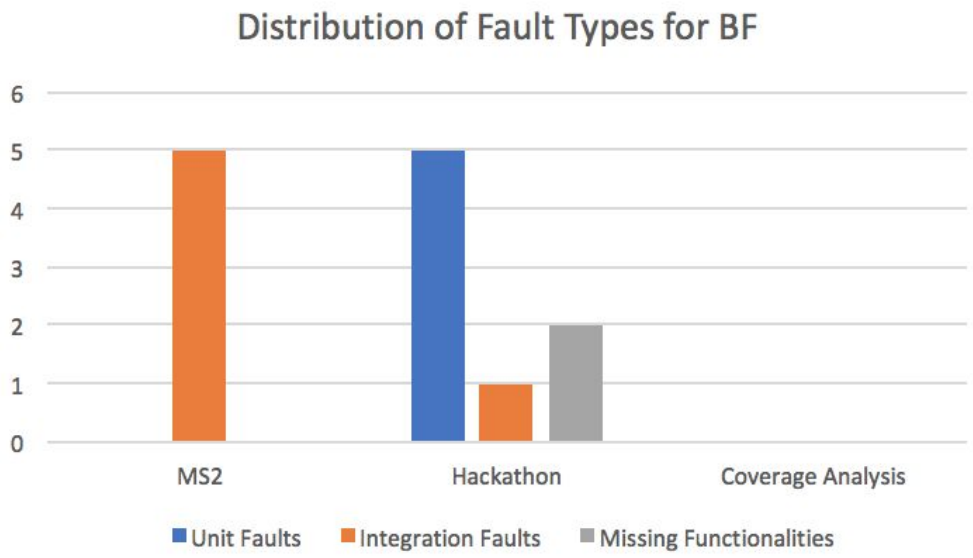
   Types of faults: unit fault (algorithmic fault), integration fault (interface mismatch), missing functionality.
   Activity: requirements review, unit testing, integration testing, hackathon, coverage analysis.
   Each diagram will have a number of faults for a given fault type vs. different activities. Discuss what activities discovered the most faults. Discuss whether the distribution of fault types matches your expectations.



We combined the diagrams to have a clearer comparison as we can now have side-by-side comparison. The distribution of fault types matches our expectation the initial stage where requirement review has most number of faults due to ambiguous requirements. Next, we have more unit faults in MS1 as we have not done most of the integration works and in MS2

we have more integration faults as that is where we did more integration of the project and also have TDD test cases where we realise some functionalities that we are missing. During the hackathon phase, number of faults is fewer as expected. Although missing functionalities are higher in this phase, it is still within expectation due to the different interpretations by different teams on the requirements. Lastly, during the coverage analysis, it is expected to have low number of faults as we should have correctly implemented most features/requirements.

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, hackathon, coverage analysis.
Discuss whether the distribution of fault types matches your expectations.

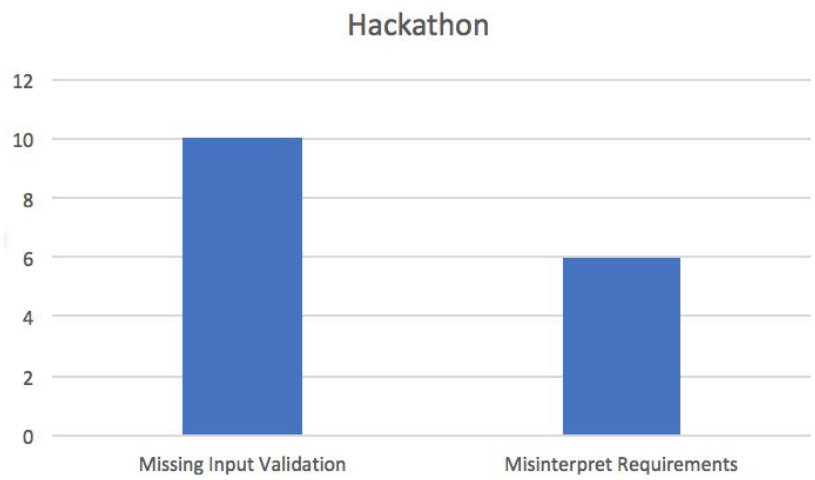


We were expecting the number of faults to decrease across the phases as the BF should be completed by hackathon. However, there is an increase in faults during the hackathon phase. This could imply that there might not have enough test case in MS1 to catch those faults.

2.3. Analyze bugs found in your project according to their type.
Analyze and plot a distribution of causes for the faults discovered by Hackathon activity.
Analyze and plot a distribution of causes for the faults discovered by Randoop, or other tools.

As we do not have any faults discovered with the use of PICT, we did not include the distribution of faults discovered by PICT. Also, as most of the default causes are not applicable to us, we will define our own causes.

Causes: Missing input validation (e.g size > 0), Misinterpret requirements



Is it true that faults tend to accumulate in a few modules?
Is it true that some classes of faults predominate? Which ones?

Faults were expected to and did accumulate in modules that are very large and have more logic component like the Shell.
The class of faults that predominated was missing input validation, such as an empty string was interpreted as an invalid app and threw an exception where it should have been discarded, and missing checks on zero/negative values when a positive value was expected.

3. **Provide estimates on the time that you spent on different activities (percentage of total project time):**
   Requirements Analysis: 15%
   Coding: 35%
   Test development: 40%
   Test execution: 10%

4. **Test-driven Development (TDD) vs. Requirements-driven Development. What are advantages and disadvantages of both based on your project experience?**

   The advantage of TDD is that a detailed specification is precisely defined through the test cases, hence ensuring better coverage of all use cases and edge cases, resulting in less bugs. Furthermore, defects in implementation are caught immediately and the faults are made obvious by the failed test cases, reducing the need for the use of debugger tools and speeding up development.
   However, it takes significant amount of time to write a test suite with good coverage, and much of the effort will be wasted if the requirements were to change constantly as some of the test cases would have to be rewritten. Hence it's more suitable for projects adopting the waterfall

model where requirements and scope are fixed (similar to this project).

The advantage of Requirements-driven Development is that the application functionalities will be developed first, allowing the users/clients to test the product as it develops, which is beneficial in an agile model or when the requirements are not clearly defined and subject to rapid changes.
However, as the bugs were not caught early on, it may be costly to debug and fix them later on, as it becomes harder to locate the faults and change the design/implementation to properly fix the bugs as the application complexity grows.

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?**

   No, the coverage metrics did not necessarily correlate to code quality. LsApplication and GrepApplication had the highest branch coverage of 95% whereas CatApplication had the lowest branch coverage of 44.4% but its quality was not much worse compared to the classes of highest coverage. For example, there are total of 5 bugs discovered within GrepApplication while there's only 2 bugs discovered within CatApplication during the hackathon.

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

   It is not necessary that the most covered classes have the highest quality.
   The class implementation and test cases may not have covered the necessary functionalities and boundary cases, hence there may be bugs even with high coverage.
   Therefore, the unit tests must first be constructed carefully to cover all functionalities and boundary cases as much as possible in order for the coverage to reflect the quality of the classes.
   Moreover, code quality also includes readability, maintainability and efficiency, which are not reflected in code coverage metrics. Hence the design and implementation of the classes may have greater impact on the quality.

6. **What testing activities triggered you to change the design of your code? Did integration testing help you to discover design problems?**

   Integration testing has shown that many of the apps do not account for a change in directory by the `cd` command as it still reads files from the original working directory. The implementation of each app has to be changed to read from the global variable which can be modified by the `cd` app instead of the actual working directory defined by the OS as the directory in which the program is ran in.

   Black-box testing triggered some changes in the design of the code. For example, originally the appending of a newline character to each command output was handled at the shell layer. We have discovered through blackbox testing that our `cat` application does not match the behavior

of the Linux implementation, as unlike most applications like `echo`, the Linux implementation of `cat` does not automatically append a new line to the output. Hence the appending of a newline character to the output was changed to be handled by each application individually instead.

7. **Automated test case generation: did automatically generated test cases (using Randoop) help you to find new bugs?**

   Since we are unsure of the correctness of each application, Randoop might generate false positive test cases that are not valid. Therefore, we use Randoop to help us with regression testing instead, hence regression was reduced during development.

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

   The use of Microsoft PICT helped us to identify pairwise combinations to test our shell functionalities which would have been more difficult to enumerate manually. Although the test cases had to be written instead of automatically generated, this was still helpful as it reduces the possibility of leaving out certain test cases.
   In terms of readability, manual effort allows us to name the method appropriately compare to automatically generated one, making it easier to identify the purpose of the test cases and the faults when the test cases failed. On the other hand, the test cases generated by Randoop made it hard to understand what exactly went wrong or what is being tested, making it hard to analyse and uncover the faults.
   Moreover, Randoop's random testing nature might provide us with redundant test cases within the same equivalence partition which can actually be covered by a representative test case or boundary values.

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

   Yes, the hackathon has helped to identify some assumptions that we made which we had thought was obvious. It has also helped us to identify some corner cases which we did not cover.

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?**

   Our group uses Travis CI as part of our testing pipeline to ensure all tests passed within the same environment configuration. These tests are automatically executed for each commit and upon submitting a pull request, thereby ensuring that the proposed changes do not result in any regression, as failing test cases can be identified and fixed before the pull request is merged.

   Would you change any coding or testing practices based on the bugs/debugging you

encountered in the CS4218 project?

We could have used Randoop more extensively in our regression testing. A Randoop test suite can be generated once we verified the correctness of the application. Our group should also focus more on unit testing the methods and interfaces by providing stubs/drivers when the implementation is not available.

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 has helped to some extent in terms of code quality such as standardising the use of curly braces and warnings on very long methods. However, it has at times raised warnings on duplicate literals which if extracted into constants can make it more difficult to understand certain test cases, as well as preserve stack trace warnings which were not very applicable to the project, and method naming warnings when trying to use a different test method naming convention.

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

As we have developed our project using object oriented programming paradigm, we can make use of O-O design metrics suites to measure the quality of our project without the need of test case. One such metric is the MOOD framework of which it has 6 different metrics to measure the project quality. They are Method Hiding Factor, Attribute Hiding Factor, Method Inheritance, Attribute Inheritance Factor, Polymorphism Factor and Coupling Factor. After calculating these metrics we are able to find out the quality of the projects between the classes and methods.

Other ways would be to check if the project follow the SOLID principles such as not having repeated code and have high cohesion and low coupling.

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

Our test coverage gives us the most confidence in the quality of our project as we have designed the test cases carefully to cover all the functionalities and boundary cases as much as possible.

13. Which of the above answers are counter-intuitive to you?

We do not have any answers that are counter-intuitive.

14. **Describe one important reflection on software testing or software quality that you have learnt through CS4218 project in particular, and CS4218 in general.**

Software testing is integral to software development and requires a lot of resources to achieve a high standard/coverage. It is important to balance the costs of implementation and testing. Requirements are often not clear, hence further clarification should be made and assumptions

on features that are not specified should be explicitly stated.

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. Please try to suggest an improvement to the project structure which would improve your testing experience in the project?**

    It could be more efficient for requirements analysis and testing if there was more emphasis on applications as modification to the shell-related code is complex and could very easily break the program which made it take a much longer time to implement or fix bugs compared to applications. Some applications could already be implemented but containing bugs to allow us to perform requirements-driven testing in MS1 and re-design/fix the code, and then leading into MS2 for implementing unimplemented functionalities using TDD.