CS4218 SOFTWARE TESTING AND DEBUGGING Spring 2014

QA Report

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 added test cases.

3 Integration test cases + bug reports + 100% method coverage.

4 Hackathon 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.

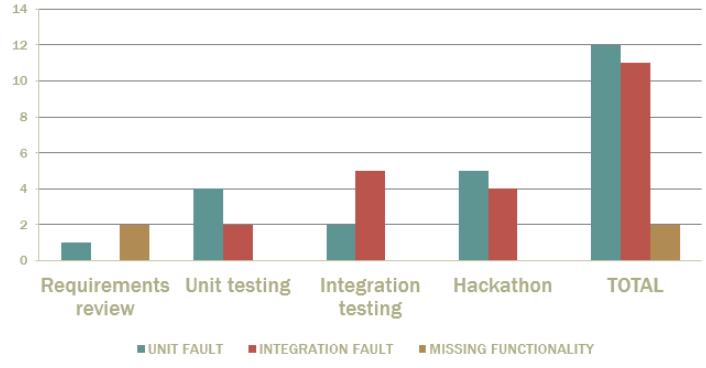
Report

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

Text code: 5641 vs. Source code: 3686 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, 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.

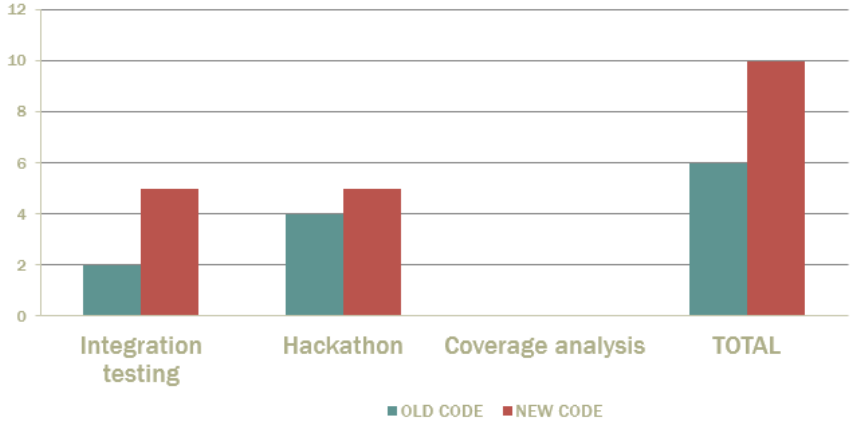


Type of faults	Requirements review	Unit testing	Integration testing	Hackathon	TOTAL
Unit fault	1	4	2	5	12
Integration fault	0	2	5	4	11
Missing functionality	2	0	0	0	2

The most number of bugs are found during Hackathon. This matched our expectation, as the other team has the motivation to find as many bugs as possible. Presumably, this is why software companies typically have separate development and testing team.

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

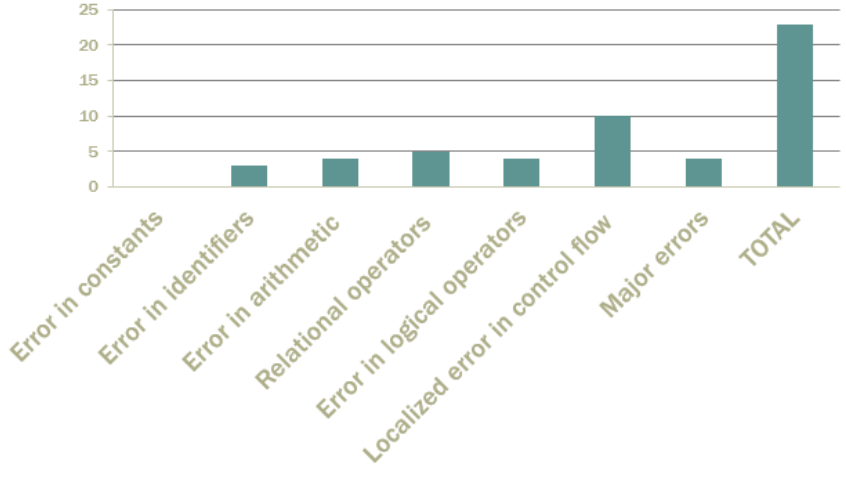
Activity: integration testing, hackathon, coverage analysis. Discuss whether the distribution of fault types matches your expectations.



Type of faults	Integration testing	Hackathon	Coverage analysis	TOTAL
Old code	2	4	0	6
New code	5	5	0	10

It matches of expectation that most of the bugs appear in the new code. From time to time we also discover regressions in the old code.

2.3. Analyze and present a distribution of causes for the faults discovered: *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').



# of faults	Error in constants	Error in identifiers	Error in arithmetic	Relational operators	Error in logical operators	Localized error in control flow	Major errors	TOTAL
	0	3	4	5	4	10	4	23

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

- requirements analysis 20%
- coding 30%
- test development 25%
- test execution 25%

4. TDD vs. Specification-Based Testing.
What are advantages and disadvantages of both based on your project experience? Can they be combined in a beneficial manner?

TDD is significantly more time-consuming. In a small, throw-away school project, TDD might not be the most efficient method. However, it scales well. When the project grows to become sufficiently big, the process of TDD provides a strong test suite along the way, and thus integrate the development and testing well.

Specification-based testing is less time-consuming. But it often misses out corner cases that are not inside the scope of the specs.

It might be possible to do TDD only for a critical feature, thus combine the two techniques.

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

Not necessarily. Especially when the target coverage is a hard value to meet and thus we create test cases just to pull it up.

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

The 10% of classes that achieved the most branch coverage are typically the ones that has *the least number of branches*. They are easy to cover.

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

Covering more branches doesn't necessarily lead to exposure of 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 more changes of designs. Some designs looked fine until during integration test we discovered that the design led to problems that were difficult to recover from.

7. 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?

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

The interface given wasn't sufficient, and the specs were not clear and detailed enough. It would help if we are given more complete interfaces and told to follow a certain implementation when in doubt (e.g. bash).
We would have thought about integration earlier when designing our components.

8. 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?

To a certain extent, yes. But there were also some pedagogical requirements that aren't helpful.

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

By manually test the functionalities and by actually using the program.

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

That would still be the test suite. If all test cases pass and we assume the test cases are representative and comprehensive, then we have confidence on the quality of our project.

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

Specification-based testing relies heavily on requirements and specifications. Before clearing out doubts on the requirements, we could barely start the testing.

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

The fact that implementing the functionality takes up only 20% of developers' time. I'm not sure whether it's common or it's just for this project.