CS4218 Software Testing Report - Milestone 1

1. Introduction

1.1. Objectives

The goal of the project is to implement and test a shell and a set of applications. For Milestone 1, we start with unit tests for basic and extended functionalities.

1.2. Test stage

- Unit test: the main focus of Milestone one, 321 test cases (98.46%) in total.
- Integration test: 5 test cases (1.54%).

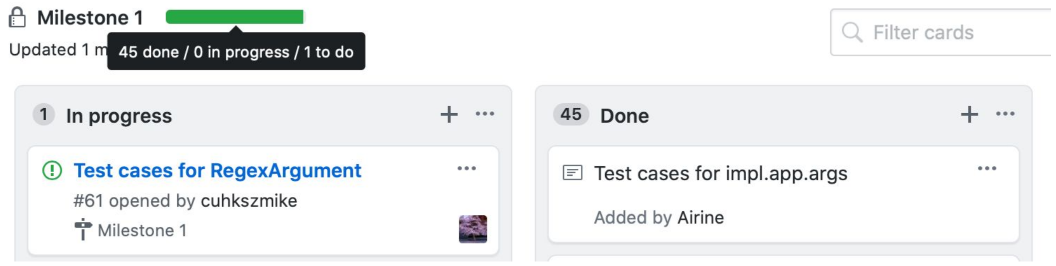
1.3. Environment Requirements

- JDK: 1.8. The Java Development Kit we used is 1.8 - lambda.

1.4. Tools

The tools we used in this project include:

- JUnit5: The JUnit version we used is JUnit 5.6.0.
- Intellij PMD Plugin: We all use Intellij, and use its PMD Plugin to analyze our code using given PMD rules.
- Maven: We use Maven to manage the dependencies consistency among our team members.
- Slack+WeChat: We use Slack as the main online communication tool, which supports thread chat.
- GitHub:
  - Slack: with GitHub extension, we can receive notification and open an issue from slack.
  - GitHub Project: the project progress manager tool. It can easily create a task as an issue and assign the issue to team members.



Milestone 1
Updated 1 m
45 done / 0 in progress / 1 to do
Filter cards

1 In progress
Test cases for RegexArgument
#61 opened by cuhkszmike
Milestone 1

45 Done
Test cases for impl.app.args
Added by Airine

2. Function Implementation

Our implementation process basically follows the process of incremental development, where the whole model is divided into many sub-modules and we implement part of it each time. The correctness of the whole model depends on the correctness of all the sub-modules. There are several strategies we follow during the whole process:

- For an unimplemented module, we first write simple implementation according to its functional description and interface. Then we write comprehensive test cases for it to search for possible bugs, we modify the implementation until it passes all test cases.
- For an implemented module, we examine its correctness by writing comprehensive test cases for it. As mentioned before, we modify the implementation until it passes all test cases
- For reusable codes and methods, we put them in utility classes.

3. Test Case Design

3.1. Test Case Design Strategy

We used both functional testing and structural testing to complete our test cases. Several strategies were followed during testing to ease our work and ensure coverage.

- We only do integration testing after we finish all unit testing to better localize bugs.
- We ensure basic condition coverage, where each choice of a condition is covered at least once.
- We use catalog testing by including error-prone test cases like boundary values or out of range values in our test suite.
- For structural testing, we check the line coverage report of our test suite and add test cases for uncovered codes.
- For integration testing
  - Test some combination of command
  - compare the running result with the expected output
  - Due to convenience, some features only are tested by integration testing (IORedirect, Pipe, a sequence of command, command substitution).

3.2. Test Case Coverage

Our test suite achieves 93.5% (1435/1535) line coverage for 52 classes. The coverage for some classes under ***app*** directory are shown in table 1.(Some test cases were excluded from our coverage result since they are for EF1 functionality which haven't been implemented yet.)

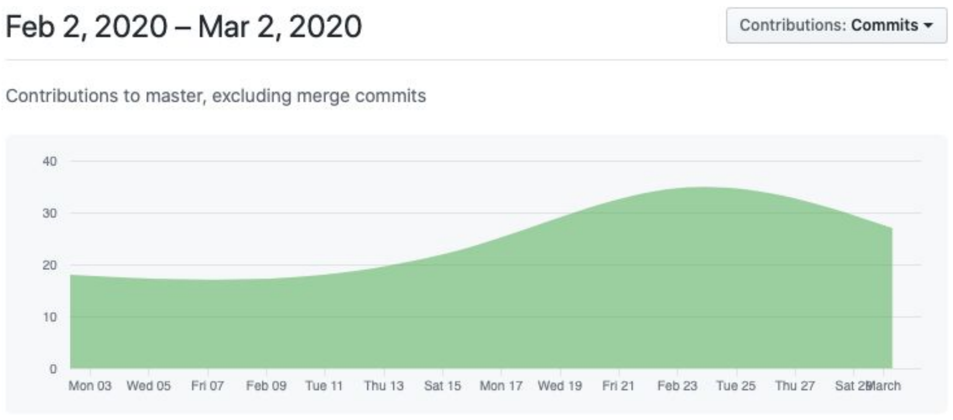
Test Case	Test Object	Method Coverage	Line Coverage
CdApplicationTest	CdApplication	100%	100%
CutApplicationTest	CutApplication	100%	97.8%
EchoApplicationTest	EchoApplication	100%	100%
ExitApplicationTest	ExitApplication	100%	100%
FindApplicationTest	FindApplication	100%	97.3%
GrepApplicationTest	GrepApplication	100%	100%
LsApplicationTest	LsApplication	100%	100%
MvApplicationTest	MvApplication	100%	100%
PasteApplicationTest	PasteApplication	100%	95.5%
RmApplicationTest	RmApplication	100%	100%
SedApplicationTest	SedApplication	100%	96.7%
SortApplicationTest	SortApplication	100%	98.6%
WcApplicationTest	WcApplication	100%	100%

Table 1. the table of coverage

For full coverage report, please refer to http://blog.aaron-xin.tech/CS4218-RESULTS/

4. Work Summary

Our test for Milestone1 starts from Feb 3, 2020 and ends on March 1, 2020. 326 test cases were created for 52 classes.
The working progress of our team is reasonable and stable:



4.1. Faults Revealed

We found many bugs in the original implementation, in addition to those caused by incompleteness of code, the following are some codes with logical errors:

Fault Type	Related Codes	Note
infinite loop	`impl.util.StringUtils.isBlank()`	`for (int i = 0; i < str.length(); )`
argument mismatch	`impl.app.SedApplication.replaceSubstringInStdin()`	the index of char in string is compared with the #th match of a pattern in string
faulty operation	`impl.app.SortArguments.parse()`	all null flag arguments are considered as filename
switch without breaks	`impl.util.ApplicationRunner.runApp()`	some switch cases do not have the break statement

untrackable exception	`EchoApplication.run()`
	`GrepApplication.grepResultsFromFiles()`
	`GrepApplication.run()`
	`LsApplication.listFolderContent()`
	`SequenceCommand.`
		Throwing a new exception from a catch block without passing the original exception into the new Exception

unhandled windows file format	`LsApplication.resolvePath()`
	`// convert String to Java.nio.Path`
		This method did not handle the windows directory.

miss function	`CommandBuilder.parseCommand()`	cannot deal with output redirection, need to delete the `break;` in case CHAR_REDIR_INPUT

miss function	`CommandBuilder.parseCommand()`	cannot start new tokens after semicolon, need to add `tokens = new LinkedList<>();` in case CHAR_SEMICOLON
incorrect condition	`ArgumentResolver.resolveOneArgument()`	the condition in `if (subOutputSegment.isEmpty())` is incorrect, should be modified as `if (!subOutputSegment.isEmpty())`
incorrect condition	`IORedirectionHandler.extractRedirOptions()`	the condition in `if (argsList == null && argsList.isEmpty())` is incorrect, should be modified as `if (argsList == null || argsList.isEmpty())`
incorrect iostream	`PipeCommand.evaluate()`	Only the `nextOutputStream` of the last command should be `stdout`