NUS
National University of Singapore

School of Computing

CS4218 Software Testing

AY21/22 Semester 2

Milestone 1 Report

Introduction

Objective

The goal for Milestone 1 is to develop an implementation for the basic functionality (BF) and extended functionality 1 (EF1). We also came up with unit tests and partial integration tests for BF, EF1, and EF2.

Project Environment

The project should run with the following environment and dependencies installed:

- Any operating system: Windows/Linux/Mac
- JDK 11 or higher

Tools Used

- JUnit 5
- PMD Plugin
- Mockito

Implementation Plan

- Ensure that we have fully understood the shell/application requirement through the specification in the project document, checking with Linux as the testing oracle, and clarifying with the teaching team before implementation or writing test cases.
- Carefully read through and understand the current implementation in the skeleton code provided and ensure that unit test cases are free of dependencies as much as possible.
- We started off working on the implementation before writing the test cases for Milestone 1.
- For shell/application with provided skeleton code, write test cases according to application's specification and tweak the code when bugs are found.
- For shell/application without any skeleton code, implement the application, write test cases and tweak the code when bugs are found.
- We strived to implement our shell program as similar to the original to our best efforts. Some are implemented according to our adaptations, which are explained in the assumptions document.
- As this is a software engineering module, we seek to write quality codes through various good software engineering practices. Examples include Do Not Repeat Yourself (DRY) and the Open-Closed Principle.
- Weekly sprints/standups are held to plan and allocate the required tasks and update individual progress.
- Each member worked independently on their assigned tasks. Members are to open pull requests whenever they wish to merge to the main branch, which is to be reviewed by one or two other members. The reviewing process ensures that the test cases are good and working on different members' operating systems ensures that our test cases work well in a cross-platform environment. It also potentially allows members to provide suggestions to improve the code quality and correctness.

Test Case Plan

Test Strategy

The team adopted the following strategies while testing:

- When approaching the generation of test cases, we mainly utilized category partition testing to identify the parameters, categories, representative values, and constraints. Using this information, we perform pairwise testing to efficiently generate a sufficient number of test cases to provide reasonable coverage in the given time.
- We introduced FileUtils and IOAssertUtils classes which provide helper methods that are commonly used across tests. This reduces clutter and also complies with some Software Design Principles such as DRY.
- As most applications deal with and manipulate files and folders, we decided to generate test files and folders during setup and teardown. The teaching team also recommends this as it allows the tests to be run on any machine and platform.
- For unit tests, mocks are created using Mockito for classes that are being depended on. This ensures that the method under test is truly isolated from other classes, which may result in an unfair test if not done so. For example, most test cases involving our application's run() method would have a mocked parser.
- We only did integration testing after completing the necessary unit tests to ensure that bugs surfaced due to interactions between classes and better identify where the bugs are coming from.
- The team also had a goal to achieve 100% class coverage and at least 80% code coverage.
- The team is committed to testing as extensively as possible, so function class dependencies such as utils and parser are also well-tested.

Test Case Coverage

Our test suite achieved 88.5% (1503/1698) line coverage across all 35 classes. The following table lists the coverage for the application classes of BF and EF1 in impl/app, including coverage of other key classes and dependencies used:

Application	Class Coverage	Method Coverage	Line Coverage
CatApplication	100% (1/1)	100% (6/6)	89% (62/69)
CdApplication	100% (1/1)	100% (3/3)	100% (15/15)
CpApplication	100% (2/2)	100% (9/9)	88% (62/70)
CutApplication	100% (1/1)	100% (9/9)	95% (67/70)
EchoApplication	100% (1/1)	100% (2/2)	85% (12/14)
ExitApplication	100% (1/1)	100% (2/2)	100% (3/3)
GrepApplication	100% (1/1)	100% (10/10)	93% (174/187)
LsApplication	100% (1/1)	100% (7/7)	94% (99/105)
RmApplication	100% (1/1)	100% (5/5)	81% (49/60)
SortApplication	100% (1/1)	100% (11/11)	91% (89/97)
TeeApplication	100% (1/1)	100% (6/6)	89% (35/39)
WcApplication	100% (1/1)	100% (7/7)	93% (108/115)

Othey Key Classes	Class Coverage	Method Coverage	Line Coverage
app/args	100% (3/3)	100% (22/22)	94% (129/136)
cmd	100% (3/3)	88% (8/9)	87% (66/76)
parser	70% (7/10)	72% (31/43)	82% (88/107)
util	91% (11/12)	93% (57/61)	88% (417/470)

Other Statistics

Type	Numbers
Unit Test	33 Classes
Integration Test	2 Classes (CallCommandIT & PipeCommandIT)
Test Cases	1223 Tests

Summary

Bugs Found

Bugs are found in the project's original implementation through a similar approach with Test-Driven Development where the team makes use of Intellij's testing toolkit. The team would try various inputs and insert breakpoints to understand the code base and test its intended behavior. Here are some snippets of the bugs we have found in the original implementation *(Note: This list is non-exhaustive)*:

Location and Bug type	**Test Input / Test Case that reviewed this error**	**Description**
ArgumentResolver.resolveOneArgument(), Missing feature	`echo “Hello World\t”`	Tabs are not handled within quotes (eg. \t not disabled in ‘’ and “”).
ArgumentResolver.resolveOneArgument(), Incorrect condition	``echo `echo Hello` ``	To append tokens into an unmatched quotes segment, the subOutputSegment should not be empty.
CallCommand.evaluate(), Wrong operation	`echo hello > a.txt`	Added to close output streams. Without this, we cannot perform any cleanup or teardown of IO-redirected files.
PipeCommand.evaluate(), Incorrect condition	`echo hello | tee a.txt`	Without the right conditions, the passing of input/output streams during piping will be incorrect for the final command. This can result in incorrect terminal output.
CommandBulder.parseCommand(), Unnecessary code	**`call command`** `< a.txt`	Call commands with input redirection was not handled. Had to delete the unnecessary “break;” under “CHAR_REDIR_INPUT” switch case for input redirection to work as intended.
IORedirectionHandler.isRedirOperator(),	**`call command`** `< a.txt > o.txt`	Output of call commands was not correctly written into the redirected output file due to

Missing condition		missing check for “>” argument. Added in the condition to check if the current argument is equal to “>” to handle output redirections.
IORedirectionHandler.extractRedirOptions(), Incorrect condition	Nil	Did not handle null or empty argument list correctly. Change from using “&&” to “||” to handle null or empty argument lists correctly.
SortApplication.sortInputString(), Incorrect implementation	Created file (a.txt) with multiple single characters and symbols/special character lines not in order of their ASCII values. Ran `sort a.txt`	Sort application did not sort according to the conditions stated in the project document, it was only sorting by ASCII value, not all symbols/special characters came before characters and numbers. Added code to ensure all symbols/special characters come first instead of sorting purely by their ASCII values.

Acceptable PMD Errors

PMD Error	**Explanation**
Closure Resources	● Overall, there are various PMD issues regarding CloseResources that were suppressed. ● This is largely due to using an external util method to close the resources which PMD does not recognize.
Long Variables	● Found in the ErrorConstant class. Long and descriptive constants help the programmer understand the error easily. Hence, the team decided to keep the constant naming, prioritizing understandability.
Excess Method Length	● Found in classes such as ArgumentResolver, CommandBuilder, GrepApplication, RmApplication, and IORedirection. ● Results due to the interface methods given or the method are rather complex, and many parameters are required.
Class Naming Conventions	● Found in CommandBuilder and Environment, the class name given seems intuitive and descriptive which the team feels is reasonable
God Class	● Despite the team's best efforts, the team feels that the complexity in the classes is high, and having a larger class is reasonable. ● **CutUtilsTest:** The test file requires a relatively large amount of constants due to repeated usage of certain values for testing and a large number of methods to test the methods of a class. ● **GrepApplication:** The original skeleton (with TODO) had already triggered this warning. However, an attempt to refactor the methods persisted with the warning. Since we are not allowed to remove any predefined method from the skeleton codes, usage of auxiliary files such as GrepArguments introduced even more PMD warnings due to redundant assignments and repeated logic that violates Software Design Principles.
Avoid String Buffer Field	● Found in RegexArgument, using string builder gives the developer more flexibility when dealing with strings.
UseVarargs	● Using string array instead of varags allows for better flexibility to the developer and the team feels that it is easier to be used.