



MONASH University

TEST STRATEGY

Kaihan Xie 28560949
Wenchu Du 290300250

Introduction

This is the test strategy for Orbital Rocket Information System, which has been written to communicate test approach to team members. It includes the objectives, testing plan, testing approach and defect tracking process.

Objectives

The following tests can be categorised into two types; One is functional testing which details the functions and capabilities that the system should be able to perform; another is non-functional testing which examines all of the critical features of the system to determine whether they meet business expectations.

In the functional test, the system will be examined through two testing methods. The first one is black-box testing which basically is conducted to detect missing functions and behaviour errors. The black-box testing will be implemented throughout the whole software development cycle and it will be applied more often as the system becomes more complex. And the other testing is called white-box testing whose test cases are written based on the analysis of the components and structure of the system. It is designed to verify the flow of inputs and outputs via the system to optimise the code design.

As for non-functional testing, the system will be tested with regards to different aspects, such as performance, usability, compatibility, etc. to evaluate the overall system design. Since we are currently on the very first stage of unit testing, this test will not be covered in the first three phases and will be conducted later during system testing and acceptance testing.

Testing Plan

The Test-Driven Development (TDD) approach will be utilized throughout the whole software development cycle. Therefore, test cases will be discussed first before any codes are written and implemented. In the current phase, as we only have to write a part of unit tests for mutator methods (validations) and other basic components of the system, black-box testing will be applied to write test cases. First, we need to specify the constraints for each field, which in this case can be categorized into two types: the basic constraints derived from the ground truth such as the price cannot be negative and special constraints based on our assumption. For example, the year founded for rocket provider should have four digits as the basic format constraint and then it cannot be over the current year, meanwhile, should at least pass the year 1900 as the range constraint based on our assumption since there is no rockets existed before 1900. After that, a number of valid and invalid values will be specified for test cases. Here, worst-case robust testing will be conducted for numbers as errors tend to occur at the boundary values for input variables. Therefore, the parameters can be taken from the value beyond the max value(max+), the max value in the range(max), normal value and the same for minimum values: min and min- for every boundary. Then, a small piece of the code will be developed based on the test cases discussed in the previous steps to check

if it passes the tests. If so, complete the code and repeat from the first step; Otherwise, refactor the code until it finally passes the tests. In addition, other types of tests will be implemented in the following phases as the structure of the system has not yet been established.

Testing tools and approach

This project is going to make use of the agile method, with 3 days iterations. Each iteration has its own test phase, the test will be generated at the beginning of each iteration; Then, the code will be written and tailored to pass each test. Apart from that, the regression test will be run if any changes to code and logic have occurred. And at the end of each iteration, the new requirements and test cases will be discussed by team members.

The following tools for software development and version control will be used in this project:

1. NetBeans IntelliJ (Java IDE)
2. GitKraken (Version Control Tool)
3. GitHub Education (Private Online Repository for code)
4. Jenkin (Continuous Integration Tool)

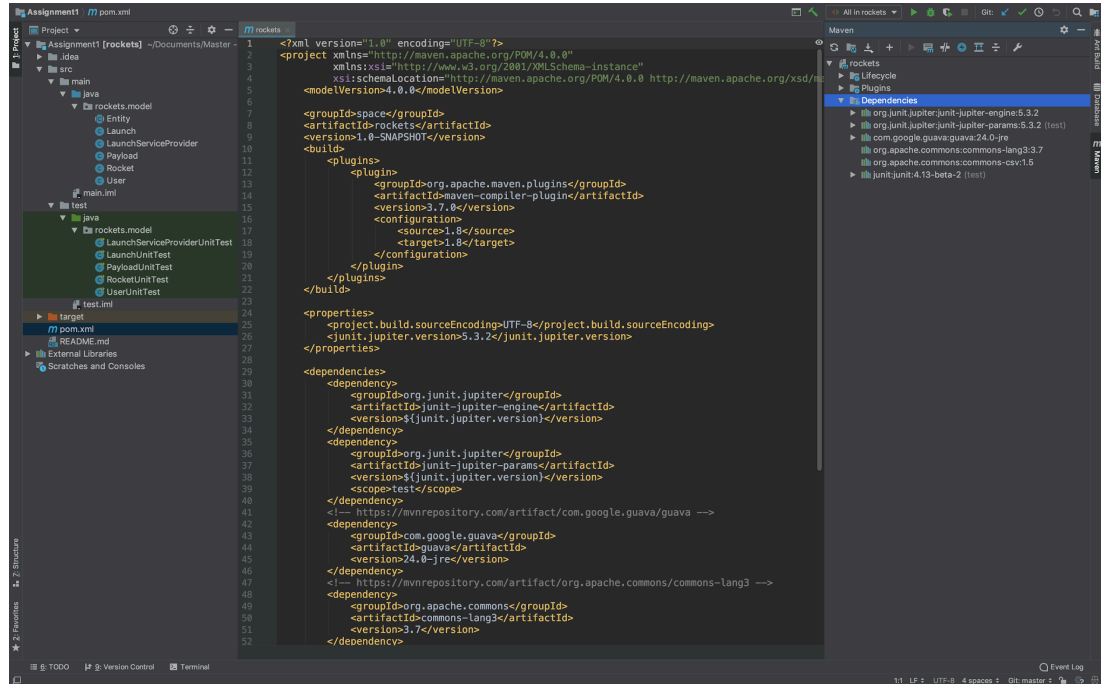
Defect Tracking Mechanism

One of the objectives of using TDD is to reduce bugs, as the code will be tailored to pass pre-written tests. However, defects still seem to be inevitable during each phase of the software development cycle. The defect tracking process of this project can be divided into four stages; The first step is analysis; the tester will examine the given test scenarios from both positives and negatives approaches. The next step is defect entry, which occurs when an abnormal behaviour turns up and happens to be a defect, then the defect will be recorded into the bug tracking tool such as Backlog and Mantis by the tester. Before we correct the defects and run the tests again, the recurring defects will be analysed on if it is a new bug or discovered bug. Finally, the process will be terminated by the team leader after all defects are handled properly.

Development & Testing Environment Setup

In this project, we are going to use IntelliJ (as shown in figure 1) as IDE which embedded with Apache Maven Engine and a number of other required Java packages to develop the Rocket web application. In addition to that, we use GitHub as the repository service to store and monitor the code along with GitKraken for version control purposes.

1. The Screenshot of Apache Maven Setup on IntelliJ





2. Java JDK & Apache Maven version check


```
wenchu — -bash — 88x12
Last login: Tue Mar 26 11:31:56 on console
Wenchus-Air:~ wenchu$ mvn -version
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5fffb20918da4f719f3; 2018-10-25T05:41:47+11:00)
Maven home: /usr/local/Cellar/maven/3.6.0/libexec
Java version: 1.8.0_201, vendor: Oracle Corporation, runtime: /Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/jre
Default locale: en_AU, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.4", arch: "x86_64", family: "mac"
Wenchus-Air:~ wenchu$
```


3. Set Tutor Arvind Kaur as the collaborator of the project

CollaboratorsPush access to the repository

**Andy Xie**
1200953



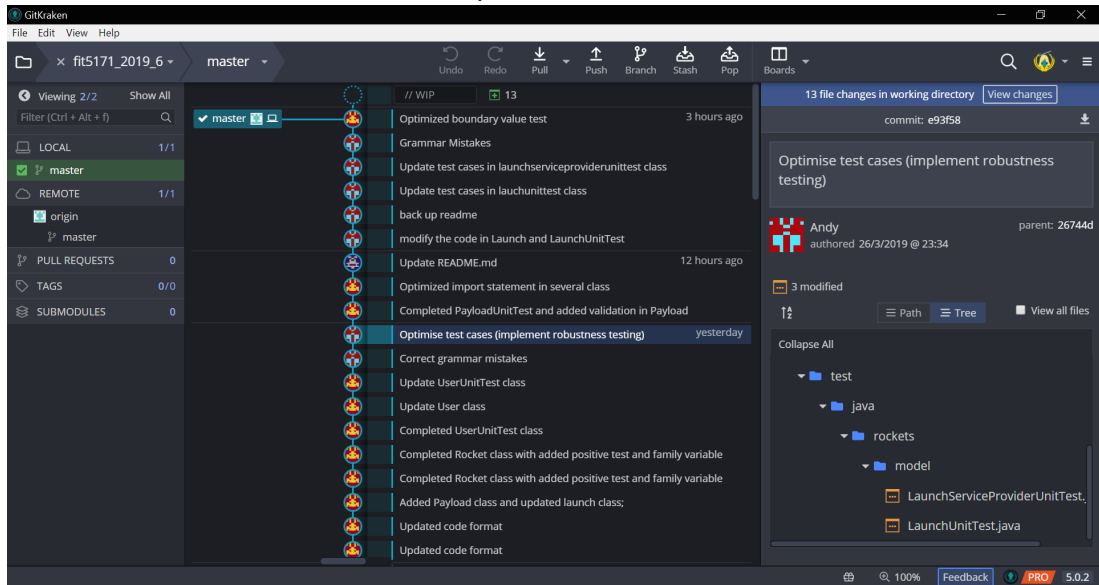
**Arvind Kaur**
ArvindKaur



Search by username, full name or email address
You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

Add collaborator

4. The screenshot of GitKraken setup



Code Base Understanding and Extension

The abstract class Entity will be inherited in every class and the field in that class will be used as the primary key for each record. However, the wiki link will not appear in User class as there should not be any Wikipedia links related to the private accounts. We also noticed that there are a number of mutator methods missing in Launch class, so some fields such as orbit and function cannot be changed once the object is instantiated. Therefore, corresponding mutator methods are complemented in the class. Moreover, we modified the return type of mutator methods from void to boolean for running positive functional tests.

The testing process follows the TDD procedure, the constraints are first discussed before any test cases or code are written. As mentioned in the test strategy, except for the ground truth, some of the assumptions are made based on the research online. The fields with string type are generally confined not to be null or empty. For usernames and country names, the first character of each word must be capitalized, and for Email and password, we use a single regex to validate each parameter. In order to avoid the illegal argument exception, the password must be at least one upper case letter, lowercase letter, one digit and one special character, and the length must be greater than 8. In addition, the Email should not be the same for the different user.

For numbers such as integer, the first constraint we came up with is the range limit. For example, the price should not be negative or zero. Then, the format for the year should be limited to 4 digits as format constraint. And again, as mentioned above, we made our own assumption here that we assume there is no rocket existed before 1900 and as the attribute yearFounded is the time happened in the past, it should not be over the current year.

Several constraints are raised to restrict reference data types as well. For instance, we have the local date for launchDate to store the date in the standard international date format 'yyyy-MM-dd'. As it is a derived data type, there is no need to make a decision table to test the built-in function for date increment. Therefore, we only made our assumption that the year first launch service provider (LSP) founded is later than the year 1900. Furthermore, the enum types are also verified such as successful and failed for launch outcome.

After the constraints are determined, we started to write test cases and the code for corresponding tests to implement these constraints. Basically, all string type variables are restricted to be a non-null and non-empty value. Therefore, the mutator methods and constructors are tested by passing null value and a series of empty strings into the parameter list. Apart from that, for a number of special cases, for instance, the country, first name and last name, which are restricted to only contain words with the first character capitalized are tested from both positive and negative approaches; In the positive tests, a set of valid names will be passed into corresponding constructors and methods while invalid names are passed in negative tests.

As for the number and date data types, the worst-case robustness test approach is implemented to examine the system behaviour. For example, as mentioned above, the year founded is confined from 1900 to current year. Thus, in the positive tests, the test cases will be the min 1900, min+ 1901, middle 1950, max- 2018 and max

2019; meanwhile, in the negative test, the test cases will be min- 1889 and max+ 2020. In addition, before testing the range constraint for the year founded, there is a test to examine if the parameter is 4 digits for format constraint.

For the LaunchDate attribute, since it is a reference type which contains 3 variables. Here, normally, the weak robust testing approach is implemented which is based on the single fault assumption. However, since we directly passed the local time object as a whole, we decided to use Boundary Value Testing approach to test our assumption that if the year is in the valid range. So, the same test case suite of the year founded is implemented again for the launch date.

After all test methods are written, the code in the main class is developed and tested by running the tests mentioned above. The validation for each attribute is included in corresponding mutator methods and constructors.

Self-Assessment

Wenchu Du, a team member in team 6, the main focus is on coding section. The contribution is described as follow:

1. Setup the environment of development and test, created GitHub repository with relevant screenshots in the report.
2. Designed test cases iteratively.
3. Created and extended code base and unit test classes with constraint validation unit test and relevant validation methods.
4. Optimized the code implemented in the initialize phase until finished the assignment.
5. Completed the extra credit report section with relevant code development.

Kaihan Xie, a team member in team 6, has played a role in coding and logging throughout phase 1. The main contribution can be described in the following aspects:

1. Finished the test strategy and the description of code understanding and extensions
2. Optimised some of the test cases for implementing testing approaches we learned and avoiding redundancies
3. Extended and refactored the tests and also the code base in some of the classes
4. Analysed and solved the problems raised in developing the test code (recorded in event log)
5. Examined the grammar for test description and revised the argument source type and exception type.
6. All collaborative work mentioned above.