



THE UNIVERSITY  
*of* ADELAIDE

# EXAMPLES OF TESTING PLAN

[adelaide.edu.au](http://adelaide.edu.au)

*seek* LIGHT

# Testing plan template

## MCI project First Milestone Report

Team number :

Project Title:

Milestone 1	Activities	Planned Outputs	Achieved Outputs
Restate the milestone from your Draft plan .	Restate the key activities from your draft plan.	Restate the planned outputs from your draft work plan.	Outline the actual outputs compared to what was projected (or type "same as planned")
Team reflection on progress	Provide some comments below regarding the completion of this milestone specifically around: 1. How is the project progressing? 2. Are there any differences between projected and actual outputs/outcomes?		

# Before to start the testing plan

- Download the testing plan template. You can find the link in Week 8

[https://myuni.adelaide.edu.au/courses/50165/files/6455516?module\\_item\\_id=1831000](https://myuni.adelaide.edu.au/courses/50165/files/6455516?module_item_id=1831000)

- Video of testing software on **echo 360**

[https://myuni.adelaide.edu.au/courses/50165/external\\_tools/82](https://myuni.adelaide.edu.au/courses/50165/external_tools/82)

- Slides of testing software. You can find the pdf file in Week 7  
"Lecture week 7-Testing " and "More resource – eXtremeTesting"

# Important Notes

- Follow the testing plan template.
- The description of the points in the template should be concise and brief. Without test reports, the document should be non-longer than 3 pages.
- The examples don't contain the same points as the testing template. However, your testing plan should include all the points of the template.
- Remember you are doing a plan, it is not needed to add all your test cases reports only some examples.

# Example 1

## Testing Plan

Automatically Documenting Jupyter Notebooks

### 1 Introduction

#### 1.1 Project overview

Research about the use of computational notebooks shows the inadequacy and undesirable quality of documentation. The goal of this project is to make documentation more readable, more reliable and be generated efficiently, ultimately minimising the gap between exploration and explanation, and easing the process of tracking, sharing and reviewing the analyses and enabling reproducibility. This goal is achieved by developing a software to document Jupyter Notebooks automatically using 200,000 source Jupyter Notebooks. A two-pass method is used in the software and the two stages are: pre-programming stage and look-up stage.

#### 1.2 Scope

This document describes the testing approaches and the framework that driving the testing of the software. Since all the tests have been done before the second milestone except the compatibility tests, this plan focuses on how the tests have been planned and organised throughout the development process. The document covers the introduction of the project, test strategy and execution strategy, but excludes the specific test cases and the results of the tests (though some examples are given to show how the tests were planned).

### 2 Test strategy

#### 2.1 Test scope

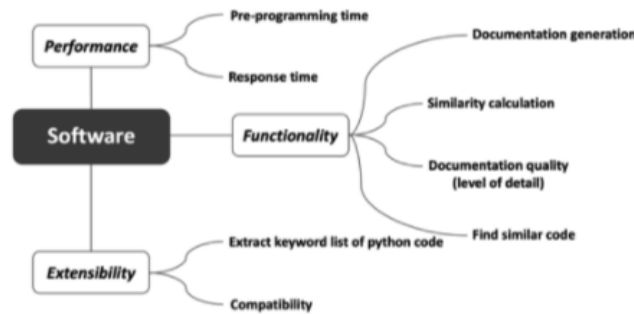
The objective of the tests is to verify that the software developed works according to the features of the software (Figure 1). There are 3 levels of features: functionality, performance and extensibility. Unit tests were done to test each of the key individual functions. Validation tests were done to test the functionalities and performance of the software, ensuring the software operates as intended. Defect tests have been being done during the whole development process whenever an error was shown. Except the compatibility tests, all other tests were done in IOS environment.

The final product of the testing procedure is twofold:

- A production-ready software
- A testing report (appendix A shows an example part of the report)

# Example 1

Figure 1 Features of the software



## 2.2 Test assumptions

- The documentation is written above the code to be explained
- At least some source notebooks contain identical blocks of code
- At least some source notebooks contain quality documentation

## 3 Test execution

### 3.1 Defect test

Defect tests have been being done throughout the whole project development process. A test was done whenever an error occurred. For example, one serious defect was shown when implementing the hashing function in pre-programming using the defect test. The test case used was an old version Jupyter Notebook file. It explained that the hashing function did not work with the old version Jupyter Notebook and thus this problem has been solved by reading new and old versions differently.

### 3.2 Unit test

Unit tests were done to test the following functions:

- `string abstract_keywordlist(string s);`  
Write few python code snippets and use them to run the function. Manually write down the expected keyword lists as the planned results before running the function.  
Feature tested: Extract keyword list of python code
- `void creat_Buckets(string keywordlist, string content);`  
Write few sample keyword lists and content of them. Use them to run the function. Check the output files. It is expected that a txt file with each keyword list (a bucket) would be created if it has not been. If there already existed a txt file with a give keyword list, the content of the lists would be appended to that txt file.  
Feature tested: Find similar code
- `double levenshtein(const string source, const string target)`  
Enter few strings and run the function. Manually calculate the levenshtein distance as planned results.  
Features tested: Similarity calculation, Documentation quality (level of detail), Find similar code

### 3.3 Integration test

Integration tests were done to test the following features:

- Pre-programming time  
Split the source notebook files into few folders, run whole pre-programming section in each folder and print out the time used. Compute the total time used for all folders.
- Response time



# Example 1

The response time is the time used to look-up documentation for a given code snippet. Input few code snippets with different lengths, run the functions for the look-up stage (note: in this stage, the programming stage is finished, i.e. the keyword lists of all 200,000 source notebooks have been extracted as txt files) and print out the time used. This test has been doing throughout the process of improving the efficiency of the look-up stage.

- Documentation generation

Run the functions for the look-up stage with few code snippets and the different weights of two similarity scores. Print out documentation found.

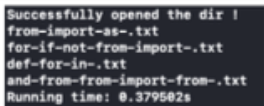
- Compatibility

The software will be run on Windows and Linux environments to test its compatibility.

## 3.4 Test management

All the tests done will be recorded (some have already been recorded) in a test result recording table (an example is shown at table 1) along with the test case and a brief description of each test in the test report (see Appendix A), so that all stakeholders can review the test results. The unit tests were done by Haojie Chen and reviewed by Qianyi Liu, and the integration tests were done by Qianyi Liu and reviewed by Haojie Chen. A checklist (table 2) will be used to make sure all features are verified.

Table 1 Test result recording table example

<b>Test</b>	Test function: <code>string abstract_keywordlist(string s);</code>	
<b>Test case used</b>	nb_218994.ipynb	
<b>Feature(s)</b>	Extract keyword list of python code	
<b>Test date</b>	21 May 2019	
<b>Planned result</b>		<b>Actual result</b>
From-import-as-.txt For-if-not-from-import-.txt Def-for-in-.txt And-from-from-import-from-.txt		
<b>Note</b>	Test passed	
<b>Tested by</b>	Haojie Chen on 21 May 2019	
<b>Reviewed by</b>	Qianyi Liu 26 May 2019	

# Example 2

## Testing plan

Project 20 Extracting the features of security software from its documentation

### 1. Introduction

The purpose of this document is to describe the scope, objective, roles and responsibilities, resource and environment, schedule and test strategies of the project 20. Our project will use machine learning (ML) algorithm to extract the features from the specific documentation and provide useful and user-friendly information. The ML models have been categorized as non-testable, it throws challenges to perform testing of ML models. However, there are some sections can be tested which play an important role in determining the success of an ML system.

### 2. Scope

This plan gives the actual test cases which are performed in milestone 2 and also defines the future testing activities of the whole project in order to fix all bugs and defects. All the test activities are determined in the Test Strategy. Additionally, to provide a quality product we also design several phases to improve our project: the system requirements review, system design review and the execution of design verification testing.

### 3. Roles and responsibilities

Role	Members	Responsibilities
Clients/ supervisor	Professor M. Ali Babar Chadni Islam	1. giving the requirements for the development of the project. 2. providing the guidance for the project and key information for the overall success of the project.
tester	Tong Yu, Liang Shi	1. understand requirements 2. write code for the requirements 3. executing test cases 4. preparation of the test data 5. retesting and regression testing 6. bug and defect review 7. coordinate with supervisor for any issues encountered during the preparation and execution

### 4. Resource and environment needs

#### 4.1 Testing tool:

Process	Tool
Test case creation & tracking	Microsoft Excel
Test execution	Manual, PyCharm, jupyter notebook
Test reporting	Microsoft Word, PDF



# Example 2

## 4.2 Configuration Management (CM)

Documents CM: GitHub

Code CM: PyCharm, jupyter notebook, GitHub

## 4.3 Test Environment

Hardware Platform: MSI (15-inch, i7, GPU: Nvidia)

MacBook Pro (Retina, 13-inch, Mid 2017)

Software Platform: PyCharm, jupyter notebook, Google Chrome v 74.0

## 5. Test strategy

### 5.1 Unit test plan

In the Unit test, we test 3 key aspects of our project. First, for the machine learning model, we check the feature selection, N-gram model, test accuracy and overfitting. The test methods of these aspects are SelectKBest function, accuracy rate, accuracy rate and plot learning curves respectively(seen as test 1,2,7 in the test matrix). We used the methods in the Scikit learn library. The second unit test is to test the pre-processing function. We use partition testing method for this test. By creating dataset which combines different situations and check whether it can be cleaned as expected(seen as test 3 in the test matrix).. The third test is GUI test, we use manual based testing and check whether the GUI can display the correct information and whether the button can work. (seen as test 9 in the test matrix).

We have already finish machine learning model tests during milestone 2. The details of the test matrix can be seen in the appendix.

### 5.2 Integration test plan

In the integration test, we will have two tests. One is a test whether GUI can connect with prediction module. The other one is to test the interface link between the prediction module and machine learning models. For these we two tests, the method we use is put text into GUI and see whether the GUI can show the expected prediction. These two tests have been finished during milestone2. The details of these two test cases can be seen in the appendix. (seen as test 3 , 4 in the test matrix).

### 5.3 System test

We will use black box test method to test the overall performance of the system. We will crawl more data from the security software website. Put these data into the GUI and check the prediction accuracy. We haven't finished this test yet. (seen as test 6 in the test matrix).

### 5.4 Acceptance test plan

In order to meet the requirements of supervisors, we will show the test result of each component in the test matrix and run some of the functions to supervisors in the client meeting.

# Example 2

## 5.5 Contingency plan for risks

The bug severity and method that we would take is list in the following table.

Bug description	Severity Description	Plan Description
Overfitting	critical	Must fix immediately
Accuracy is relatively low	critical	Must fix immediately
GUI cannot connect with machine learning model	critical	Must fix immediately
GUI lack of performance	Medium	Fix When Have Time
User's input is not clean up	Medium	Fix When Have Time

## 6. Scheduling

The detail is shown in the Gantt chart in the appendix. Overall, we have already finished build the test case matrix and machine learning model test. We will continue with the other test this week. We will debug and retest next.

# Example 3

## Testing plan

### 1. Introduction

The Test Plan will describe how to test a web-based submission system that can test code in the Docker container. It mainly includes the objectives, scope, risk, test environment, approach and schedule.

#### 1.1 Objectives and tasks

The objective of the test is to test that the whole system can work as same as design. Most of tests will use the mocha framework to execute test scripts and identify all defects and mistakes in the significant parts of this system. After that, we will try to fix and retest those the most significant parts. If there is more time, we will fix other parts that have mistakes later.

### 2. Scope

The test plan describes the strategies of the unit testing, integration testing, system testing and acceptance testing. Risks of the testing are identified. The testing schedule is listed. Performance testing and stress testing are not considered in this plan because of the limitation of the time.

### 3. Testing strategy

There are four phases in testing. In the initial phase, the main four features and container will be tested. The second phase will implement the integration testing from bottom to up. The third phase is system testing by using black-box. The last phase is acceptance testing.

#### 3.1 Unit testing

Four main functions and the container part will be done the unit testing. They are separately updating data function, checking data function, uploading files function and downloading files function. Besides these four main functions, we

# Example 3

also need to test the container part, including creating, starting, stopping and removing. Details about unit testing are in the Appendix A.

## 3.2 Integration testing

In the integration test part, we will use the bottom-top strategy. Bottom-up integration is that we integrate infrastructure components firstly and then add functional components, which can simplify error localisation during the process of adding components. Thus, considering the customer's operating process, we will add functions along with the following sequence. Appendix B shows the details about integration testing.

## 3.3 System testing

In system testing, the project will be tested as a black box. In this part, we will assess the project with the specified requirements as following:

- (1) Create a course      (2) Upload a dockerfile      (3) Upload code file
- (4) Get results for code      (5) Store results

## 3.4 Acceptance testing

In this test part, we will distribute this system to some teachers and students. After they use this system, we can get real input from target users and may find other problems, which will decide if this system is ready to be deployed into the customer environment.

## 4. Test environment

The software needed in the test are listed as following:

- (1) Chrome Version 74.0.3729.169, Safari Version 12.0
- (2) Mocha (Testing framework for unit testing and integration testing)
- (3) Chai (Assert library for unit testing and integration testing)

# Example 3

## 5. Test schedule

We have finished the unit testing and the part that creating the container in the integration testing. The rest of integration testing, system testing and acceptance testing will be finished before 15/6/2019. Appendix C illustrates the details of the test schedule in the appendix.

## 6. Risks

The risks have been identified and the extent of the impact is defined according to how the project would be affected when the risk happened. The trigger is the event which causes by the risk. The mitigation plan is an action to relieve the risk. The details of risk are listed in the figure 1:

No.	Risk	Impact	Trigger	Mitigation Plan
1	Excessive delays	High	Defects are founded in the testing and testers spend much time fixing the defects.	If the defect does not affect the main functions, the defect can be fixed later.
2	The tests written before are negated	High	Changes in the functionality cause these functions need to be retested.	The functionality is not changed unless the change is necessary.
3	Wrong results of testing	Medium	Wrong test cases because testers incorrectly analyse the boundaries in the partition testing	Testers develop testing as a team. One tester design the test cases and the other review the test cases.

# Example 3

## Appendixes

### Appendix A Unit testing details

Unit testing	Test design	Input	Expected Output	Actual output	Responsibility	Reviewer
Update data part	Type data in the text area for testing if data are saved into the particular file	new data that are typed in the text area	The particular files contain new data if the data type is suitable to this area, or return "wrong data type" to the console	Same as expected	W. Zhou	Y. Xue
Check data part	Use an existing JSON file to test if the text area can read and present previous data	A JSON file containing data	Text area can present data from a designated file	Same as expected	W. Zhou	Y. Xue
Upload files part	Upload a file to test if this file can be saved in the particular folder of the server.	A file	The complete file can be found in the particular path if the uploading file format is correct, or return "wrong file format" to the console	Same as expected	W. Zhou	Y. Xue
Download files part	Prepare many files and testing if the function can pick and compress files that user need, and then put this zip file in a particular position	Different files that are put in different folders	A zip file that contains all files that user need	Same as expected	W. Zhou	Y. Xue
Docker part – create image	Use a dockerfile to test if this part can create a new image	A dockerfile	A new image in the docker folder	Same as expected	Y. Xue	W. Zhou
Docker part – create container	Use an existing image to create a new container.	A image	A new container	Same as expected	Y. Xue	W. Zhou
Docker part –run code in the container	Prepare a test code and run a container to test this code file	A code file and an existing container	The feedback that is generated by container for the code file	Same as expected	Y. Xue	W. Zhou
Docker part – remove container	Container can be removed automatically after running.	A existing container	This container is removed from the container list	Same as expected	Y. Xue	W. Zhou