Test Plan

Contents

1 Introduction

This test plan document will discuss the test methods used in the ChocAn project. These testing methods include unit testing, smoke testing, and system testing. After going through these tests, our team can discover the weaknesses of this project and improve the project against these weaknesses.

1.1 Purpose and Scope

The purpose of this file is to test whether the ChocAn project can meet the system requirements and whether it can meet the needs of users. In this test plan, these goals need to be achieved:

- Through unit testing, smoke testing, and system testing to ensure that the entire system is error-free and can meet the expected functional requirements.
- During the test, some improvements were made for some unsatisfactory parts.
- Improve understanding of testing.

1.2 Target Audience

The target audience of this test plan document is the maintenance team and test team for this project. If anyone is interested in the testing process of this project, you can also read this document. Before reading this document, you need to understand the basic structure of the project. You can read the design document or read the code of the document.
Relevent links:

- Project repository
- Design document

1.3 Terms and Definitions

Here are some definitions that might be used:

- Unit Testing: UNIT TESTING is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.
- Smoke Testing: SMOKE TESTING, also known as “Build Verification Testing”, is a type of software testing that includes of a non-exhaustive set of tests that aim at ensuring that the most important functions work. The result of this testing is used to decide if a build is stable enough to proceed with further testing.
- System Testing: SYSTEM TESTING is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications.

2 Test Plan Description

2.1 Scope of Testing

- Terminal test module
  Test whether the program can complete the basic operation in the computer terminal and reach the required menu as required.

- Maps Data Structure module
  Test whether the corresponding structure of the data storage location is implemented.
- User Hierarchy module
  Testing to ensure members, providers and managers have the right data and function.
- Service Record module
  Test to make sure a service is selected and recorded correctly as Providers and members.

2.2 Testing Schedule

The first test to be performed is a unit test. This will ensure that each sub- component (unit) works properly and that all errors and incorrect outputs found are fixed.
After unit testing is complete, the next testing phase will be smoke testing. This will test the system as a whole, with each slave unit interacting with all its systems and dependent units. Once the main functions of the system as a whole are tested and completed, we will transition to system testing.
The final testing phase will be system testing. This test will ensure that the software meets the overall required specifications.

2.3 Release Criteria

Because the document involves the user's personal privacy, we must make reasonable adjustments to the final released document.

- Because the data contains sensitive information of members and providers For ChocAn, our final requirements will require the security of all our data and our data structure does not contain memory leaks that can be exploited.
- Our program should allow logins from different devices or terminals, and must identify different levels of personnel information.
- Allows the administrator to manage the services that are present in the system, and the details related to the services.
- Allow customers to get their own weekly reports, and display the current weekly report in real time when the user needs it.

3 Unit Testing

In the unit testing section, we will list the testing strategies and some functions that need to be tested. In unit testing, we need to test the individual functions in each object, and make sure that each object works correctly before linking them together.

3.1 Strategy

In our system, we have built these classes:

- InteractiveModule
- ManagerModule
- Member
- Person
- Provider
- ProviderDirectory

- ProviderModule
- Service

Some of these classes exist independently and have nothing to do with other classes, so we decided to test these classes first. There are some classes that need to interact with other classes. For these classes, we will complete this part of the test after completing the previous part of the test.

3.2 InteractiveModule

InteractiveModule is implemented by relying on Member and Provider, so InteractiveModule will be tested after testing Person, Member, and Provider. InteractiveModule's private domain has two unordered_maps that store information about Members and Providers, and public functions mainly operate on these two unordered_maps, including add, remove, edit, display. In the constructor, the data in the file is read and stored in the two maps. In the write_out function, the data in the map is written to the file. The init function will call other functions.
We will test the functions for the map operation and the functions of the read and write parts, because these parts of the function are more likely to make errors.

3.3 ManagerModule

The second object is the ManagerModule. This object depends on Service, Member, Person, and Provider, so it will be tested later. This object's private domain has three maps to store service, member, and provider. Among its public functions, weekly_report and summary_report are responsible for generating reports, and person_report, provider_report, and member_report are responsible for sending reports to specific people. Similarly, this object will read data in the constructor.
We will test the weekly_report function of this object and focus on seeing if this function selects the date correctly. We will also test the person_report function, because provider_report and member_report both call this function, so we must ensure the correctness of this function.

3.4 Person

Person is a completely independent object, and it is also a subclass of member and provider. Its private domain is name, city, state, id, and zip. And its common functions are used to get this information and edit this information.
This is a simple structured object, we only need to test whether the functions can get and change the information stored in it.

3.5 Member

Member is a subclass of Person. Compared with person, member has one more suspended variable. At the same time, get and set functions for suspended variables have also been added.
We will test the get_suspended and set_suspended functions.

3.6 Provider

Provider is a subclass of person, without any new variables and functions, so there is no need to test.

3.7 Service

Service is an independent class that is used to store service information and provide external interfaces to access and modify the service.
We need to test whether the interfaces of these services are valid so that no errors occur when calling these interfaces.

3.8 ProviderDirectory

ProviderDirectory has a map data structure that stores services, so it needs to be tested after service unit testing. The common function of this object is generate_directory (), which can write directory to directory.txt file.
We are going to test the generate_directory () function to make sure the format of the data meets our expectations.

3.9 ProviderModule

An instance of the providerdirectory object exists in the private domain of the ProviderModule, so you need to test it after you finish testing the providerdirectory. The public functions to be tested in this object are: validate_member, find member by member id, and check suspended status of member. provide_service, output service information, and confirm that there is no error in the service information. get_provider_-directory, write the service information to the directory.txt file.

4 Smoke Testing

In the above, we tested the actual work of each function independently, and we actually verified whether the function's return value is reasonable and effective. And we verified the processing status of the data by a single function. In this section, we will combine all the previous test results to perform linkage between multiple functions to determine whether the cooperation between multiple functions is effective.

4.1 ManagerModule, ProviderModule, InteractiveModule Login

Case	Test	Expected Result
1	Login to manager module with valid ID	logs in
2	Login to manager module with invalid ID	login is rejected
3	Repeat 1 and 2 for InteractiveModule and ProviderModule	

4.2 InteractiveModule Searching and Editing Members and Providers

Case	Test	Expected Result
1	Search for provider and member that exist	person is found
2	Edit a provider and member that exist	data structure is updated correctly
3	Remove a provider and member that exist	data structure is updated correctly
4	Search for provider and member that doesn't exist	nothing is found
5	Edit a provider and member that doesn't exist	data structure is unchanged and error is returned
6	Remove a provider and member that doesn't exist	data structure is unchanged and error is returned

4.3 Reports

Case	Test	Expected Result
1	Run provider report	correct data is reported
2	Run member report	correct data is reported
3	Run weekly report	correct data is reported

4.4 Provide Service

Case	Test	Expected Result
1	Provide service with valid info	service data structure is updated
2	Proved service with invalid info	failure is returned and data structure is unchanged

5 System Testing

In this part we go through three steps to test the integrity and functionality of the program as expected. The three tests are: joint test, structural test, and performance test.

5.1 Joint Test

In this test we test the complete program by combining all the units. The main purpose of this test is to find out whether the link between the various units and interfaces is smooth, and to troubleshoot some errors that are only found at runtime.

5.2 Permission Test

In this test we will test whether the permissions of each type of user are correct. For example, members can only change their basic information by logging in to their account. Providers can only make changes to their users; records of this service and cannot see the basic information and other records of members.

5.3 Storage test

This test will check that the user information and records are stored correctly and that the automatically sent abstracts and reports are properly structured.

5.4 Disk Recording

It will include the following info:

- Current date and time
- Date of service
- Provider's number
- Member number
- Service code

5.5 Summary Report

It will include the following info:

- A list of providers to pay this week.
- Number of consultations per provider.
- The total cost to be paid by the provider during the week.
- Total number of providers providing services.
- Total number of consultations from all providers.

5.6 Generated Form

It will include the following info:

- Member name
- Member number
- Member address

- Member cities
- Member States
- Member postal code

For each service provider, the following should exist:

- Date of service
- Provider name
- service name

On the other hand, the provider's form should contain the following data:

- Provider name
- Provider number
- Provider address
- Provider city
- Provider status
- he postal code of the provider
- Total number of consultations
- Total cost of the week

For each service, the following data:

- Date of service
- The system receives date and time data
- Member name
- Member number
- Service code
- Fees to be paid

6 Performance Test

6.1 Stress test

In this test, the program's endurance limit is tested by entering a large amount of data into the program at one time.

6.2 Time test

This test is to check whether the program produces results at the right time for each input.

6.3 Security Test

In this test we will test the program's response by intentionally entering an error code or performing an invalid operation.