Test Plan for the Chocoholic Anonymous System

A data processing system to support membership and services provided to the chocolate dependent population.

CS 300 PSU WINTER 20’

Professor Chris Gilmore

Group 1

Christian Andersen

Darika Batbayar

James Bao

Minjin Enkhjargal

Yan Li

Deep Patel

Tenzing Rither

Term Project: *ChocAn*

Test Plan Document

**Table of Contents**

[**1 Introduction**](#_wezrtebn66y3) **2**

[**2 Test Plan Description**](#_al2pbikojl30) **4**

[2.1 Scope of Testing](#_stwbkburu6z3) 4

[2.2 Testing Schedule](#_qwoggectdmif) 4

[2.3 Release Criteria](#_sy3xqqv8312z) 5

[**3 Unit Testing**](#_u9gfn8agnw9y) **6**

[3.1.1 Person](#_dqbklc8gr509) 6

[3.1.2 Provider](#_j4mg1fvzcf63) 7

[3.1.3 Member and History](#_hcy21697d89d) 7

[3.1.4 ARR and Head](#_jzau0g9qkrmf) 8

[3.2 Strategy - James](#_6gyge1fj2uz7) 8

[**4 Smoke Testing**](#_yt5a3rx4btc4) **9**

[4.1 Provider Functionality](#_m8o49lf9rzky) 9

[4.2 Admin functionality](#_gfe3c8v54lel) 10

[4.3 Member functionality](#_18sripnqid5u) 11

[**5 System Testing**](#_x19rzbme0i4o) **12**

[5.1 Member and Provider Functionality](#_d8rcukbyvno3) 12

[5.2 Member and Admin Functionality](#_xczyrcr9wqs1) 12

[5.3 Provider and Admin Functionality](#_61eocgb9h04p) 12

[5.4 Component Functionality](#_3oxg6sipfeq4) 12

# 

# 1 Introduction

The test plan will go over how our team will be testing the software for ChocAn. The document will detail the types of tests to be implemented at each step of the design, what to check for and why we’re checking for certain components. The later sections in the document will cover important terms and definitions, an overview of the tests that will be run, why those tests are important, the test plan description, the release criteria, member functionality, provider functionality, admin functionality, and component functionality.

**1.1 Purpose and Scope**

To make sure the software our group creates is correct, we have to run tests to verify and validate the code. The test plan will provide insight into what kind of tests will be run to check our progress along the way and their significance. The tests that will be implemented in our project include: unit testing, smoke testing, and system testing.

**1.2 Target Audience**

The target audience for the test plan are the engineers involved in developing the software, managers, members and health providers. The engineers have to oversee what kind of tests to build and run. Managers, members and health providers need to see what kind of verification will be placed for the software to ensure the quality and accuracy of the product.

**1.3 Terms and Definitions**

|  |  |
| --- | --- |
| **Terms** | **Definitions** |
| ChocAn | Chocoholics Anonymous is an organization committed to helping people with chocolate addiction. |
| Unit testing | Individual program units or object classes are tested. |
| Smoke testing | Non-exhaustive set of tests that aim at ensuring that the most important functions work. |
| System testing | Some or all of the components in a system are integrated and the system is tested as a whole. |
| Provider | Healthcare professional provider. |
| Member | Customer of ChocAn seeking aid from a provider. |
| Admin | Administrator for ChocAn |
| ChocAn Data Center | Data center responsible for processing member’s information. |

# 2 Test Plan Description

This test plan document aims to demonstrate the scope of testing that will be done to ensure the software meets its requirements. The goal from validation testing and defect testing is to test that the software meets requirements, operates as intended, and that defects/bugs are exposed.

## 

## 2.1 Scope of Testing

The three different tests that will be done are unit tests, smoke tests, and system tests. Unit testing will test individual components for functionality and normal operation, as well as with abnormal inputs to ensure that component does not break. The scope of this testing mainly lies on observing output behaviours based on input data of each component. Once the main components are in place, smoke testing will test the scope of the overall software and if critical functionalities are acceptable and stable. After these tests, system testing checks the integration and interaction of components. The scope of system tests will be focused on different areas of the software, wherever components have come together. Testing on the security of the software will be omitted.

## 2.2 Testing Schedule

|  |  |  |
| --- | --- | --- |
| **Period** | **Test Type** | **Test Summary** |
| 2/23 - 2/29 | Unit Testing | - User input and outputs, read in functions, delimiters, function parameters, data access |
| 3/1 - 3/7 | Smoke Testing | - Verify critical functionalities and overall functionality of software is in place |
| 3/8 - 3/13 | System Testing | - System integration, observing interactions between components, ensuring components are compatible |

## 2.3 Release Criteria

**2.3.1 Definition of success**

Success will be defined by the availability of correct functionality as demanded by client requirements at all three interfaces: Member interface, provider interface and administrator interface. The software must be able to operate at all three interfaces smoothly and provide information and data requested.

**2.3.2 Functionality and Intended Purpose**

At the member level, data that does not belong to the individual shall not be displayed; adequate security through data verification or member ID card will be required to permit information to be displayed to the member. The software must be able to adequately store data and retrieve data when requested, such as for the weekly reports. This also means that the data being stored must also be correctly written out to file.

Another criteria that must be met is the ability to correctly keep a running balance of all fees for the member services that have occurred the previous week. This running balance must be correct, so that the correct bill can be sent to Acme Accounting Services to ensure that each provider receives correct payment from ChocAn.

Errors relating to abnormal inputs must have been well-tested and thus will not be tolerated. Errors apparent to a single component should be caught and fixed. The main functionalities as required by the requirement documents must be adhered to.

# 3 Unit Testing

The ChocAn data processing program consists of four units and therefore, this section introduces test plans for each unit of the system. Each unit, further, has many different functionalities and methods to be tested for validation and verification of the system. In order to test the functionalities of each unit, there are calls to the methods with different input parameters. Due to the object oriented programming this system employs, this section covers all the major features of the objects and puts them into all the possible states to prevent spontaneous state changes that may occur. Each unit is tested in isolation as well as the methods found in the objects. The four units to be tested are stated below.

## 3.1.1 Person

This is the parent class to two other derived classes. Although there are two classes dependent on it, an object of this class is able to be tested in isolation. The purpose of an object of this class is to contain the same data that is then specialized into two other categories. This class contains the following as protected data members:

1. Name (25 characters)
2. ID (9 digits)
3. Street (25 characters)
4. City (14 characters)
5. State (2 characters)
6. Zip (5 digits)

This information is entered by operators at the ChocAn Data Center. The testing of this unit includes the input values entered by the operators. For each data member that is a string or an array of characters, the test ensures there are no special characters or any integers. For the data members that are integers, the test ensures there are no characters. In order to test this unit, an automated exception handling is implemented to catch all possible inputs. When a wrong input is entered, the program catches the error and makes sure that the program doesn’t abort immediately. This class is implemented first in order to test the derived classes mentioned below.

## 3.1.2 Provider

The provider class is derived from the Person class in 3.1.1. This class is difficult to be tested in isolation and therefore, the programmer implements the Person class first. When an object of a Provider class is created, it inherits all the data members of a Person class along with its methods. One of the tests to be done on an object of this class is to test the inheritance. Calling a display method for the parent object using the current object of the Provider class is sufficient to test the inheritance. The Provider class has other functionalities such as looking up service codes and entering necessary information to fill the form in order to add service history to a particular patient or a member. The inputs entered by the provider using this class are tested in the same way as the parent class tests, Person. Catching all possible input errors is done using the same test plan used for the parent class as well. When a method to add a history to a member is called through the object of the Provider class, the method has an integer as a return value to indicate a successful addition. This value is caught in another method when the program is run but it is caught in the main file during testing for isolation purposes. This class must be able to access the Provider Directory file at all times and this is tested after each function call during testing to validate the consistent record of the same file.

## 3.1.3 Member and History

The member class is also derived from the Person class in 3.1.1. A member class object contains a pointer to a linear linked list of member [patient] history objects, each of which contains a date string in [MM-DD-YYYY] format plus other inputs. When an object of a Member class is created, it inherits all the data members of a Person class along with its methods and this is tested using the same strategy in testing inheritance in the Provider class. This class has methods that are similar to the Provider class but it does not add history to itself. This class acts as a manager class for the history it contains. When building the history in a linear linked list data structure, the head of the list is held by the object of the Member class. This is tested by directly displaying the contents of the first node or the first history in the data structure. A second test is done by displaying the entire contents of the history and the data structure. The contents are automatically populated by a text file specifically produced to test them before entering histories when the program is run.

## 3.1.4 ARR and Head

This is the data structures unit and therefore, much of the building of the initial data is done through the ARR and Head classes. The ARR class is the main manager class that contains a number of methods due to the many tasks it must perform for a successful and safe entries of inputs. These two classes are impossible to be tested in isolation because they are the overarching structure for the entire program. All the methods of these two classes directly interact with the hierarchy created above. Due to the complexity of the program, these classes and objects of these classes are tested as the last unit test when the program is built.

## 3.2 General Strategy

List of unit subtests for verifying data elements

1. find\_ID() : check for spaces, bounds check, check non alphabetic
2. find\_name() : check for spaces, bounds check, check if alphabetic
3. get\_address() : individually write tests for street, city, state, and zip. each has its own bounds. “street” - 25 char limit, city - 14 char limit, state - 2 char limit, zip - 5 char limit.
4. date\_format() : check for spaces , check non alphabetic, bounds check --- it should be in the MM-DD-YYYY format
5. get\_cost(): the cost of services for each patient is determined by how many services they have incurred, with each service being a flat cost. This is done by verifying that the service fee is correct, and counting the number of services that the member has.

Basically, the testing process for every major function in the ChocAn application (verifying user credentials, generating a report, etc.) will use these unit subtests to make sure that the data objects are being handled or copied correctly. For example- the unit test for the Person class will only require subtests 1-3, while the unit subtests for Person’s derived classes Provider and Member will use all 5 subtests, since both the Provider and Member classes must work with date-time data objects and need to keep track of some number of services. Reuse of the code from one basic subtest to another will ensure high code coverage and minimize redundancy. The service and report generation smoke tests (to assess functionality for different user roles) will utilize functions that involve all of the major subtests since they require tabulating data for one or more member objects. The main problems that may arise during integration testing will likely involve storing different sets of ID data within the system for authentication, rather than maintaining the data structures used to generate reports, so we predict minimal issues with transitioning from smoke testing to integration testing.

# 4 Smoke Testing

This section outlines the testing of typical paths a user will experience while using the software. Each path consists of functions that makeup the most important functionality of the system. These tests will ensure the minimum functionality of the system has been tested, and the system is ready for basic use.

## 4.1 Provider Functionality

These tests will ensure the provider unit can find members, administer services to members, and print a report of a provider’s member service history. These tests will ensure the provider unit has basic functionality.

**4.1.1 Provider Login Functionality**

This test will ensure the provider is able to login and logout of the system. Upon entering the provider interface, the user will be prompted for credentials, which will be used to search through the ARR unit. Once the credentials have been found the interface will change and allow the provider to choose from a list of options from the provider unit. If the user’s credentials aren’t found the user will be prompted to enter in valid credentials. Once the user has completed their work, choosing the option to exit will cause the new data to be saved on disk via the ARR unit.

**4.1.2 Provider Service Functionality**

Finding a member, looking up a service, and adding that service to that member’s service history will be tested. This test will ensure the provider is able to find a member and administer a service. Once logged on, the user will be prompted for a member ID. The Provider unit will then search for an existing member using their name or member ID inside the ARR unit. Once a member is found the Provider unit will search the Services unit to find a service code for the session administered to the member. Once chosen, the service code will be stored into the Member unit as member history.

**4.1.3 Provider Report Functionality**

Producing a report entialing provider service history will be tested. This test will ensure a provider can login and pull a report for their service history. Once logged in the user will be able to produce reports from the Provider unit interface. Once a report option is chosen, the specifications will be passed to the ARR unit. The ARR unit will be searched for matching specifications and printed as a report to the user.

## 

## 4.2 Admin functionality

These tests will ensure the admin interface has basic functionality.

**4.2.1 Logging in/out as an admin**

The admin credential has been set in chocAn and has been provided to the admin of the system. When we try to log in with incorrect login information, it generates a password invalid error. We verify that admin is able to log in with the correct password. Admin is able to log out by entering ‘E’ key. When admin enters the wrong key, it generates error. ARR class has switched statements that allow to log out admin from the admin interface.

**4.2.2 Admin reports generating functionalities**

**4.2.2a Provider’s services report from ChocAn :** This function displays the used services by the member per week. The admin will enter the provider id number. System ensures that provider id exists in the system. If provider id does not exist in the system, function displays “provider does not exist” message into terminal. Otherwise, it will display all the members information, who has been taken services from the provider.

**4.2.2b Member’s service report from ChocAn admin:** This function displays the used services by each member per week. The admin will enter member id. System ensures that member id exists in the system. If member id does not exist in the system, function displays “member does not exist” message into terminal. Otherwise, it displays all the services provider information to an individual member in sorted order of service date.

**4.2.2c Brief summary of number of providers and the overall fee :** This function displays all the provider service, total number of consultation and the overall fee. The function has been implemented and tested.

**4.2.3 Adding/removing a member:** Admin is able to remove and add members to the ChocAn system. The environment deletes the members from the text file database. When admin enters the new member it will read in text file as well.

## 

## 4.3 Member functionality

**4.3.1 Logging in/out as an member:** Member log in with their member id. The system will either find the member or it will display “member not exist in the system” message back into the terminal. Members will be able to log out from their session by pressing ‘E’.When member presses the wrong character. It will display an error message and switch statement repeatedly asking for the correct key.

**4.3.2 Display service history for member:** Member will be able to print service history from member interface. This unit will display the total number of services and total amount need to be paid to ChocAn.

# 5 System Testing

The purpose of this section is to lay out some test cases for testing the interactions between the different components in the ChocAn system. This can ensure whether the parts of the system are compatible with each other. Each test will test the functionality of the three interfaces with relation to one another. The following has been listed and described below to explain the testing once all components are integrated.

## 5.1 Member and Provider Functionality

This test will ensure the components that makeup the member and provider interface work when used together. Each option in the member interface will be tested before moving into the provider interface. Each option in the provider interface will then be tested to ensure the transition from the member component to the provider component won’t produce any errors. The same tests will be conducted starting with the provider interface and switching to the member interface. This test will ensure the provider component works when switching to it from the member component.

## 5.2 Member and Admin Functionality

This test will ensure the components that makeup the member and admin interface work when used together. Each option in the member interface will be tested before moving into the admin interface. Each option in the admin interface will then be tested to ensure the transition from the member component to the admin component won’t produce any errors. The same tests will be conducted starting with the admin interface and switching to the member interface. This test will ensure the member component works when switching to it from the admin component.

## 5.3 Provider and Admin Functionality

This test will ensure the components that makeup the provider and admin interface work when used together. Each option in the provider interface will be tested before moving into the admin interface. Each option in the admin interface will then be tested to ensure the transition from the provider component to the admin component won’t produce any errors. The same tests will be conducted starting with the admin interface and switching to the provider interface. This test will ensure the admin component works when switching to it from the provider component.

## 5.4 Component Functionality

This test will ensure all components work together regardless of transition. Test 5.1 will be run before switching to test 5.2. Upon completion, test 5.2 will be run before switching to test 5.3. Finally, test 5.1 will be run before running test 5.3. These tests will ensure that each component works with every other component in the system regardless of which component ran before it.

## 