Term Project: *ChocAn*

Test Plan Document

Table of Contents

1 Introduction 3

1.1 Purpose and Scope 3

1.2 Target Audience 3

1.3 Terms and Definitions 3, 4

2 Test Plan Description 5

2.1 Scope of Testing 5

2.1.1 Unit Tests 5

2.1.2 Integration Tests 5

2.1.3 System Tests 5

2.2 Testing Schedule 5

2.3 Release Criteria 6

2.3.1 Information Integrity 6

2.3.2 Performance 6

2.4 Testing Constraints 6

3 Unit Testing 7

3.1 Strategy 7, 8

3.2 Unit Tests 8

3.2.1 Entity Class 8

3.2.2 Patient Class 8

3.2.3 Provider Class 9

3.2.4 Service Class 9

3.2.5 Transaction Class 9

3.2.6 Database Wrapper Class 9

3.2.7 Terminal 9

4 Smoke Testing 10

4.1 Provider Terminal 10

4.1.1 Process Service Transaction 10

4.1.2 Check Patient's Membership Status 10, 11

4.1.3 Show Services Directory 11

4.2 Manager Terminal 11

4.2.1 CRUD Service 11

4.2.2 Print/Send Reports 11

4.3 Operator Terminal 11

4.3.1 CRUD Patient 11

4.3.2 CRUD Provider 11

5 System Testing 12

5.1 Provider Testing 12

5.2 Manager Testing 12, 13

5.3 Operator Testing 13

# Introduction

This document describes the processes that will be used to test CA during its development. It discusses how the tests should be performed and what specific criterion are needed to satisfactorily pass these tests. In order to formulate a comprehensive plan, the unit tests, smoke tests and system tests have been outlined and analyzed in the following pages.

## Purpose and Scope

The purpose of the test plan document is to specify a structure for testing different aspects of the software being created and the system as a whole. The focus will be primarily on unit tests and smoke tests. Unit tests will describe the parameters each method must meet for implementation, where smoke tests will follow various user-system paths to assess required functionality. Furthermore, system tests will verify the system as a whole. The scope of testing will be limited to the CA system components as virtualized through a command-line interface.

## Target Audience

The target audience for this document is the CA development team.

## Terms and Definitions

**CA**–Chocoholics Anonymous, the software product specified in this document.

**CA Data Center–**The server hosting the CA database within the data center, owned and operated by ChocAn.

**CRUD**–Create, Read, Update, Delete. The responsibilities of CA.

**EFT Report** – A record of services charged by Providers during a one-week period.

**Provider Report --** A report of all services provided by a Provider during a one-week period.

**Provider Directory**–A list of Services provided by ChocAn.

**Software Development Team**–The group of programmers in charge of creating the software.

**Summary Report**–A report of providers providers to be paid, number of consultations and total fees.

**Submodules**–Encapsulated object groups within CA. Developed by teams of two.

1) Entity, Patient, and Provider

2) Service and Transaction

3) DatabaseWrapper

4) Terminal interface. Provider, Operator, and Manager terminals. User input and output

# Test Plan Description

## Scope of Testing

CA will follow the test-driven-development (TDD) model. Unit tests will be written before implementation to encourage high-quality code and minimize code regression. We will have three levels of testing as outlined below:

**2.1.1 Unit Tests:**

* Objects: constructor, setters, getters, toString()
* Interface / Menu-system: input error detection, both from user or csv files.

**2.1.2 Integration Tests:**

* SQL queries from DatabaseWrapper. Ensures queries are well-formed and return the desired data.
* Entity, Patient, and Provider object creation from csv files or user-input.
* Transactions verify that all involved Patients, Providers, and Services exist and are active.

**2.1.3 System Tests:**

* Use-cases from end-user perspective. Client validation

## Testing Schedule

11/15/16: Unit tests outlined for all objects. From then on, unit tests shall be run regularly as new code is written.

11/28/16: Submodules shall be integrated by this point. Begin integration tests.

12/01/16: System Tests shall be completed by this point. Verification and validation complete.

## Release Criteria

**2.3.1 Information Integrity**

The CA system maintains confidential patient information that must be protected. Operations involving Operator and Manager logins and the retrieval of patient records shall be error free to protect patient privacy. In addition, the CA system shall validate all data inputs to ensure that no database corruption occurs.

**2.3.2 Performance**

The CA system shall perform its operations quickly and consistently. Performance is particularly important when printing Patient, Provider, and EFT reports since large amounts of data may be collated simultaneously. CA should maintain at least 98% uptime while returning queries and updating the database.

## Testing Constraints

Since the CA system is a simulation, Provider, Operator and Manager terminals will be tested in a single console application with a menu-based system. The terminals will not be tested independently or remotely. In addition, no tests tests for database corruption will be performed beyond those provided by JDBC.

# Unit Testing

This section describes the unit tests and unit testing strategy that will be used during the implementation of the CA software. The units tested shall include:

## Strategy

The CA development team will use test-driven development. For each object class or program unit, unit tests will be written before the coding begins. Unit testing will continue throughout development and will be used for regression testing in later stages of the project.

While developing an object class or program unit, developers will work on a branch off of master. The unit tests will run with each build of the code, and developers will only make pull requests to the master once their implementation is complete and all unit tests have passed.

The CA team will use the JUnit unit testing framework to manage tests. All tests will be shared across the development team using git and GitHub.

The CA development will have at least 85% code coverage of object classes and program units with unit tests. Time permitting, a tool will be used to measure code coverage.

For the CA project, unit testing will focus solely on object classes. Unit tests will not have any dependency on external systems or between units. Integration tests will be used once the fundamental units are completed, and will test their interaction. In particular, these tests will focus on database queries and file system interactions.

Unit tests will not cover JDBC or SQlite internal operations. These projects are well established and beyond the scope of unit testing for this project.

**3.2 Unit Tests**

**3.2.1 Entity Class**

The Entity class is the superclass of Patient and Provider, and provides most of the functionality for its subclasses. This includes all getters, setters, and printing functions. Since the Entity class is an abstract class it cannot be tested directly. The Patient and Provider class each test methods inherited from Entity, as well as any additional methods they possess.

The unit tests for inherited functionality are described in this section. The getters for idNumber, name, address, city, state, zipcode, and status are verified with unit tests to ensure they output values match those set by the setters. The setters for ID, name, address, city, state, zipcode, and status, verify that setters correctly set and validate members. The unit tests ensure that setters properly handle all invalid data. Examples of invalid data include invalid input lengths and letters in zipcode. For each subclass, unit tests ensure the correct output is produced by the toString method. Note that each subclass has a different output. The “helper” methods for Entity are tested while calling the unit tests described above.

**3.2.2 Patient Class**

The unit tests for the Patient class test the getters, setters, and toString methods of the Patient class as described in the previous section. In addition, the getters and setters for financialStanding are tested.

**3.2.3 Provider Class**

The unit tests for the Provider class test the getters, setters, and toString methods as described in the previous section. The Provider class does not have any additional methods or members.

**3.2.4 Service class**

The unit tests for the Service class test the getters, setters, and constructors. For the constructors, instantiation and data validation are tested. In addition, the toString override, equals override, and private helper function compareStrings are tested, including any conditional branching within them.

**3.2.5 Transaction class**

The unit tests for the Transaction class test the getters, setters, and constructors. For the constructors, instantiation and data validation are tested. In addition, the toString override and helper functions getTodayDateLong, isValidShortDate, and isValidLongDate are tested, including any conditional branching within them.

**3.2.6 Database Wrapper class**

The unit tests for the DatabaseWrapper class test getter methods. The unit tests also verify that Patients, Services, Providers and Transactions are added, updated and removed correctly to and from the database by DatabaseWrapper methods.

**3.2.7 Terminal**

Since the Terminal is an interactive, menu-based system, the majority of its functionality will be tested using bash scripts with predefined use cases. The output from these scripts will be diffed against an expected outcome to determine success or failure.

# Smoke Testing

This section describes the smoke tests that will be used during the implementation of the CA software. The purpose of the smoke tests is to determine if CA provides the critical functionality described in “Requirements\_Doc\_FINAL\_10.18.16”.

At a high level, CA will be tested using a series of bash scripts that simulate real use cases as if an end-user was using the software. For example, the script “Create a New Patient” will take an input *xyz* and should return successfully. The scripts will pass a series of commands and data in *xyz* to CA, and CA’s response will indicate success or failure.

The specifics for each use case are described below.

## Provider Terminal

### 4.1.1 Process Service Transaction

A script will run the sequence of commands necessary to create a service transaction. The first attempt will be with valid input, then a second with an erroneous input. In both cases, the database response will be compared to the expected response.

### 4.1.2 Check Patient’s Membership Status

A script will check a Patient’s membership status. It will first attempt to retrieve a patient known to be in the database to ensure the database returns objects correctly and the terminal displays properly. A second attempt with an invalid idNumber will be made to ensure the database detects an erroneous input.

### 4.1.3 Show Provider Directory

A script will attempt to list the Provider Directory. Success is signified by displaying the Provider Directory from the database.

## 4.2 Manager Terminal

### 4.2.1 Create, Update, or Remove Services

A script will attempt each action in turn, using database queries to confirm that the database was updated successfully. Success is indicated by the absence of error messages. A second pass will test fail conditions due to a non-existent or invalid service.

### 4.2.2 Print Reports

A script will attempt to access Patient and Provider reports separately. It will then attempt to send Patient and Provider reports separately. Success is indicated by the absence of error messages.

## Operator Terminal

### 4.3.1 Create, Update, or Remove Patient

A script will attempt each action in turn, using database queries to confirm that the database was updated successfully. Success is indicated by the absence of error messages. A second pass will test fail conditions due to a non-existent or invalid service.

### 4.3.2 Create, Update, or Remove Provider

The same process will be followed as in section 4.3.1.

# System Testing

In this section we describe the testing of overall system functionality. Testing will be performed by live testers using the CA terminal, and will ensure that each terminal operation performs as expected. Each system test will be performed multiple times with good inputs, bad inputs, and inputs to capture edge cases. System testing will cover the Provider, Manager, and Operator terminals.

## Provider Testing

Our system testing for the Provider terminal ensures that each path the provider chooses has the correct output for a given input. Below are options and suboptions that must be tested. If available, each sub-option will be verified with a separate system test.

* List Providers
* Check Patient Status
* Create Consultation
  + Create Transaction(s)

## Manager Testing

Our system testing for the Provider terminal ensures that each path the provider chooses has the correct output for a given input. Below are options and suboptions that must be tested. If available, each sub-option will be verified with a separate system test.

* Service Management
  + List Services
  + Add Service
  + Update Service
  + Delete Service
* Report Management
  + Print Weekly Summary
  + Print EFT
  + Print Patient
  + Print Provider

## Operator Terminal

Our system testing for the Provider terminal ensures that each path the provider chooses has the correct output for a given input. Below are options and suboptions that must be tested. If available, each sub-option will be verified with a separate system test.

* Provider
  + Add Provider
  + Update Provider
  + Delete Provider
* Patient
  + Add Provider
  + Update Provider
  + Delete Provider