**JAIL INFORMATION MANAGEMENT SYSTEM**

Review Milestone

Test Plan Report Outline

October 21, 2015

Team Members

Print Name: Joseph Comeau

Signature:

Print Name: David Edwards

Signature:

Print Name: Mike Pasek

Signature:

Print Name: Eric McCoy

Signature:

Print Name: Erik Micklesen

Signature:

Introduction

As a replacement to the current sheriff’s booking and inmate database system, the Jail Information Management System (JIMS) team will provide a more streamlined and user-friendly inmate management system. The JIMS team will provide an easy-to-use inmate database system that will be used by the sheriff’s department, jail system, probation offices, and other authorized users to store and handle the processing of inmate information.

Our upgraded jail information management system will utilize a COTS framework capable of providing a seamless navigation throughout the menu-based user interface. The software system will be capable of storing all relevant information regarding an inmate (ie. name, height, gang affiliation, health problems, etc.) in a secure database, with access via web application. This will allow users to access information regardless of location. After logging in to the system, users shall be able to process inmate property and financial affairs, as well as process the length of each inmate’s sentence.

The system will rely on a standard password protected validation, and limit users to only accessing data to which they are allowed. Entry of information into the system will be checked for validity, and any improper input shall result in an error message. Some fields of entry in JIMS will allow free-form text entry, to help specify/clarify any issues regarding that area of interest. Because JIMS will contain private information, maintaining a secure system is vital. With the ability to support multiple users, JIMS must be able to handle and respond quickly to several users in the same environment.

Proper testing will be conducted throughout the development of the system to guarantee proper functionality and reliability. Some of these testing methods shall include: input validation for the login/inmate information entry, access verification to ensure users are only allowed access to data they have privilege, and stress testing to ensure the system maintains functionality if overloaded with multiple users simultaneously.

Software Test Environment

Our test environment (where we will carry out our testing) will include various components ranging from software to hardware.

**Software:**

The product is designed to be OS independent, but it is preferred to run the software on a Unix/Linux distro due to the stability and security of the OS. For this reason, our test environment will be designed for a Unix/Linux system for our first increment, but in the future it will be adapted to support other OSs, i.e. OS X and Windows. The product is developed using HTML/CSS, Python, and JavaScript, so the appropriate dependencies must be installed on the system in order to test the software. To be able to run python scripts, the latest version of Python must be installed on the system. The benefit of using Unix/Linux distro is that they usually come pre-installed with Python 2.7.3, which is sufficient to run our Python test cases. Python comes with a unit-testing framework referred to as PyUnit, which we are using to write our test cases. The test cases can be performed using a CLI (command line interface) or using an IDE. The CLI requires no additional software tools, but if the tester prefers to use a GUI to assist him or her in the testing, it must be downloaded and installed in addition to Python. We recommend using PyCharm or Eclipse with the Python plugin. For HTML/CSS and JavaScript, you simply need a code editor and a browser. Any text editor will suffice, but if the tester decides to use PyCharm or Eclipse they can use these IDEs to write their HTML/CSS and JavaScript. Unix/Linux come with some default CLI text editors, such as vi and emacs. You will need access to a browser (Chrome, Firefox, Internet Explorer) in order to test the user interface aspect of the product. We will be using Selenium WebDriver to automate our UI test cases. To use the Selenium, you must download the python version at <http://www.seleniumhq.org/download/>. Along with unit PyUnit framework and Selenium WebDriver, we will be using GenerateData to obtain sample inmate data to plug in to our database to ensure that the data is being stored appropriately. You can find a step-by-step instruction to setup GenerateData at <https://www.percona.com/blog/2014/02/10/generating-test-data-mysql-tables/>.

**Problem Report System:**

To document our development and to keep track of bugs discovered we will be using Bugzilla, which is a defect tracking system that allows developers to keep track of bugs in a product. Bugzilla gives us the ability to prioritize issues we run into and help manage the quality of our product. It gives us a central location to see the latest issues discovered by a team member, increasing efficiency in communication and development.

**Hardware:**

The hardware requirements are not demanding due to the fact that python can be run on virtually any platform. The basic requirement is that the hardware supports one of the following OSs; Unix/Linux, OS X, or Windows.

Test Descriptions

***JIMS system framework tests:***

Tests the functionality of the COTS web application used as the JIMS framework. We used graph-based testing to ensure proper navigation throughout the system. After proper login, our menu-driven interface shall produce correct redirection from its current page to the next page. The requirements tested for this are be 1.8.a and 2.1. The pass/fail criteria for this test would be correct navigation throughout the system. This means that when a link is clicked by a user (ie. Inmate Info is clicked from the Main Menu), the user will be directed to the proper page. In conducting this test, we are assuming that the whole JIMS framework is established. Because this is the first increment, we will not have completed certain sections, such as Financial Records and Medical Information. Therefore, we will most likely constrain our tests to ensure proper navigation from a limited amount of pages, such as the Main Menu, Inmate Information, etc.

***User input tests:***

Tests the validity of the input entered by users. For many of the inmate information fields, users will be able to enter text. In order to ensure that proper information is entered into the respective field, we used equivalence partitioning and input validation and syntax testing. Equivalence partitioning was used to check the length of the characters entered, or possible ranges in values (ie. divide height and weight into several partitions). Input validation and syntax testing were used to verify that proper data types were entered (ie. numbers would be invalid in a name field). The requirements tested for this are 1.6.a, 3.2.a-3.2.g, 3.2.i, 3.2.n-3.2.s, 3.2.w-3.2.z, 3.2.bb-3.2.kk, 3.2.mm, 3.2.oo, and 3.5.f. The pass/fail criteria would be if invalid information were entered, the system should return an error message and indicate where the error occurred. For the test, we will assume that users will not use unexpected characters into the fields (ie. percent signs, asterisks, etc.). While our current tests will not catch those special characters, later increments shall be more comprehensive and test for such issues.

***User-interface/Demonstration tests:***

Tests the user interface and functionality of the system components. Here we used demonstration testing and Selenium in order to show that certain aspects of the system, and the user interface were working properly. The requirements tested were 1.1.a, 1.8.a, 3.7.d, 3.8.a, and 3.14. While we used Selenium to ensure our UI was performing properly, we simply had to demonstrate that certain areas were functioning accordingly. The pass/fail criteria was simply if the portion of the system was working properly. We assumed that the outcome for each area we tested would yield only two outcomes, correctly or incorrectly.

***User-maintained lists tests:***

Tests the functionality of user inputs according to user-maintained lists. For these tests, we used comparison testing. We used this approach because we were comparing the expected, or known, outputs against a set of predetermined values. The requirements tested were 3.2.k-3.2.m, 3.2.p, 3.2.s-3.2.v, 3.2.x-3.2.bb, 3.2.gg, 3.2.jj, 3.2.ll, 3.2.nn, and 3.7.a. The pass/fail criteria for these tests were if the results matched the expected results. For these tests, we assumed that each field being tested had a limited number of acceptable answers. Because the lists are predetermined, we already know what should occur when an option is selected. Therefore, we can compare the results of when each option is selected by the user to the results we know.

***User-System interaction tests:***

Tests the interactions between the user and system. For these tests, we used usage-based tests in order to ensure proper user-system interaction. Most of the items tests were boolean questions for fields (ie. yes/no, male/female, etc.). The requirements tested were 1.4.b, 3.2.h, 3.2.j, 3.4.a, 3.5.a-3.5.e, 3.5.g-3.5.h, 3.6.a-3.6.l, 6.1.a, 6.1.c-6.1.e. For these tests, each field was given two options. The pass fail criteria was if the user chose either option, the system would respond correctly, without any errors. Because many of the items we were testing were very similar and we were limited on time, we constrained this test to about half of the items. We thus assumed the system would yield similar results for the other half.

***Database functionality tests:***

Tests the proper functionality of information entered into JIMS. For these tests, we used database testing. The requirements tested were 1.4.a, 3.1, and 3.3. These tests were conducted to ensure the databases used for storing information in JIMS was properly functioning, and that info was going into the proper database. The pass/fail criteria was checking if information entered was stored properly, and in the correct location. Because this is the first increment still, we have not fully developed certain parts in JIMS. Therefore we constrained our tests to only test information regarding the first increment.

Data Recording, Reduction, Analysis

**Overview**:

For the first increment, the data that is being entered into the system include a wide variety of information, such as name, date of birth, sex, Social Security Number, physical appearance, and address. When reviewing the defined requirements for the first increment, our team realized the broadness of the input data and has made it a priority to figure out a sufficient way to test all possible options that a user could enter. We have come up with a combination of techniques to use to ensure that our test cases cover the complete data set with out exhausting the tests. The techniques that will be used are Equivalence Class Partitioning, Boundary Value Analysis, and Comparison Testing. In order to keep track of the data tested, we will log the results in a spreadsheet. We would like to run weekly tests, so that we can compare the results overtime to see how they change as we continue through the development process.

**Brief Summary of Possible Input Types**:

General Text-Based data – name, appearance, comments, descriptions

Boolean data – medical conditions, sex

Numerical data – SSN, Department/Officer ID, weight, height, date/time

Unique data – addresses, license number, phone number, arrest charges

**Techniques used to reduce dataset**:

Equivalence Class Partitioning – *based on dividing the input into several classes that are deemed equivalent for the purposes of finding errors*. For general text-based data, we decided to use equivalence class partitioning to test the data. For example, to the test inmate names we used name generator software to generate a large list of names. We then divided those list of names based on the length of the name. We would have two different sets of classes, one for male names and another for female names.

|  |  |
| --- | --- |
| **Class** | **Representative** |
| Inmate Name | Empty |
| Inmate Name w/ Middle Name | Length > 20 |
| Inmate Name w/ Middle Name | Length < 20 |
| Inmate Name w/o Middle Name | Length > 20 |
| Inmate Name w/o Middle Name | Length < 20 |

Boundary Value Analysis – *uses the same classes as equivalence partitioning, testing at the boundaries rather than just an element from the class*. We used BVA to test our numerical data. Although there was not set range for some of the numerical data, we created the classes based on a realistic scale. For example, when using BVA for weight, we created two sets of classes one for women and another for men. We set a minimum value (0) and a maximum value for the weight. Example for men:

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **All Cases** | **Belonging Cases** | **Reduced Cases** |
| 0-90 | -1, 0, 1 89, 90, 91 | 0,1 89,90 | 0 90 |
| 91-180 | 90,91,92 179,180,181 | 91,92 179,180 | 91 180 |
| 181-270 | 180,181,182 269,270,271 | 181,182 269,270 | 181 270 |
| 271-360 | 270,271,272 359,360,361 | 271,272 359,360 | 271 360 |
| 361-450 | 360,361,362 449,450,451 | 361,362 449,450 | 361 450 |

Comparison Testing – *comparing the data against actual results, highlighting the differences between expected and actual results.* Due to the fact that a majority of the unique input data is generated from a list maintained by the Jail, we can use a comparison testing to ensure that the data entered by the user is valid. With the use of equivalence class partitioning to divide up the possible inputs provided by the Jail System, we would test a random valid and invalid input for each class to get a general sense of whether the system is responding appropriately. Also, we are using a form of comparison testing for data that had to be in a certain format, such as social security numbers, addresses and date/time. We compare the inputted data to the set format to make sure everything aligns properly.

Test Conditions and Schedule

\*\*See OpenProj schedule\*\*

Requirements

\*\*See Requirements Database spreadsheet\*\*