CTC Test Plan

On Track Trainwreck

Meyling Taing

1 Introduction

1.1 Scope

This is the component test plan for the Centralized Traffic Control of the North Shore Extension transit system. This document shall include the description of testing throughout the system. Specifically, this document will emphasize individual unit testing and component testing, along with interface as well as load testing. This plan will also demonstrate that if the software does fail, it will do so in a safe manner. The purpose of the CTC Test Plan is to test the CTC to ensure that the component functions as is expected by the Software Requirements Specification and that it is what the customer wants. All software that is written by the members of On Track Trainwreck will be tested. Any methods taken directly from the Java standard library will be assumed to work and will not be tested individually.

1.2 References

1.2.1 External References

1.3.1.1 IEEE Standard 829-2008 – Standard for Software and System Test Documentation

1.2.2 Internal References

1.3.2.1 Software Requirements Specification for Port Authority North Shore Extension

1.3.2.2 CTC Use Cases

1.3 System overview and key features

The CTC is one component of the North Shore Extension transit system. It is responsible for providing the main user interface to the system with high level features of editing tracks, opening and closing tracks for maintenance, suggesting setpoint and authority, and monitoring the entire system. The CTC is mainly used by a dispatcher. For a more detailed list of functions of the CTC and the entire transit system, refer to the SRS.

1.4 Test Overview

1.5.1 Organization

Unit, component, and system testing will be done during development to ensure that the software's functionality works as expected. The processes of creating test plans for each unit and developing each unit will be interleaved. There will be a main test program that tests methods from all of the components in the system and this will be added to and run every single time a single update is made to the codebase. Code will only be committed to the central repository if all of the tests completed successfully. The successful completion of tests will allow the team to measure progress.

1.5.2 Test Schedule

Testing for the CTC will be done in several parts. During initial development of the CTC, unit and component testing will be interleaved and done in an incremental fashion. Tests will be written for each method implemented. The steps below are taken for each increment.

1.5.2.1 Choose use case to develop

1.5.2.2 Develop component interface test for that use case

1.5.2.3 Break use case into individual Java methods and variables

1.5.2.4 Write tests for each method which includes possible inputs and expected outputs (variable values).

1.5.2.5 Develop method

1.5.2.6 Test method with tests already written

1.5.2.7 Integrate method with other methods in use case. Keep developing and testing methods until all of them are done.

1.5.2.8 Perform overall use case test.

The CTC also has many interactions with the other modules of the system. Once all of the six components have enough functionality to integrate with each other, they will all be integrated together to form the whole system. A new tester program will be made that already includes all of the tests from the individual components, and this will be built on as the methods involving interaction are added.

Once the system is considered complete, final testing will be done. All of the use cases will be checked to make sure they follow their descriptions. Further attempts will be made to break the system.

1.5.3 Resources and Responsibilities

Each of the six members of On Track Trainwreck will be responsible for a different component of the system. The components are CTC, Train Model, Track Model, Train Controller, Track Controller, and Moving Block Overlay. Each member will do unit and component testing for his/her component. All of the members will work on system testing together. JUnit integrated into Eclipse will be used as a testing framework. A github repository will be used to store all of the current working code.

1.5.4 Tools, techniques, methods, and metrics

Unit testing will comprise of at least one test for every method. First, if possible, an individual test will be done to ensure that the method works on its own. For testing that the new method works with the rest of the CTC code, a tester program will be used. More tests will be added to the tester program as more individual units of the CTC are implemented. Every time a new method is added, the new test as well as all other tests will be run and checked for correctness. The tests will be written so that they can all be run in a single step, and so that the result is a definite success or failure. Component testing will check that all of the use cases for the CTC are satisfied. These tests mostly deal with user interface so that will involve manually running the program.

2 Details of the Test Plan

2.1 Component Testing

Since the Use Case diagram shows the highest level view of the CTC, the component interface tests will be described first. The subsequent sections will describe a few of the use case test plans and the metrics used to determine success or failure. More test scenarios will be documented as each unit is developed. In each use case, the following will be defined: the initialization and process of the use case, the inputs to be tested (if any), and the success criteria. Each of these tests will have to be done manually because they test the component interface.

2.2 Use Cases

2.2.1 Start up

2.2.1.1 Process

The user of the system starts the North Shore Extension Transit System application.

2.2.1.2 Success Criteria

2.2.1.2.1 Login prompt appears and user is able to type in the username and password fields.

2.2.1.2.2 The password field is hidden.

2.2.1.2.3 The main display is visible, but the buttons are all disabled.

2.2.2 Log In

2.2.2.1 Process

The application has been started. The user types in a username and password and hits submit.

2.2.2.2 Different Inputs

2.2.2.2.1 The user inputs a username that is in the user database and the password for that user is correct.

2.2.2.2.2 The user inputs a username that is in the user database and the password for that user is incorrect.

2.2.2.2.3 The user inputs a username that is not in the user database.

2.2.2.2.4 The user does not input anything and hits submit.

2.2.2.2 Success Criteria

2.2.2.3.1 If the user inputs a username that is in the users database and the password for that user is correct, the login prompt closes and the buttons on the main display are enabled.

2.2.2.3.2 Any other type of input results in an error message being displayed, and the login prompt remains.

2.2.3 Add Track

2.2.3.2 Process

The application has been started and the user has logged in. The user clicks on Edit Tracks and then on Add Track. Then the user enters a csv file with all the necessary track information in the correct format and hits Submit.

2.2.3.2 Different Inputs

2.2.3.2.2 The user has track creator permissions.

2.2.3.2.3 The user does not have track creator permissions.

2.2.3.2.4 The user enters an incorrectly formatted track file.

2.2.3.2.5 The user enters a physically impossible track file.

2.2.3.3 Success Criteria

2.2.3.3.1 If the user does not have track creator permissions, then the Add Track button is disabled so the user cannot click on it.

2.2.3.3.2 If the user has track creator permissions and enters a valid track file, then the prompt will be closed and the track layout will be displayed on screen.

2.2.3.3.3 If the user enters an invalid track file, then an error message will be displayed and the prompt remains.

2.3 Unit Testing

Once a single component test plan is created, the Java methods and the variables needed for that test plan are decided upon. At least one test is done for each Java method that is to be implemented. Each test checks the value of variables that may be modified during the process of executing the method. These tests are done automatically.

2.3.2 Parts of a unit test

2.3.2.1 Method

2.3.2.2 Preconditions

2.3.2.3 Inputs

2.3.2.4 Assertions

2.4 Unit Test Example

2.4.1 login(String username, String pw)

2.4.1.1 Preconditions

users.csv file contains line with following: root,password,1,1,0,1

2.4.1.2 Inputs

username = “root”, pw = “password”

2.4.1.3 On success, all of these conditions must be true

currUser != null

currUser.name.equals(root)

currUser.isAdmin()

currUser.isTrackCreator()

!currUser,isMaintenance()

```
currUser.isDispatcher()
```

2.5 Test Documentation Requirements

In the test results document, each test will be included with the information described above, as well as if it passed or failed, when it was run, and how many times it was run.

2.6 Test Administration Requirements

2.6.1 Anomaly Resolution and Reporting

If during testing, any result is found to be unclear, the test plan will be re-examined and possibly re-written to account for any extra cases and details that were not originally accounted for.

2.6.2 Task Iteration Policy

All automatic unit tests will be rerun every time a change is made to the code base. If a test fails, the code will be re-examined and adjusted. If a test fails multiple times and a fix cannot be found, the use case will be re-examined and possibly adjusted.

2.7 Test Reporting Requirements

A test log will be created when testing starts and it will be updated every time a test is run. An entry to the log will be added every time a test is run which will include what tests were run, what changes were made to the code, and success or failure.

3 General

3.1 Document Change Procedures and History

If any changes are made to this document, it will be maintained in a log. Each entry will include what change was made, who made the change, and when the change was made.