Low-motovation: Group Test Plan

Michael Ghaben

Contents

1 Introduction

1.1 Document Identifier

This document is the Low-Comotovation Group Test Plan for the design review phase of the Spring 2017 software engineering project.. In this document we detail the testing specifications, requirements, and procedures for the implementation and evaluation of testing procedures.

1.2 Scope

In this document a number of assumptions regarding the projects scope and lifecycle are made. Specifically:

1. Due to the short development cycle associated with this project, some non-negligble defects will likely persist
2. Under the iterative development methodology this group is following, the development of additional tests may be completed to address defects is expected

To better address these constraints, we will utilize a continuous integration methodology around iterative development and testing.So that we may ensure rapid development with minimal time overhead, we shall utilize continuous integration tools and methodology. Testing will be broadly divided into two categories: subsystem, and integration. Subsystem testing will be primarily focused on testing an individual system to ensure minimal functionality of a given subsystem and be done primarily independently by each group member. Integration testing will be designated as tests which require two or more modules functionality to satisfy requirements.

Specifically, it is anticipated that software defects will be found and require that testing which was not anticipated. As a result, we shall primarily focus on defining general tsts to be implemented and leave the exact testing requirements as implementation is completed. Additionally, will primarily focus on subsytems to be delivered to the end-user, the train company.

Examples of these subsystems would be the CTC module or the train controller, which are expected to be integrated into the final deliverable. Subsystems which are not examples of the final deliverables are the Train Model and Track Model subsystems, as they will ultimately be removed and replaced with the physical subsystems.

1.3 System Overview and Key Features

The purpose of the system under development is a to provide a train system for the Pittsburgh North Shore Rail system.

This system broadly consists of 6 subsystems:

1. Track Model - a physical model of a track to be used for testing

2. Train Model - a physical model of a train to be used for testing
3. CTC System - A CTC system to be implemented on the final train system
4. MBO - ???
5. Wayside System - A wayside for coordinating with all models

For a further discussion of the system, we refer the reader to the project reuqirements and discussion board.

2 Test Overview

The test organization is broadly divided into two sections: subsystem and integration testing. Subsystem tests are regarded as tests associated with only a single system at a time. Integration tests broadly refer to the tests of the integration of more than a single subsystem. In this view, testing is accomplished by each subsystem independently at the discresion of the individual developing the subsystem. Then, as members of the team develop their subsystem, each member shall attempt to integrate and develop tests for their integration. Due to the nature of continuous integration, tests are expected to be developed in parallel to the development of the program modules.

2.1 Master Test Schedule

The test schedul will be implemented as follows::

2.2 Integrity Level Schema

For this project, we utilize three integrity levels. The integrity levels for this project are as follows:

1. The lowest severity level. This is reserved for tasks which will ultimately not be passed onto the finished product and pose no threat to catastrophic failure.
2. This severity level is reserved for failures which may lead to errors in subsystem integration or incorrect information delivered to critical subsystems
3. The most severe integrity level. Is a vital system or otherwise threatens life or limb in an imlemented subsystem

2.3 Responsibilities

Michael Ghaben will be responsible for the integrity of the automated build system as well as the integrity of the master branch.

Table 1: Test Plan

Task	Test Design
Methods	How?
Inputs	What Inputs?
Outputs	What is a successful output?
Expected Completion	When will it be done?
Risks and Assumptions	What are you assuming?
Responsibility	Who are you?

2.4 Tools, Techniques, and Metrics

To implement the test environment and better facilitate a continuous integration test environment, we utilize Travis-CI[1] with GitHub[2] integration with Slack[3] and Jupiter JUnit [4] integration. By utllizing these tools, we allow automated tests to be run remotely using the Maven[5]. This allows for rapid feedback into the developmental process, allowing for better integration and testing of the team. The usage of these tools creates feedback loop between implementation, testing, and integration, leading to significant productivity gains. In each case ,testing will be carried out remotely by the build system.

The system shall be quantitatively evaluated on percentage of percentage of test passing. Each subsystem shall be responsible for the determination of importance of testing individual components of their subsystem, with the exception of vital components.

3 Details

3.1 Process

A test shall be detailed in the following manner: In this, we expect to utilize both integration as well as unit tests. After each member pushes to GitHub on his or her respective branch, Travis-CI will provide regression testing on all unit and integration tests that have been implemented. To merge into master, all tests must be passing.

3.2 Test Documentation Requirements

Each test shall be documented using the above table as well as any auxillary information by whomever holds testing responsibilities. Each module shall have a specified test plan for each module covering their individual component for

[1]Travis-CI.org
[2]github.com
[3]slack.com
[4]junit.org
[5]maven.apache.org

testing. Each subsystem test plan shall detail unit tests for his or her own module. Additionally, integration testing will be accomplished in a similar fashon completed by the group.

Furthermore, the unit and integration testing procedure will be supplemented by functional testing. Each group member shall conduct user testing of each other module. During this time, the testing member will attempt to cause defective behavior at any level. These defects will be tracked via GitHub issues. This testing will occur weekly.

3.3 Test Administration Requirements

For a unit or integration test to be considered complete, it must successfully build on the build server utilizing the Maven build system. This ensures consistent repeatable builds to attempt to ensure the clients functional requirements will be met.

Additionally, for functional testing it is expected that each group member submits either a bug report via GitHub issues. Should a group member fail to find any defects and register them, he or she must challenge Professor Profeta to find a defect at the next class meeting. If the professor discovers a defect, the group member(s) who failed to find defects owe the other group members pizza. It is hoped that this procedure will lead to people finding more defects.

3.4 Test Reporting Requirements

Each written unit and integrationtest report will be provided by the Maven automated build system.

To document user testing (e.g. by working with the user interface and attempting to find defects in that manner), a developmer may supplement this test report with the following syntax to automatically document the bug utilizng the following syntax:

/**
* @bug $< Descriptive\ message >$
*/

This will lead to documentation of the defect in the autmatically generated Doxygen documentation. Note that this should be used for defects which are not necessarily covered under tests at a given time. By utilizing this process, an iterative cycle of development these defects may be tracked and inclusion in later tests to ensure proper functionality.

4 Changelog

Table 2: Change

Date	Change
Methods	How?
Inputs	What Inputs?
Outputs	What is a successful output?
Expected Completion	When will it be done?
Risks and Assumptions	What are you assuming?
Responsibility	Who are you?