SE 3XA3: Test Plan
PGH Software Solutions

Team 35, Team Name
Pratyush Bhandari, bhandarp
Gazenfar Syed, syedg1
Hamid Ghasemi, ghasemih

October 26, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Oct 25	1.0	Completing parts 1 and 2 of document
Date 2	1.1	Notes

1 General Information

1.1 Purpose

The purpose of testing our project is to allow us to build more confidence in knowing that our product works as it is intended to.

1.2 Scope

This test plan will provide us with a method to exhaustively test our MovieGuide re-implementation. Our re-implementation must be able to display, sort and update a list of movies from an API. As well as display summaries, trailers and ratings for each respective movie. The objective of our test plan is to prove that the MovieGuide re-implementation can successfully perform these main tasks.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
PoC	Proof of Concept
GUI	Graphical User Interface
SRS	Software Requirements Specification

1.4 Overview of Document

This document aims to formalize an exhaustive test plan for the MovieGuide re-implementation. It aims to arrange testing activities and delegate testing responsibilities among team members. As well as setting the initial groundwork for testing to begin, the document will also cover the specifics of testing certain characteristics of the code and will define what testing methods are to be used to perform each test.

Table 3: **Table of Definitions**

Term	**Definition**
Automated Testing	A series of tests that are run automatically usually by a testing framework
Unit Testing	A Method of testing centred on testing functions or methods
Integration Testing	A method of testing that tests the entire system at once
Static Testing	A method of testing that is done without executing the code
Dynamic Testing	A method of testing that is done while the code is executed

2 Plan

2.1 Software Description

The software that is being developed will allow users to navigate through a list of movies loaded from an API. The user can then click on any movie to view a summary, a rating and watch a trailer. In technical terms, when a movie is loaded from the API, it is parsed into a java Movie object with summary, rating, trailer and title data. This data is then accordingly shown to the user through the GUI buttons.

2.2 Test Team

The team members involved in testing the application are Pratyush Bhandari, Gazenfar Syed and Hamid Ghasemi. Testing responsibilities will be split equally among team members.

2.3 Automated Testing Approach

Our approach to automated testing is bottom-up testing, which in essence tests each component at a lower hierarchy before going on to test the components higher in the hierarchy. As such, there will be a focus on unit testing all of the methods within classes first. Once there is certainty about the reliability of methods in the classes, the focus will shift to integration testing, which will test the functionality of each class. Finally there will be full application testing, which will involve testing the GUI of the application to perform actions that may involve the use of multiple classes.

2.4 Testing Tools

The primary tool to be used for running all of the tests for the MovieGuide reimplementation will be JUnit; this framework will allow us to use Automated Testing to test our program.

2.5 Testing Schedule

https://gitlab.cas.mcmaster.ca/syedg1/MovieGuide_PGH/blob/master/BlankProjectTemplate/ProjectSchedule/3XA3%20Gantt.pdf

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

Input:

Output:

How test will be performed:

3.1.2 Area of Testing2

...

3.2 Tests for Nonfunctional Requirements

3.2.1 Area of Testing1

Title for Test

1. test-id1

   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

3.2.2 Area of Testing2

...

3.3 Traceability Between Test Cases and Requirements

4 Tests for Proof of Concept

For proof of Concept testing we will be focusing on verifying and validating the means by performing automated testing. Automated testing will require the ability to use sort and search for a particular outputs and verify the output. Therefore, testing for proof concept will contain using of sorting and searching for movies and displaying movie information such as rating, summary and its genre.

4.1 Using Sorting and Searching to Display a Movie

1. FC-1
   Type: Functional, Dynamic, Automated
   Initial State: There is an input
   Input: A complete movie name
   Output: All the movies that have the exact same name or their names contain the input name will be displayed on the screen in alphabet order.
   How Test Will Be Performed: One movie name will be picked from the movie list. Then what happens is it will go through the movie list, compares it to each movie name. If there is only one movie with that name the system will be returning one movie, if there are more than one movie, it will sort them based on the alphabet then exhibit those movies on the screen.

2. FC-2
   Type: Functional, Dynamic, Automated
   Initial State: There is an input
   Input: An incomplete movie name as an example (Pu–)
   Output: All the movies that contain the exact same letters in their name will be displayed on the screen in alphabet order.
   How Test Will Be Performed: Two letters will be typed in the movie search designated place. Then the program must compare those letters to each movie name in order to find movies that have those two letters in their names. Then movies must be shown on the screen in alphabet order, unless there is no movie in the list with those two letters.

4.2 Displaying Movie Information

How Test Will Be Performed: In each of following tests, one movie will be chosen and then it will be checked if the information which movie provides is exactly as expected information.

1. FC-3
   Type: Functional, Dynamic, Manual
   Initial State: Set an input
   Input: A movie name
   Output: Provide all the information about the movie that has been picked

2. FC-4
   Type: Functional, Dynamic, Manual
   Initial State: No input
   Input: One movie be randomly picked
   Output: Provide all the information about the movie which has been picked

3. FC-5
   Type: Functional, Dynamic, Manual
   Initial State: Set an input
   Input: an incomplete movie name be randomly chosen
   Output: Provide all the information about the movie that has been picked

5 Comparison to Existing Implementation

There are three tests that compare the program to the Existing Implementation of the program. Please refer to:
FC-1 in Tests for Proof of Concept
//test SS-5 in Tests for Nonfunctional Requirements - Usability
//test SS-10 in Tests for Nonfunctional Requirements - Performance

6 Unit Testing Plan

JUNIT framework will be used to do unit testing for this project.

6.1 Unit testing of internal functions

To create unit test for internal functions of the program, methods that have a return value can be used to be tested. To be able to use those methods, particular inputs would be given to the methods and be checked if the result is similar to expected outputs. Series of unit tests can be created in this way. Unit tests will include tests that contain proper inputs and inputs that generate exceptions. Individually, all the classes will take care of importing everything that is needed for unit tests. This project will not need any stubs or drivers to be imported for testing. In the case of checking how much code we have covered we use coverage metrics. We will test majority of the codes which are not simple functions. Our goal percentage to beat will be 86 percent.

6.2 Unit testing of output files

To construct unit tests, which will test the accuracy of the output movies through searching, we will need movies which must be displayed and what it actually displays as outputs.There are cases that a user searches for a specific movie by inputting movie name and that name is part of the other movie name, in that case output file must exhibit all movies that consist of input name. For this case we will use methods which compare if the output file name and output file are exactly what have been expected. In order for a test case to be considered passed, movies should be at least 99 percent the same. Less than 99 percent and the test case fails. For this set of unit tests the required imports are: java.io.IOException, java.io.File, and java.io.File.compareTo.

References

7 Appendix

N/A

7.1 Symbolic Parameters

N/A

7.2 Usability Survey Questions?

N/A