Test Document

Team 1

8 April 2012

Table 1: Team

Name	ID Number
Jonathon Bergeron	9764453
Marc-Andre Faucher	9614729
Jeffrey How	9430954
Dmitry Kuznetsov	5679311
Willian Ling	9193480
Thomas Paulin	9333630
Alain Sakha	9770836
Kai Wang	5652723

1 Introduction

This testing plan document outlines the testing to be done on the system that ensures that it meets specified requirements in a proper, organized way, and provides a reasonable user experience within those requirements and that feature set.

Specifically, the goals of this document are to formulate a scheduled plan to thoroughly validate that system requirements are met at the system, subsystem and unit levels, along with the inclusion of integration and regression testing, and to list the results of these tests in a manner in which the requirements they meet are known. Additionally, this document aims to verify that the way in which the requirements are implemented provides a certain feature set that has the ability to surprise and delight users.

The information contained in this document includes a brief overview of the forms of testing done on the system, the schedule and resources required and used for testing, brief notes on the design of the test cases, as well as detailed test results at each aforementioned level, along with their results and the system requirements which they are intended to satisfy.

2 Test Plan

The testing planned for the system spans the various levels of production. As the system was tested starting at the unit level, the method of testing was bottom up. Regardless, the tests will be outlined in a top-down fashion, based on the complete design and implementation of the system.

2.1 System Level Test Cases

These system test cases define overarching test cases for the software that are designed to test both the design and behavior of the system and its main functional requirements. Specific, detailed scenarios will be covered in the sections describing subsystem and unit tests.

Test Case 1 System Execution

Purpose The purpose of this system test is to ensure the user can run the program on all platforms capable of running the Java Virtual Machine (JVM).
Input Specification

For this case, there is no input files or methods needed to be called. The test simply is designed to ensure that the application is portable, and thus can execute on any system configuration capable of running the JVM.
Expected Output
The system is expected to be able to run the application regardless of hardware and software configuration, including, but not limited to Windows (XP/Vista/7, 32 bit and 64 bit), Max OS X (10.X), various Linux distributions, etc.
Traces to Use Cases
This system test relates directly to the non-functional requirement O1 from the requirements document (also included in Appendix C) which states It shall be possible to run the system on any computer system that run the JVM.

Test Case 2 System Navigation and Use

Purpose
The purpose of this system test is to ensure the user has access to all system functions, interface options, and generally can perform any given specified requirement of the system. **Input Specification**
For this case, there are no explicit input files that need to be inputted to the system. All system XML files, however, need to be included as expected with the system. The files, people.xml and task.xml, can be found in detail in the Appendix A at the end of this document
Expected Output
The system is expected to be able to run without error, and the user has access to all tabs (Tasks, People, Gantt Chart, and Tree View) and can perform all operations on each tab. These cases will be explained in greater detail in the subsystem and unit level test descriptions, and the only scenario in which an output file is produced can be found in Appendix B (people.txt).
Traces to Use Cases
The system test should validate all use cases, scenarios and requirements implemented for the system. Specifically, the requirements F2 F13 as specified by the requirements document, and included for reference in Appendix C.

2.2 Subsystem Level Test Cases

The subsystem tests for the system are designed to test specific collections of units of functionality within the system. As the system separates requirements by the Model, View, Controller architecture, those will serve as the major subsystems of the system.

Model Subsystem

Purpose

The purpose of this system test is to ensure the data model always maintains integrity and is not affected by the operations performed by user. While data will change, the way the data it stored should not be affected.

Input Specification

For this test case, the input specification should simply be valid format XML files. As long as the system is able to process the files, the data should be stored properly in appropriate data structures.

Expected Output

The system is expected to function using the proper data format in combination with their storage in proper data structures.

Traces to Use Cases

This test case does not directly relate to any functional or non-functional requirements, however is implicitly essential to all requirements and functionality.

View Subsystem

Purpose

The view subsystem is designed to handle all the required display options for the data. This includes the tree view, gantt chart, tables, etc.

Input Specification

For this test case, the input specification should simply be valid format XML files. As long as the system is able to process the files, the view should be able to accurately display regardless of which option is selected, and how the data is changed in the model.

Expected Output

The system is expected to function by displaying any requested view to the user using the data specified in the model.

Traces to Use Cases

The view subsystem has several direct traces to use cases and requirements of the system. Specifically, functional requirements F2, 6, 7, 10 as indicated in Appendix C and the requirements document. These functional requirements dictate the many ways in which the data can be displayed graphically to the end user.

Controller Subsystem

Purpose

The controller subsystem is designed to perform operations on the model as dictated by the operations the user performs on the view. This includes update XML files, output text files, etc.

Input Specification

The input XML files should simply be of valid format. Any data combinations

should work. The workflow process can begin at one of many points, including the addition/removal of tasks or employees, modification of employees, etc.
Expected Output
The system is expected to function by appropriately modifying data stored in structures, and any needed XML files. Specific units need to be tested as the process continues.
Traces to Use Cases
As evidenced by its purpose and input/output specifications, the controller subsystem is related directly to several functional requirements. These requirements are as follows: F3, F4, F5, F8, F9, F11, F12, F13, all described in Appendix C.

2.3 Unit Test cases

The unit and their tests desrcibed in this section represent the individual methods of the system, most of which are small and inputs predictable. Details follow.

updateXML() Test for if people or task are empty;

getTaskData() Test if a task object is empty or null

getPeopleData() Test if a person object is empty or null

getTotalHoursOnProjects() Test if the total hours on a project is negative, or if the testing total hour is same as expected value.

getListOfProjects() Test if the project list is empty or null, and check if the output vaue is the same as expected value.

nextAvailableId() Test if the nextAvailableID() is proper value or null

findTask() Test if by using the taskID, we can get the expected task

assignStringOfID() Test if the assigned ID is the expected ID.

setIdentifier() Test if the identifier fits its proper type pattern.

testSetFName() Test if the first name fits its proper type pattern.

testSetLName() Test if the last name fits its proper type pattern.

testSetJobTitle() Test if the job title fits the proper type pattern. **testSetJobDescription()** Test if the job description fits the proper type pattern. **testSetTotalHours()** Test if the total hours set for a project is an integer number.

setCompletion() Test if completion is a percentage number.

setDeadline() Test if the deadline is a valid date.

setDeliverable() Test if the deliverable follows correct type pattern.

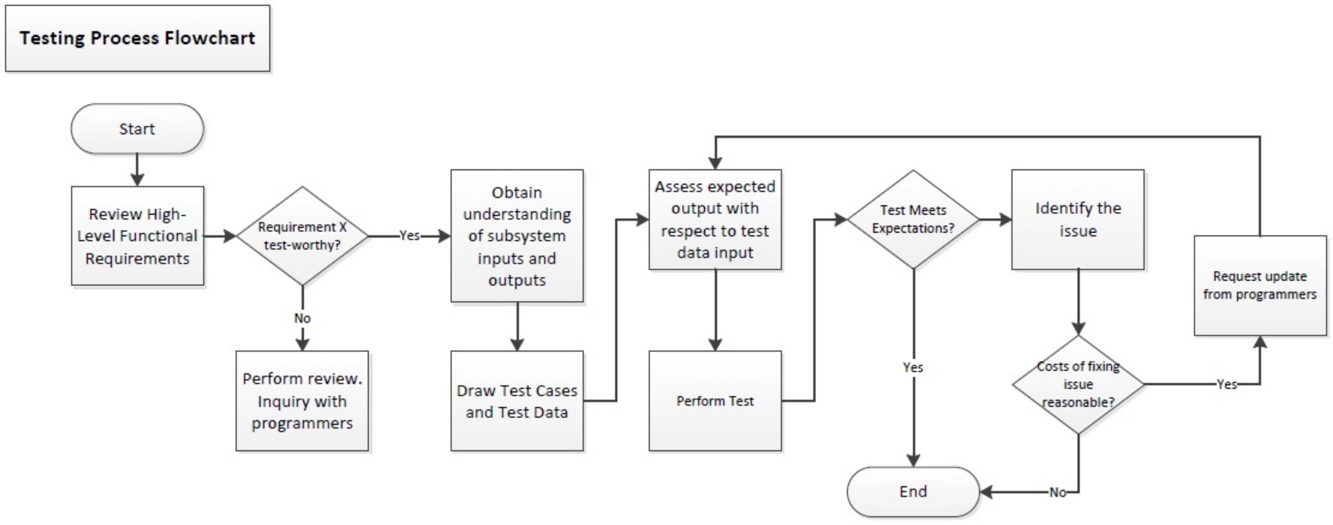
setDescription() Test if the description follows correct type pattern.

setDuration() Test if duration is an integer number.

setIdentifier() Test if the Identifier ID is an integer number.

setPeopleAssignedarray() Test if all the ID's of people assigned are integers.

setTitle() Ensure that the title being set is valid according to its type.



Testing Process Flowchart
Start
Review High-Level Functional Requirements
Requirement X test-worthy?
Yes
No
Perform review. Inquiry with programmers
Obtain understanding of subsystem inputs and outputs
Draw Test Cases and Test Data
Assess expected output with respect to test data input
Perform Test
Test Meets Expectations?
Yes
End
Identify the issue
Costs of fixing issue reasonable?
Yes
No
Request update from programmers

Figure 1: Sample Flow for unit testing

2.4 Regression Testing

Regression testing performed on the system involved ensuring that all previous code and functionality is not effective after changes made for the current increment. The regression tests were divided into three major test cases, file format validation, MVC architecture validation, and functionality verification.

Test Case 1 XML File Formats Validation

Purpose
The purpose of this regression test is to ensure that the validation methods on the XML data format still function as intended and can handle erroneous cases.

Input Specification
The system state should be normal, reading in the XML files as intended. The regression test includes invalid file formats specified through unit tests. These invalid XML files are attached and part of the Appendix A.

Expected Output
The expected system response is to be able to handle and report appropriate data errors. For most cases, blank fields, unclosed tags on non-critical fields

will be handled by the system in a manner that allows the user to continue on execution and operation of all intended functionality. For certain critical fields, such as user ID, task ID, hours assigned, etc. invalid XML formats will be handled in a way by the system that removes that record from operation, and allows the user to continue operation and execution of the system with other, valid records. In both cases the only system output is console-level error messages, detailing how the system recovered from the error.

Traces to Use Cases

This test case does not satisfy an explicit user requirement or use case, but does satisfy an implicit system requirement, that the XML input files must be readable, and the system must be able to survive data errors in a user-friendly way.

Test Case 2 MVC Architecture Validation

Purpose

The purpose of this regression test is to ensure that the Model View Controller architecture implemented in Increment 2 is not affected by the changes made in implementing Increment 3 functionalities

Input Specification

The system state should be maintained in a way that separates the data from the user interface and the methods that manipulate it. There is no specific input file needed, other than the default, valid XML file needed for normal program execution.

Expected Output

The system is expected to function as normal. There is no specific output other than the user being able to view each view properly as they manipulate data. The pople.txt output file should be produced as normal.

Traces to Use Cases

This test case does not satisfy an explicit user requirement or use case, but does satisfies an implicit requirement that the MVC architecture is used to implement the system.

Test Case 3 Functionality Verification

Purpose

The purpose of this regression test is to ensure that all previous functionality available in previous increments is still available and working correctly with the changes added by the current increment.

Input Specification

The system state should be viewed as normal, that is, a regular execution is expected with all exceptions handled. The default input should be used, which includes generic, valid input XML files.

Expected Output
The expected system response is to be able to perform the functionalities added in Increments 1 or 2, including being able to produce a valid people.txt file, and the view of the people, and tasks table, and any operations able to be

Traces to Use Cases
This requirement has several traces to use cases and requirements, specifically requirements: F2, F3, F4, F5, F8, F10. Detailed information for requirements are available in Appendix C.

2.5 Integration Testing

Implementation testing was performed as components were added to the system. While there are no specific test cases, the regression test were run as components were added.

As stated earlier, the system was implemented in a bottom-up manner, meaning that modules were slowly coupled together, and implemented beginning at the unit. Below is a diagram of the way in which the modules of the system were integrated:

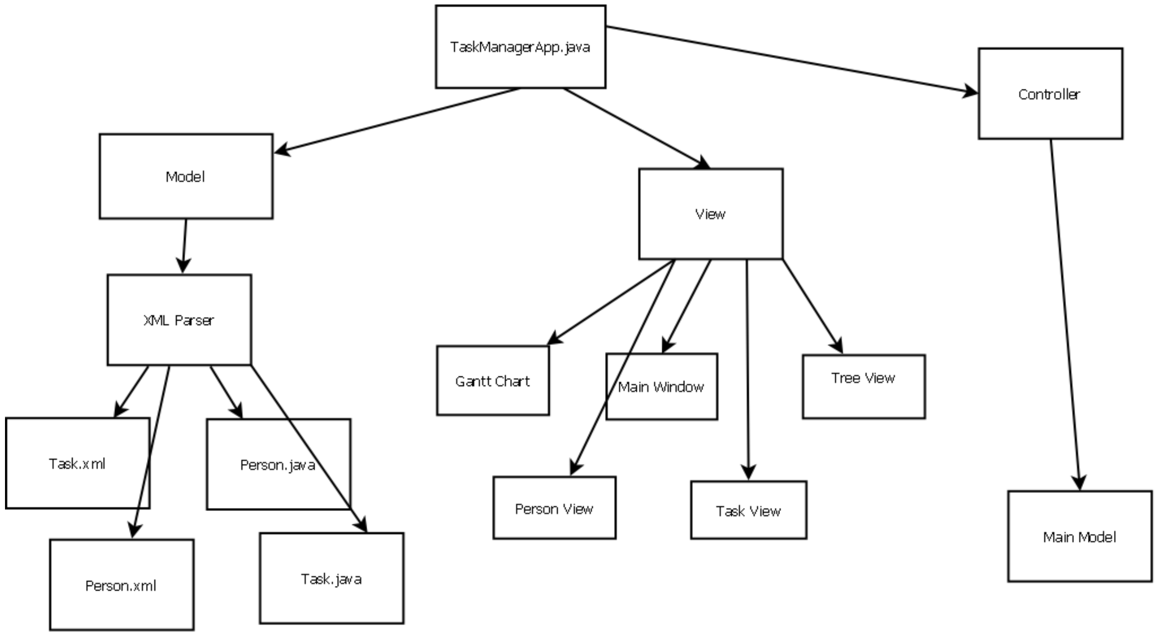


Figure 2: Bottom-Up Integration of System

2.6 Schedule and Resources for Testing

This section defines the schedule and resources used in the testing of the system.

2.7 Notes on Test Case Design

The test cases were designed to be operated within the confines of the MVC system, meaning that each case was written and performed keeping the notion of a separate Model, View and Controller in mind. This means that test cases need to ensure that each separate portion of the system maintained integrity both as a whole, and separately.

2.8 Requirements Tested and Omitted

Requirements Tested

The requirements tested in the development of the system are listed in the requirements document and in Appendix C of this document. The specific requirements are labeled functional requirements F2-F10, F13, as well as operational requirement O1, with the omission of functional requirement F1, F11, F12 covered in detail below.

Requirements Omitted

The requirements listed in the requirements document (included in Appendix C) omitted from the testing and eventual development of the system are requirements F1 and N1.

Functional requirement F1 states Only one person should be logged in at a time. This requirement was later dropped from development during the development of Phase 2, as it was deemed beyond the scope of the application, as the application functioned better in a more general form. This functionality was thus not tested as part of the requirements.

Functional requirement F11, F12 were discarded, as the notion of administrator users was omitted as part of F1. As this is the case,

Non-functional requirement N1, specifically the system must know who the user is in order to order the appropriate options, was thus rendered invalid, as the entire notion of different users using the system was discarded.

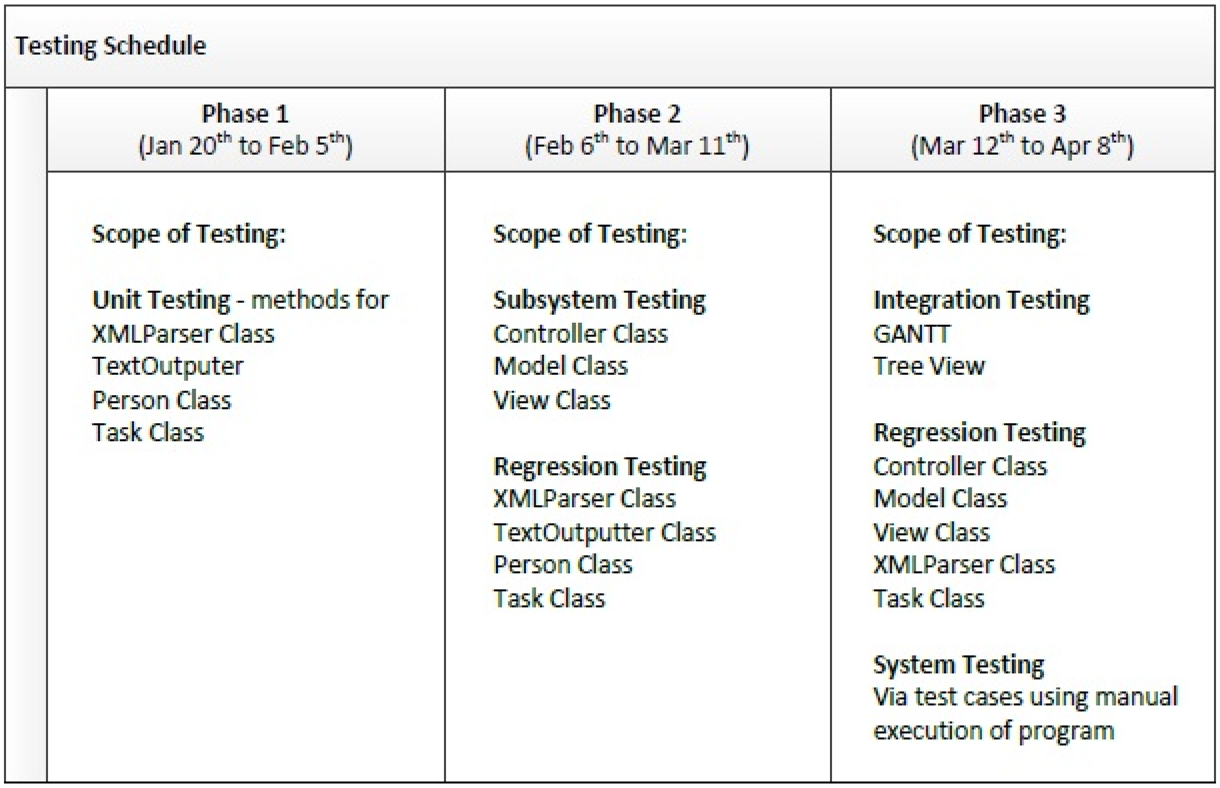
Testing Schedule

Resources Used

3 Test Results

Objective

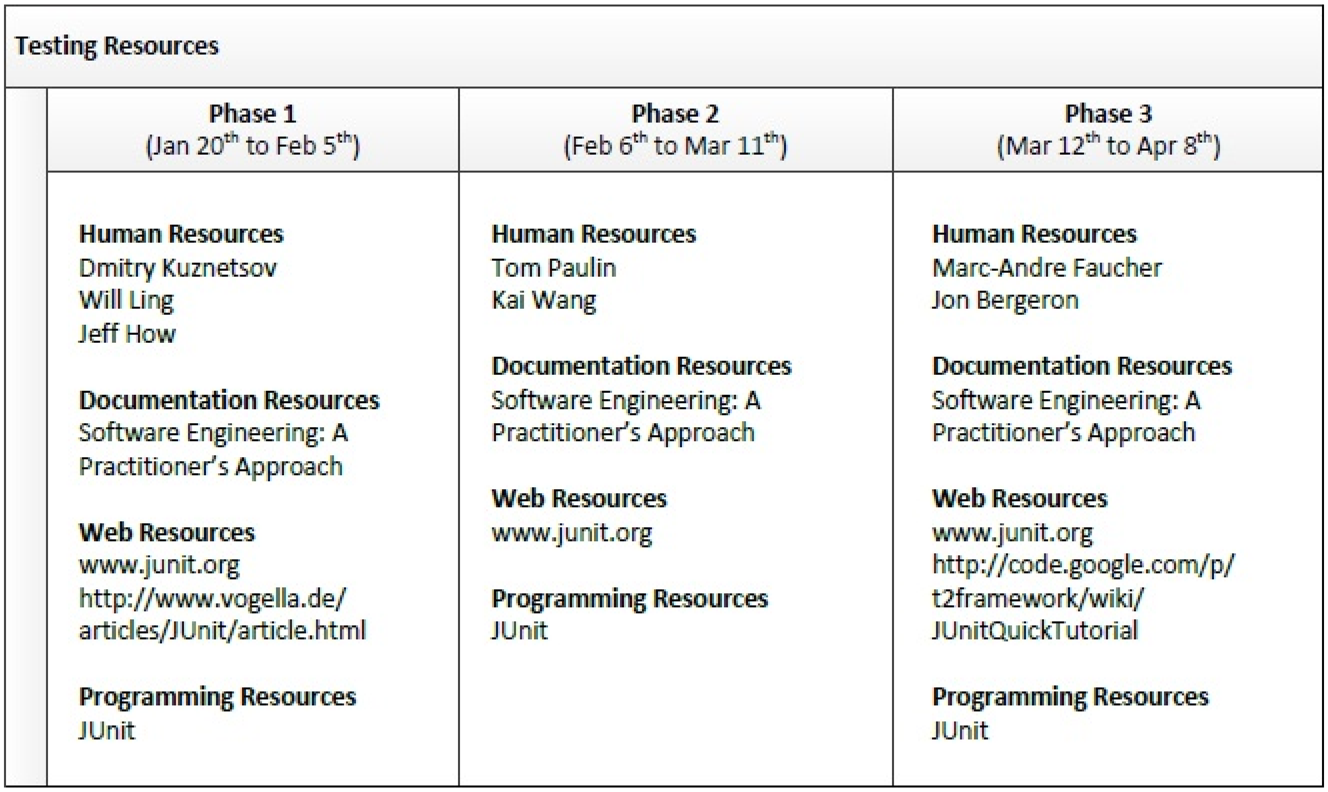
Test system requirements to verify that they are operational and met by the implementation.



Testing Schedule

Phase 1 (Jan 20th to Feb 5th)	Phase 2 (Feb 6th to Mar 11th)	Phase 3 (Mar 12th to Apr 8th)
Scope of Testing: **Unit Testing** - methods for XMLParser Class TextOutputer Person Class Task Class	**Scope of Testing:** **Subsystem Testing** Controller Class Model Class View Class **Regression Testing** XMLParser Class TextOutputter Class Person Class Task Class	**Scope of Testing:** **Integration Testing** GANTT Tree View **Regression Testing** Controller Class Model Class View Class XMLParser Class Task Class **System Testing** Via test cases using manual execution of program

Figure 3: Testing Schedule



Testing Resources

Phase 1 (Jan 20th to Feb 5th)	**Phase 2** (Feb 6th to Mar 11th)	**Phase 3** (Mar 12th to Apr 8th)
Human Resources Dmitry Kuznetsov Will Ling Jeff How	**Human Resources** Tom Paulin Kai Wang	**Human Resources** Marc-Andre Faucher Jon Bergeron
Documentation Resources Software Engineering: A Practitioner's Approach	**Documentation Resources** Software Engineering: A Practitioner's Approach	**Documentation Resources** Software Engineering: A Practitioner's Approach
Web Resources www.junit.org http://www.vogella.de/articles/JUnit/article.html	**Web Resources** www.junit.org	**Web Resources** www.junit.org http://code.google.com/p/t2framework/wiki/JUnitQuickTutorial
Programming Resources JUnit	**Programming Resources** JUnit	**Programming Resources** JUnit

Figure 4: Testing Resources

Requirements Tested

F1 Only one person should be logged in at a time. After increment 1, this requirement was deemed unnecessary.

F2 A view of tasks in the company is to be shown to the user in order for him to manage them. The table view is operational. Tested via manual executions of the user interface.

F3 The user is to be able to enter a new task.

Tested via manual executions of the user interface. Empty task added in Task view.

F4 The user is to be able to remove a previously created task. Tested via manual executions of the user interface.

Initial testing resulted in three failures:

1. Removal of task did not update People View appropriately. Project was not removed from employee data.

F5 The user is to be able to modify a previously created task.

Tested via manual executions of the user interface.

Initial testing resulted in three failures:

1. Verifying the correct format of data inputted into the system.

2. Inconsistency in dates between the Task View and XML document.

3. Inconsistency in data between the Task View and Person View. Changing project duration in Task View did not update employees hours in Person View.

F6 The user is to be able to view a Tree graph of tasks. The tree view is operational. Tested via manual executions of the user interface.

F7 The user is to be able to view a Gantt chart of tasks. The gantt view is operational. Tested via manual executions of the user interface.

F8 The user must be able to manage employees on a task.

Tested via manual executions of the user interface.

Initial testing resulted in a failure:

1. Management of employees did not update People View appropriately.

F9 The user must be able to assign employees to a task.

Tested via manual executions of the user interface.

Initial testing resulted in a failure:

1. Assignment of employees did not update People View appropriately.

F10 The user must be able to view a list of employees in the company with their respective data. The Person View is operational. Tested via manual executions of the user interface.

F11 The user is to be able to add a new employee. After increment 1, this requirement was deemed unnecessary.

F12 The user is to be able to remove an employee. After increment 1, this requirement was deemed unnecessary.

F13 The user must be able to withdraw employees from a task. Tested via manual executions of the user interface.

Initial testing resulted in a failure:

1. Withdrawal of employees did not update People View appropriately in regards to allocating hours.

Conclusion

In regards to requirements deemed necessary, any failures were communicated with the programmers, who fixed the code. Follow-up tests were performed to ensure the issues were resolved. The testers are satisfied with the final results.

4 References

A Description of Input Files

A.1 People.xml

<people> : Root node used to specify the beginning of the listing of people

<person> : Node used to specify a new person

<identifier> : A unique, alphanumberic identifier for each person

<fname> : Node used to store the first name of the person

<lname> : Node used to store the last name of the person

<jobtitle> : Node used to track the job title of the person. Used to differentiate between-roles

<jobdescription> : Node used for the person's job description.

<clearance> : Node used to specify the clearance a person within the organization. Restricts permission based on value. 0 - Employee, 1 - Manager, 2 - Administrator

A.2 A.2 Tasks.xml

<tasks> : Root node used to specify the beginning of the listing of tasks

<task> : Node used to specify a new task

<identifier> : A unique, alphanumberic identifier for each task

<title> : Node to store a short title for the task

<description> : Node to store a full description for the task

<startdate> : Node to store the appropriate start date for the task

<duration> : The duration, in hours of the task.

<deliverable> : The expected physical deliverable upon task completion

<deadline> : The date by which the task must be completed

<peopleassigned> : A listing of the individuals assigned to the task. The duration will be split evenly among assignees.

<id> : The corresponding identifier for each person assigned

<completion> : The percentage of completion for the task

A.3 Sample Invalid XML Files

B Description of Output Files

B.1 People.txt

The list of people with a summary of their names, the total amount of time spent on all projects, and the specific projects to which they are assigned. Sample format:

Name — Total Hours — Project List ———————————————————————— LName , FName — X.X — A, B, C

C List of Requirements

C.1 Functional Requirements

F1

Only one person should be logged in at a time.

F2

A view of tasks in the company is to be shown to the user in order for him to manage them.

F3

The user is to be able to enter a new task.

F4

The user is to be able to remove a previously created task.

F5

The user is to be able to modify a previously created task.

F6

The user is to be able to view a dependency graph of tasks.

F7

The user is to be able to view a Gantt chart of tasks.

F8

The user must be able to manage employees on a task.

F9

The user must be able to assign employees to a task.

F10

The user must be able to view a list of employees in the company in order for him to manage them.

F11

The user is to be able to add a new employee.

F12

The user is to be able to remove an employee.

F13

The user must be able to withdraw employees from a task.

C.2 C.2 Non-Functional Requirements

N1

The system must know who the user is in order to order the appropriate options.

O1

It shall be possible to run the system on any computer system that runs the JVM.