Paul Jemmott
Sean Gahagan
John Carlo Salter

PROJECT LEGACY

Software Quality Assurance Plan

Version: (1) **Date: (05-11-2015)**

Document History and Distribution

1. Revision History

Revision #	Revision Date	Description of Change	Author
1	5-4-15	Initial Creation	Paul Jemmott et. Al.
2	5-7-15	Filled in some of section 2 and 5	Paul Jemmott, et Al.
3	5-17-15	Filled in all remaining sections.	John Carlo Salter, Sean Gahagan, Paul Jemmott

TABLE OF CONTENTS

1. INTRODUCTION

(NOTE 1: THE SOFTWARE TEST PLAN GUIDELINES WERE DERIVED AND DEVELOPED FROM IEEE STANDARD FOR SOFTWARE TEST DOCUMENTATION (829-1998)).

The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The plan must identify the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.

1.1 Objectives

The overall goal of this project is to improve the nonfunctional requirements of the open-source project *Project Legacy.* Specifically, we want to improve the readability, maintainability, security, and reliability of the game. We will do this through a combination or static analysis, testing, and refactoring.

1.2 Testing Strategy

Specific test plan components include:

- Refer to section 5 of this document for a detailed list of specific test plans
- Refer to section 3 for features to be tested.
- Refer to section 4 for features not to be tested.
- Refer to section 6 for pass/fail criteria.
- Refer to section 7.5 for the overall testing schedule.
- Refer to section 9 for risk management.

1.3 Scope

Our goal is to provide a one-time quality-assurance upgrade to *Project Legacy*. We intend to have our full-update package ready for export by May 18th, 2015, at which point we will push our updates out to GitHub.

1.5 Definitions and Acronyms

N/A - Not Applicable

2. TEST ITEMS

- Project Legacy: https://sourceforge.net/projects/plegacy/
- Game folder: https://github.com/JohnCarloSalter/cosc442_final_project
- SnowyEvening: https://snowy-evening.com/paul-et-al/project-legacy-cosc-442-final-project

2.1 Program Modules

We are not building any new modules to this project with regards to its actual functionality. We will only be refactoring code that already exist and has been tested by the developer, as well as adding test classes for existing code.

2.2 User Procedures

We did not edit any of the user documentation. Our JUnit tests can be run through Eclipse by right-clicking on the project root folder and selecting “Run as -> JUnit test”.

3. FEATURES TO BE TESTED

- Tools will cover the entire application based on the tools specific design.
- *GameObjectManager.java*
- *Ship.java*

  Classes to be refactored:

  Test.java refactored into a functional Junit test class(es). (EDIT: Test.java is vital running the application and is therefore not able to be refactored into a junit test.)

4. FEATURES NOT TO BE TESTED

We will avoid writing extensive tests for classes and methods that are heavily involved with the user interface, as these sections rely too much on code that is outside of our control. We also will avoid writing extensive tests for classes that are incredibly coupled. The coupling in these situations seems necessary based on the current design of the program, but it also makes it difficult to develop effective unit tests in a time-efficient manner.

5. APPROACH

5.1 Component Testing

Each of the classes listed will be tested against a testing suite to see if they find any coding issues, bugs, bad coding, etc.

Changes will be made to the code based on the output of the testing suite.
The issues found by every testing tool ran will be evaluated and removed.

5.2 Integration Testing

Unfortunately, the code is not partitioned in a way that makes integration testing time-efficient. When all unit testing is completed and any changes have been made, we will run the application again and perform another round of black-box beta-like testing to verify that our changes have been accepted.

5.3 Interface Testing

Interface testing will not be covered in the scope of our software testing plan.

5.4 Security Testing

A tool (TCM Early Vulnerability Detector) will be chosen to test the security of the application.

5.5 Performance Testing

When all individual testing is completed and any changes have been made, the code will be ran again to see that the program performs at a similar quality to the original code.

5.6 Regression Testing

When all individual testing is completed and any changes have been made, the code will be ran again to see that the program still functions as intended.

5.7 Acceptance Testing

Since we do not have a customer there is no acceptance criteria to follow. Acceptance testing will be done at the end of our project, but is not an integral part of our project scope.

5.8 Beta Testing

We ran the application before doing any code analysis in order to black box test the application. The code seemed to function as intended, we did not encounter any failures during initial testing. We will also perform black box tests at the end of the project lifespan to ensure the quality of the application has not diminished.

6. PASS / FAIL CRITERIA

6.1 Suspension Criteria

We will suspend testing on a portion of the code if we decide that that code should undergo heavy refactoring.

6.2 Resumption Criteria

We will resume suspended testing after necessary refactoring is complete and we are satisfied with the updated design.

6.3 Approval Criteria

We consider "sufficient refactoring" of a class to entail that the code is more readable and object-oriented.

7. TESTING PROCESS

7.1 Test Deliverables

Our goal is to make the developer's life easier during future development. Our deliverables will be the ability to run unit tests in Eclipse, which will then produce clear output that the developers can use to find where their code is failing.

7.2 Testing Tasks

There should be no required preparation with regards to running our tests -- the developer only needs to make changes to his or her code and run the new test suites.

7.3 Responsibilities

Group meetings will allow us to delineate tasks to group members, allowing each to focus on partitionable parts of the testing process. For this purpose we will all be members of the testing group, since we do not have a requirements document, and our main focus is creating

test cases for and improving the quality of the code.

Testing specific components will be assigned through an issue tracker at: snowyevening.com

7.4 Resources

Sean Michael Gahagan:

UCDetector

- Find code that is not being used.

FindBugs

- Find common errors in code.

Paul Jemmott

Checkstyle

- Evaluate code compared to coding standard

PMD

- testing rulesets to see if code follows rules and standards
    - basic.xml
    - design.xml
    - unnecessary.xml
    - unused.xml

John Carlo Salter:

TCM Early Vulnerability Detector

- Checks for common security vulnerabilities associated with web applications and other programs.

ThreadSafe

- Checks for common issues related to multi-thread programming and multi-processor environments.

7.5 Schedule

1. Perform black-box testing of our current product to see if we can identify any new bugs.
    a. If bugs are identified, go to 2. Otherwise, go to 3.
2. Find the areas of code that are likely the cause of those bugs. Create extensive unit tests to find the source of those bugs.
3. Create unit tests for classes specified in ***Features to be Tested*** if these classes were not covered in part two.
4. Run the unit tests, try to find the source of the errors, and correct those faults.

Repeat until all unit tests pass.

5. Examine the modified code and refactor for readability and object-oriented adherence where applicable.
6. Re-run black box testing from part one to assure that everything still works.
7. Repeat from step two.

8. ENVIRONMENTAL REQUIREMENTS

(Specify both the necessary and desired properties of the test environment including the physical characteristics, communications, mode of usage, and testing supplies. Also provide the levels of security required to perform test activities. Identify special test tools needed and other testing needs (space, machine time, and stationary supplies. Identify the source of all needs that is not currently available to the test group.)

8.1 Hardware

There are no hardware requirements in order to test the project.

A stable connection to github is required for version control.

8.2 Software

Eclipse IDE

windows environment

8.3 Security

As the code is open source, security of the testing environment is not a major concern of our project.

TCM early vulnerability detector is a tool that will evaluate the security of the code.

8.4 Tools

Paul Jemmott:

Eclipse-pmd

Checkstyle

Sean Gahagan:

findbugs

UCDetector

John Carlo Salter:

TCM Early Vulnerability Detector

ThreadSafe

8.5 Risks and Assumptions

Our allotted time to complete our testing was a month. Understanding the code will take a significant amount of time to complete, as each member must learn and understand the functionality of each component. Meeting time will also pose an issue based on other members schedules.

As the project is a video game's source code, the classes are heavily coupled as many objects require each other in order to render and update the GUI.

9. CHANGE MANAGEMENT PROCEDURES

Change will be initiated when one of our test cases fails. Since we are mostly working at the unit-level, changes do not have to undergo extensive review -- the tester/developer should double check his logic with a colleague, make the change, and then re-run the tests to validate the results.