TEST PLAN (TP)

FOR

Project UML (PUMMEL): A UML Diagramming Tool for the 2012 CS 384 Class

Version 1.0

March 19, 2012

Prepared for:

Dr. Clinton Jeffery

Prepared by:

Coleman Beasley, Alex Dean, Austin Enfield, Jason Fletcher, Cable Johnson, Adrian Norris, Theora Rice, and David Summers

University of Idaho

Moscow, ID 83844-1010

CS384 TPD

RECORD OF CHANGES (Change History)

Change Number	Date completed	Location of change (e.g., page or figure #)	A M D	Brief description of change	Approved by (initials)	Date approved

A - ADDED M - MODIFIED D – DELETED

[put program /system name here]

TABLE OF CONTENTS

Section Page

1 TEST PLAN IDENTIFIER

PUML_V1_CS384

2 REFERENCES

- **SYSTEMS AND SOFTWARE REQUIREMENTS SPECIFICATION (SSRS) FOR PUML : Project UML** Version 1.1
- **SYSTEM AND SOFTWARE DESIGN DESCRIPTION (SSDD): Incorporating Architectural Views and Detailed Design Criteria FOR Project: UML** Version 1.1

3 INTRODUCTION

This test plan is designed to test and evaluate functionality in the Pummel UML editor. It describes a master test design. GUI elements, file operations, and diagram error checking will be addressed.

4 TEST ITEMS

Functions To Be Tested

- Icon Manipulation
    - Selection (including selecting multiple icons)
    - Insertion
    - Deletion (keyboard, GUI, and right-click)
    - Changing Properties (Inserting/editing text, etc.)
    - Movement
    - Scaling
    - Overlap
- Line Manipulation
    - Labelling
    - Drawing
    - Deleting
    - Snapping to Icons

    - Collisions
- General Editing Tools
    - Copy (keyboard, GUI)
    - Cut (keyboard, GUI)
    - Paste(Keyboard, GUI)
    - Toggle Grid On/Off
    - Change Insertion Options (Switching between icons and lines to be inserted)

5 SOFTWARE RISK ISSUES

Due to the scholarly nature of this work, it may be hard to update this UML editor or find support for future users. The development team this semester may not be able to provide ongoing support for this product in the years to come. Because of this, those who come to use this program will need to become familiar with the source code in order to fix possible future errors.

6 FEATURES TO BE TESTED

- High Priority:
    - Icon Selection (including selecting multiple icons)
    - Icon Deletion (keyboard, GUI, and right-click)
    - Changing Icon Properties
- Medium Priority:
    - Change Insertion Options
    - Moving Icons
    - Icon Scaling
    - Drawing Connecting Lines
    - Deleting Connecting Lines
    - Labelling Lines
- Low Priority:
    - Toggle Grid On/Off
    - Copy Icon (keyboard, GUI)
    - Cut (keyboard, GUI)
    - Paste(Keyboard, GUI)
    - Undo
    - Redo

7 FEATURES NOT TO BE TESTED

- Undo/Redo - Have been removed from the 1.0 feature list.
- Some individual icons will not be as rigorously tested, as they are all based on the same format.

8 APPROACH

We use three different phases of testing for the entire project:

1. Unit Testing
2. Integration Testing
3. Functional Testing

8.1 Unit Testing

Since the project is almost entirely GUI based, satisfactory unit testing can be a difficult task. To account for this, we refactored our code to follow a more rigorous Model View Control implementation. This was accomplished by separating the logical parts of the code from the code that invokes GUI events (which cannot be truly unit tested). Take the saveAsFile method as an example. This method is composed of three parts:

- Invoking a dialogue box to enter the filename
- Writing the current diagram into XML
- Opening a new tab

Since the first and third parts are invoking GUI events, this method cannot be fully unit tested independently of the GUI itself. It can only be tested during an integration test. However, since XML writing contains all the logic of the method (branching, loops, etc.) with no GUI interaction, it can be moved to a helper method and be unit tested independently of GUI events. At that point, when GUI events have been tested in the integration testing phase, and the logical portion is independently unit tested, we can conclude that the saveFileAs process is tested to a satisfactory level.

In order to perform thorough unit testing, two things must be achieved: data coverage and branch coverage. The tool we use for data coverage is CXXTest. This is the framework that will test the logical portions of the code and supply a variety of data to test all possible inputs.
The tool used for coverage testing is Trucov. Coverage testing is done on both the logical portions of the code as well as the GUI event portions. To test for coverage on the logical portions, the tool is run over the unit tests. The GUI event coverage testing is automated with a tool called Open HMI Tester, which is simply a GUI recorder that will launch the program and walk through a series of actions within the GUI.

8.2 Integration Testing

In this phase the actual GUI events are tested. During this phase, all of the use cases will be walked through manually while confirming that the right events are being invoked based on input. For these individual use case tests, each test must be run as independently as possible, with minimal setup. This is to observe the behaviour of each use case completely independent of the others.

As these individual test cases are walked through, the GUI events taking place are recorded with a GUI recorder (Open HMI Tester). This way the process of rerunning the test cases is automated so that as adjustments are made one could simply launch a script and let the integration tests happen by themselves. Since the test cases are automated, coverage testing is also run over the project as the integration tests are executed. In this manner we are able to get coverage reports on the GUI portions of the code that could not be unit tested. Unit testing coverage in addition to this style of integration testing coverage results in the ability to get coverage on the entire project and leave no line of code untouched.
The integration testing phase tests GUI events. During this phase, the application is fully assembled. All integration tests will be run on the full application. The testers will follow the test cases included in this test plan. Each test case draws from a use case from the Use Case diagram. Each test case contains its own setup instructions, and its own criteria for success.

8.3 Functional Testing

The functional phase is where the usage of the UML editor is tested to a much higher degree. Rather than testing each use case individually, there will be a variety of users selected to attempt to produce UML diagrams of different types and magnitudes. This will produce a very large variety of permutations of use cases, and allow us to observe how the use cases behave when used together. The functional testers will have a minimum set of tasks that must be performed within the application for it to be considered a thorough enough run. These minimum requirements include:

- At least one of every icon available
- At least ten icons total
- At least one of every line available
- At least one note icon
- Must save the diagram at least one time
- Must re-open the diagram after exiting for the first time
- Must attempt all above criteria with each available diagram type

- **Tools**
  - CXX unit testing tool
- **Platforms**
  - Windows 7
  - Ubuntu Linux
  - Mac OS X

9 ITEM PASS/FAIL CRITERIA

The completion criteria for this plan are as follows:

- All Unit Tests complete with 95 percent success rate.
- All Unit Tests report 95 percent coverage, barring documented exceptions.

- All Integration tests complete with 95 percent success rate, barring documented exceptions.

- Functional tests are completed by users who report no large problems, for a 95 percent effectiveness rating.

10 SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

- Unit Testing

    - Stopping a series of tests: More than 80 percent failure.
    - Acceptable level of defects is 20 percent failure for continued testing before refactoring.

- Integration Testing

    - Stopping a series of tests: More than 80 percent failure.
    - Acceptable level of defects is 20 percent failure for continued testing before refactoring.

- Functional Testing

    - Stopping a series of tests: Users report errors that render program useless.
    - Acceptable defects include problems with graphical interface or rules, that still allow user to continue using program. These will later be refactored.

11 TEST DELIVERABLES

- Test plan document
- Test cases
- Error log of failed tests
    - Actions Taken

12 REMAINING TEST TASKS

- Undo/Redo have been postponed until Version .01

- Further diagrams will be added to future versions, and are not to be sought out in the current edition.

13 ENVIRONMENTAL NEEDS

For this small scale of testing there are very little as far as outside needs. All tests will be conducted on local machines on a variety of platform, using manual insertion, and GUI testing tools. Testing will continue on each part until the results match the desired results for each respective part.

- Software
    - CXXTest
    - Valgrind
    - Trucov
    - Wiki Knowledge
    - Qt Framework

14 STAFFING AND TRAINING NEEDS

Training will be required for the members of the team doing the testing to be able to work the testing software. Also any outside testers that are brought on will have to be trained on the workings of the program.

We designed the software so that minimal training is needed to run it. However, testers are assumed to have familiarity with UML diagrams. Each tester will be given a quick tutorial (or quick reference card?) on the software's basic functionality, and will be expected to refer to the help documentation if any difficulties arise.

Staff training will include the training in the use of the Cxx unit testing framework, the trucov coverage tester, the Open HMI GUI recorder, and the in-house makefile that automates the majority of tool use and also generates test reports on the wiki repository.

15 RESPONSIBILITIES

Each team member will individually add their own parts to the test plans, that includes creating unit tests for individual sections of code and being able to run the unit test framework(Cxx Test) as well as the coverage tester (trucov) on their individual machines or on a VM.

16 SCHEDULE

- Unit Testing is to be conducted April 16-26th.
- Integration Testing is to be conducted April 27th-May 3rd.
- Functional Testing is to be conducted May 3rd-9th

17 PLANNING RISKS AND CONTINGENCIES

Ample time is allotted for testing even given possible delays, but in case of running short on given time, tests will begin before the remaining functionalities of the program are functional, or even included. This may require multiple retests as each new function is implemented and weaken the final test suit, but will have each piece tested before final date

18 APPROVALS

Approvals for test will originate within the group from individuals working on specific parts that are to be tested. As each different function is added it will be approved for the given test case corresponding to it, and testing will begin as soon as milestone 3 is reached, or when the delivery date for milestone 3 has been passed and there is sufficient testable product.

With regards to unit testing, a module is considered sufficiently tested when its report on the wiki repository displays satisfactory test and coverage results. These reports will be verified by a minimum of two team members (in addition to the author) and then be signed off after the report and the tests themselves are deemed sufficient.

19 GLOSSARY

Icon - A graphic object used in UML Diagrams, such as an ellipse, classbox, rounded square, etc.