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

1 TEST PLAN IDENTIFIER

Some type of unique company generated number to identify this test plan, its level and the level of software that it is related to. Preferably the test plan level will be the same as the related software level. The number may also identify whether the test plan is a Master plan, a Level plan, an integration plan or whichever plan level it represents. This is to assist in coordinating software and testware versions within configuration management.

[Insert text here.]

2 REFERENCES

List all documents that support this test plan. Refer to the actual version/release number of the document as stored in the configuration management system. Do not duplicate the text from other documents as this will reduce the viability of this document and increase the maintenance effort.

[Insert text here.]

3 INTRODUCTION

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

4 TEST ITEMS

These are things you intend to test within the scope of this test plan. Essentially, something you will test, a list of what is to be tested. This can be developed from the software application inventories as well as other sources of documentation and information.

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

This is a listing of what is to be tested from the USERS viewpoint of what the system does. This is not a technical description of the software, but a USERS view of the functions.

- 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

This is a listing of what is NOT to be tested from both the Users viewpoint of what the system does and a configuration management/version control view. This is not a technical description of the software, but a USERS view of the functions.

[Insert text here.]

8 APPROACH

This is your overall test strategy for this test plan; it should be appropriate to the level of the plan (master, acceptance, etc.) and should be in agreement with all higher and lower levels of plans. Overall rules and processes should be identified.

- *Are any special tools to be used? What are they?*
- *What metrics will be collected for this test?*
- *How many configurations/platforms are to be tested?*
- *How will elements in the design deemed "untestable" be processed?*

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. Our code allows for a rigorous Model View Control implementation. This was accomplished by separating the logical and GUI based code, as they cannot be tested in the same manner.

Using 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, and opening a new tab. The first and third parts are invoking GUI events, which means this method cannot be fully unit tested independent of the GUI itself. This means it can only truly be tested during an integration test. However, since writing the XML document contains all the logic of the method (branching, loops, etc.) with no GUI interaction, it can be moved to a helper method and unit tested completely independent of GUI events. The GUI events can be tested in the later integration testing phase. Thus, we can confidently conclude that the saveFileAs process is tested to a satisfactory level.

In order to actually unit test these methods, we will use the CxxTest framework. To write a test for a given class or file, the class and function definitions are written in a header file. From that header file is generated a .cpp file which is compiled individually and linked with the project, producing an executable unit test. Because every building step is automated by an in-house written makefile, the only training needed to use the tool is covered by the also in-house example template that displays proper class syntax and test macro calls.

8.2 Integration Testing

NEEDS EDITING This is the phase where 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 set-up. This is to observe the behaviour of each use case completely independent of the others.

8.3 Functional Testing

NEEDS EDITING 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.

9 ITEM PASS/FAIL CRITERIA

What are the Completion criteria for this plan? This is a critical aspect of any test plan and should be appropriate to the level of the plan. [Insert text here.]

10 SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

If the number or type of defects reaches a point where the follow on testing has no value, it makes no sense to continue the test; you are just wasting resources.

Specify what constitutes stoppage for a test or series of tests and what is the acceptable level of defects that will allow the testing to proceed past the defects. [Insert text here.]

11 TEST DELIVERABLES

What is to be delivered as part of this plan?

- *Test plan document*
- *Test cases*
- *Relevant error logs or problem reports*

One thing that is not a test deliverable is the software itself that is listed under test items and is delivered by development. [Insert text here.]

12 REMAINING TEST TASKS

If this is a multi-phase process or if the application is to be released in increments there may be parts of the application that this plan does not address. These areas need to be identified to avoid any confusion should defects be reported back on those future functions. This will also allow the users and testers to avoid incomplete functions and prevent waste of resources chasing non-defects. [Insert text here.]

13 ENVIRONMENTAL NEEDS

Are there any special requirements for this test plan, such as:

- *Special hardware such as simulators, static generators etc.*
- *How will test data be provided. Are there special collection requirements or specific ranges of data that must be provided?*

[Insert text here.]

14 STAFFING AND TRAINING NEEDS

Training on the application/system.

Training for any test tools to be used. [Insert text here.]

15 RESPONSIBILITIES

Who is in charge?

This issue includes all areas of the plan. Here are some examples:

- *Selecting features to be tested and not tested.*
- *Ensuring all required elements are in place for testing.*

[Insert text here.]

16 SCHEDULE

Should be based on realistic and validated estimates. If the estimates for the development of the application are inaccurate, the entire project plan will slip and the testing is part of the overall project plan.

[Insert text here.]

17 PLANNING RISKS AND CONTINGENCIES

What are the overall risks to the project with an emphasis on the testing process? Specify what will be done for various risk events.

[Insert text here.]

18 APPROVALS

Who can approve the process as complete and allow the project to proceed to the next level (depending on the level of the plan)?

[Insert text here.]

19 GLOSSARY

Used to define terms and acronyms used in the document, and testing in general, to eliminate confusion and promote consistent communications.

[Insert text here.]

20 APPENDIX A. [insert name here]

Include copies of test examples, etc. supplied or derived from the customer. Appendices are labeled A, B, . . . n. Reference each appendix as appropriate in the text of the document.

[insert appendix A here]

21 APPENDIX B. [insert name here]

[insert appendix B here]