

Acceptance Test Plan
For
Mashup Visual Programming Environment
(MVP-E)
Submitted by
Team 10 (Team MVP)

Instructor: Professor Werner Krandick

Advisor: Professor William Regli

Team Members: Tim Cheeseman

Dan De Sousa

Ngoc Nguyen

Jordan Osecki

Martin Piecyk

Version: 1.0

Date Submitted December 1, 2009

Grading Rubric – Acceptance Test Plan

This rubric outlines the grading criteria for this document. Note that the criteria represent a plan for grading. Change is possible, especially given the dynamic nature of this course. Any change will be applied consistently for the entire class.

Achievement		Minimal	Limited	Satisfactory	Exemplary	Score
Content		Section(s) missing, not useful, inconsistent, or wrong.	Serious omissions or problems with content.	Some problems with completeness or details of content	Provides all relevant information correctly and with appropriate detail	
Sections 1 - 5	10					
Functional Test Cases	35					
Non-functional Test Cases	25					
Grammar and Spelling	10	Many serious mistakes in grammar or spelling	Several large issues or many smaller ones	Some small grammar or spelling issues	Grammar, punctuation, and spelling all correct	
Expression	10	Very difficult to understand	Hard to follow or poor word choices	Mostly easy to read and understand	Clear and concise. A pleasure to read	
Tone		Tone not appropriate for technical writing	Tone somewhat unprofessional	Mostly professional tone	Tone is consistently professional	
Organization	10	Very hard to find information	Information difficult to locate	Can find information with slight effort	All information is easy to find and important points stand out	
Layout		Layout makes it harder to understand and use the document	Layout is inconsistent or not visually appealing or supportive	Layout is reasonable, consistent and generally helpful	Layout is attractive, consistent, and helps guide the reader	
Late Submission						
Total	100					

Table of Contents

1	Introduction.....	7
1.1	Background	7
1.2	Structure of the Document	7
1.3	References	7
1.4	Glossary.....	7
2	Test Approach and Constraints.....	9
2.1	Introduction	9
2.2	Test Objectives.....	9
2.3	Test Structure	9
3	Test Assumptions, Exclusions, and Methodology.....	10
3.1	Introduction	10
3.2	Assumptions.....	10
3.3	Exclusions	10
3.4	Methodology	10
3.5	Outcomes.....	10
4	Entry and Exit Criteria.....	11
4.1	Introduction	11
4.2	Schedule Analysis	11
4.3	Entry Criteria.....	12
4.4	Exit Criteria.....	12
5	Testing Participants.....	13
5.1	Introduction	13
5.2	Roles and Responsibilities	13
5.3	Training Requirements.....	13
5.4	Reporting Procedures for Test Plan	13
5.5	Reporting Procedures for Gathering Feedback	14
6	Project Test Cases.....	15

6.1	Introduction	15
6.2	Test Cases for All Users.....	15
6.2.1	Create a new project.....	15
6.2.2	Load a project	15
6.2.3	Save a project.....	16
6.2.4	Add static web components (different kinds)	16
6.2.5	Create a new mashup – Map.....	16
6.2.6	Create a new mashup – Table/Graph	17
6.2.7	Set mashup access.....	18
6.2.8	Set mashup name	18
6.2.9	Select an existing mashup.....	19
6.2.10	Select various web services to use	19
6.2.11	Create a new source from a web service.....	20
6.2.12	Create a new source from a data feed or website.....	20
6.2.13	Semantically annotate a source.....	21
6.2.14	Wire together web services through the interpreter and a mashup.....	21
6.2.15	Load a mashup (using tap menu or side bar)	22
6.2.16	Save a mashup (using tap menu or side bar).....	22
6.2.17	Access the help menu.....	22
6.2.18	Publish a website/project	23
6.2.19	View a website/project.....	23
6.2.20	Publish a mashup	24
6.3	Test Cases for System Qualities.....	25
6.3.1	MVP Editor Reliability.....	25
6.3.2	Server Latency	25
6.3.3	System Scalability.....	26
6.3.4	Execution Engine Flexibility	26
6.3.5	Portability of the MVP Editor.....	26
6.3.6	Portability of the Execution Engine	27
6.3.7	WSDL Definition for Execution Engine’s Web Service	27

6.3.8	Security	27
6.3.9	Standards Compliance	28
6.3.10	Wrappers	28
6.3.11	Performance	28
6.3.12	Design Constraints	29
6.3.13	Reliability.....	29
6.3.14	Availability	29
6.3.15	Maintainability.....	30

Table of Contributions

The table below identifies contributors to various sections of this document.

	Section	Writing	Editing
1	Introduction	N. Nguyen	Everyone
2	Test Approach and Constraints	J. Osecki	Everyone
3	Test Assumptions and Exclusions	M. Piecyk	Everyone
4	Entry and Exit Criteria	T. Cheeseman	Everyone
5	Testing Participants	D. De Sousa	Everyone
6	Project Test Cases	Everyone	Everyone

Revision History

Members	Date	Version	Reason For Changes
All	November 24, 2009	1.0	Initial Version for Consideration.

1 Introduction

The purpose of this document is to provide the testing criteria for the Acceptance Test Plan (ATP) for the MVP-E system. It is meant to validate if MVP-E is in accordance with the requirement of the customers.

1.1 Background

Mash-up Visual Programming (MVP-E) Editor is a system that allows users to create, compile, and run mash-ups of different services. It is describe in greater detail in the Software Requirements Specification for MVP-E.

1.2 Structure of the Document

This document contains the following sections:

- Section 2 – Describes the test approach and constraints of the ATP process.
- Section 3 – Describes the test assumptions and exclusions of the ATP process.
- Section 4 – Describes the entry and exit criteria of the ATP process.
- Section 5 – Describes the testing participants of the ATP process.
- Section 6 – Describes the test cases of the ATP process.

1.3 References

This document draws references from the following:

- Software Requirements Specification for MVP-E

1.4 Glossary

This document utilizes the following terms and their definitions:

- Unit Tests: a method to verify and validate each unit (smallest testable part) of a application
- Integration Tests: the testing of combined individual software modules, usually done after unit testing
- System Tests: a complete test of the system for compliance to specific requirements, usually done after integration testing
- Test Overview: provide a brief description of the testing process
- Pre-condition: a condition that must be true before the execution of some section of code
- Action: The section of code performs some action
- Post-condition: : a condition that must be true after the execution of some section of code

- Acceptance Test Plan (ATP): the testing of an application to see if it satisfies all the software requirements
- Software Requirements Specification (SRS): provide the complete requirements for a software application

2 Test Approach and Constraints

This section deals with the test approach and constraints. The following contains sections including an Introduction, Test Objectives, and Test Structure.

2.1 Introduction

This section describes the overall approach, particular techniques, and testing tools that will be used during the Acceptance Test Plan of the MVP-E system and any constraints that may apply. The section is broken up into the sections Test Objectives and Test Structure. The Test Objectives section will describe the objectives of the Acceptance Test Plan. The Test Structure section will put some constraints on how the Acceptance Test Plan is executed.

2.2 Test Objectives

The Acceptance Test Plan process will prompt the user to evaluate the MVP-E system and verify whether it performs in accordance with the user's requirements, listed in the MVP-E Software Requirements Specification.

The Acceptance Test Plan is composed of Test Cases, which are expected to cover all of the functions that users of all types (Army, Scientists, and regular people) will intend to use. The Acceptance Test Plan is essentially a walk-through of all of the system's features that a user would use.

In addition, the Test Cases that are included in the Acceptance Test Plan are expected to cover all requirements listed in the Software Requirements Specification. There is not a one-to-one matching, but through the execution of all of the Test Cases located in this document, each requirement in the Software Requirements Specification is expected to have been exercised at least once.

2.3 Test Structure

The Acceptance Test Plan consists of a set of Test Cases, which will rigorously test the MVP-E system. The Acceptance Test Plan will consist of a subset of Test Cases and methods, from the previously utilized Test Cases in Unit Tests, Integration Tests, and System Tests conducted during the actual coding process of the MVP-E system.

The Test Cases will be carefully selected and agreed upon by both the development team and the user, and will allow for the most adequate verification of all of the features the user will use and all of the function requirements of the MVP-E system, as listed in the Software Requirements Specification, without the extensiveness of the full-scale System Test.

The Acceptance Test Plan will originally be populated with the set of key features that the user will be expecting in the final product. As design and implementation commence and Unit Tests, Integration Tests, and System Tests are devised, these will be migrated into the Acceptance Test Plan, as appropriate, to ensure that all features and functional requirements are accounted for and evaluated.

It is essential that all appropriate Unit Tests, Integration Tests, and System Tests were successfully performed for the MVP-E system prior to the Acceptance Test Plan being executed, and all of their results being reported and presented to the client.

3 Test Assumptions, Exclusions, and Methodology

This section deals with the test assumptions, exclusions, and methodology. The following contains sections including an Introduction, Assumptions, exclusions, Methodology, and Outcomes.

3.1 Introduction

This section describes the assumptions, exclusions, and the methodology with the MVP-E test plan. It describes in greater details which issues and features of MVP-E will and will not be covered by the Acceptance Test Plan.

3.2 Assumptions

There are several assumptions with this test plan. This document assumes that unit tests have been written for all automatable tests. Moreover, it assumes that non-automatable tests will be manually checked and verified. Also, the plan assumes that a reasonable number of white-box unit tests have been written to check the internals of the MVP-E. These white-box tests were generally not described in this test plan. Finally, this document assumes that the SRS is complete.

The Acceptance Test Plan is expected to cover the following:

- The functional and non-functional Requirements listed in the SRS
- Usability of the System by Clients
- User-related System Documentation

3.3 Exclusions

White-box tests of the methods of classes are not described in this plan. In addition, security exploit tests are not covered in this document. Finally, the integrity of the source code will also not be tested.

3.4 Methodology

The test cases in this document were created using the SRS. This allowed consistency between the two documents. In order to verify that the MVP-E works correctly, an end-user could iterate through each of the test cases and verify that the MVP-E passes all of them.

White-box tests were not included in this document. These were not appropriate since we did not write our source code yet, which is required for most white-box tests.

3.5 Outcomes

In the event of all of the tests passing, the MVP-E will be prepared for release. In the event of some non-critical tests failing, the MVP-E will be considered for release and the MVP-E will be attempted to be fixed. If critical tests fail, the MVP-E will not be considered to be released until the code is fixed.

4 Entry and Exit Criteria

This section deals with the test entry and exit criteria. The following contains sections including an Introduction, Schedule Analysis, Entry Criteria, and Exit Criteria.

4.1 Introduction

This section describes the analysis of the testing schedule, the entry criteria to ensure that the proper environment is in place for testing, and the exit criteria to ensure that testing was successful and the development process is complete.

4.2 Schedule Analysis

Included below is an excerpt from the project schedule pertaining to the acceptance test plan:

Task	Duration	Start date	End date	People
Create Acceptance Test Plan (Initial Draft)	13 days	11/12/2009	11/30/2009	Everyone
Compile Acceptance Test Plan	3 days	11/20/2009	11/24/2009	Everyone
Revise Acceptance Test Plan For Changes In Final Requirements (CS version due on 12/1 and SE version due on 11/30)	2 days	11/27/2009	11/30/2009	Everyone
Revise and Create Final Drafts of Requirements Document / Acceptance Test Plan	8 days	2/5/2010	2/16/2010	Everyone
Perform Surveys for End Users, to Gather Feedback	10 days	3/10/2010	3/20/2010	Everyone
Create Beta Test Report	18 days	4/2/2010	4/27/2010	Everyone
Develop Test Suites	15 days	4/2/2010	4/22/2010	Everyone
Create Test Report	3 days	4/23/2010	4/27/2010	Everyone

Most of the time in creating the ATP is in the initial draft as this is when the majority of original work is done. Further time is allocated to revise the plan as the project evolves, but this time is considerably less. Similarly, the majority of time for creating the test report is in the overlapping

tasks of developing and running the test suites and creating the initial draft of the test report. A shorter amount of time is allocated for refining and expanding the test report after the testing has been completing.

4.3 Entry Criteria

- The MVP system has successfully undergone unit tests, the integration test, and the system test.
- This acceptance test plan and all of its test cases must be approved.
- All of the tests must be implemented.
- All of the developer documentation (e.g. Software Requirements Specification) must be up to date.
- All of the user-related documentation must be up to date.
- All test hardware platforms must be installed and properly configured for testing.
- All software to be tested must be installed and properly configured for testing.
- All testing tools must be installed and properly configured.
- All people involved in the testing (Project Team Leader and Test Engineer) must be familiar with the tests and testing tools and consent to beginning the tests.

4.4 Exit Criteria

- All of the developer documentation (e.g. Software Requirements Specification) must be completed.
- All of the user-related documentation must be completed.
- All Priority 1 requirements must be implemented.
- All critical bugs must be fixed.
- All planned deliverables must be completed.

5 Testing Participants

This section deals with the test participants. The following contains sections including an Introduction, Roles and Responsibilities, Training Requirements, and Reporting Procedures for both a test plan and gathering feedback plan.

5.1 Introduction

The following section outlines all roles, responsibilities, training requirements, and reporting procedures involved with the Acceptance Test Plan.

5.2 Roles and Responsibilities

The following roles exist within the context of this Acceptance Test Plan:

- Test Coordinators
 - Assignment: Team 10 (Team MVP)
 - Responsibilities:
 - Assign and schedule testers to particular test cases.
 - Receive and review problem and progress reports.
 - Conduct surveys and review survey results.
- User
 - Assignment: An individual or party representative of an MVP-E benefactor or use case subject.
 - Responsibilities:
 - Review testing process.
 - Take a survey regarding the product.
- Tester
 - Assignment: TBD
 - Responsibilities:
 - Execute a particular test case as assigned by the test coordinator.

5.3 Training Requirements

Prior to engaging in the test plan, all individuals listed in (5.2) should be familiar with all aspects of the MVP Environment, including:

- MVP Editor User Interface
- MVP Web page output
- MVP Server Configuration

All individuals should also be familiar with the most recent MVP-E Software Requirements Specification.

5.4 Reporting Procedures for Test Plan

During test iteration, the Test Coordinators will create a schedule and assign individual testers to each test case.

Each tester will execute their test case and document which test cases passed or failed. For each pass or failure, the tester shall document all known parameters and environment state present

when the test result was observed. In the case of a failure, the test shall also document any notes they believe to be useful to correcting the test failure. This information shall be filled in a standard template and presented to the Test Coordinators. These aggregate reports shall be combined and be known as the ***Results Report***.

The Test Coordinators shall review each results report and optionally request that a tester reproduce the result. If the result is verified to the Test Coordinator's satisfaction it shall be included in the final results report which will be given to the Team Lead to be reviewed at a bug fix or staff meeting.

5.5 Reporting Procedures for Gathering Feedback

In addition, the Test Coordinators will conduct a survey to give out to potential users of the software, with the object not to obtain testing results, but to gather feedback regarding the software product itself. The required feedback would be used to help improve the final product before the final phase of implementation is due.

The survey would be conducted in March 2010 with students at Drexel University. The students would be gathered by their responding to either e-mail or flyer advertisements of the survey. Professor William Regli would provide funds in order to pay each participant approximately \$10 to conduct the survey, with the goal being to interview up to 30 participants. The survey process will require approval from the university, as does any survey which utilize human subject testing.

The survey would consist of a set of slides which will have preface information to provide the participants the background training discussed in section 5.3, along with slides of test cases and use cases. It would require the participants to do exercises within the program and then fill out a survey of their experience.

The results of the survey would provide the following three types of feedback:

- **Interaction Issues**: Test Coordinators will study how the participants interact with the software, to detect both positive and negative interactions and detect trends.
- **Process Issues**: Test Coordinators will study and survey questions will analyze the processes that each participant uses to solve the exercises, and see if there are any process issues with the software itself.
- **Qualitative Assessment**: The survey will provide qualitative assessment information to include in final reports and documentation for the product. The two main assessments will be analysis of time saved and also expertise required, comparing the process of mashing web services both before and after this product's release.

6 Project Test Cases

This section deals with the project test cases. The following contains sections including an Introduction, Test Cases for all users, and System Qualities.

6.1 Introduction

This section will cover the Test Cases required for both the standard functional test cases and the non-functional test cases. They will all be tests performed by the user to test requirements that have been listed in the SRS.

6.2 Test Cases for All Users

Below in this section are the Test Cases for all users. They test product features in their entirety, in order to cover all functional requirements as listed in the SRS. The tests can be performed in iteration by the client/customer.

6.2.1 Create a new project

Test Overview	This test will cover the case when a user creates a new project.
Pre-conditions	None
Actions	In order to create a new project, the user will open the main menu and choose the option to create a new project.
Post-conditions	A new project is created.

6.2.2 Load a project

Test Overview	This test will cover the case when a user loads a project.
Pre-conditions	The project and mashup files must exist on the local file system.
Actions	In order to load a project, the user will open the main menu and choose the option to load a project. The client will check the necessary mashup servers for new versions of the mashup files and download them if they were updated.
Post-conditions	The project is loaded and the mashup files are made up-to-date.

6.2.3 Save a project

Test Overview	This test will cover the case of the user saving the project.
Pre-conditions	The user has the system on and a new or existing project file open in the editor.
Actions	The user brings up the system menu by double tapping on the work area, or right clicks on the work area, or opens the sidebar. The user selects the Save option in the menu. The user is asked to “Save” or “Save As”. The user will navigate to where they want the mashup to be saved. Depending on the user’s selection, the most standard action for Save and Save As is observed.
Post-conditions	The project is saved, and the user is taken back to their current window.

6.2.4 Add static web components (different kinds)

Test Overview	This test will cover the case when a user adds static web components to the web page.
Pre-conditions	The web page must be visible.
Actions	The user drags and drops static web components on the web page.
Post-conditions	The static web components are placed on the web page.

6.2.5 Create a new mashup – Map

Test Overview	This test will cover the case where a user wants to create a new mashup component in their project, specifically a map.
Pre-conditions	The user has the system on and a new or existing project file open in the editor.
Actions	The user draws a box using their finger on the multi-touch screen or clicks on the new mashup component button from the side bar, to

	<p>indicate the creation of a new mashup. The user is presented with the choices of a new mashup and selects the map choice. The user chooses the appropriate settings to customize the map, such as the location of the view, scale of the view, and types of pins to use. The user clicks okay and is directed to a screen to choose which web services to use. This will be covered in a separate test case. Once this is finished, the user selects that the mashup creation is complete.</p>
Post-conditions	<p>The user has the same project file open, but in the area where they indicated to create a new mashup, a representation of the new map mashup just created is shown.</p>

6.2.6 Create a new mashup – Table/Graph

Test Overview	<p>This test will cover the case where a user wants to create a new mashup component in their project, specifically a table or graph.</p>
Pre-conditions	<p>The user has the system on and a new or existing project file open in the editor.</p>
Actions	<p>The user draws a box using their finger on the multi-touch screen or clicks on the new mashup component button from the side bar, to indicate the creation of a new mashup. The user is presented with the choices of a new mashup and selects the table or graph choice. The user chooses the appropriate settings to customize the map, such as the type of table or graph, the scales of the axes, and the type of key. The user clicks okay and is directed to a screen to choose which web services to use. This will be covered in a separate test case. Once this is finished, the user selects that the mashup creation is complete.</p>
Post-conditions	<p>The user has the same project file open, but in the area where they indicated to create a new mashup, a representation of the new table/graph mashup just created is shown.</p>

6.2.7 Set mashup access

Test Overview	This test will cover the case where a user wants to set the user access properties for a mashup that they have already created.
Pre-conditions	The user has the system on and a new or existing project file open, with at least one mashup that has been created using 6.2.5 or 6.2.6 to do so.
Actions	The user selects the mashup either using their finger in the multi-touch interface environment or a mouse in the traditional interface environment. The user selects to edit the mashup access properties. From here, the user can set permissions on the mashup, including read, write, and execute privileges for themselves, groups of people, or everyone. Once the user has made their choice, the user confirms their setting.
Post-conditions	The user still has the system on and the same project with the same mashup open, but the mashup access properties have been changed accordingly.

6.2.8 Set mashup name

Test Overview	This test will cover the case where a user wants to set the name property for a mashup that they have already created.
Pre-conditions	The user has the system on and a new or existing project file open, with at least one mashup that has been created using 6.2.5 or 6.2.6 to do so.
Actions	The user selects the mashup either using their finger in the multi-touch interface environment or a mouse in the traditional interface environment. The user selects to edit the mashup name properties. From here, the user can set the mashup name, to be referred to both in the project and also if the mashup is published on to the server. Once the user has

	made their choice, the user confirms their setting.
Post-conditions	The user still has the system on and the same project with the same mashup open, but the mashup name properties have been changed accordingly.

6.2.9 Select an existing mashup

Test Overview	The test covers the case of the user import/loading an existing mashup.
Pre-conditions	The user is in the Select Mashup configuration.
Actions	The user selects Import Mashup with a single tap or a left single mouse click. The user is asked to navigate the file structure and select the desired mashup. The user selects the mashup and hits open.
Post-conditions	The user selected mashup is loaded and the user is taken to the Main Window.

6.2.10 Select various web services to use

Test Overview	This test covers the case when the user selects a web component to add into the project.
Pre-conditions	The user is in the Configuration Window, and selects a web service icon, either from the option bar or an existing web service icon in the work area.
Actions	The user is taken to the Select Web Service window. The user selects one of the available web services but a single tap with their finger or single click with a mouse.
Post-conditions	The user is taken back to the Configuration Window and the web service component is added.

6.2.11 Create a new source from a web service

Test Overview	This test will cover the case where a user imports a web service to be used as a source in a mashup.
Pre-conditions	The web service must exist, must have a WSDL, and must be semantically annotated.
Actions	The user clicks the Add New Source button, enters the URL of the WSDL, and clicks OK.
Post-conditions	The mashup editor is able to use the new source, has identified its inputs and outputs correctly from the semantic markup, and can combine it with other sources in a mashup.

6.2.12 Create a new source from a data feed or website

Test Overview	This test will cover the case where a user creates a new source from a data feed or website and adds it to a mashup.
Pre-conditions	Use a built-in service <ul style="list-style-type: none">• The built-in service must be available. Use existing web service or data feed <ul style="list-style-type: none">• The service must be available. Scrape website <ul style="list-style-type: none">• The user must have access to a computer that can run a custom server and client program and knowledge of how to write code.
Actions	Use a built-in service <ul style="list-style-type: none">• The built-in service is executed. Use existing web service or data feed <ul style="list-style-type: none">• The service is automatically converted to a SOAP message output. Scrape website <ul style="list-style-type: none">• The user writes a program that

	downloads and parses a data feed or website and outputs it to the MVP server in a SOAP message. The user runs this program on a server.
Post-conditions	The MVP server receives SOAP messages of the results.

6.2.13 Semantically annotate a source

Test Overview	This test will cover the case where a user creates a source from a web service that is not semantically annotated.
Pre-conditions	The web service must exist and have a WSDL.
Actions	The user clicks the Add New Source button, enters the URL of the WSDL, and clicks OK.
Post-conditions	The mashup editor is able to use the new source, has identified its inputs and outputs correctly from the semantic markup, and can combine it with other sources in a mashup.

6.2.14 Wire together web services through the interpreter and a mashup

Test Overview	This test will cover the case where a user combines two sources in a mashup.
Pre-conditions	The two sources have at least one pair of compatible outputs between them (e.g. each service has a location output).
Actions	The user highlights the two sources and selects Add Join from the menu. A window opens displaying all of the pairs of compatible outputs on which the sources can be combined. The user selects the desired pair and clicks the OK button.
Post-conditions	The two sources are now joined in the editor, and execution of the mashup correctly combines the output from each source.

6.2.15 Load a mashup (using tap menu or side bar)

Test Overview	This test will cover the case of loading a mashup using a tap menu or side bar.
Pre-conditions	The user has the system on or existing project file open in the editor.
Actions	The user brings up the system menu by double tapping on the work area, or right clicks on the work area, or opens the sidebar. The user selects the open option in the menu. The user will navigate to where the mashup exists, and selects the desired mashup.
Post-conditions	The user selected mashup is displayed in the editor.

6.2.16 Save a mashup (using tap menu or side bar)

Test Overview	This test will cover the case where a user saves a mashup that they have been editing.
Pre-conditions	The mashup is a complete mashup (i.e. there are defined inputs and outputs for the mashup as a whole, and all internals are connected between them).
Actions	The user clicks the Save Mashup button in the menu.
Post-conditions	The new mashup definition is saved to the MVP server linked to the MVP editor.

6.2.17 Access the help menu

Test Overview	This test will cover the case where a user attempts to access the help menu.
Pre-conditions	The tester is at any non-dialog window in the MVP Editor and the sidebar is visible.
Actions	The tester selects the help icon from the sidebar. or

	<p>The tester single taps a blank area of the main palette with two fingers, then selects the “help” option from the radial menu.</p> <p>or</p> <p>The tester selects menu from the options sidebar, then selects the “help” option from the radial menu.</p>
Post-conditions	The tester is presented with a full-context help menu.

6.2.18 Publish a website/project

Test Overview	This test will cover the case where a user creates a website from an MVP projet.
Pre-conditions	The tester has opened and modified a saved MVP Project file using only mashups previously published on a server.
Actions	<p>The tester single taps a blank area of the main palette with two fingers, then selects the “create webpage” option from the radial menu.</p> <p>or</p> <p>The tester selects menu from the options sidebar, then selects the “create webpage” option from the radial menu</p> <p>then</p> <p>The tester selects a local file system location to create the webpage file(s).</p>
Post-conditions	One or more HTML, CSS, and/or Javascript files are created on the local file system at the location specified.

6.2.19 View a website/project

Test Overview	This test will cover the case where a user views a website created from an MVP project.
Pre-conditions	The tester has executed (1.1.2) and a webpage has been created.

	<p>and</p> <p>The tester is using a computer connected to the mashup server over a network.</p>
Actions	<p>The tester, using the local operating system, opens the index.html file in a compliant web browser.</p>
Post-conditions	<p>The webpage appears in a compliant browser according to the layout specified in the MVP Editor and performing the function created in the mashup description.</p>

6.2.20 Publish a mashup

Test Overview	<p>This test will cover the case where a user saves a mashup to the mashup server.</p>
Pre-conditions	<p>The tester has opened and modified a saved MVP Project file.</p> <p>and</p> <p>The tester is using a computer connected to the mashup server over a network.</p>
Actions	<p>The tester single taps a blank area of the main palette with two fingers, then selects the “publish” option from the radial menu.</p> <p>or</p> <p>The tester selects menu from the options sidebar, then selects the “publish” option from the radial menu</p> <p>Then</p> <p>The tester fills in information, including mashup name, publishing server, etc.</p>
Post-conditions	<p>The publish progress window appears, and after an initial upload period, the tester is informed that the mashup was published.</p> <p>or</p> <p>The publish progress windows appears and</p>

	after an initial upload period, the test is informed that the mashup could not be published because the tester was not authorized.
--	--

6.3 Test Cases for System Qualities

Below in this section are the System Qualities Test Cases. They test product qualities in their entirety, in order to cover all non-functional requirements as listed in the SRS. The tests can be performed in iteration by the client/customer to assess things such as performance.

6.3.1 MVP Editor Reliability

Test Overview	The system will provide 24 hours a day, 7 days a week accessible programming environment to mashup creations.
Pre-conditions	The user is running the MVP Editor on a reliable computer.
Actions	The user uses the MVP Editor.
Post-conditions	The user can write mashups and save them to the local file system. If a mashup server is running, the user can publish to the mashup server as well.

6.3.2 Server Latency

Test Overview	Servers should handle about 100 users adequately at any time by sending the beginning of replies to requests in less than a second.
Pre-conditions	100 users are concurrently accessing a mashup server or web server and the network latency between the servers and users is less than 300 milliseconds.
Actions	The servers will begin to reply in less than a second.
Post-conditions	The servers will send the rest of the replies after a second.

6.3.3 System Scalability

Test Overview	The system should be adaptive to load increases in excess of 500 users
Pre-conditions	More than 500 users should access the system.
Actions	A system administrator should be able to add more servers to the system to handle the load without interrupting access to the system.
Post-conditions	The system will be able to handle the load.

6.3.4 Execution Engine Flexibility

Test Overview	The execution engine is available through a web service interface, so that the actual implementation of the engine is not coupled with the clients that use it.
Pre-conditions	The execution engine is running.
Actions	Any change in the implementation of the engine (e.g. programming language, mashup algorithm) is transparent to the client, making the system more maintainable.
Post-conditions	The execution engine continues running and clients do not notice any changes.

6.3.5 Portability of the MVP Editor

Test Overview	The MVP Editor shall be developed on technology that is platform independent (e.g. the Common Language Runtime, which can run on the .NET framework on Windows, or Mono on Linux and OS X) for maximum portability.
Pre-conditions	A user wants to run the MVP Editor on Windows, Linux, and OS X.
Actions	The user can run the MVP Editor on all three of these operating systems.

Post-conditions	The program runs on all three operating systems.
------------------------	--

6.3.6 Portability of the Execution Engine

Test Overview	The execution engine shall be developed on technology that is platform independent (e.g. Python, which can run on all major operating systems) for maximum portability.
Pre-conditions	A system administrator wants to run the Execution Engine on Windows, Linux, and OS X.
Actions	The system administrator can run the Execution Engine on all three of these operating systems.
Post-conditions	The program runs on all three operating systems.

6.3.7 WSDL Definition for Execution Engine's Web Service

Test Overview	The web service provided by the execution engine will have a standard WSDL definition.
Pre-conditions	A user wants to get the WSDL definition of the execution engine.
Actions	The user downloads the WSDL definition of the execution engine.
Post-conditions	None

6.3.8 Security

Test Overview	This test will ensure that users cannot gain more access than they are granted through their permissions.
Pre-conditions	The servers, client, and web browser are running.
Actions	The user cannot find a way to get privileged access to the system without the proper

	credentials.
Post-conditions	The system and data are kept safe.

6.3.9 Standards Compliance

Test Overview	This test will ensure that the final product meets the “Standards Compliance” requirements set forth in the SRS.
Pre-conditions	The mashup registry is populated with multiple mashups generated by the MVP Editor.
Actions	The tester creates a dump of the mashup registry into xml files and runs them through a standard XML compliance verifier.
Post-conditions	The verifier returns that the mashup descriptions are XML 1.1 and RDF compliant.

6.3.10 Wrappers

Test Overview	This test will ensure that the final product meets the “Wrappers” set forth in the original requirements.
Pre-conditions	The user has the program open.
Actions	The user creates a mashup.
Post-conditions	The XML description telling where the WSDL and OWLS is generated correctly.

6.3.11 Performance

Test Overview	This test will ensure that the final product meets the “Performance” set forth in the original requirements.
Pre-conditions	The user has the program open.
Actions	The user creates a series of mashup to specific requirements.
Post-conditions	The mashup produces the expected behavior.

6.3.12 Design Constraints

Test Overview	This test will ensure that the final product meets the “Design Constraints” set forth in the original requirements.
Pre-conditions	The user is evaluating the system as a whole.
Actions	The user first checks to determine if the final product is web-based. The customer checks to see if the product finished in budget.
Post-conditions	The program will be unchanged, but the user and customer will have resulted to report back to the project team.

6.3.13 Reliability

Test Overview	This test will ensure that the final product meets the “Reliability” requirements set forth in the SRS.
Pre-conditions	The user has the program open.
Actions	The user runs a series of tests (the number of which is determined beforehand) and confirms whether or not the mashup loaded each time.
Post-conditions	The program will be unchanged, but the user will report back the reliability percentage. 100% would satisfy the “Reliability” requirement.

6.3.14 Availability

Test Overview	This test will ensure that the final product meets the “Availability” requirements set forth in the SRS.
Pre-conditions	The servers, client, and web browser are running.
Actions	The system achieves at least 99.9% uptime.
Post-conditions	The system is usually up and running.

6.3.15 Maintainability

Test Overview	This test will ensure that the product meets the “Maintainability” requirements set forth in the SRS.
Pre-conditions	The system is current functioning properly according to the SRS.
Actions	The system is introduced a new requirement.
Post-conditions	The system still performs all the requirements in the SRS as well as the newly introduced requirement.