Beulah Works LLC

Software Testing Plan (STP) Version 1.2

UML Sequence Diagram File Generator

April 19, 2019

Isis Curiel, Bruno Hnatusko III, Brayden Mccoy,

Dhyey Patel, Jesse Primiani, Jacob Taylor,

and Syed Arshiyan Ali Zaidi

Approvals:

Title	**Signature**	**Date**
Testing Engineer	Bruno Hnatusko III	04/19/2019
Project Manager	Isis Curiel	04/19/2019

Revision History

Date	Revision	Description	Author
4-12-19	0.1	Created rough draft	Isis Curiel
4-17-19	0.2	Spelling fixes, few additions	Bruno Hnatusko III
4-17-19	0.3	Test plan introduction, test items, environment needs, space for test plan approvals	Bruno Hnatusko III
4-18-19	0.4	Added content to section 2.3 and 2.4	Isis Curiel
4-19-19	0.5	Major formatting changes, few spelling fixes	Bruno Hnatusko III
4-19-19	0.6	System test explanation and acceptance test plans	Bruno Hnatusko III
4-19-19	0.7	Edited unit tests	Jacob Taylor
4-19-19	1.0	Indentation adjustments, small fixes	Bruno Hnatusko III
4-26-19	1.1	Reorganized content and added more info	Jesse Primiani
4-26-19	1.2	Reviewed by team	Isis Curiel

Table of Contents

1 Introduction

This document contains the STP for the UML Sequence Diagram Generator. The categories of testing addressed in this document include: unit, system, performance, and acceptance testing. This document describes the testing required to validate the first build of the UML Sequence Diagram File Generator. IEEE standard 829-1998 for Software Testing Documentation is used at every level of testing.

2 UML Sequence Diagram Generator Test Documentation

The STP for UML Sequence Diagram Generator covers test planning, specification, and reporting. There are separate test plans for unit, system, performance, and acceptance testing. Each test plan references its test design, test case, and test procedure specifications. The test reporting documentation consists of a test summary report. Please see Figure 1 (Approach to Tests and their Documentation) in the Appendix for information on testing methodologies for each type of test, as well as the configuration items being tested.

2.1 Environment Needs

- Windows / Mac OS
- Eclipse IDE
- Java JDK & JUnit 4
- Input SDM Model Library
- Aspose.Diagram Library
- Visio Masters File

2.2 Responsibilities

Bruno Hnatusko III, testing engineer (TE), is responsible for managing, preparing, and executing all tests. In addition, the UML Sequence Diagram File Generator development group addresses technical questions and responds to test incident reports.

2.3 Approvals

The completion of a test requires approval of project manager and other team members. If any implementation defects are found, they are reported to the programmer who wrote the faulty code. Design defects will be reported to a Software Architect, and Requirements defects to a Requirements Engineer.

3 Unit Tests

See Figure 2 for a list of classes to be Unit Tested.

3.1 Unit Test Plan

JUnit 4 will be used for unit testing. All unit tests will be run whenever any changes to the code are made.

3.2 Unit Test Design

Test_InputBeulahWorks

Test Method	**Input**	**Expected Output**
Constructors	An empty SDM diagram	Returns a new InputBeulahWorks() object
	A null diagram parameter	Throws an SDMException
getActorCount()	0 actors in the input SDM	Returns int = 0
	1 actor in the input SDM	Returns int = 1
getActorName(index)	2 actors with names in the input SDM. parameter(s): index = 0	Returns string = The name of the first Actor
	2 actors with names in the input SDM. parameter(s): index = 1	Returns string = The name of the second Actor
	0 actors in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 actor in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 actor without a name in the input SDM. parameter(s): index = 0	Returns string = “”
getClassBlockCount()	0 classes in the input SDM	Returns int = 0
	1 class in the input SDM	Returns int = 1
getClassBlockInstanceName (index)	2 classes with instance names in the input SDM. parameter(s): index = 0	Returns string = The instance name of the first Object

	2 classes with instance names in the input SDM. parameter(s): index = 1	Returns string = The instance name of the second Object
	0 classes in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 class in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 class without an instance name in the input SDM. parameter(s): index = 0	Returns string = “”
getClassBlockClassName(index)	2 classes with class names in the input SDM. parameter(s): index = 0	Returns string = The class name of the first Object
	2 classes with class names in the input SDM. parameter(s): index = 1	Returns string = The class name of the second Object
	0 classes in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 class in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 class without a class name in the input SDM. parameter(s): index = 0	Returns string = “”
getActivationBlockCount()	0 activation boxes in the input SDM	Returns int = 0
	1 activation box in the input SDM	Returns int = 1
getLifelineCount()	0 lifelines in the input SDM	Returns int = 0
	1 lifeline in the input SDM	Returns int = 1
getLifelineFromIndex(index)	A lifeline connecting a class to an activation box in the input SDM. parameter(s): index = 0	Returns int = The index of the class

	0 lifelines in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 lifeline with a source in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 lifeline without any source in the input SDM. parameter(s): index = 0	Throws an SDMException
getLifelineToIndex(index)	A lifeline connecting a class to an activation box in the input SDM. parameter(s): index = 0	Returns int = The index of the activation box
	0 lifelines in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 lifeline with a destination in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 lifeline without any destination in the input SDM. parameter(s): index = 0	Throws an SDMException
getLifelineActive(index)	An active lifeline connecting a class to an activation box in the input SDM. parameter(s): index = 0	Returns boolean = true
	An inactive lifeline connecting a class to an activation box in the input SDM. parameter(s): index = 0	Returns boolean = false
	0 lifelines in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 lifeline in the input SDM. parameter(s): index = -1	Throws an SDMException
getMethodCount()	0 message in the input SDM	Returns int = 0
	1 message in the input SDM	Returns int = 1

getMethodFromIndex(index)	A method connecting activation box 1 to box 2 in the input SDM. parameter(s): index = 0	Returns int = The index of the first activation box
	0 methods in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 method with a source in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 method without any source in the input SDM. parameter(s): index = 0	Throws an SDMException
getMethodToIndex(index)	A method connecting activation box 1 to box 2 in the input SDM. parameter(s): index = 0	Returns int = The index of the second activation box
	0 methods in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 method with a destination in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 method without any destination in the input SDM. parameter(s): index = 0	Throws an SDMException
getMethodText(index)	A method with text connecting activation box 1 to box 2 in the input SDM. parameter(s): index = 0	Returns string = “message()”
	0 methods in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 method with text in the input SDM. index = -1	Throws an SDMException
	1 method without text in the input SDM. parameter(s): index = 0	Returns string = “”

getConstraintCount()	0 constraint elements in the input SDM	Returns int = 0
	1 constraint element in the input SDM	Returns int = 1
getConstraintText(index)	2 constraints with text in the input SDM. parameter(s): index = 0	Returns string = The constraint text of constraint 1
	2 constraints with text in the input SDM. parameter(s): index = 1	Returns string = The constraint text of constraint 2
	0 constraints in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 constraint with text in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 constraint without text in the input SDM. parameter(s): index = 0	Returns string = “”
getLoopCount()	0 loop elements in the input SDM	Returns int = 0
	1 loop element in the input SDM	Returns int = 1
getLoopText(index)	2 loops with text in the input SDM. parameter(s): index = 0	Returns string = The loop text of loop 1
	2 loops with text in the input SDM. parameter(s): index = 1	Returns string = The loop text of constraint 2
	0 loops in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 loop with text in the input SDM. parameter(s): index = -1	Throws an SDMException

	1 loop without text in the input SDM. parameter(s): index = 0	Returns string = “”
getAlternativeCount()	0 if-else elements in the input SDM	Returns int = 0
	1 if-else element in the input SDM	Returns int = 1
getAlternativeText(index)	2 if-else elements with text in the input SDM. parameter(s): index = 0	Returns string = The if constraint text of if-else 1
	2 if-else elements with text in the input SDM. parameter(s): index = 1	Returns string = The if constraint text of if-else 2
	0 if-else elements in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 if-else element with text in the input SDM. parameter(s): index = -1	Throws an SDMException
	1 if-else element without text in the input SDM. parameter(s): index = 0	Returns string = “”
getAlternativeTextElse(index)	2 if-else elements with text in the input SDM. parameter(s): index = 0	Returns string = The else constraint text of if-else 1
	2 if-else elements with text in the input SDM. parameter(s): index = 1	Returns string = The else constraint text of if-else 2
	0 if-else elements in the input SDM. parameter(s): index = 0	Throws an IndexOutOfBoundsException
	1 if-else element with text in the input SDM. parameter(s): index = -1	Throws an SDMException

	1 if-else element without text in the input SDM. parameter(s): index = 0	Returns string = “”

Test_OutputAspose

Test Method	**Input**	**Expected Output**
Constructors	N/A	Returns a new OutputAspose() object
initializeDiagram()	N/A	Does not throw an Exception
addActor(name)	name = “Bob”	Returns a string stating an actor named “Bob” was successfully added.
	name = null	Returns a string stating an actor was successfully added.

addClassBlock(instanceName, className)	instanceName = “Bob”
	className = “BobClass”
		Returns a string stating a class of instance “Bob” and class “BobClass” was successfully added.

	instanceName = null className = null	Returns a string stating a class was successfully added.
addActivationBlocks(count)	count = 5	Returns a string stating that 5 activation boxes was successfully added.
	count = 0	Returns a string stating that 0 activation boxes were added.
	count = -1	Returns a string stating that 0 activation boxes were added.
AddLifeline(start, end, active)	3 Activation Boxes were previously added. start = 0, end = 1, & active = true.	Returns a string stating that a lifeline from index 0 to 1 was added.

	3 Activation Boxes were previously added. start = -1, end = 1, & active = true.	Throws an SDMException
	3 Activation Boxes were previously added. start = 0, end = -1, & active = true.	Throws an SDMException
	3 Activation Boxes were previously added. start = 8, end = 1, & active = true.	Throws an SDMException
	3 Activation Boxes were previously added. start = 0, end = 9, & active = true.	Throws an SDMException
AddMethod(start, end, text)	3 Activation Boxes were previously added. start = 0, end = 1, & text = “Bob()”.	Returns a string stating that a method from index 0 to 1 with text “Bob()” was added.
	3 Activation Boxes were previously added. start = -1, end = 1, & text = “Bob()”.	Throws an SDMException
	3 Activation Boxes were previously added. start = 0, end = -1, & text = “Bob()”.	Throws an SDMException
	3 Activation Boxes were previously added. start = 8, end = 1, & text = “Bob()”.	Throws an SDMException
	3 Activation Boxes were previously added. start = 0, end = 9, & text = “Bob()”.	Throws an SDMException
	3 Activation Boxes were previously added. start = 0, end = 1, & text = null.	Returns a string stating that a method from index 0 to 1 was added.
addConstraint(text)	text = “Bob”	Returns a string stating a constraint with text “Bob” was successfully added.

	text = null	Returns a string stating a constraint was successfully added.
addLoop(text)	text = “Bob”	Returns a string stating a loop with text “Bob” was successfully added.
	text = null	Returns a string stating a loop was successfully added.

addAlternative(text, else)	text = “Bob”
	else = “else”
		Returns a string stating a if-else with text “Bob” was successfully added.

	text = null
	else = null
		Returns a string stating a if-else was successfully added.

finalizeDiagram()	N/A	Does not throw an Exception
saveToFile(path, name, type, overwrite)	path = null, name = null, type = null, overwrite = true	Throws an SDMException
	path = null, name = “file”, type = new OutputTypeAsposeVSDX(), overwrite = true	Throws an SDMException
	path = “dir”, name = null, type = new OutputTypeAsposeVSDX(), overwrite = true	Throws an SDMException
	path = “dir”, name = “file”, type = null, overwrite = true	Throws an SDMException

Test_OutputTypeAsposeVSDX

Test Method	**Input**	**Expected Output**
getExtension()	N/A	Returns string = “.vsdx”

getType()	N/A	Returns int = SaveFileFormat.VSDX

Test_SDMtoFile

Test Method	**Input**	**Expected Output**

Constructors	6 constructors:
	
	#1 = SDMtoFile();
	
	#2 = SDMtoFile("dir", "file");
	
	#3 = SDMtoFile(new OutputTypeAsposeVSDX());
	
	#4 = SDMtoFile(new OutputTypeAsposeVSDX(), new OutputAspose());
	
	#5 = SDMtoFile("dir", "file", OutputTypeAsposeVSDX());
	
	#6 = SDMtoFile("dir", "file", OutputTypeAsposeVSDX(), new OutputAspose());
		Returns 6 new SDMtoFile() objects, without throwing any exception, with the configurations specified in each of the 6 constructors

	4 constructors with null parameter(s):
	
	#1 = SDMtoFile(null);
	
	#2 = SDMtoFile(new OutputTypeAsposeVSDX(), null);
	
	#3 = SDMtoFile(null, null. null);
	
	#4 = SDMtoFile(null, null, null, null);
		Throws an SDMException for each of these 4 constructors

setOverwrite(overwrite)	overwrite = true	Does not throw an Exception
setOutputFile(path, name)	path = “”, name = “test”	Does not throw an Exception
	path = null, name = “test”	Throws an SDMException
	path = “”, name = null	Throws an SDMException
	path = null, name = null	Throws an SDMException
setOutputType(type)	type = new OutputTypeAsposeVSDX()	Does not throw an Exception
	type = null	Throws an SDMException
setOutputAdapter(adapter)	adapter = new OutputAspose()	Does not throw an Exception
	adapter = null	Throws an SDMException

exportFile(adapter, log)	adapter = null
	log = null
		Throws an SDMException

3.3 Unit Test Results

JUnit will notify the user if a unit test fails. All 42 Unit Tests were last run on 04/25/2019, and their actual outputs matched the expected outputs for each test.

4 System Tests

Since this project only has a couple highly-coupled methods, integration testing will be covered by the system tests. This project is uses a big bang integration methodology for system / integration testing. These System Tests will test the requirements specified in the SRS, and the architecture and design specified in the SDD. The tests verify that the UML Sequence Diagram File Generator project correctly extracts SDM data and exports it in the correct file format.

4.1 System Test Plan

This test plan covers the tests for the entire SDMfileGenerator package. Tested methods cover the interactions between input from a Beulah Works SDM, output to Aspose, output type subclasses, the SDMtoFile class, and file exportation. After each system test is run, the resulting file will be uploaded to Lucidchart to verify that the expected content was properly added to it. A System Testing interface is included to simplify this process.

4.2 System Test Design

Each System Test is designed based on the specific requirements found in section 3.6 of the SRS.

4.3 System Test Results

A System Test will fail if the generated file does not contain all the expected and required elements after being uploaded to Lucidchart. All 22 System Tests were last run on 04/25/2019, and the exported files contained all the expected and required diagram elements.

5 Performance Tests

5.1 Performance Test Plan

Each Performance Test will generate a file, while measuring both the amount of CPU and Wall-Clock time it took to generate it.

5.2 Performance Test Design

The four performance tests are given as follows:

- Creating and exporting 100 object elements.
- Creating and exporting 100 activation block elements with lifelines.
- Creating and exporting 100 messages between 2 activation blocks.
- Creating and exporting 200 activation blocks.

5.3 Performance Test Results

A Performance Test will fail if a file is not generated within 2 seconds. All performance tests ran in under 1.5 seconds on a Windows 10 Laptop with 8 GB of RAM and an Intel Core i7-4500U Quad Core 2.40 GHz CPU.

6 Acceptance Tests

A single acceptance test is given in a form of a presentation to the customer.

6.1 Acceptance Test Plan

All team members will work on creating a presentation for our customer meant to demonstrate and highlight all the functionality of the project, and to verify that all requirement specifications were met.

6.2 Acceptance Test Design

A presentation will be given by the project manager and one of the software architects. The customer, Victor Guo of BeulahWorks, will be present to view the final presentation which includes a demo of the final product. Along with a product demo, the presentation will include the following: showing the customer how to use the product, presenting a possible maintenance plan, and seeing if the customer is content with the overall design.

6.3 Acceptance Test Results

Victor was satisfied with the project and agreed that it met all of its requirements.

Appendix

Figure 1: Approach to Tests and their Documentation

Test Type	**Approach**	**Corresponding Documents' Section / Configuration Item**
Unit	White and black box; method and class tests, test against detailed requirements and design.	STP section 3 SDD section 6 Source Code
System	Black box, all packages; whole system; test against functional requirements, use cases, architecture, and high-level requirements.	STP section 4 SRS section 3.6 SDD section 5.2 Source Code
Performance	Black box, all packages; whole system; test against performance and other nonfunctional requirements.	STP section 5 SRS section 3.3 SDD section 5.2 Source Code
Acceptance	Given as a presentation to the customer.	STP section 6 SRS all sections Compiled .jar file from Source

Figure 2: Classes to Be Unit Tested

Name	**Version**
InputBeulahWorks	1.0
OutputAspose	1.0
OutputTypeAsposeVSDX	1.0
SDMtoFile	1.0