

# Project Title: Unit Verification and Validation Plan for GlassBR

Vajiheh Motamer

December 16, 2018

## 1 Revision History

Date	Version	Notes
27/11/18	1.0	initial UnitVnVPlan based on new template
07/12/18	2.0	Major changes based on existing modules in the MIS
11/12/18	2.1	Updated changes in some of the modules

# Contents

<b>1</b>	<b>Revision History</b>	<b>i</b>
	<b>List of Tables</b>	<b>ii</b>
<b>2</b>	<b>Symbols, Abbreviations and Acronyms</b>	<b>iii</b>
<b>3</b>	<b>General Information</b>	<b>1</b>
3.1	Purpose . . . . .	1
3.2	Scope . . . . .	1
<b>4</b>	<b>Plan</b>	<b>1</b>
4.1	Verification and Validation Team . . . . .	1
4.2	Automated Testing and Verification Tools . . . . .	1
4.3	Non-Testing Based Verification . . . . .	2
<b>5</b>	<b>Unit Test Description</b>	<b>2</b>
5.1	Tests for Functional Requirements . . . . .	2
5.1.1	Input Module . . . . .	2
5.1.2	LoadASTM Module . . . . .	4
5.1.3	Calc Module . . . . .	5
5.1.4	GlassType ADT Module . . . . .	5
5.1.5	Thickness ADT Module . . . . .	6
5.1.6	SeqServices Module . . . . .	6
5.1.7	FuncADT Module . . . . .	8
5.1.8	ContoursADT Module . . . . .	10
5.1.9	Output Module . . . . .	12
5.2	Traceability Between Test Cases and Modules . . . . .	12
<b>6</b>	<b>References</b>	<b>12</b>

## List of Tables

1	Default Inputs . . . . .	2
2	Valid User Input Test Cases . . . . .	3
3	Derived Quantities Test Cases . . . . .	3

## 2 Symbols, Abbreviations and Acronyms

The symbols, abbreviations, and acronyms used in this document include those defined in the table below, as well as any defined in the tables found in Section ?? [\[this reference didn't work - the other external references also don't seem to work —SS\]](#) of the Software Requirements Specification (SRS) document.

This document provides the unit Verification and Validation (VnV) plan for the software. Section 4 outlines at a high level the plan for verifying and validating the software. Section ?? gives more details about the specific tests that will be used to verify each module and Access Routin [spell check —SS] Semantics. [This shouldn't be in all caps. —SS]

## 3 General Information

### 3.1 Purpose

This document will identify what testing is going to be done on GlassBR. This document is different from the System VnV Plan in that it considers each module as already written according to the Module Instance Specification (MIS) document. These [proof read - watch spacing —SS] test cases can be tested using a white box approach, where the inputs, outputs, and transitions between the inputs and outputs are known.

The purpose of the unit verification and validation activities is to confirm that every module of GlassBR performs its expected actions correctly. The tests described in this document cannot definitively prove correctness, but they can build confidence by verifying that the software is correct for the cases covered by tests.

### 3.2 Scope

The Hardware-Hiding module is implemented by the operating system of the hardware on which GlassBR is running, will not be unit tested as and is assumed to work correctly. Verification of the non-functional requirements is not included in the unit verification plan because they are also sufficiently covered by the System VnV Plan System VnV Plan document. [Don't list System VnV Plan twice. —SS] [List the names of the specific modules that will be tested. Reference your MG —SS]

## 4 Plan

### 4.1 Verification and Validation Team

Responsible member for the verification and validation of GlassBR is Vajiheh Motamer.

### 4.2 Automated Testing and Verification Tools

According to that all test files are going to develop based on Python. The following tools have been considered:

- Unit Testing Tools :
  - pytest: no API. Automatic collection of tests; simple asserts; strong support for test fixture/state management via setup/teardown hooks; strong debugging support via customized traceback. In additional, it is considered as tests runner which Selectively run tests; Stop on first failure
  - unittest : Strong support for test organization and reuse via test suites

[Don't say what tools are being considered. Say what tool you are going to use. —SS]

### 4.3 Non-Testing Based Verification

This section for GlassBR is not applicable. [Why not? Did you say “not applicable” because this is the easiest option for you, or because it isn’t justified for this problem? We are already doing informal review, so with no extra effort, informal review can be part of your verification. —SS]

## 5 Unit Test Description

Test cases have been selected to verify that each module conforms to the specification for the module described in the [Module Interface Specification \(MIS\) document](#). The test cases has been intended to cover all Access Routine Semantics [This shouldn’t be in all caps —SS] both with exceptions and without exceptions.

Many of the tests refer to variables or constants as part of the initial state, input, or expected output. The specifications for these variables and constants can be found in the MIS document for this project.

The values in Table 1 will be used as input for many of the test cases described throughout this section. These values were taken from the [Default Values](#) for this project. Individual test cases will reference the table as input, but specify new values for any input parameter that should have a different value than specified by the table.

Input	Value	Unit
a	1600	m
b	1500	m
g	HS	-
$P_{tol}$	0.008	-
$SD_x$	0	m
$SD_y$	1.5	m
$SD_z$	11.0	m
t	10.0	mm
TNT	1.0	-
w	10.0	kg

Table 1: Default Inputs

### 5.1 Tests for Functional Requirements

#### 5.1.1 Input Module

As described in the MIS, the Input module is expected to read in many [remove the word ‘many’ —SS] user inputs from a file. For each value contained in the file, there is a corresponding test case verifying that the value was properly read into the data structure containing the input parameters.

The input module is also responsible for verifying the input using describing [clarify what you mean here —SS] in `verify_params()` of ?? from MIS, so for each possible violation of an input constraint, there is a corresponding test case verifying that the correct exception was thrown.

## Valid User Input

The test cases described in Table 2 verify that each user input is correctly read. These test cases are identical to each other with the exception of the expected output on which they assert. The input for each is a file containing the inputs specified in Table 1. The type of these test cases is automatic. The initial state for each is a new session. The expected output for each is given in Table 2. The expected output is derived based on the given inputs. The tests will be performed as automated tests on a unit testing framework.

Test Case	Test Name	Expected Output
TC1	test-input_a	$a = 1600$
TC2	test-input_b	$b = 1500$
TC3	test-input_g	$g = HS$
TC4	test-input_ $P_{b_{tol}}$	$P_{b_{tol}} = 0.008$
TC5	test-input_ $SD_x$	$SD_x = 15000$
TC6	test-input_ $SD_y$	$SD_y = 1.5$
TC7	test-input_ $SD_z$	$SD_z = 11.0$
TC8	test-input_t	$t = 10.0$
TC9	test-input_TNT	$TNT = 1.0$
TC10	test-input_w	$w = 10.0$

Table 2: Valid User Input Test Cases

## Derived Quantities

The test cases described in Table 3 to ensure initial inputs from Table 1 regarding to the Constants Values and Calculations [remove the misplaced capitals, clarify this sentence —SS] describing in `load_params(s)` of ?? from MIS have been correctly converted into derived quantities. The expected output for each test case have been considered in Table 3. The type of these test cases is automatic. The initial state for each test case is a new session. The tests will be performed as automated tests on a unit testing framework. [How were the derived quantities calculated? Where are the equations found? Are they in a specific module of the MIS? —SS]

Test Case	Test Name	Expected Output
TC11	test-input_LDF	$LDF = 0.2696493494752911$
TC12	test-input_h	$h = 9.02$
TC13	test-input_GTF	$GTF = 2.0$
TC14	test-input_SD	$SD = 11.10180165558726$
TC15	test-input_AR	$AR = 1.0666666666666667$

Table 3: Derived Quantities Test Cases

## Invalid User Input

The test cases described in Table ?? from System VnV Plan document are intended to cover all invalid input possibilities. Invalid input is input that defies the data constraints described in Section 6.2.6 of the SRS. These test cases are identical to each other with the exception of their input.

TC?? - TC?? in the System VnV Plan document cover these set of tests.

### 5.1.2 LoadASTM Module

This module Reads [you insert capitals in odd locations; read should be lower case; the other odd capitals should also be removed from this document —SS] the data from each ASTM file as a text file and create an object of FunctT and add it to the ContoursT object that is described in ?? from MIS as two (Access Routins Semantics) : i) LoadTSD( $s$ ) which is for 3 second equivalent, pressure ( $q$ ) versus stand off distance (SD) versus charge weight ( $w$ ) and ii) LoadSDF( $s$ ) which is Non dimensional lateral load ( $\hat{q}$ ) versus aspect ration versus Stress distribution factor ( $J$ ).

TC16: test-input\_sTSD

Type: Automatic

Initial State: New Session

Input: testTable1.txt [where is this file? the reader needs to know if they are to conduct the tests —SS]

Output: *out* :=

Expected  $w$  : [4.5, 9.1, 14., 18.]

Expected  $SD$  : [[[6.143397364345, 6.128184215473, 6.344639409554, 7.199850837298],[6.158648279689, 6.158648279689, 6.344639409554, 7.21772438659],[6.189263785047, 6.189263785047, 6.376179502933, 7.253604707984],[6.189263785047, 6.189263785047, 6.407876386546, 7.271611700659],[6.204628563268, 6.22003148438, 6.407876386546, 7.289663395493]]]

Expected  $q$  : [[[4.411877630513, 7.62703125734, 9.952136903603, 9.992650492472],[4.388319475839, 7.575020192327, 10.00554408521, 9.924527029632],[4.364878545185, 7.494339327108, 9.898975926537, 9.871513535251],[4.341579876052, 7.534556975369, 9.793542815831, 9.833410793036],[4.318397091121, 7.434385814135, 9.846098917578, 9.766372924029]]]

Test Case Derivation: None [There is a derivation. How were the values calculated above? How does the reader know they are correct? You don't have to go through all of the details, but they should at least know what steps were involved. Are you just loading the files and then verifying that the input matches? How are you going to access the  $w$ ,  $SD$  values etc. LoadASTM doesn't give you access programs for these, does it? (I haven't checked the MIS to see one way or the other.) You make a similar omission with many of your other test cases. You should say where the expected output comes from, or how it was derived. —SS]

How test will be performed: Automated test on unit testing framework.

### 5.1.3 Calc Module

This Module has been designed for the equations for predicting the probability of glass breakage, capacity, and demand and other routines in the Calc module in the MIS using the input parameters.

The test cases for this Module from TC?? - TC?? in the System VnV Plan document cover these set of tests.

### 5.1.4 GlassType ADT Module

From Section 8 in the MIS, The implementation of the “glass type” and ‘Glass Type Factor’. In this section it represents test cases which covers all semantic routines in this module.

TC17: test\_constructor\_GlassTypeT

Type: Automatic

Initial State: New Session

Input: ( $s$  : “HS”).

Output:  $out := gtf.HTS$

Test Case Derivation: None

How test will be performed: Automated test on unit testing framework.

TC18: test\_GTF

Type: Automatic

Initial State: New Session

Input: ( $s$  : “HS”).

Output:  $out := gtf.HTS = 2.0$

Test Case Derivation: None

How test will be performed: Automated test on unit testing framework.

TC19: test\_toString

Type: Automatic

Initial State: New Session

Input: ( $g$  : gtf.AN ).

Output:  $out := \text{“AN”}$

Test Case Derivation: None

How test will be performed: Automated test on unit testing framework.



### 5.1.5 Thickness ADT Module

From Section 9 in the MIS, the implementation of the thickness type and it Defines abstract data type for thickness. In this section it represents test cases which covers all semantic routins in this module.

TC20: test\_constructorEXP\_ThicknessT

Type: Automatic

Initial State: New Session

Input:  $x : 1.0$

Output: *out* : raise ValueError("Invalid input") [Your quotes are in the wrong direction "Invalid input". This mistake is made throughout the document. —SS]

Test Case Derivation: According to the MIS, if  $x$  be  $(\neg(x \in T))$ , the system throws an exception. So, this test case covers this exception.

How test will be performed: Automated test on unit testing framework.

TC21: test\_toMinThick

Type: Automatic

Initial State: New Session

Input:  $x := t$  in MIS in the section 9

Output: *out* := According to the MIS in the section 9

Test Case Derivation: None

How test will be performed: Automated test on unit testing framework.

TC22: test\_toFloat

Type: Automatic

Initial State: New Session

Input:  $x : 2.5, 8.0, 22.0$

Output: *out* :  $x$

Test Case Derivation: None

How test will be performed: Automated test on unit testing framework.

### 5.1.6 SeqServices Module

TC23: test\_isAscending\_true

Type: Automatic

Initial State: New Session

Input:  $(X : [1, 2, 3, 4, 5.5])$ .

Output: *out* := true

Test Case Derivation: This test case has been designed to Check to see if a given list is monotonically increasing. In this test case, the input has been sorted in the ascending sequence. So, the expected result would be true.

How test will be performed: Automated test on unit testing framework.

TC24: test\_isAscending\_false

Type: Automatic

Initial State: New Session

Input: (X : [1, 2, 3, 4, 0]).

Output: *out* := false

Test Case Derivation: In this test case, the input has not been sorted in the ascending sequence. So, the expected result would be false.

How test will be performed: Automated test on unit testing framework.

TC25: test\_isInBounds\_true

Type: Automatic

Initial State: New Session

Input: (X : [1, 2, 3, 4, 5.5] , x : 3.5)

Output: *out* := true

Test Case Derivation: Regarding with MIS, this routine Checks to see if a given value is within the bounds of a provided list. In this test case, the input has not been is within the bounds of the given list. So, the expected result would be true.

How test will be performed: Automated test on unit testing framework.

TC26: test\_interpLin\_x

Type: Automatic

Initial State: It is assumed that isAscending is True and isInBounds is True as well.

Input: (X : [1, 2, 3, 4, 5.5] , x : 1 , x : 3.12345678 )

Output: *out* := 0 , *out* := 2

Test Case Derivation: Regarding to th MIS, this test case Determines the best "placement" for a value against a provided list. In this test case, in the first x, it is present in list. In the second x,  $x < X(2 + 1)$  and it will return *i* which is 2.

How test will be performed: Automated test on unit testing framework.

TC27: test\_interpQuad\_x

Type: Automatic

Initial State: It is assumed that isAscending is True.

Input:  $(x_0, y_0, x_1, y_1, x_2, y_2, x) = (0, 0, 3, 6, 4.5, 5.67, 4)$

Output:  $out := y_1 + \frac{y_2 - y_0}{x_2 - x_0}(x - x_1) + \frac{y_2 - 2y_1 + y_0}{2(x_2 - x_1)^2}(x - x_1)^2 = 5.8533333333333335$

Test Case Derivation: Regarding to the MIS, *out* and input values are members of  $\mathbb{R}$ , this test case is to test of Quad interpolation function which returns interpolated value which is calculated using provided inputs and .

How test will be performed: Automated test on unit testing framework.

TC28: test\_index\_x

Type: Automatic

Initial State: It is assumed that isAscending is True.

Input: (X : [1, 2, 3, 4, 5.5] )

Output:  $out := y_1 + \frac{y_2 - y_0}{x_2 - x_0}(x - x_1) + \frac{y_2 - 2y_1 + y_0}{2(x_2 - x_1)^2}(x - x_1)^2 = 5.8533333333333335$

Test Case Derivation: Regarding to the MIS, *out* and input values are members of  $\mathbb{R}$ , this test case is to test of Quad interpolation function which returns interpolated value which is calculated using provided inputs and .

How test will be performed: Automated test on unit testing framework.

Output: *out* := false

Test Case Derivation: In this test case, the input has not been sorted in the ascending sequence. So, the expected result would be false.

How test will be performed: Automated test on unit testing framework.

TC29: test\_isInBounds\_true

Type: Automatic

Initial State: New Session

Input: (X : [1, 2, 3, 4, 5.5] , x : 3.5)

Output: *out* := true

Test Case Derivation: Regarding with MIS, this routine Checks to see if a given value is within the bounds of a provided list. In this test case, the input has not been is within the bounds of the given list. So, the expected result would be true.

How test will be performed: Automated test on unit testing framework.

### 5.1.7 FuncADT Module

From Section 10 in the MIS, this module is used to represent a two dimensional function. For instance, the data may be represented by a table of data, or a polynomial, or a piecewise linear function, etc. It Defines the abstract data type for functions, providing a constructor and a means to evaluate the function for a given value of the independent variable. In this section it represents test cases which covers all semantic routines.

TC30: test\_minD

Type: Automatic

Initial State: New Session

Input: ( X : [1, 2, 10] , Y : [1, 2, 10] , o : 1 )

Output:  $out := \min X : 1$

Test Case Derivation: None

How test will be performed: Automated test on unit testing framework.

TC31: test\_maxD

Type: Automatic

Initial State: New Session

Input: (  $X : [1, 2, 10]$  ,  $Y : [1, 2, 10]$  ,  $o : 1$  )

Output:  $out := \min X : 10$

Test Case Derivation: None

How test will be performed: Automated test on unit testing framework.

TC32: test\_eval

Type: Automatic

Initial State: New Session

Input: (  $X : [1, 2, 10]$  ,  $Y : [1, 2, 10]$  ,  $x = 3$  ,  $o : 1$  )

Output:  $out := \text{interpLin}(X_i, Y_i, X_{i+1}, Y_{i+1}, x) = 3$

Test Case Derivation: None

How test will be performed: Automated test on unit testing framework.

TC33: test\_FuncT\_notAscendingErr

Type: Automatic

Initial State: New Session

Input: ( $X : [5, 4, 6]$  ,  $Y : [0, 1, 3]$  ,  $i : 2$ ).

Output:  $out := \text{raise IndepVarNotAscending}(\text{'Independent variables are not in ascending order!'})$

Test Case Derivation: This test case uses of isAscending from SeqServices module. Because X is not in the ascending sequence, the system must raise an exception.

How test will be performed: Automated test on unit testing framework.

TC34: test\_FuncT\_ssmErr

Type: Automatic

Initial State: New Session

Input: ( $X : [0, 1, 2]$  ,  $Y : [0]$  ,  $i : 1$ ).

Output:  $out := \text{raise SeqSizeMismatch}(\text{'Sequences are not of the same length!'})$

Test Case Derivation: Because Sequences are not in the same length, the system raises an exception.

How test will be performed: Automated test on unit testing framework.

TC35: test\_FuncT\_InvalidInterpErr

Type: Automatic

Initial State: New Session

Input: (X : [1, 2, 3], Y: [0, 1, 3], i : 0 or 3).

Output: *out* := raise InvalidInterpOrder("Invalid interpretation error!")

Test Case Derivation: According to the MIS, *i* must be between [1..MAX\_ORDER]. So, this test case covers this exception.

How test will be performed: Automated test on unit testing framework.

TC36: test\_FuncT\_oodErr

Type: Automatic

Initial State: New Session

Input: ( X : [1, 2, 10] , Y : [1, 2, 10] , o : 1 , x: 100).

Output: *out* := raise OutOfDomain('Out of domain!')

Test Case Derivation: *x* must be between minX and maxX. So, this test case covers this exception.

How test will be performed: Automated test on unit testing framework.

### 5.1.8 ContoursADT Module

From Section 11 in the MIS, the data structure used to represent a sequence of two dimensional functions in the form of contours. Each function is associated with a particular value, like a contour in a contour map. It Defines the abstract data type for a sequence of functions. Allows adding/removing functions to/from the sequence. Provides evaluation of the 3d function at values between the contours. In this section it represents test cases which covers all semantic routines in this module.

TC37: test\_constructorEXP\_ContoursT

Type: Automatic

Initial State: (*S* : [], *Z* : [], *o* : i)

Input: (i : 0 or 3).

Output: *out* := raise InvalidInterpOrder("Invalid interpretation error!")

Test Case Derivation: According to the MIS, *i* must be between [1..MAX\_ORDER]. So, this test case covers this exception.

How test will be performed: Automated test on unit testing framework.

TC38: test\_addEXP\_ContoursT

Type: Automatic

Initial State: (*S* : [], *Z* : [], *o* : i)

Input: (i : 0 or 3).

Output:  $out := \text{raise InvalidInterpOrder}(\text{"Invalid interpretation error!"})$

Test Case Derivation: According to the MIS,  $i$  must be between  $[1..MAX\_ORDER]$ . So, this test case covers this exception.

How test will be performed: Automated test on unit testing framework.

TC39: Add\_constructorEXP\_ContoursT

Type: Automatic

Initial State:  $(S : [], Z : [], o : i)$

Input:  $(i : 0 \text{ or } 3)$ .

Output:  $out := \text{raise InvalidInterpOrder}(\text{"Invalid interpretation error!"})$

Test Case Derivation: According to the MIS,  $i$  must be between  $[1..MAX\_ORDER]$ . So, this test case covers this exception.

How test will be performed: Automated test on unit testing framework.

TC40: test\_addEXP\_ContoursT

Type: Automatic

Initial State:  $(S : [], Z : [], o : i)$

Input:  $(S : [1, 2, 10], z : 0)$

Output:  $out := \text{add}(s, z) : \text{raise InvalidInterpOrder}(\text{"Independent variables not in ascending order!"})$

Test Case Derivation: According to the MIS, this routine appends elements to state variable sequences if  $(|Z| > 0 \wedge z < Z_{|Z|-1} \Rightarrow \text{IndepVarNotAscending})$  system throws an exception. So, this test case covers this exception.

How test will be performed: Automated test on unit testing framework.

TC41: test\_getC\_ContoursT

Type: Automatic

Initial State: New Session

Input:  $(S : [[1, 2, 10], [2, 3, 11]], i : 1)$

Output:  $out := \text{getC}(1) : [2, 3, 11]$

Test Case Derivation: This method gets the  $i$ th element from the sequence  $S$  and it returns the  $i$ th element from  $S$ .

How test will be performed: Automated test on unit testing framework.

TC42: test\_getCEXP\_ContoursT

Type: Automatic

Initial State: New Session

Input:  $(S : [[1, 2, 10], [2, 3, 11]], i : 3)$

Output:  $out := \text{getC}(3) : \text{raise InvalidIndex}(\text{"Index out of range"})$

Test Case Derivation: According to the MIS, if  $(\neg(0 \leq i < |S|))$ , system throws an exception. So, this test case with  $i=3$  covers this exception.

### 5.1.9 Output Module

The verification of the output is covered by ?? in the System VnV Plan document. The delivery of each output is covered by TC?? to TC?? in the System VnV Plan document. This document therefore only adds test cases for items not covered by the System VnV Plan document.

## 5.2 Traceability Between Test Cases and Modules

Every module is covered by test cases except for those that were declared out of scope in Section 3.2 of this document.

## 6 References

[There is no bibliography in this document, and if there was, its name wouldn't be SRS. —SS]