Glass Breakage Analysis: System Verification and Validation Plan for GlassBR [Include the name GlassBR here —SS][Fixed —VM]

Vajiheh Motamer

December 26, 2018

1 Revision History

Date	Version	Notes
10/26/18	1.0	Initial Draft
12/21/18	1.1	Updates based on Dr. Smith's feedback

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 2.3 of the Software Requirements Specification (SRS) document.

symbol	description
TC	Test Case
VnV	Verification and Validation
MIS	Module Interface Specification
MG	Module Guide

Contents

List of Tables

This document presents the system verification and validation plan for the software. General information regarding the system under test and the objectives of the verification and validation activities is provided in Section 3. Overviews of verification plans for the SRS, design, and implementation are given in Section 4, along with a summary of the validation plan for the software. Section 5 details specific system test cases for verifying the requirements outlined in Section 7 of the SRS. A summary of planned static verification activities can be found in Section **??** of this document.

3 General Information

3.1 Summary

The software being tested is the Glass Breakage Analysis Program (GlassBR). Based on user-defined glass type and blast properties, GlassBR interprets the inputs to give out the outputs which predict whether the glass slab can withstand the blast under the given conditions. The blast under consideration is a type of blast load. Software is helpful to efficiently and correctly predict the blast risk involved with the glass slab using an intuitive interface.

3.2 Objectives

The purpose of the verification and validation activities is to confirm that GlassBR exhibits desired software qualities. The primary objective is to build confidence in the correctness of the software. 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. Other important qualities to be verified are the portability and reusability of the software.

3.3 Relevant Documentation

Extensive information about the purpose and requirements of GlassBR can be found in the SRS document. This System VnV Plan is complemented by the System VnV Report, where the results of the tests planned in this document are discussed. For more details on test cases specific to the implementation of GlassBR, consult the Unit VnV Plan document. The latest documentation for GlassBR can be found on GitHub, at https://github.com/smiths/caseStudies/tree/master/CaseStudies/glass. [Good! —SS][Thanks! I have fixed it throughout the document. —VM]

4 Plan

4.1 Verification and Validation Team

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

4.2 SRS Verification Plan

The SRS for the project will be reviewed by Dr. Smith and some of the class mates in CAS741. So, feedback will be provided. Some SRS feedback for this project have been provided and addressed using github issue tracker. Besides, during preparing of all steps SRS will be reviewed and if would be required, the changes would be made.

4.3 Design Verification Plan

The design of GlassBR 's documents will be verified by getting feedback from Dr. Spencer Smith and my CAS 741 classmates : Malavika Srinivasan for the MG and Robert White for the MIS. The Module Guide and Module Interface Specification will contain information about the software design. Feedback is expected to be provided by reviewers via github issue tracker. [You can be more descriptive here, since you know the names of who is going to review each of your documents (through Repos.xlsx) —SS][Fixed. —VM]

4.4 Implementation Verification Plan

The implementation of the GlassBR program will be verified by performing statically code review with Dr. Smith [LaTeX has a rule that it inserts two spaces at the end of a sentence. It detects a sentence as a period followed by a capital letter. This comes up, for instance, with Dr. Smith. Since the period after Dr. isn't actually the end of a sentence, you need to tell LaTeX to insert one space. You do this either by Dr. Smith (if you don't mind a line-break between Dr. and Smith), or Dr. Spencer Smith (to force LaTeX to not insert a line break). —SS][Fixed —VM] and the classmates in CAS741 and dynamically by executing the test cases detailed in this plan and the unit VnV plan using Python [spell check! —SS][Fixed —VM] testing frameworks.

4.5 Software Validation Plan

There is no validation plan for GlassBR because there is not any external validation source.[You should the reason why there is no software validation plan. —SS] [Done —VM]

5 System Test Description

System testing for the GlassBR ensures that the correct inputs produce the correct outputs. The test cases in this section are derived from the instance models and the requirements detailed in the tool's SRS. These values were taken from the test input files for GlassBR which can be found on GitHub, at https://github.com/smiths/caseStudies/tree/master/CaseStudies/glass. [proof read! —SS][Done —VM]. Individual test cases will reference the table as input in Table 1 but specify new values for any input parameter that should have a different value than specified by the table.

5.1 Tests for Functional Requirements

5.1.1 User Input Tests

Valid User Input
The following set of test cases is intended to cover different forms of valid user input. These values were taken from the test input files for GlassBR can be found on GitHub, at https://github.com/smiths/caseStudies/tree/master/CaseStudies/glass/src. [You should say where the values in Table 1 come from. —SS] [I 'm going to move test cases in the "src" to the test folder, so if you approve, I will do. So, maybe it would be better to reference to exact path of the project. would'nt it? —VM]

TC1: Tst_Pb_DefaultValues

Control: Automatic

Initial State: New session

Input: As described in Table 1.

Output:

$$P_b = (1.301524590203718e - 04) < P_{b_{\text{tol}}} \;,$$
$$Demand(q) = (3.258285992018616e + 00),$$
$$Capacity(LR) = (6.843002155788037e + 00) > q,$$

is_safePb = True and is_safeLR = True, The glass is considered safe.

[Where did these expected values come from? You should tell the reader so that they can judge whether these values make sense. —SS] [it has described in the test case derivation. —VM] [You don't need brackets around your numbers. —SS] [For scientific notation, you don't use the e notation in LaTeX, you can write the multiplication and the power of 10 explicitly. For instance, $P_b = 1.301524590203718 \times 10^{-4}$ —SS]

How test will be performed: Unit testing using PyUnit.

Test Case Derivation: These expected values has been derived using of equations from Risk of Failure (B) from DD1 and "Probability of Glass Breakage" from DD12 based on the input values in the Table 1.

Input	**Value**	**Unit**
a	1600	m
b	1500	m
g	HS	-
$P_{b_{\text{tol}}}$	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: Inputs for Tst_Pb_DefaultValues

[You don't normally need to hard code in newpages —SS][Thanks! Done throughout the document. —VM]

TC2: Tst_ Pb_ SmallDimensionValues [I don't understand how the dimensions are small values. They look like regular values. —SS] [It has been considered smaller than dimentions in the table 1. Besides, I have taken these values from TestingPythonGlassBR.pdf in the path of project that it has referenced to testinput1.txt for small dimentions as input file. —VM]

Control: Automatic

Initial State: New session

Input: As described in Table 2.

Output:

$$P_b = (1.8246621494248 94e-03) < P_{b_{\text{tol}}},$$
$$Demand(q) = (3.658003449421614e+00),$$
$$Capacity(LR) = (4.916016610996773e+00) > q,$$

is_safePb = True and is_safeLR = True, The glass is considered safe.

How test will be performed: Unit testing using PyUnit.

Test Case Derivation: 1 and 1 in SRS.

Input	**Value**	**Unit**
a	1200	m
b	1000	m
g	AN	-
$P_{b_{\text{tol}}}$	0.010	-
SD_x	0	m
SD_y	2.0	m
SD_z	10.0	m
t	8.0	mm
TNT	1.0	-
w	10.0	kg

Table 2: Inputs for Tst_ Pb_ SmallDimensionValues

TC3: Tst_Pb_LargeDimensionValues

Control: Automatic

Initial State: New session

Input: As described in Table 3.

Output:

$$P_b = (3.459068155453604e - 04) < P_{b_{\mathrm{tol}}},$$
$$Demand(q) = (5.777809021268771e + 00),$$
$$Capacity(LR) = (1.092974208994522e + 01) > q\ ,$$

is_safePb = True and is_safeLR = True, The glass is considered safe.

How test will be performed: Unit testing using PyUnit.

Test Case Derivation: 1 and 1 in SRS.

Input	**Value**	**Unit**
a	1600	m
b	1500	m
g	HS	-
$P_{b_{\mathrm{tol}}}$	0.010	-
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 3: Inputs for Tst_ Pb_ LargeDimensionValues

TC4: Tst_ Pb_ LowPbTol

Control: Automatic

Initial State: New session

Input: As described in Table 4.

Output:

$$P_b = (1.301524590203718e - 04) > P_{b_{\text{tol}}},$$
$$Demand(q) = (3.258285992018616e + 00),$$
$$Capacity(LR) = (3.124424950223241e + 00) \not> q \ ,$$

is_safePb = False and is_safeLR = False, The glass is NOT considered safe.

How test will be performed: Unit testing using PyUnit.

Test Case Derivation: 1 and 1 in SRS.

Input	**Value**	**Unit**
a	1600	m
b	1500	m
g	HS	-
$P_{b_{\text{tol}}}$	0.008	-
SD$_x$	0	m
SD$_y$	2.5	m
SD$_z$	6.0	m
t	10.0	mm
TNT	1.0	-
w	10.0	kg

Table 4: Inputs for Tst_ Pb_ LowPbTol

TC5: Tst_ Pb_ DiffSDValues

Control: Automatic

Initial State: New session

Input: As described in Table 5.

Output:

$$P_b = (1.185574651484522e - 02) > P_{b_{\text{tol}}},$$
$$Demand(q) = (7.377747177423622e + 00),$$
$$Capacity(LR) = (6.843002155788037e + 00) \not> q \ ,$$

is_safePb = False and is_safeLR = False, The glass is NOT considered safe.

How test will be performed: Unit testing using PyUnit.

Test Case Derivation: 1 and 1 in SRS.

Input	**Value**	**Unit**
a	1600	m
b	1500	m
g	HS	-
$P_{b_{\text{tol}}}$	0.008	-
SD$_x$	0.0	m
SD$_y$	2.5	m
SD$_z$	6.0	m
t	0.008	mm
TNT	1.0	-
w	10.0	kg

Table 5: Inputs for Tst_ Pb_ DiffSDValues

TC6: Tst_ Pb_ HighChgWght

Control: Automatic

Initial State: New session

Input: As described in Table 6.

Output:

$$P_b = (2.497577817262034e - 01) > P_{b_{\text{tol}}},$$
$$Demand(q) = (1.428204355548630e + 01),$$
$$Capacity(LR) = (6.843002155788037e + 00) \not> q,$$

is_safePb = False and is_safeLR = False, The glass is NOT considered safe.

How test will be performed: Unit testing using PyUnit.

Test Case Derivation: DD12 and IM?? in SRS.

Input	**Value**	**Unit**
a	1600	m
b	1500	m
g	HS	-
$P_{b_{\text{tol}}}$	0.008	-
SD$_x$	0	m
SD$_y$	1.5	m
SD$_z$	11.0	m
t	10.0	mm
TNT	1.0	-
w	60.0	kg

Table 6: Inputs for Tst_ Pb_ HighChgWght

TC7: Tst_ Pb_ LowThickness

Control: Automatic

Initial State: New session

Input: As described in Table 7.

Output:

$$P_b = (2.528418262282350e - 01) > P_{b_{\text{tol}}},$$
$$Demand(q) = (3.258285992018616e + 00),$$
$$Capacity(LR) = (1.716982174845693e + 00) \not> q,$$

is_safePb = False and is_safeLR = False, The glass is NOT considered safe.

How test will be performed: Unit testing using PyUnit.

Test Case Derivation: 1 and 1 in SRS.

Input	**Value**	**Unit**
a	1600	m
b	1500	m
g	HS	-
$P_{b_{\text{tol}}}$	0.008	-
SD_x	0	m
SD_y	1.5	m
SD_z	11.0	m
t	3.0	mm
TNT	1.0	-
w	10.0	kg

Table 7: Inputs for Tst_ Pb_ LowThickness

[The input and output values for the above test cases do not seem to follow a discernible pattern. Multiple values are modified between tests, but I don't know the rationale for the modification. You would usually hold all values constant and then only modify 1 value (or maybe two values, if they are related like a and b). This approach is more systematic, and it also means you don't have to repeat the entire table every time. You certainly do want test cases where the glass is safe and where it isn't, but you didn't distinguish the names of any of your test cases based on this. —SS][Dr. Smith I did not make sense of your comment, as you mentioned I held all values constant and only modify one of the values, for example in this case I modified just thickness which is too low and the result is not considered safe. —VM]

Invalid User Input

The test cases described in Table 8 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. The input for each is specified in Table 8. For each test case, the inputs which have not been specified in this table, have been specified in Table 1. Besides,The [proof read —SS] expected output has been specified for each test case and the control method for these test cases is automatic. The initial state for each is a new session. The tests will be performed as automated tests on the PyUnit.

Table 8: TestCheckConstraints

Test Case	Test Name	Significant Input	Expected Output
TC8	checkAPositiveTest	a = -1600	InputError: a and b must be greater than 0
TC9	checkBPositiveTest	b = -1500	InputError: a and b must be greater than 0
TC10	checkSmallAspectRTest	b = 2000	(a/b=0.8<1); InputError: a/b must be between 1 and 5
TC11	checkLargeAspectRTest	b = 200	(a/b=8>5); InputError: a/b must be between 1 and 5
TC12	checkValidThicknessTest	t = 7	InputError: t must be in [2.5,2.7,3.0,4.0,5.0,6.0,8.0, 10.0,12.0,16.0,19.0,22.0]
TC13	checkLowerConstrOnWTest	w = 3	InputError: wtnt must be between 4.5 and 910
TC14	checkUpperConstrOnWTest	w = 1000	InputError: wtnt must be between 4.5 and 910
TC15	checkTNTPositiveTest	tnt = -2	InputError: TNT must be greater than 0
TC16	checkLowerConstrOnSDTest	sdx = 0; sdy = 1.0; sdz = 2.0	InputError: SD must be between 6 and 130
TC17	checkUpperConstrOnSDTest	sdx = 0; sdy = 200; sdz = 100	InputError: SD must be between 6 and 130
TC18	incorrectA0Test	a = 0	InputError: a and b must be greater than 0
TC19	incorrectB0Test	b = 0	InputError: a and b must be greater than 0
TC20	incorrectTNT0Test	tnt = 0	InputError: TNT must be greater than 0
TC21	incorrectAspectREqLwrBndTest	a = 1500; b = 1500	(a/b = 1); “a/b must be between 1 and 5, equation would be uncertain.”
TC22	incorrectAspectREqUpprBndTest	a = 7500; b = 1500	(a/b = 5); “a/b must be between 1 and 5, equation would be uncertain.”
TC23	incorrectWEqLwrBndTest	w = 4.5	”wtnt must be greater than or equal $Wmin$ = 4.5, equation would be uncertain.”
TC24	incorrectWEqUpprBndTest	w = 910	”wtnt must be less than or equal $Wmax$ = 910, equation would be uncertain.”
TC25	incorrectSDEqLwrBndTest	sdx = 0; sdy = 6; sdz = 0	“SD must be greater than or equal $SDmin$ = 6, equation would be uncertain.”
TC26	incorrectSDEqUpprBndTest	sdx = 130; sdy = 0; sdz = 0	“SD must be greater than or equal $SDmax$ = 130, equation would be uncertain.”

[The error messages for the last test cases are all the same “Encountered an unexpected exception”. Is this what the SRS says? We really could have a more descriptive error message, as for the other cases. —SS] [Although wtnn and SD equation in the UpperBand and Lowerband has been considered in the SRS as valid value, however it throw an error which it will been reported in the SystemVnVReport. —VM]

[Use “uses” to get the correct opening and closing quotes. —SS][Thanks! Fixed throughout the document. —VM]

5.1.2 Output Tests

Test Derived Values

The following set of test cases are intended to verify initial inputs have been correctly converted into derived quantities. These test cases follow term definitions and equations in the Section 6.2.4 of the SRS. [proof read —SS][Done! —VM] For each test case, one input column references to the specified input table. Besides,The expected output has been specified for each test case and the control method for these test cases is automatic. The initial state for each is a new session. The tests will be performed as automated tests on the PyUnit.

[The text is better for version control, and for reading in other editors, if you use a hard-wrap at 80 characters —SS][Done! —VM]

Table 9: TestDerivedValues

Test Case	Test Name	Input Table	AR Expected	SD Expected	LDF Expected	wTNT Expected	h Expected	GTF Expected
TC27	Table 1	1.0666666666666667	11.10180165558726	0.2696493494752911	10.0	9.02	2	
TC28	Table 2	1.2	10.198039027185569	0.2696493494752911	10.0	7.42	1	
TC29	Table 3	1.25	9.093404203047394	0.2696493494752911	15.0	9.02	4	

5.2 Tests for Nonfunctional Requirements

5.2.1 Portability test

TC30: test-portability

Type: Manual

Initial State:There is a completed implementation of GlassBR

Input/Condition: -

Output/Result:-

How test will be performed: All unit tests will be run on all environments. Running GlassBR on Mac, Windows and Linux operating systems has been considered to test this requirement. [You can be more specific here. My suggestion is that you explicitly say that all unit tests will be run on all environments. —SS] [Thanks! —VM]

5.2.2 Reusability test

TC31: test-reusability

Type: Manual

Initial State: There is a completed implementation of GlassBR

Input/Condition: -

Output/Result: An alternative version of GlassBR that uses the input code. All modules in the alternative version should be identical to the existing GlassBR modules, with the exception of the Input Module.

How test will be performed: If only the Input Module is changed from the existing version, then the test passes and confidence in the reusability of GlassBR's modules is increased. If other modules need to be changed, the test fails and the other modules must be modified to be completely reusable.

5.2.3 Correctness Tests

The correctness NFR is covered by the functional requirement test cases for calculations, found in Section 5.1.1.

5.3 Traceability Between Test Cases and Requirements

The purpose of the traceability matrix shown in Table 10 is to provide easy references on which requirements are verified by which test cases, and which test cases need to be updated if a requirement changes. If a requirement is changed, the items in the column of that requirement that are marked with an "X" may have to be modified as well.

	R1	R2	R??	R4	R5	R6
TC1 - TC7	X					
TC1 - TC7		X				
TC8 - TC??			X			
TC1 - TC7				X		
TC1 - TC7					X	
TC27 - TC29						X

Table 10: Traceability Matrix Showing the Connections Between Requirements and Test Cases

[You can remove this section. —SS][Done! —VM]

[Since the system tests for GlassBR are fairly simple, you should be looking at incorporating additional tests in your plan. For your program a usability survey would be a good addition. We want it to be easy for users to get their results from GlassBR. You could put some survey questions in your Appendix and introduce this idea in the main body of your plan. —SS] [I put this section in the document, however I am not sure this questions would be appropriate or not! —VM]

5.4 Usability Survey Questions

1. Was the output delivered more quickly or more slowly than expected?
2. Did the program accept the valid inputs?
3. Did initial inputs have been correctly converted into derived quantities (AR, SD, LDF, $wTNT$, h, GTF)?
4. Were you able to install and test the program using the instructions?