System Verification and Validation Plan for Truss

Ting-Yu Wu

October 29, 2020

1 Revision History

Date	Version	Notes
October 29, 2020	1.0	Initial version of VnV plan

Contents

List of Tables

List of Figures

Remove heading if you don't need it

2 Symbols, Abbreviations and Acronyms

symbol	description
FR	Functional Requirements
NFR	Nonfunctional Requirements
R	Requirements
SRS	Software Requirements Specification
VnV	Verification and Validation

This document provides an overview of the Verification and Validation (VnV) plan for Truss. The general information is introduced in section 3. Verification plans and test description are in section 4 and section 5, respectively.

3 General Information

3.1 Summary

The software being test in this document is Truss. Users can input the external force and the structure of the truss, the software will calculate all the internal forces within truss members and ouput the result with a .txt file.

3.2 Objectives

The objective of the VnV plan is to verify the FR and NFR described in the SRS. We will test all the functional requirements and nonfunctional requirements in Section 5. The most important goals are building confidence in the software correctness and increasing the reliability of the software.

3.3 Relevant Documentation

- SRS for Truss

[Other documents will be included after building —Author]

You can include from now

4 Plan

This section lists the VnV plan of Truss. Section 4.1 introduces the members of the VnV team. Verification plans of SRS, design, and implementation are covered in section 4.2, section 4.3, and section 4.4, respectively. Section 4.5 outlines the tools that are used for automated testing. Section 4.6 outlines the validation plan of the software.

4.1 Verification and Validation Team

This section lists the members of verification and validation team.

- Ting-Yu Wu
- Dr. Spencer Smith
- Dr. Jacques Carette
- Tiago de Moraes Machado reviews the whole project
- Xuanming Yan reviews the SRS
- Mohamed AbuElAla reviews the VnV plan

4.2 SRS Verification Plan

The SRS can be reviewed by the author's supervisors and classmates from the class. Reviewers can give feedbacks and revision suggestions to the author by creating issues on GitHub. It is author's responsibility to check the submitted issues regularly and make necessary revisions.

4.3 Design Verification Plan

The design of this software will be verified by testing functional requirements and nonfunctional requirements. Test cases for functional requirements are listed in section 5.1, and test cases for nonfunctional requirements are listed in section 5.2.

4.4 Implementation Verification Plan

The implementation verification plan includes the followings:

- Code walkthroughs. The rubber duck testing method will be implemented. The procedure involve explaining the code line by line to the duck and go into detail of what the code is supposed to do.
- Expert review. This verification will be performed by individuals in the verification and validation team, as listed in section 4.1, by paying close attention and looking for potential implementation errors.
- Unit testing. Tool we used for unit testing is PyUnit, the python unit testing framework. More details are outlined in section 6.

4.5 Automated Testing and Verification Tools

We implement PyUnit for automated unit testing.

[More details and tools will be included later —Author]

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS] [The details of this section will likely evolve as you get closer to the implementation. —SS]

4.6 Software Validation Plan

The software will be validated by testing the correctness of outputs, which is covered in section 5.1.2.

5 System Test Description

5.1 Tests for Functional Requirements

The functional requirements are described in the SRS. Truss shall verify that the inputs are valid and the calculated outputs are correct. FR1 and FR2 will be tested in section 5.1.1. FR3 and FR4 will be tested in section 5.1.2.

5.1.1 Input Verification

According to FR1 and FR2 in the SRS, Truss shall take inputs from users and verify whether the inputs meet the data constraints, as described in the section 4.2.6 in SRS. If the input values are incorrect or out of bounds, the software shall display an error message.

Input Verification test

1. Valid inputs

You can replace this test with a test that actually looks at the output to verify its correctness

Control: Automatic

Initial State: Truss is started and running

Input: $F_1 = 500$, $x_1 = 3$, $x_2 = 7$, $\theta_1 = 30$, $\theta_2 = 30$

Output: Generate an output file and display a success message

Test Case Derivation: Successfully generate an output file

How test will be performed: Automated system test

2. Invalid distance

Does Pascal generate exception for the code? (I hopes!) This would be a ValueError in Python

Control: Automatic

Initial State: Truss is started and running

Input: $x_1 = 0$; $x_1 = -3$; $x_2 = 0$; $x_2 = -3$

Output: An error message of "Please input positive value for the distance"

Test Case Derivation: Successfully display the error message

How test will be performed: Automated system test

3. Invalid angle

Control: Automatic

Initial State: Truss is started and running

Input: $\theta_1 = 0$; $\theta_1 = 90$; $\theta_1 = -45$; $\theta_2 = 0$; $\theta_2 = 90$; $\theta_2 = -45$

Output: An error message of "Please input the value greater than 0 and less than 90 for the angle"

Test Case Derivation: Successfully display the error message

How test will be performed: Automated system test

[More specific input cases will be included later. —Author]

← You can write these test now.

5.1.2 Output Verification

According to FR3 and FR4 in the SRS, Truss shall calculate equations and output the correct values for all internal forces.

Output Correctness test

1. Simple case

   Control: Automatic

   Initial State: Not applicable

   Input: $F_1 = 500$, $x_1 = 3$, $x_2 = 7$, $\theta_1 = 30$, $\theta_2 = 30$

   Output: $F_{AC} = -700$, $F_{AD} = 606.22$, $F_{BC} = -300$, $F_{BD} = 259.81$, $F_{CD} = 500$

   Test Case Derivation: Successfully calculate the expected outputs

   How test will be performed: Automated system test

2. Correctness of output file

   Control: Automatic

   Initial State: Not applicable

   Input: the output file

   Output: the relative error between each value at each time step

   Test Case Derivation: Successfully calculate the expected outputs

   How test will be performed: Automated system test

5.2 Tests for Nonfunctional Requirements

The nonfunctional requirements are described in the SRS. All the qualities of Truss will be tested in the following section. Some requirements can be measured by the grade sheet, such as table 1 for understandability. In some cases a superscript * is used to indicate that a response of this type should be accompanied by explanatory text Smith et al. (2018).
NFR1 correctness and NFR2 verifiability will be tested in section 5.2.1. NFR3 understandability, NFR4 portability, NFR5 maintainability, and NFR6 reliability will be tested in section 5.2.2, section 5.2.3, section 5.2.4, and section 5.2.5, respectively.

5.2.1 Correctness and Verifiability

The correctness test covers the NFR1, and the verifiability test covers the NFR2. Both tests are to ensure that the software meets the SRS, and they can be assess through this document.

5.2.2 Understandablility

The understandability test covers the NFR3.

Understandablility test

1. Code review

   Type: Manual

   Initial State: Not applicable

   Input/Condition: Review the source code

   Output/Result: How easy can a new developer understand the source code

   How test will be performed: Understandability can be measured by the grade sheet in Table 1

Questions	Answer set
Consistent indentation and formatting style?	{yes, no, n/a}
Explicit identification of a coding standard?	{yes*, no, n/a}
Are the code identifiers consistent, distinctive, and meaningful?	{yes, no*, n/a}
Are constants (other than 0 and 1) hard-coded into the program?	{yes, no*, n/a}
Comments are clear, indicate what is being done, not how?	{yes, no*, n/a}
Is the name/URL of any algorithms used mentioned?	{yes, no*, n/a}
Parameters are in the same order for all functions?	{yes, no*, n/a}
Is code modularized?	{yes, no*, n/a}
Descriptive names of source code files?	{yes, no*, n/a}
Is a design document provided?	{yes*, no, n/a}
Overall impression?	{1 .. 10}

Table 1: Understandability grade sheet

5.2.3 Portablility

The portability test covers the NFR4.

Portablility test

1. Portability on Windows system

   Type: Manual

   Initial State: Truss has been successfully installed on a Windows system

   Input/Condition: Perform basic functions of the software

   Output/Result: Successfully perform the functions

   How test will be performed: Portability can be measured by the grade sheet in Table 2. It will be performed by test team manually

2. Portability on Linux system

   Type: Manual

Why not just repeat your ~~test~~ automated test on the different platform OSs?

Are you testing will all of the Daon generated language?

Initial State: Truss has been successfully installed on a Linux system

Input/Condition: Perform basic functions of the software

Output/Result: Successfully perform the functions

How test will be performed: Portability can be measured by the grade sheet in Table 2. It will be performed by test team manually

3. Portability on MacOS system

   Type: Manual

   Initial State: Truss has been successfully installed on a MacOS system

   Input/Condition: Perform basic functions of the software

   Output/Result: Successfully perform the functions

   How test will be performed: Portability can be measured by the grade sheet in Table 2. It will be performed by test team manually

not the best way to measure - You have access to the source code, and test (which you are writing)

Questions	Answer set
What platforms is the software advertised to work on?	{Type of OS}
Are special steps taken in the source code to handle portability?	{yes*, no, n/a}
Is portability explicitly identified as NOT being important?	{yes, no, n/a}
Convincing evidence that portability has been achieved?	{yes*, no,}
Overall impression?	{1 .. 10}

Table 2: Portability grade sheet

5.2.4 Maintainability

The maintainability test covers the NFR5.

Maintainability test

1. Version control

   Type: Manual

Initial State: Not applicable

Input/Condition: Existing Truss system

Output/Result: Exist multiple versions of the system

How test will be performed: Test team will perform manually to check whether there exists a history of multiple versions of the software

This is your software - you know how many versions

2. Issue tracking

Type: Manual

Initial State: Not applicable

Input/Condition: Existing Truss system

Output/Result: Implementing issue tracking tools

How test will be performed: Test team will perform manually to check whether a issue tracking tool is implemented

You know this already

5.2.5 Reliability

The reliability test covers the NFR6.

Reliability test

1. Software running

Type: Manual

Initial State: Not applicable

Input/Condition: Existing Truss system

Output/Result: Successfully operate the software

How test will be performed: Test team will run the software manually to check whether it breaks during installation and operation

2. Performance time

Type: Manual

Initial State: Not applicable

Input/Condition: Truss is started and running

Output/Result: Time duration of the software to perform required functions

How test will be performed: The test will be performed by test team manually

5.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
5.1.1	X	X				X				
5.1.2			X	X	X	X				
5.2.1					X	X				
5.2.2							X			
5.2.3								X		
5.2.4									X	
5.2.5										X

Table 3: Traceability Between Test Cases and Requirements

6 Unit Test Description

[Reference your MIS and explain your overall philosophy for test case selection. —SS] [This section should not be filled in until after the MIS has been completed. —SS]

6.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

6.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

6.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

6.2.2 Module 2

...

6.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

6.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

W Spencer Smith, Zheng Zeng, and Jacques Carette. Seismology software: State of the practice. *Journal of Seismology*, 22(3):755–788, 2018.