System Verification and Validation Plan for Truss

Ting-Yu Wu

December 9, 2020

1 Revision History

Date	**Version**	**Notes**
October 29, 2020	1.0	Initial version of VnV plan
December 6, 2020	1.1	Modification according to feedback
December 8, 2020	2.0	Add unit test description

Contents

List of Tables

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 provide the external force and define the properties of the truss structure by giving the distances and angles. The software will calculate all the internal forces within truss members and output 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

- Problem Statement
- Manual SRS for Truss
- Drasil-generated SRS
- Drasil-generated code
- VnV report

[Hyperlink will be included after building —Author]

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 review the whole project as the author.
- Dr. Spencer Smith and Dr. Jacques Carette review the whole project as supervisors.
- Tiago de Moraes Machado reviews the whole project as a domain reviewer.
- Xuanming Yan reviews the SRS as a secondary reviewer.
- Mohamed AbuElAla reviews the VnV plan as a secondary reviewer.
- Andrea Clemeno reviews the Drasil generated SRS as a secondary reviewer.

4.2 SRS Verification Plan

The SRS will be reviewed by Dr. Smith, Dr. Carette, Tiago, and Xuanming. Roles of each reviewer are mentioned in 4.1. 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

There is no manual verification plan for this project since the software are auto-generated by the Drasil.

4.4 Implementation Verification Plan

The implementation verification plan includes the followings:

- Code walkthroughs. The rubber duck debugging Wikipedia (2020) method will be implemented by the author. The procedure involve explaining the code line by line to the duck, including the flow of the whole functions and methods. Go into detail of all the intermediate states and transitions. If any defect is noticed during the process, trace back to its source, find out where does the code first go wrong, and fix it.
- 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

Following tools are used to verify the Truss software:

- System testing:
- Unit testing: Pytest will be implemented for automated unit testing. More details can be found in Section 6.
- Code coverage: Coverage.py automatically measure code coverage of programs when executing the python scripts.
- Code linting: Pylint and flake8 will be implemented to check against coding standard and analyze the source code for potential errors, such as syntax errors and structural problems.
- Continuons Integration: Travis CI is used to verify the code in conjunction with Drasil.

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

   Control: Automatic

   Initial State: Truss is started and running

   Input: Test case TC-1-1

   Output: Generate an output.txt file and display derived calculated values F_{Ax}, F_{Ay}, and F_{By}

   Test Case Derivation: Successfully generate an output file and verify the output values. Output correctness test is in section 5.1.2

   How test will be performed: It will be performed by test classes built with the help of Pytest

2. Invalid external force

   Control: Automatic

   Initial State: Truss is started and running

   Input: Test cases TC-2-1 and TC-2-2

   Output: A specific error message of each test case showned in Table 1

	Input					Output
TestID	F_1 (N)	x_1 (m)	x_2 (m)	θ_1 (°)	θ_2(°)	Error message
TC-1-1	500	3.0	3.0	45	45	-
TC-2-1	-100001	3.0	3.0	45	45	F_1 is out of bounds.
TC-2-2	100001	3.0	3.0	45	45	F_1 is out of bounds.
TC-3-1	500	0	3.0	45	45	x_1 is out of bounds.
TC-3-2	500	-1	3.0	45	45	x_1 is out of bounds.
TC-3-3	500	100001	3.0	45	45	x_1 is out of bounds.
TC-3-4	500	3.0	0	45	45	x_2 is out of bounds.
TC-3-5	500	3.0	-1	45	45	x_2 is out of bounds.
TC-3-6	500	3.0	100001	45	45	x_2 is out of bounds.
TC-4-1	500	3.0	3.0	0	45	θ_1 is out of bounds.
TC-4-2	500	3.0	3.0	-1	45	θ_1 is out of bounds.
TC-4-3	500	3.0	3.0	90	45	θ_1 is out of bounds.
TC-4-4	500	3.0	3.0	45	0	θ_2 is out of bounds.
TC-4-5	500	3.0	3.0	45	-1	θ_2 is out of bounds.
TC-4-6	500	3.0	3.0	45	90	θ_2 is out of bounds.

Table 1: Input parameters test

Test Case Derivation: Successfully display the error message

How test will be performed: It will be performed by test classes built with the help of Pytest

3. Invalid distance

   Control: Automatic

   Initial State: Truss is started and running

   Input: Test cases from TC-3-1 to TC-3-6

   Output: A specific error message of each test case showned in Table 1

   Test Case Derivation: Successfully display the error message

How test will be performed: It will be performed by test classes built with the help of Pytest

4. Invalid angle

   Control: Automatic

   Initial State: Truss is started and running

   Input: Test cases from TC-4-1 to TC-4-6

   Output: A specific error message of each test case showned in Table 1

   Test Case Derivation: Successfully display the error message

   How test will be performed: It will be performed by test classes built with the help of Pytest

5.1.2 Output Verification

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

Output correctness test

1. Simple case

   Control: Automatic

   Initial State: Truss is started and running

   Input: Test case TC-1-1

   Output: $F_{AC} = -353.553$N, $F_{AD} = 250.0$N, $F_{BC} = -353.553$N, $F_{BD} = 250.0$N, $F_{CD} = 500.0$N, and stress distribution of each force

   Test Case Derivation: Compare the output with which generated from Truss Calculator. A relative error of 1% is applicable

   How test will be performed: It will be performed by test classes built with the help of Pytest

2. Correctness of other input sets

   Control: Automatic

Initial State: Truss is started and running

Input: Valid input sets

Output: An outputfile with calculated internal forces

Test Case Derivation: Compare the output with which generated from Truss Calculator. A relative error of 1% is applicable

How test will be performed: It will be performed by test classes built with the help of Pytest

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 2 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 Accuracy and Verifiability

The accuracy 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 Understandability

The understandability test covers the NFR3.

Understandability 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: After reading the source code, understandability can be measured by the grade sheet in Table 2. The test will be performed by test team manually

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 2: Understandability grade sheet

5.2.3 Portability

The portability test covers the NFR4.

Portability 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 and implement it in Drasil

Output/Result: Successfully perform the functions and generate the SRS and code in Drasil

How test will be performed: Execute python test scripts, verify the test case TC-1-1 passes and check the generated documents exist. The test will be performed by test team manually

2. Portability on Linux system

   Type: Manual

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

   Input/Condition: Perform basic functions of the software and implement it in Drasil

   Output/Result:Successfully perform the functions and generate the SRS and code in Drasil

   How test will be performed: Execute python test scripts, verify the test case TC-1-1 passes and check the generated documents exist. The test 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 and implement it in Drasil

   Output/Result: Successfully perform the functions and generate the SRS and code in Drasil

   How test will be performed: Execute python test scripts, verify the test case TC-1-1 passes and check the generated documents exist. The test will be performed by test team manually

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: Multiple versions of the system

   How test will be performed: Check the effectiveness of version control and the completeness of the documents of multiple versions of the software. It will be performed by the test team manually

2. Issue tracking

   Type: Manual

   Initial State: Not applicable

   Input/Condition: Existing Truss system

   Output/Result: Implementing issue tracking on GitHub

   How test will be performed: Check whether the bugs, problems, and tasks for the system are well-organized on the GitHub issues. It will be performed by the test team manually

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: Test team will calculate the execution time and analyze whether it is reasonable. The test will be performed by test team manually

5.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	R4	R5	NFR1	NF2	NF3	NF4	NF5	NF6
5.1.1	X	X				X					
5.1.2			X	X	X	X					
5.2.1					X	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

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

The purpose of the unit test is to ensure that each module is behaving accurately and that each module is satisfying the SRS requirements.

6.2.1 Input Parameters Module

The following tests are created to ensure that the input parameters validate the user input sets. This first step is critical as all of the other modules rely on these parameters.

1. Input parameter

   Type: Automatic

   Initial State: Truss is started and running

   Input: import input.txt

   Output: assert = True

   Test Case Derivation: The parameters inputted by the user should match the state variable from InputParameters. An assert statement will return True if these are equal

   How test will be performed: It will be performed by test classes built with the help of Pytest

6.2.2 Input Constraints Module

The following tests are created to ensure that the system can successfully verify the validation of inputs. The system should do the calculations if valid parameters are given, and display error messages if invalid ones are given.

1. Valid parameter

Type: Automatic

Initial State: Truss is started and running

Input: import input.txt with valid input sets

Output: assert = True

Test Case Derivation: The system should generate an output file with the internal forces displayed. An assert statement will return True if the input parameters satisfy the constraints

How test will be performed: It will be performed by test classes built with the help of Pytest

2. Invalid parameter

   Type: Automatic

   Initial State: Truss is started and running

   Input: testcase_invalidforce.txt, testcase_invaliddistance.txt, testcase_invalidangle.txt

   Output: assert = True

   Test Case Derivation: Invalid inputs will be inputed to the system. It should display the correct error message. An assert statement will return True if the system successfully detects the invalid inputs

   How test will be performed: It will be performed by test classes built with the help of Pytest

6.2.3 Calculation Module

The module should do the calculations to solve the internal forces. The following tests are created to check the correctness of the solutions by comparing them with expected outputs.

1. Calculation

   Type: Automatic

   Initial State: Truss is started and running

   Input: import input.txt with valid input sets

Output: assert = True

Test Case Derivation: Outputs of the system should match the values calculated by an online software, Truss Calculator. An assert statement will return True if the relative error between calculated outputs and expected outputs is within 1%

How test will be performed: It will be performed by test classes built with the help of Pytest

6.2.4 Output Module

The module should do the calculations to solve the internal forces. The following tests are created to check the correctness of the solutions by comparing them with expected outputs.

1. Output

   Type: Maunal

   Initial State: Truss is started and running

   Input: import input.txt with valid input sets

   Output: Generate output.txt file

   Test Case Derivation: Outputs should include all the calculated internal forces and stress distribution of each force

   How test will be performed: Unit test will call the function with valid inputs, and check if the output.txt file is generated in the project folder

6.3 Tests for Nonfunctional Requirements

Testing nonfunctional requirements of units is not required for Truss. Tests for nonfunctional requirements of system are introduced in section 5.2.

6.4 Traceability Between Test Cases and Modules

The purpose of the traceability matrices is to provide easy references on what has to be additionally modified if a certain component is changed. Table 4 shows the dependencies between the test cases and the requirements.

Test Cases	**Modules**
6.2.1	Input Parameters Module
6.2.2	Input Constraints Module
6.2.3	Calculation Module
6.2.4	Output Module

Table 4: Traceability Matrix of the connections between modules and tests

References

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

Wikipedia. Rubber duck debugging. `https://en.wikipedia.org/wiki/Rubber_duck_debugging`, 2020.

Ting-Yu Wu. SRS. `https://github.com/tingyuw/cas741/blob/master/docs/SRS/SRS.pdf`, 2020.