Verification and Validation Report: MTOBridge

Team 15, Alpha Software Solutions
Badawy, Adham
Yazdinia, Pedram
Jandric, David
Vakili, Farzad
Vezina, Victor
Chiu, Darren

March 8, 2023

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

[symbols, abbreviations or acronyms – you can reference the SRS tables if needed —SS]

Contents

List of Tables

List of Figures

This document serves as a plan to verify and validate the design documents and outlines a series of system and unit tests for the functional and non functional requirements of the program. It is intended to be implemented as written, and for its results to be reported on in a future document.

3 General Information

3.1 Summary

The software whose Verification and Validation is planned for and referred to throughout this document is the User Interface component of MTO Bridge, a software developed in partnership with the Civil Engineering Department at McMaster university to assist MTO engineers in load rating bridges. Its primary function is to present visually, in an intuitive and digestible way, the results of a simulation written by a Civil Engineering Professor and her lab in MATLAB of the forces imparted on a bridge by a platoon of trucks driving over it. There are two different solvers for two different results about the forces imparted on the bridge. The program will allow users to specify details of the bridge, the truck platoon driving over it, and which of the two results they are interested in knowing more about.

3.2 Objectives

There are two overarching objectives to any V&V plan, ensuring we are actually building what we want to build correctly, or verification, and ensuring that we think we want to build is what the client actually wants from us, or validation. Verification is where a lot of the domain specific definitions of "correct" actually come in. As far as the verification plans for the design documents, the main goal is to build confidence that the design documents are complete, verifiable, unambiguous, and generally of high quality. The design documents inform the implementation to a huge degree, so ensuring we are building on a solid foundation is one of the main goals of this plan. As we are building a user interface, non functional qualities such as usability and performance/responsiveness are of utmost importance, and many of the tests outlined in this document will center around building confidence that our program has those qualities, or at least exposing if it doesn't early on.

As well, like any program, a number of our tests focus around ensuring that our program as written actually does what we think it does, in a predictable way. Ensuring our program properly executes all the functions we laid out in the design documents, and nothing more or less.

3.3 Relevant Documentation

Essentially every design document written for this project will be relevant documentation here, a mostly exhaustive list of this would include: The Problem Statement
The Development Plan
The SRS
The Hazard Analysis
The Module Guide: This document is not yet written, but will be extremely relevant for this plan, particularly for unit testing
The Module Interface Spec: Same as the Module Guide.

4 Document Evaluation

4.1 SRS Evaluation

4.2 Design Evaluation

4.3 Verification and Validation Plan Evaluation

4.3.1 Completeness

The primary portion of the V&V Plan that is not complete is the section on unit testing. While many unit tests have now been written and performed, the V&V Plan was written at a time when no unit tests were created. The now completed unit tests will be added into the V&V Plan document so that this section can be fully complete once the project is over. Besides this issue, the V&V Plan is also missing system tests designed to test NFR.14 and NFR.15. These tests are either going to be added or the NFRs will be reevaluated. Section 2 of the V&V Plan is also missing a number of abbreviations that are used later in the document.

4.3.2 Feasibility

For input tests, testing every possible input to confirm that the system responds correctly to each one is unreasonable. Instead, inputs are grouped into categories and it is assumed that an input from that category is representative of the whole, such as positive numbers, negative numbers, letters, and symbols for text field inputs.

Tests for Look and Feel and Usability and Humanity stated that surveys or tests would be performed on civil engineers using the program. In practice, the system was verified through meetings with the client and her students who possess a civil engineering background but are not necessarily MTO bridge engineers themselves.

5 System Evaluation

5.1 Functional Requirements Evaluation

5.1.1 MATLAB

MATLAB Communication

1. FR1.ST1

   Control: Automatic

   Initial State: None

   Input: None

   Output: MATLAB engine started

   Test Case Derivation: The first step in the communication is to get the MATLAB engine to start.

   How test will be performed: The test will instantiate our MATLAB engine wrapper.

2. FR1.ST2

   Control: Automatic

   Initial State: None

Input: Truck, Bridge, and Analysis configurations

Output: Any result from standard out or standard error

Test Case Derivation: We just need to confirm that the engine is processing our input in some way, whether it gives an error or not.

How test will be performed: Start the MATLAB engine through our wrapper, create some default truck, bridge, and analysis configurations, and try to run the analysis.

5.1.2 Truck Configuration

Truck Configuration Input

1. FR2.ST1

   Control: Automatic

   Initial State: GUI exists

   Input: Axle load, axle spacing, number of trucks, headway

   Output: GUI fields are filled

   Test Case Derivation: If the data of the input fields can be entered, then the user should also be able to fill in the fields.

   How test will be performed: Input fields will filled, and checked if they have the correct value.

2. FR2.ST2

   Control: Automatic

   Initial State: GUI exists

   Input: Invalid axle load, axle spacing, number of trucks, headway

   Output: GUI fields are filled, highlighted as incorrect

   Test Case Derivation: The user should be able to enter data, but be informed if the entered information is invalid.

   How test will be performed: Input fields will filled with incorrect information, and the fields should be highlighted as incorrect.

Truck Platoon Visualization

1. FR3.ST1

   Control: Automatic

   Initial State: GUI exists

   Input: Valid truck configuration

   Output: A valid visualization of the truck platoon shown

   Test Case Derivation: If a valid truck configuration is given, the user should be able to visualize the truck platoon.

   How test will be performed: Given a valid truck configuration, a visualization will be generated. The visualization will be saved as a picture and compared to the expected visualization.

2. FR3.ST2

   Control: Automatic

   Initial State: GUI exists

   Input: Invalid truck configuration

   Output: A message informing the user of an incorrect truck configuration

   Test Case Derivation: If an invalid truck configuration is given, the user should be informed that the visualization cannot be created without a valid truck configuration.

   How test will be performed: Given an invalid configuration, the program will not show a visualization, but a message informing the user that a visualization cannot be created/shown until the configuration is valid.

Save Truck Configuration

1. FR4.ST1

   Control: Automatic

Initial State: GUI exists

Input: A truck configuration

Output: A file with the correctly saved configuration

Test Case Derivation: The program should be able to take the existing truck configuration and save it so that it can be loaded or used later.

How test will be performed: Given a truck configuration, a file will be created. Its contents will be compared with the expected contents.

Load Truck Configuration

1. FR5.ST1

   Control: Automatic

   Initial State: GUI exists, a truck configuration file exists

   Input: Truck configuration file

   Output: Truck configuration input fields will be filled correctly

   Test Case Derivation: After loading a previously saved truck configuration file, the configuration should apply and fill the input fields for the truck configuration.

   How test will be performed: Given a truck configuration, a file will be created. Its contents will be compared with the expected contents.

5.1.3 Bridge Configuration

Bridge Configuration Input

1. FR6.ST1

   Control: Automatic

   Initial State: GUI exists

   Input: Number of spans, bridge length

   Output: GUI fields are filled

   Test Case Derivation: If the data of the input fields can be entered, then the user should also be able to fill in the fields.

How test will be performed: Input fields will filled, and checked if they have the correct value.

2. FR6.ST2

   Control: Automatic

   Initial State: GUI exists

   Input: Invalid number of spans, bridge length

   Output: GUI fields are filled, highlighted as incorrect

   Test Case Derivation: The user should be able to enter data, but be informed if the entered information is invalid.

   How test will be performed: Input fields will filled with incorrect information, and the fields should be highlighted as incorrect.

Bridge Visualization

1. FR7.ST1

   Control: Automatic

   Initial State: GUI exists

   Input: Valid bridge configuration

   Output: A valid visualization of the bridge shown

   Test Case Derivation: If a valid bridge configuration is given, the user should be able to visualize the bridge.

   How test will be performed: Given a valid bridge configuration, a visualization will be generated. The visualization will be saved as a picture and compared to the expected visualization.

2. FR7.ST2

   Control: Automatic

   Initial State: GUI exists

   Input: Invalid bridge configuration

Output: A message informing the user of an incorrect bridge configuration

Test Case Derivation: If an invalid bridge configuration is given, the user should be informed that the visualization cannot be created without a valid bridge configuration.

How test will be performed: Given an invalid configuration, the program will not show a visualization, but a message informing the user that a visualization cannot be created/shown until the configuration is valid.

Save Bridge Configuration

1. FR8.ST1

   Control: Automatic

   Initial State: GUI exists

   Input: A bridge configuration

   Output: A file with the correctly saved configuration

   Test Case Derivation: The program should be able to take the existing bridge configuration and save it so that it can be loaded or used later.

   How test will be performed: Given a bridge configuration, a file will be created. Its contents will be compared with the expected contents.

Load Bridge Configuration

1. FR9.ST1

   Control: Automatic

   Initial State: GUI exists, a bridge configuration file exists

   Input: Bridge configuration file

   Output: Bridge configuration input fields will be filled correctly

   Test Case Derivation: After loading a previously saved truck configuration file, the configuration should apply and fill the input fields for the truck configuration.

How test will be performed: Given a bridge configuration, a file will be created. Its contents will be compared with the expected contents.

5.1.4 Solver Setup

Solver Selection

1. FR10.ST1

   Control: Automatic

   Initial State: None

   Input: input to indicate the desired solver

   Output: Calculation results based on the selected solver

   Test Case Derivation: The current logic allows for calculation of the demand at a concerned section, or at the critical section, and there are separate calculations for each.

   How test will be performed: The test involves running a series of predetermined calculations using both solvers and then comparing with the expected output.

Section of Concern

1. FR11.ST1

   Control: Automatic

   Initial State: GUI exists

   Input: location of concerned section

   Output: GUI field is filled

   Test Case Derivation: If the data of the input fields can be entered, then the user should also be able to fill in the fields.

   How test will be performed: Input fields will be filled, and checked if they have the correct value.

Discretization Length

1. FR12.ST1

   Control: Automatic

   Initial State: GUI exists

   Input: discretization length

   Output: GUI field is filled

   Test Case Derivation: If the data of the input fields can be entered, then the user should also be able to fill in the fields.

   How test will be performed: Input fields will be filled, and checked if they have the correct value.

Force Type

1. FR13.ST1

   Control: Automatic

   Initial State: GUI exists

   Input: force type

   Output: GUI field is filled

   Test Case Derivation: The user can select the type of force to calculate for. If they select the force type it should reflect in the GUI by filling the field properly.

   How test will be performed: Input fields will filled, and checked if they have the correct value.

2. FR13.ST2

   Control: Automatic

   Initial State: None

   Input: text input to indicate the desired moment

   Output: Calculation results based on the selected parameters

Test Case Derivation: The current implementation of the engine allows for calculation using positive and negative moment

How test will be performed: The test will be performed by using our pool of pre-determined bridge and truck platoon configs to test for both positive and negative moment given a certain force type.

5.1.5 Result Visualization

Concerned Section Result Visualization

1. FR14.ST1

   Control: Automatic

   Initial State: None

   Input: calculations based on the concerned section

   Output: Animation and visualization highlighting the resulting properties on the specified bridge and load

   Test Case Derivation: As one of the main purposes of the program, the visualization will be interactive to show various views

   How test will be performed: Relying on the pre-determined and validated data, we will graph the visualizations of each through the system as well as manually through the data produced by the engine. We can then compare the the expected and actual result across all available views.

Critical Section Result Visualization

1. FR15.ST1

   Control: Automatic

   Initial State: None

   Input: calculations based on the discretization length

   Output: Animation and visualization highlighting the resulting properties on the specified bridge and load

Test Case Derivation: As one of the main purposes of the program, the visualization will be interactive to show various views

How test will be performed: Relying on the pre-determined and validated data, we will graph the visualizations of each through the system as well as manually through the data produced by the engine. We can then compare the the expected and actual result across all available views.

5.1.6 Exectution Flow Logging

Report Generation

1. FR16.ST1

   Control: Automatic

   Initial State: None

   Input: text input to indicate logging flags

   Output: text output, "logs"

   Test Case Derivation: With the engine built in another framework, its important to log every state change throughout the calculations as triggered by the system

   How test will be performed: While running multiple calculations, the system is asked to comprehensively log the flow in the engine. The system will be interrupted at times to test the robustness as well. The resulting logs should be clearly highlighting the steps taken including any errors or warnings.

5.2 Nonfunctional Requirements Evaluation

5.2.1 Look and Feel Requirements

1. NFR1.ST1

   Type: Manual, Static

   Initial State: Bridge UI mockups containing graphic elements will be prepared

Input/Condition: Civil engineers are presented with a UI mockup as seen in Figure **??**

Output/Results: The engineers can correctly identify what part of a bridge each UI element corresponds to in 90% of cases.

How test will be performed: The developers will present a UI mockup to civil engineers and will assess how accurately the engineers can identify the UI elements.

5.2.2 Usability and Humanity Requirements

1. NFR2.ST1

   Type: Manual, Dynamic

   Initial State: Application is opened with no past data

   Input/Condition: Civil engineers are presented with the application and a brief explanation

   Output/Results: 90% of the civil engineers are able to perform bridge analysis within 5 minutes of introduction

   How test will be performed: Civil engineers will be presented with the application and a brief explanation by the developers, then the amount of time it takes them to complete a bridge analysis will be measured.

2. NFR3.ST1

   Type: Manual, Dynamic

   Initial State: Application is opened

   Input/Condition: Application is running on a computer with 1280x720 display

   Output/Results: All buttons and animation window remain visible and accessible

   How test will be performed: The application will be manually opened on a 1280x720 display and checked to ensure that all buttons and animation windows are still visible.

3. NFR3.ST2

   Repeat test NFR.ST3 for 1366x768 display

4. NFR3.ST3

   Repeat test NFR.ST3 for 1920x1080 display

5. NFR3.ST4

   Repeat test NFR.ST3 for 2560x1440 display

6. NFR3.ST5

   Repeat test NFR.ST3 for 3840x2160 display

7. NFR4.ST1

   Type: Manual, Dynamic

   Initial State: Application is opened with no past data

   Input/Condition: Font size setting is changed to 8pt

   Output/Results: All buttons and animation window remain visible and accessible

   How test will be performed: The application will be manually opened and set to use 8pt font, then checked to ensure that all buttons and animation windows are still visible.

8. NFR4.ST2

   Repeat test NFR.ST8 with font size 16pt

9. NFR4.ST3

   Repeat test NFR.ST8 with font size 24pt

10. NFR4.ST4

    Repeat test NFR.ST8 with font size 32pt

11. NFR5.ST1

    Type: Manual, Dynamic

    Initial State: Application is not installed

    Input/Condition: Application is downloaded and installed

    Output/Results: Application is ready to use within 30 minutes

    How test will be performed: The application will be manually downloaded (on an internet connection with at least a 10Mbps download speed), installed, and run. This process will be timed to ensure that it takes less than 30 minutes.

12. NFR6.ST1

    Type: Manual, Static

    Initial State: Visually similar UI elements are grouped into categories by developers

    Input/Condition: Civil engineer are asked to associate UI elements with their respective categories

    Output/Results: Civil engineer sorts at least 90% of UI elements into their predefined categories

    How test will be performed: The developers will present a UI mockup to civil engineers and will assess how accurately the engineers can correctly group the UI elements.

5.2.3 Performance Requirements

1. NFR7.ST1

    Type: Automatic, Dynamic

    Initial State: Application is opened with no past data

Input/Condition: The application is run with invalid inputs (zeroes, negative numbers, strings, etc.)

Output/Results: The application does not freeze or crash

How test will be performed: The application will be automatically run with many different invalid inputs, and it will be ensured that the application does not freeze or crash

2. NFR8.ST1

   Type: Manual, Dynamic

   Initial State: Application is installed but the MATLAB files are not installed

   Input/Condition: The application is opened

   Output/Results: The application will display an error regarding the missing MATLAB files

   How test will be performed: The application will be manually installed without the required MATLAB files, and then it will be opened

3. NFR9.ST1

   Type: Automatic, Dynamic

   Initial State: Application is opened with no past data

   Input/Condition: Various UI elements are interacted with

   Output/Results: The UI elements react to the interaction within 100ms

   How test will be performed: Various interactions with UI elements will be simulated on a computer with hardware similar to that of an MTO engineer's, and the speed of the UI response will be measured automatically

4. NFR10.ST1

   Type: Automatic, Dynamic

   Initial State: Application is opened with no past data

   Input/Condition: Various valid inputs

Output/Results: Total execution time of calculations will not exceed underlying MATLAB script's execution time by more than 10%

How test will be performed: Many different analyses will be performed and timed on a computer with hardware similar to that of an MTO engineer's.

5.2.4 Maintainability and Support Requirements

1. NFR13.ST1

   Type: Automatic, Static

   Initial State: The program code is contained in multiple files

   Input/Condition: Measure code file length in lines

   Output/Results: At least 75% of files will contain 750 lines of code or less

   How test will be performed: The clang-tidy linter will be used to automatically measure file length

2. NFR13.ST2

   Type: Automatic, Static

   Initial State: The program code is contained in multiple files

   Input/Condition: Measure method length in lines

   Output/Results: At least 75% of methods will contain 75 lines of code or less

   How test will be performed: The clang-tidy linter will be used to automatically measure method length

3. NFR13.ST3

   Type: Automatic, Static

   Initial State: The program code is contained in multiple files

   Input/Condition: Measure length of code lines in characters

Output/Results: At least 75% of lines will contain 120 characters or less

How test will be performed: The clang-tidy linter will be used to automatically measure line length

4. NFR13.ST4

   Type: Automatic, Static

   Initial State: The program code is contained in multiple files

   Input/Condition: Measure nesting depth of methods

   Output/Results: At least 75% of methods will have a maximum nesting depth of 5 or less

   How test will be performed: The clang-tidy linter will be used to automatically measure method nesting depth

6 Unit Testing

7 Comparison to Existing Implementation

This section will not be appropriate for every project.

8 Changes Due to Testing

[This section should highlight how feedback from the users and from the supervisor (when one exists) shaped the final product. In particular the feedback from the Rev 0 demo to the supervisor (or to potential users) should be highlighted. —SS]

9 Automated Testing

10 Trace to Requirements

11 Trace to Modules

12 Code Coverage Metrics

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection. Please answer the following question:

1. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)