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	Developer(s)	Change
March 6	David	NFR System Testing
March 6	Adham	Design Evaluation
March 7	Pedram	VnV Plan Unambiguous
March 7	Victor	VnV Plan Completeness
March 7	Adham	VnV Plan Overall
March 7	All	FR System Testing
March 8	Victor	SRS Ease of Understanding
March 8	David	SRS Completeness
March 8	Adham	Changes Due to Testing
March 8	All	Unit Testing
March 8	Pedram	SRS Ambiguity
March 8	Darren	VnV Plan Feasibility
March 8	Adham	SRS Testability
March 8	Darren & Farzad	SRS Abstraction & Spelling
March 8	Victor	Traceability
March 8	All	Reflection
March 8	Victor	Automated Testing

2 Symbols, Abbreviations and Acronyms

Refer to section 1.4 of the SRS.

Contents

List of Tables

3 Introduction

The following document is a Verification and Validation Report for the MTO-Bridge Software. It details the results of carrying out the previously drafted Verification and Validation Plan. That document can be found here

4 Document Evaluation

4.1 SRS Evaluation

4.1.1 Abstraction

FRs were reviewed and no abstraction issues were found. The fact that the sentences start with "The program should allow the user" helps the author with focusing on what should be provided to the user and not how it would be provided.

NFRs were reviewed and no significant abstraction issues were found.

4.1.2 Testability

For Testability, the FRs are all relatively atomic, with clear fit criteria, they should be easily testable in combination with the VnV plan.

The NFRs are a bit muddier, though many of them also have quantitative fit criteria that make them much more testable. One notable exception is the localization requirement, while this is testable in theory, in practice we do not have reasonable access to a french translator nor do we have much experience in localization. This could still be a requirement, but the fit criterion should probably change from measuring the time taken to actually localize the program to a more high level evaluation of the difficulty of translation. Something like a word count could work, but given a bit more thought it is likely a better fit criterion could be chosen. It would likely be of that general type, however.

4.1.3 Ambiguity

In terms of ambiguity, the Functional Requirements correctly reflected the abstract properties of the software with a clear language. Nevertheless, there are changes that can improve the clarity. In the third Functional Requirement, the requirement states the capabilities of the software in visualizing previously specified truck platoons. The wording in this case can be specified to indicate changes can only be made in the truck tab and the final platoon will be an exact copy. We can see a very similar wording with Functional Requirement 7 which is ambiguous in reflecting the source of changes. We also are concerned with Functional Requirements 14 as well as 15 as the definition of "visualizing" can be up to interpretation in these scenarios. We can remove the ambiguity by breaking down the visualization to various components where each would be a separate Functional Requirements as these can often include graphs, charts, animations and more complex objects.

The Non-Functional Requirements which are meant to describe the qualities of the software are also written in a descriptive manner, nevertheless there is room for improvement. Starting with the very first Requirement, it states that graphics of the software will be informative. This a broad statement which can be ineffective in visualizing the final look. This can be changed to specify simple yet key elements of the UI such as "The graphics will be tabular with a visual element on the right side of the each tab". Similarly, the second Non-Functional Requirement captures the usability of the software through an intuitive design. This can also be broken down to highlight the intuitive design for each section such as the Truck and the Bridge. This can include familiar design patters or dynamic feedback. Moving on the Performance section, Non-Functional Requirements 10 and 11 aim to evaluate the "slowness" as well as the precision of the program. While there is fit criterion available, there is a need for a further context in each scenario. As for the precision, it is vital to understand the target metrics. Finally, the thirteenth requirement mentions the maintainability aspects of the software. With a standalone windows-based application, there is a guarantee that the program will run the same in any machine. As such, the maintainability would really involve "updates" in this scenario.

4.1.4 Completeness

For completeness, we are mainly concerned with the Functional and Nonfunctional requirements, the rest of the document seems fine from this standpoint.

As far as the FRs, there seem to be no notable omissions from the 16 stated. Nothing has been exposed through this review, reviews with Dr Yang and her team, or through other design documents, with one notable exception. The Hazard Analysis identified a need for thread safety in the program, and while implemented in the Rev0 implementation, there is no mention of this in the FRs. This will need to be added for Rev1.

For NFRs, completeness is much harder to judge due to their more nebulous and subjective nature. However, once again no glaring omissions were found by this review.

Though this was mentioned in the VnV Plan, it bears repeating that the document is likely not 100 percent complete, but outside of the mentioned thread safety requirements, we have found it to be close enough to approximate completeness.

4.1.5 Spelling and Grammar

FRs were reviewed and no significant spelling and grammar errors were found other than small improvements such as using "above-mentioned" instead of "above mentioned" or "long-term" instead of "long term" where a hyphen can be used to join two or more words to form a compound adjective.

NFRs were reviewed and no significant spelling and grammar errors were found.

4.1.6 Ease of Understanding

The SRS is fairly legible and easy to understand, but it has areas in which it could be improved. The first of these is in the traceability section of the functional and non-functional requirements, which are currently difficult to understand as they are simply the id of the constraint or assumption (e.g., C.1, A.1, etc.). Turning these ids into links which point to the appropriate

constraint or assumption would help the reader understand the requirements by allowing them to read the constraint or assumption again if they do not remember them.

Another area in which the SRS lacks legibility is in the figures. None of the figures contain legends which detail what the various symbols and arrows used within the figure represent. This is especially important for the use case diagram (Figure 2), which uses various arrows with different meaning. Adding a simple legend to each figure would solve this issue and improve the readability of the SRS.

4.2 Design Evaluation

4.2.1 Usability

In a meeting with Dr Yang and her team, we collectively agreed upon a score out of 5 for each of the following Usability metrics the program should have. Below are the scores decided upon along with a paraphrased summary of the group's reasoning.

- **Keeps Users informed about its status:** 3/5 for the wrong reasons. Error messages pop up when something goes wrong, but they also tend to crash the program.

- **Show information in a way users understand:** 4/5. Overall acceptable, but the bridge visualization is not very accurate to what a Civil Engineer would expect

- **Offer users control:** 2/5. User often loses control of the program due to errors that lead to crashes, and there is no way to undo an error.

- **Be consistent:** 4/5. Not bad. A couple of the button labels need to be changed to be consistent across tabs.

- **Prevent errors:** 1/5. The program does basically none of this. Error prevention is almost entirely up to the user. Very bad in its current

state.

- **Have visible information to let users recognize options:** 3/5. The program is fairly simple, and the labels on the buttons imply function well enough to make it intuitive, but some actual on screen instructions or help hovers would be useful.

- **Be flexible:** 1/5. There is only really one way to operate the program, and if the user skips any steps to try and speed things up it usually leads to error.

- **Have no clutter:** 5/5. No discussed issues here, no extraneous information was found on any of the tabs.

- **Provide plain-language help:** 1/5. No help/documentation tab, and when errors pop up they are C++ jargon and not helpful to a layman user.

There is quite a lot to improve upon here, and we plan to take this into Rev1 and hopefully remeasure all of these at the end with Yang and her team.

4.2.2 Flexibility

After a review of the module guide, stretch goals, and some hands on experience of expanding and extending the Rev0 demo with more features, we came to the conclusion that the design is quite flexible as is, and not in need of much correction. It is highly modular, and accomplishes low coupling and high cohesion relatively well. Changes can be made to the inner workings of one module without really being felt by the other modules. This means that stretch goals such as accepting 2D bridge sketches or having the UI be customizable should be feasible, or at least not held back by the flexibility and modularity of the design.

4.2.3 Maintainability

In the interest of time, this review was performed in a more high-level way than originally planned, and no outside volunteers were recruited. With than in mind, the overall findings were that while file to file information hiding and separation of concerns were quite good, as described above, there is some cleanup of internal file and code structure would help with bug-fixing and maintainability. The issues found were not severe, and many of the classes are relatively simple, but there were examples of functions that could be split into a couple of smaller functions to ensure atomicity among functions, not just classes. This is that will be improved for Rev1.

4.2.4 Software Validation Plan

As mentioned in the VnV plan, this section was a little vague. However, overall everything is going according to plan. The biweekly meetings have kept up consistently, and in them we present a summary of what we are currently working on and what are goals are, and get feedback on that from Dr Yang, we also discuss anything she or her team wish to bring up. These meetings have been invaluable to ensure continued alignment between us and them, and will hopefully continue to do so into Rev1. More recently the meetings have included demonstrations of what we have created for Rev0, and discussions centered around what could be changed or improved to better align with what Dr Yang wants, such as the Usability section of the Design Verification Plan.

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 Unambiguous

When evaluating the verification plan in terms of ambiguity, we are mainly concerned with the ability to simply follow the plan as intended with no parts that can be interpreted differently. Overall, each "Verification Plan" section is clearly described with little ambiguity, nevertheless we can make the following changes. Starting with the SRS Verification Plan, part of the completeness verification requires presentation to the client teams at different stages to make sure we are not missing any desired functions. This section can be improved by suggesting a structured set of questions for each presentation that would then help us compare and track our progress much better. As for the Design Verification Plan, defining the aspects of the categories usability, flexibility and maintainability could be better understood by connecting these sections to the actual implementation limitations. Lastly, the Software Validation Plan aims to capture the correctness of the final solution mainly through unstructured meetups and reviews with the client. Although vague by nature, there are ways to improve this step. This can include further qualitative methods such as surveys and public interviews to quantitative methods such as the total number of tests passing.

4.3.3 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.

4.3.4 Overall Objective Accomplishment

There were three stated objectives outlined in Section 3.2 of the VnV Plan:

- Build confidence that the design documents are complete, verifiable, unambiguous, and generally of high quality
- Build confidence that in important non functional qualities such as Usability, or expose if those qualities aren't there
- Build confidence that the implementation does what we thought it did when we built it

Reviewing the plan holistically, it seems to accomplish the first two objectives rather well. The section on the document reviews is complete, unambiguous, and feasible, with the notable exception that the last point of section 4.4 regarding this very review should explicitly mention the objectives of section 3.2 for clarity. Overall though, no issues. Usability and other non functional qualities are also heavily focused on in sections 4.3, 4.7, and in the non functional tests of Section 5. and finally, section 5 and 6 for System and Unit Testing respectively cover the third objective. The obvious problem here is that as mentioned in the completeness section, the unit testing section is basically entirely missing in the plan as is. If that is fixed and well filled out, however, the third objective will also be accomplished.

5 System Evaluation

5.1 Functional Requirements Evaluation

5.1.1 MATLAB

MATLAB Communication

1. FR1.ST1

Control: Manual

Initial State: None

Input: None

Expected Output: MATLAB engine started

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

Actual Output: MATLAB engine started

Results Summary: the MATLAB engine starts and commands are ready to be sent to MATLAB.

2. FR1.ST2

Control: Manual

Initial State: None

Input: Truck, Bridge, and Analysis configurations

Expected Output: Any result from standard out or standard error

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.

Actual Output: Produces the correct results for the configurations.

Results Summary: MATLAB returned a result, which was the correct result. It was verified against the corresponding standalone MATLAB calculation.

5.1.2 Truck Configuration

Truck Configuration Input

1. FR2.ST1

Control: Manual

Initial State: GUI exists

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

Expected Output: GUI fields are filled

How Test Was Performed: Input fields were manually filled, preview button was hit, and GUI fields were manually checked

Actual Output: GUI fields were filled

Results Summary: Pass

2. FR2.ST2

   Control: Manual

   Initial State: GUI exists

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

   Expected Output: GUI fields are filled, highlighted as incorrect

   How Test Was Performed: Input fields were manually filled with invalid values, preview button was hit, and GUI fields were manually checked

   Actual Output: Program Crashed

   Results Summary: Fail, does not behave as expected, and fails to handle invalid input.

Truck Platoon Visualization

1. FR3.ST1

   Control: Manual

   Initial State: GUI exists

   Input: Valid truck configuration

   Expected Output: A valid visualization of the truck platoon shown

   How Test Was Performed: A valid truck configuration was inputted, the preview button was hit, and the visualization was compared against the oracle.

   Actual Output: A valid visualization of the truck platoon shown

   Results Summary: Pass

2. FR3.ST2

   Control: Manual

   Initial State: GUI exists

   Input: Invalid truck configuration

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

   How Test Was Performed: An invalid truck configuration was inputted, the preview button was hit, and the result was compared against the oracle.

   Actual Output: Program Crashed

   Results Summary: Fail, does not behave as expected, and fails to handle invalid input.

Save Truck Configuration

1. FR4.ST1

   Control: Manual

   Initial State: GUI exists

   Input: A truck configuration

   Expected Output: A file with the correctly saved configuration

   How Test Was Performed: Given a truck configuration, the "save configuration" button was hit and a file saved. The contents of the file were compared against the oracle

   Actual Output: A file with the correctly saved configuration

   Results Summary: Pass

Load Truck Configuration

1. FR5.ST1

   Control: Manual

Initial State: GUI exists, Truck configuration file exists.

Input: A truck configuration file

Expected Output: Truck configuration input fields will be filled correctly

How Test Was Performed: Given a truck configuration file, the "load configuration" button was hit and the file was loaded. The contents of the input fields were then compared against the oracle.

Actual Output: Truck configuration input fields was be filled correctly

Results Summary: Pass

5.1.3 Bridge Configuration

Bridge Configuration Input

1. FR6.ST1

   Control: Manual

   Initial State: GUI exists

   Input: Number of spans, span length, discretization length, concerned section

   Expected Output: GUI fields are filled

   How Test Was Performed: Input fields were manually filled, preview button was hit, and GUI fields were manually checked

   Actual Output: GUI fields are filled

   Results Summary: Pass

2. FR6.ST2

   Control: Manual

   Initial State: GUI exists

   Input: Invalid number of spans, bridge length

   Expected Output: GUI fields are filled, highlighted as incorrect

How Test Was Performed: Input fields was filled with incorrect information

Actual Output: GUI fields are filled, not highlighted as incorrect

Results Summary: Fail

Bridge Visualization

1. FR7.ST1

   Control: Manual

   Initial State: GUI exists

   Input: valid bridge configuration

   Expected Output: A visualization of the effects of their bridge characteristics is displayed.

   How Test Was Performed: User clicks "Preview" button after filling in all the other input fields inside the bridge tab

   Actual Output: A visualization of the effects of their bridge characteristics is displayed.

   Results Summary: Pass

2. FR7.ST2

   Control: Manual

   Initial State: GUI exists

   Input: Invalid bridge configuration

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

   How Test Was Performed: User clicks "Preview" button after filling in all the other input fields inside the bridge tab

   Actual Output: Program Crashes

   Results Summary: Fail

Save Bridge Configuration

1. FR8.ST1

   Control: Manual

   Initial State: GUI exists

   Input: Invalid bridge configuration

   Expected Output: A new window pops up asking the user where to save their bridge inputs and the file is saved with the correct format after the location is specified.

   How Test Was Performed: User clicks on the "Save Config" button after filling in all the other input fields inside the bridge tab

   Actual Output: A new window pops up asking the user where to save their bridge inputs and the file is saved with the correct format after the location is specified.

   Results Summary: Pass

Load Bridge Configuration

1. FR9.ST1

   Control: Manual

   Initial State: GUI exists

   Input: A bridge configuration File

   Expected Output: A new window pops up asking the user to specify which configuration file they want to load. After specifying the file, input fields get filled.

   How Test Was Performed: User clicks on the "Save Config" button after filling in all the other input fields inside the bridge tab

   Actual Output: A new window pops up asking the user to specify which configuration file they want to load. After specifying the file, input fields get filled.

   Results Summary: Pass

5.1.4 Solver Setup

Solver Selection

1. FR10.ST1

   Control: Manual

   Initial State: GUI exists

   Input: Solver choice of concerned section or critical section made by user

   Expected Output: Calculation results are based on the solver choice made by the user

   How Test Was Performed: This test was performed by running pre-determined calculations after selecting each of the solver choices and comparing the data output to pre-calculated results, ensuring that the selected solver was being used for calculations.

   Actual Output: Calculation results are based on the solver choice made by the user

   Results Summary: This test passed as the system correctly allowed the user to modify the solver being used for calculations

Section of Concern

1. FR11.ST1

   Control: Manual

   Initial State: GUI exists

   Input: Concerned section inputted by user via text box

   Expected Output: Calculation results are for the concerned section specified by the user

   How Test Was Performed: This test was performed by running pre-determined calculations after specifying different concerned sections and comparing the data output to pre-calculated results, ensuring that the specified concerned section was being used for calculations.

Actual Output: Calculation results are for the concerned section specified by the user

Results Summary: This test passed as the system correctly allowed the user to modify the concerned section being used for calculations

Discretization Length

1. FR12.ST1

   Control: Manual

   Initial State: GUI exists

   Input: Discretization length inputted by user via text box

   Expected Output: Calculation results are based on the discretization length specified by the user

   How Test Was Performed: This test was performed by running predetermined calculations after specifying different discretization lengths and comparing the data output to pre-calculated results, ensuring that the specified discretization length was being used for calculations.

   Actual Output: Calculation results are based on the discretization length specified by the user

   Results Summary: This test passed as the system correctly allowed the user to modify the discretization length being used for calculations

Force Response

1. FR13.ST1

   Control: Manual

   Initial State: GUI exists

   Input: Force response choice of positive moment, negative moment, or shear made by user

   Expected Output: Calculation results are based on the force response choice made by the user

How Test Was Performed: This test was performed by running predetermined calculations after selecting each of the force response choices and comparing the data output to pre-calculated results, ensuring that the selected force response was being used for calculations.

Actual Output: Calculation results are based on the force response choice made by the user

Results Summary: This test passed as the system correctly allowed the user to modify the desired force response used for calculations

5.1.5 Result Visualization

Concerned Section Result Visualization

1. FR14.ST1

   Control: Manual

   Initial State: GUI exists

   Input: The user performs calculations using the concerned section solver

   Expected Output: A visualisation of the calculation results are displayed to the user

   How Test Was Performed: This test was performed by running predetermined concerned section calculations and comparing the visualised output data to pre-calculated results, ensuring that the calculation results are being correctly shown to the user.

   Actual Output: A visualisation of the calculation results are displayed to the user

   Results Summary: This test passed as the system correctly visualised the results of concerned section calculations

Critical Section Result Visualization

1. FR15.ST1

   Control: Manual

Initial State: GUI exists

Input: The user performs calculations using the critical section solver

Expected Output: A visualisation of the calculation results are displayed to the user

How Test Was Performed: This test was performed by running predetermined critical section calculations and comparing the visualised output data to pre-calculated results, ensuring that the calculation results are being correctly shown to the user.

Actual Output: A visualisation of the calculation results are displayed to the user

Results Summary: This test passed as the system correctly visualised the results of critical section calculations

5.1.6 Execution Flow Logging

Report Generation

1. FR16.ST1

Control: Manual

Initial State: Application has been opened with configurations set and able to run analysis

Input: Save button pressed and name of file to be saved given as "report.txt" in local directory

How Test Was Performed: Fields were set for bridge, truck, and solver configurations and then an analysis run to generate results. The report save was then tested by clicking the button, defining the file name to save to, and opening the created file to ensure it contained the necessary information.

Expected Output: "report.txt" generated containing configurations and analysis results

Actual Output: "report.txt" generated containing configurations and analysis results

Results Summary: Pass

5.2 Nonfunctional Requirements Evaluation

5.2.1 Look and Feel Requirements

1. NFR1.ST1

   Type: Manual, Static

   Initial State: Bridge UI mock ups containing graphic elements will be prepared

   Input: Civil engineers/students are presented with a UI mock up.

   Expected Output: The engineers/students can correctly identify each UI element physically corresponds to in 90% of cases.

   How Test Was Performed: The developers presented the program to civil engineers/students, and assessed how accurately the engineers could identify the UI elements.

   Actual Output: Over 90% of engineers/students understood almost all of the UI elements were for.

   Result Summary: Pass

5.2.2 Usability and Humanity Requirements

1. NFR2.ST1

   Type: Manual, Dynamic

   Initial State: Application is opened with no past data

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

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

   How Test Was Performed: Civil engineers/students were 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.

   Actual Output: Over 90% of the civil engineers/students were able to perform the bridge analysis within 3 minutes of introduction

Results Summary: Pass

2. NFR3.ST1

   Type: Manual, Dynamic

   Initial State: Application is opened

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

   Expected Output: All UI elements remain visible and accessible

   How Test Was Performed: The application was opened on a 1280x720 display and checked to ensure that all UI elements were still visible and accessible.

   Actual Output: All UI elements remained visible and accessible

   Result Summary: Pass

3. NFR3.ST2

   Repeat test NFR.ST3 for 1366x768 display

   Result Summary: Pass

4. NFR3.ST3

   Repeat test NFR.ST3 for 1920x1080 display

   Result Summary: Pass

5. NFR3.ST4

   Repeat test NFR.ST3 for 2560x1440 display

   Result Summary: Pass

6. NFR3.ST5

   Repeat test NFR.ST3 for 3840x2160 display

   Result Summary: Pass

7. NFR4.ST1

   Type: Manual, Dynamic

   Initial State: Application is opened with no past data

   Input: Font size setting is changed to 8pt

   Expected Output: All UI elements remain visible and accessible

   How Test Was Performed: The application was set to use 8pt font and opened, then checked to ensure that all UI elements were still visible and accessible

   Actual Output: All UI elements remained visible and accessible

8. NFR4.ST2

   Repeat test NFR.ST8 with font size 16pt

   Result Summary: Pass

9. NFR4.ST3

   Repeat test NFR.ST8 with font size 24pt

   Result Summary: Pass

10. NFR4.ST4

    Repeat test NFR.ST8 with font size 32pt

    Result Summary: Pass

11. NFR5.ST1

    Type: Manual, Dynamic

    Initial State: Application is not installed

    Input: Application is downloaded and installed

    Expected Output: Application is ready to use within 30 minutes

    How Test Was Performed: The application and MATLAB downloaded (on an internet connection with at least a 10Mbps download speed)

and installed, and the application was run. The process was timed to ensure that it takes less than 30 minutes.

Actual Output: Application was ready to use within 15 minutes

Result Summary: Pass

12. NFR6.ST1

    Type: Manual, Static

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

    Input: Civil engineers/students are asked to associate UI elements with their respective categories

    Expected Output: Civil engineers/students sort at least 90% of UI elements into their predefined categories

    How test will be performed: The developers presented the names/functions of UI elements to civil engineers and assessed how accurately the engineers/students could correctly group the UI elements.

    Actual Output: Civil engineers/students sorted over 80% of UI elements into their predefined categories

    Result Summary: Fail, concerned section and critical section was categorized by the engineers/students into the analysis tab instead of the bridge tab.

5.2.3 Performance Requirements

1. NFR7.ST1

    Type: Automatic, Dynamic

    Initial State: Application is opened with no past data

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

    Expected Output: The application does not freeze or crash

How Test Was Performed: The application was automatically run with many different invalid inputs, and checked that the application does not freeze or crash

Actual Output: The application crashed with many different invalid inputs

Results Summary: Fail, the application could not reliably handle invalid inputs, crashing with almost all tested invalid inputs

2. NFR8.ST1

   Type: Manual, Dynamic

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

   Input: The application is opened

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

   How Test Was Performed: The application was manually installed without the required MATLAB files, and was then opened

   Actual Output: The application displayed an error message regarding the missing MATLAB files

   Results Summary: Pass, the system was able to give an error message in this situation.

3. NFR9.ST1

   Type: Automatic, Dynamic

   Initial State: Application is opened with no past data

   Input: Various UI elements are interacted with

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

   How Test Was Performed: Various interactions with UI elements were simulated on a computer with hardware similar to that of an MTO engineer's, and the speed of the UI response was measured automatically

Actual Output: The UI elements reacted well within 100ms, approaching almost instantaneous.

Results Summary: Pass, the system was very responsive, and the UI elements responded immediately to input

4. NFR10.ST1

   Type: Automatic, Dynamic

   Initial State: Application is opened with no past data

   Input: Various valid inputs

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

   How Test Was Performed: Many different analyses were performed and timed on a computer with hardware similar to that of an MTO engineer's.

   Actual Output: The execution time of calculations was at most 5% of the underlying MATLAB script's execution time.

   Results Summary: Pass

5.2.4 Maintainability and Support Requirements

1. NFR13.ST1

   Type: Manual, Static

   Initial State: The program code is contained in multiple files

   Input: Measure code file length in lines

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

   How Test Was Performed: File length was checked manually using Visual Studio

   Actual Output/Results: Over 75% of files contain <750 lines of code

   Result Summary: Pass

2. NFR13.ST2

   Type: Manual, Static

   Initial State: The program code is contained in multiple files

   Input: Measure method length in lines

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

   How Test Was Performed: Method length was checked manually using Visual Studio

   Actual Output: Over 75% of methods contain <75 lines of code

   Result Summary: Pass

3. NFR13.ST3

   Type: Automatic, Static

   Initial State: The program code is contained in multiple files

   Input: Measure length of code lines in characters

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

   How Test Was Performed: The clang-tidy linter was used to automatically measure line length

   Actual Output: Over 75% of lines contain <120 characters

   Result Summary: Pass

4. NFR13.ST4

   Type: Automatic, Static

   Initial State: The program code is contained in multiple files

   Input: Measure nesting depth of methods

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

How Test Was Performed: The clang-tidy linter was used to automatically measure nesting depth

Actual Output: Over 75% of methods have a nesting depth of 5 or less

Result Summary: Pass

6 Unit Testing

6.1 MATLAB

1. CalculationCaller.UT1

   Control: Automatic

   Method: std::shared_ptr<matlab::cpplib::MATLABLibrary> getMatlab()

   Input: None

   Expected Output: MATLAB pointer exists and is valid

   Actual Output: MATLAB pointer exists and is valid

   Results Summary: Pass

2. CalculationCaller.UT2

   Control: Automatic

   Method: CalculationOutputT runCalculation(CalculationInputT)

   Input: The default truck, bridge, and solver configuration

   Expected Output: The output is valid, with correct first axle position, equal number of axle positions and forces, and critical section and force undefined.

   Actual Output: Incorrect critical section.

   Results Summary: Fail, the critical section is uninitialized, instead of being an impossible value (-1)

3. CalculationCaller.UT3

   Control: Automatic

   Method: CalculationOutputT runCalculation(CalculationInputT)

Input: Invalid Axle Load

Expected Output: An empty CalculationOutputT

Actual Output: Exception, incorrect number of array indices

Results Summary: Fail, the invalid axle load sequence is not caught, causing an uncaught exception.

4. CalculationCaller.UT4

   Control: Automatic

   Method: CalculationOutputT runCalculation(CalculationInputT)

   Input: Invalid Axle Spacing

   Expected Output: An empty CalculationOutputT

   Actual Output: Exception, incorrect number of array indices

   Results Summary: Fail, the invalid axle spacing sequence is not caught, causing an uncaught exception.

5. CalculationCaller.UT5

   Control: Automatic

   Method: CalculationOutputT runCalculation(CalculationInputT)

   Input: Negative Number of Trucks

   Expected Output: An empty CalculationOutputT

   Actual Output: Exception, array indices must be positive integers or logical values

   Results Summary: Fail, the invalid number of trucks is not caught, causing an uncaught exception.

6. CalculationCaller.UT6

   Control: Automatic

   Method: CalculationOutputT runCalculation(CalculationInputT)

   Input: Negative Headway

   Expected Output: An empty CalculationOutputT

   Actual Output: Debug assertion, out of bounds access violation

   Results Summary: Fail, the invalid headway is not caught, causing potential catastrophic failure.

7. CalculationCaller.UT7

   Control: Automatic

   Method: CalculationOutputT runCalculation(CalculationInputT)

   Input: Invalid Number of Bridge Spans

   Expected Output: An empty CalculationOutputT

   Actual Output: Debug assertion, out of bounds access violation, or segmentation fault

   Results Summary: Fail, the invalid number of spans is not caught, causing catastrophic failure.

8. CalculationCaller.UT8

   Control: Automatic

   Method: CalculationOutputT runCalculation(CalculationInputT)

   Input: Invalid Span Length

   Expected Output: An empty CalculationOutputT

   Actual Output: Exception, array indices must be positive integers or logical values

   Results Summary: Fail, the invalid span lengths not caught, causing an uncaught exception.

9. CalculationCaller.UT9

   Control: Automatic

   Method: CalculationOutputT runCalculation(CalculationInputT)

   Input: Negative Concerned Section

   Expected Output: An empty CalculationOutputT

   Actual Output: Command properly runs and returns some output

   Results Summary: Fail, the invalid concerned section not caught, causing incorrect output.

10. CalculationCaller.UT10

    Control: Automatic

    Method: CalculationOutputT runCalculation(CalculationInputT)

Input: Invalid Discretization Length

Expected Output: An empty CalculationOutputT

Actual Output: Command properly runs and returns some output

Results Summary: Fail, the invalid concerned section not caught, causing incorrect output.

11. Engine.UT1

    Control: Automatic

    Method: Engine getInstance()

    Input: None

    Expected Output: Engine starts and initializes MATLAB

    Actual Output: Engine starts and initializes MATLAB

    Results Summary: Pass

12. Engine.UT2

    Control: Automatic

    Method: QThread thread()

    Input: None

    Expected Output: Main thread and Engine thread are different

    Actual Output: Main thread and Engine thread are different

    Results Summary: Pass

13. Engine.UT3

    Control: Automatic

    Method: CalculationOutputT runCommand(CalculationInputT)

    Input: The default truck, bridge, and solver configuration

    Expected Output: Engine finishes running the command without exception

    Actual Output: Engine finishes running the command without exception

    Results Summary: Pass

14. Engine.UT4

    Control: Automatic

    Method: void stopEngine()

    Input: The default truck, bridge, and solver configuration

    Expected Output: Engine kills MATLAB and it's own thread

    Actual Output: Engine kills MATLAB and it's own thread

    Results Summary: Pass

6.2 Bridge Configuration

1. BridgeConfiguration.UT1

   Control: Automatic

   Method: void updateNumberOfSpans(QString newNumberOfSpans)

   Input: QString::fromStdString("3")

   Expected Output: NumberofSpans changes to 3

   Actual Output: NumberofSpans changes to 3

   Results Summary: Pass

2. BridgeConfiguration.UT2

   Control: Automatic

   Method: void updateNumberOfSpans(QString newNumberOfSpans)

   Input: QString::fromStdString("5")

   Expected Output: InvalidConfiguration Exception

   Actual Output: NumberofSpans changes to 5

   Results Summary: Fail

3. BridgeConfiguration.UT3

   Control: Automatic

   Method: void updateNumberOfSpans(QString newNumberOfSpans)

Input: QString::fromStdString("number")

Expected Output: InvalidConfiguration Exception

Actual Output: NumberofSpans changes to 0

Results Summary: Fail

4. BridgeConfiguration.UT4

   Control: Automatic

   Method: void updateConcernedSection(QString newConcernedSection)

   Input: QString::fromStdString("15")

   Expected Output: ConcernedSection changes to 15

   Actual Output: NumberofSpans changes to 15

   Results Summary: Pass

5. BridgeConfiguration.UT5

   Control: Automatic

   Method: void updateConcernedSection(QString newConcernedSection)

   Input: QString::fromStdString("NUMBER")

   Expected Output: InvalidConfiguration Exception

   Actual Output: NumberofSpans changes to 0

   Results Summary: Fail

6. BridgeConfiguration.UT6

   Control: Automatic

   Method: void updateDiscretizationLength(QString newDiscretization-Length)

   Input: QString::fromStdString("0.5")

   Expected Output: DiscretizationLength changes to 0.5

   Actual Output: DiscretizationLength changes to 0.5

   Results Summary: Pass

7. BridgeConfiguration.UT7

   Control: Automatic

   Method: void updateDiscretizationLength(QString newDiscretization-Length)

   Input: QString::fromStdString("NUMBER")

   Expected Output: InvalidConfiguration Exception

   Actual Output: DiscretizationLength changes to 0

   Results Summary: Fail

8. BridgeConfiguration.UT8

   Control: Automatic

   Method: void updateSpanLength (QString newNumberOfSpans)

   Input: QString::fromStdString("20 30")

   Expected Output: SpanLength changes to 20, 30

   Actual Output: SpanLength changes to 20, 30

   Results Summary: Pass

9. BridgeConfiguration.UT9
   Control: Automatic

   Method: void updateSpanLength (QString newNumberOfSpans)

   Input: QString::fromStdString("20,30")

   Expected Output: SpanLength changes to 20, 30

   Actual Output: SpanLength changes to 20

   Results Summary: Fail

10. BridgeConfiguration.UT10
    Control: Automatic

    Method: void updateSpanLength (QString newNumberOfSpans)

    Input: QString::fromStdString("number1 number2")

    Expected Output: InvalidConfiguration Exception

Actual Output: Program Crashes

Results Summary: Fail

11. BridgeConfiguration.UT11

    Control: Automatic

    Method: vector<double> convertQstringToDoubleVector(QString qstring)

    Input: QString::fromStdString("30.5 40")

    Expected Output: 30.5, 40

    Actual Output:30.5, 40

    Results Summary: Pass

12. BridgeConfiguration.UT12

    Control: Automatic

    Method: vector<double> convertQstringToDoubleVector(QString qstring)

    Input: QString::fromStdString("20,30")

    Expected Output: 20, 30

    Actual Output: 20

    Results Summary: Fail

13. BridgeConfiguration.UT13

    Control: Automatic

    Method: vector<double> convertQstringToDoubleVector(QString qstring)

    Input: QString::fromStdString("number1 number2")

    Expected Output: InvalidConfiguration Exception

    Actual Output: Program Crashes

    Results Summary: Fail

6.3 Platoon Configuration

1. PlatoonConfiguration.UT1

   Control: Automatic

   Method: UpdateAxleLoad()

   Input: QString "53.4 75.6 75.6 75.6 75.6"

   Expected Output: Platoon Configuration AxleLoads field should be a list that equals the input

   Actual Output: Platoon Configuration AxleLoads field was a list that equalled the input

   Results Summary: Pass

2. PlatoonConfiguration.UT2

   Control: Automatic

   Method: UpdateAxleLoad()

   Input: QString "53.4,75.6,75.6,75.6,75.6"

   Expected Output: Platoon Configuration AxleLoads field should be a list that equals the input

   Actual Output: Platoon Configuration AxleLoads field was a mess that did not match input, but no exception thrown

   Results Summary: Fail, needs to be able to process comma separated lists as well as space separated lists.

3. PlatoonConfiguration.UT3

   Control: Automatic

   Method: UpdateAxleLoad()

   Input: QString "I like turtles"

   Expected Output: Function should return and no changes should be made to PlatoonConfiguration AxleLoads field

   Actual Output: Exception thrown.

   Results Summary: Fail, failed to handle invalid input.

4. PlatoonConfiguration.UT4

   Control: Automatic

   Method: UpdateAxleSpacing()

   Input: QString "3.6576 1.2192 9.4488 1.2192"

   Expected Output: Platoon Configuration AxleSpacings field should be a list that equals the input

   Actual Output: Platoon Configuration AxleSpacings field was a list that equalled the input

   Results Summary: Pass.

5. PlatoonConfiguration.UT5

   Control: Automatic

   Method: UpdateAxleSpacing()

   Input: QString "3.6576,1.2192,9.4488,1.2192"

   Expected Output: Platoon Configuration AxleSpacings field should be a list that equals the input

   Actual Output: Platoon Configuration AxleSpacings field was a mess that did not match input, but no exception thrown

   Results Summary: Fail, needs to be able to process comma separated lists as well as space separated lists.

6. PlatoonConfiguration.UT6

   Control: Automatic

   Method: UpdateAxleSpacing()

   Input: QString "I hate turtles"

   Expected Output: Function should return and no changes should be made to PlatoonConfiguration AxleSpacings field

   Actual Output: Exception thrown.

   Results Summary: Fail, failed to handle invalid input.

7. PlatoonConfiguration.UT7

   Control: Automatic

   Method: UpdateNumberOfTrucks()

   Input: QString ”15”

   Expected Output: Platoon Configuration NumTrucks field should be 15

   Actual Output: Platoon Configuration NumTrucks field was 15

   Results Summary: Pass.

8. PlatoonConfiguration.UT8

   Control: Automatic

   Method: UpdateNumberOfTrucks()

   Input: QString ”-8”

   Expected Output: Function should return and no changes should be made to PlatoonConfiguration NumTrucks field

   Actual Output: Platoon Configuration NumTrucks field was -8

   Results Summary: Fail, should not allow a negative number of trucks.

9. PlatoonConfiguration.UT9

   Control: Automatic

   Method: UpdateNumberOfTrucks()

   Input: QString ”Booyakasha”

   Expected Output: Function should return and no changes should be made to PlatoonConfiguration NumTrucks field

   Actual Output: Exception thrown.

   Results Summary: Fail, failed to handle invalid input.

10. PlatoonConfiguration.UT10

    Control: Automatic

Method: UpdateHeadway()

Input: QString "10"

Expected Output: Platoon Configuration Headway field should be 10

Actual Output: Platoon Configuration Headway field was 10

Results Summary: Pass.

11. PlatoonConfiguration.UT11

    Control: Automatic

    Method: UpdateHeadway()

    Input: QString "-5"

    Expected Output: Function should return and no changes should be made to PlatoonConfiguration Headway field

    Actual Output: Platoon Configuration Headway field was -5

    Results Summary: Fail, should not allow a negative number of trucks.

12. PlatoonConfiguration.UT12

    Control: Automatic

    Method: UpdateHeadway()

    Input: QString "Three minutes of playtime"

    Expected Output: Function should return and no changes should be made to PlatoonConfiguration Headway field

    Actual Output: Exception thrown.

    Results Summary: Fail, failed to handle invalid input.

6.4 Truck Configuration

6.5 Saver

1. Saver.UT1

   Control: Automatic

Method: void savePlatoonConfiguration(TruckT Platoon)

Input: TruckT Platoon = {.axleLoad = {53.4, 75.6, 75.6, 75.6, 75.6}, .axleSpacing = {3.6576, 1.2192, 9.4488, 1.2192}, .numberOfTrucks = 3, .headway = 5}

Expected Output: .trk file created at the path

Actual Output: .trk file created at the path

Results Summary: Pass

2. Saver.UT2

   Control: Automatic

   Method: void savePlatoonConfiguration(TruckT Platoon)

   Input: TruckT Platoon = {.axleLoad = {53.4, 75.6, 75.6, 75.6, 75.6}, .axleSpacing = {}, .numberOfTrucks = 3, .headway = 5}

   Expected Output: Warning Message due to the empty axleSpacing

   Actual Output: Application Crash

   Results Summary: Fail

3. Saver.UT3

   Control: Automatic

   Method: void saveBridgeConfiguration(BridgeT Bridge)

   Input: BridgeT Bridge = {.numberSpans = 2, .spanLength = {20, 20}, .concernedSection = 10, .discretizationLength = 0.1}

   Expected Output: .brg file created at the path

   Actual Output: .brg file created at the path

   Results Summary: Pass

4. Saver.UT4

   Control: Automatic

   Method: void saveBridgeConfiguration(BridgeT Bridge)

Input: BridgeT Bridge = {.numberSpans = 2, .spanLength = {}, .concernedSection = 10, .discretizationLength = 0.1}

Expected Output: Warning Message due to the empty spanLength

Actual Output: Application Crash

Results Summary: Fail

5. Saver.UT5

   Control: Automatic

   Method: void saveSolverConfiguration(SolverT Solver)

   Input: SolverT Solver = {solverConfig = {.forceType = mtobridge::MockSolverT::POSITIVE
   .solverType = mtobridge::MockSolverT::CONCERNED}

   Expected Output: .slv file created at the path

   Actual Output: .slv file created at the path

   Results Summary: Pass

6. Saver.UT6

   Control: Automatic

   Method: void saveSolverConfiguration(SolverT Solver)

   Input: SolverT Solver = {solverConfig = {.forceType = NULL, .solverType = mtobridge::MockSolverT::CONCERNED}

   Expected Output: Warning Message due to the empty forceType

   Actual Output: Application Crash

   Results Summary: Fail

6.6 Loader

1. Loader.UT1

   Control: Automatic

   Method: void loadPlatoonConfiguration()

Input: String Platoon = {.axleLoad = {53.4, 75.6, 75.6, 75.6, 75.6}, .axleSpacing = {3.6576, 1.2192, 9.4488, 1.2192}, .numberOfTrucks = 3, .headway = 5}

Expected Output: Truck Object File returned

Actual Output: Truck Object File returned

Results Summary: Pass

2. Loader.UT2

   Control: Automatic

   Method: void loadPlatoonConfiguration()

   Input: String Platoon = {.axleLoad = {53.4, 75.6, 75.6, 75.6, 75.6}, .axleSpacing = {}, .numberOfTrucks = 3, .headway = 5}

   Expected Output: Warning Message due to the empty axleSpacing

   Actual Output: Application Crash

   Results Summary: Fail

3. Loader.UT3

   Control: Automatic

   Method: void loadBridgeConfiguration()

   Input: String Bridge = {.numberSpans = 2, .spanLength = {20, 20}, .concernedSection = 10, .discretizationLength = 0.1}

   Expected Output: Bridge Object File returned

   Actual Output: Bridge Object File returned

   Results Summary: Pass

4. Loader.UT4

   Control: Automatic

   Method: void loadBridgeConfiguration()

   Input: String Bridge = {.numberSpans = 2, .spanLength = {}, .concernedSection = 10, .discretizationLength = 0.1}

Expected Output: Warning Message due to the empty spanLength

Actual Output: Application Crash

Results Summary: Fail

5. Loader.UT5

   Control: Automatic

   Method: void loadSolverConfiguration()

   Input: String Solver = {solverConfig = {.forceType = mtobridge::MockSolverT::POSITIVE_M
   .solverType = mtobridge::MockSolverT::CONCERNED}

   Expected Output: Solver Object File returned

   Actual Output: Solver Object File returned

   Results Summary: Pass

6. Loader.UT6

   Control: Automatic

   Method: void loadSolverConfiguration()

   Input: String Solver = {solverConfig = {.forceType = NULL, .solver-
   Type = mtobridge::MockSolverT::CONCERNED}

   Expected Output: Warning Message due to the empty forceType

   Actual Output: Application Crash

   Results Summary: Fail

6.7 Solver Configuration

1. Solver.UT1

   Control: Automatic

   Initial State: Solver.force == POSITIVE_MOMENT

   Method: static std::string getForceType()

   Input: None

Expected Output: "Positive Moment"

Actual Output: "Positive Moment"

Results Summary: Pass

2. Solver.UT2

   Control: Automatic

   Initial State: Solver.solver == CONCERNED

   Method: static std::string getSolverType()

   Input: None

   Expected Output: "Concerned Section"

   Actual Output: "Concerned Section"

   Results Summary: Pass

3. Solver.UT3

   Control: Automatic

   Initial State: Solver.force == POSITIVE_MOMENT

   Method: static void updateForceType(std::string forceStr)

   Input: forceStr = "Negative Moment"

   Expected Output: Solver.force == NEGATIVE_MOMENT

   Actual Output: Solver.force == NEGATIVE_MOMENT

   Results Summary: Pass

4. Solver.UT4

   Control: Automatic

   Initial State: Solver.force == NEGATIVE_MOMENT

   Method: static void updateForceType(std::string forceStr)

   Input: forceStr = "Shear"

   Expected Output: Solver.force == SHEAR

Actual Output: Solver.force == SHEAR

Results Summary: Pass

5. Solver.UT5

   Control: Automatic

   Initial State: Solver.force == SHEAR

   Method: static void updateForceType(std::string forceStr)

   Input: forceStr = "Positive Moment"

   Expected Output: Solver.force == POSITIVE_MOMENT

   Actual Output: Solver.force == POSITIVE_MOMENT

   Results Summary: Pass

6. Solver.UT6

   Control: Automatic

   Initial State: Solver.force == POSITIVE_MOMENT

   Method: static void updateForceType(std::string forceStr)

   Input: forceStr = "Negative"

   Expected Output: invalidConfigurationValue exception is thrown and Solver.force == POSITIVE_MOMENT

   Actual Output: invalidConfigurationValue exception is thrown and Solver.force == POSITIVE_MOMENT

   Results Summary: Pass

7. Solver.UT7

   Control: Automatic

   Initial State: Solver.force == POSITIVE_MOMENT

   Method: static void updateForceType(std::string forceStr)

   Input: forceStr = ""

Expected Output: invalidConfigurationValue exception is thrown and Solver.force == POSITIVE_MOMENT

Actual Output: invalidConfigurationValue exception is thrown and Solver.force == POSITIVE_MOMENT

Results Summary: Pass

8. Solver.UT8

   Control: Automatic

   Initial State: Solver.solver == CONCERNED

   Method: static void updateSolverType(std::string solverStr)

   Input: solverStr = "Critical Section"

   Expected Output: Solver.solver == CRITICAL

   Actual Output: Solver.solver == CRITICAL

   Results Summary: Pass

9. Solver.UT9

   Control: Automatic

   Initial State: Solver.solver == CRITICAL

   Method: static void updateSolverType(std::string solverStr)

   Input: solverStr = "Concerned Section"

   Expected Output: Solver.solver == CONCERNED

   Actual Output: Solver.solver == CONCERNED

   Results Summary: Pass

10. Solver.UT10

    Control: Automatic

    Initial State: Solver.solver == CONCERNED

    Method: static void updateSolverType(std::string solverStr)

    Input: solverStr = "Critical"

Expected Output: invalidConfigurationValue exception is thrown and Solver.solver == CONCERNED

Actual Output: invalidConfigurationValue exception is thrown and Solver.solver == CONCERNED

Results Summary: Pass

11. Solver.UT11

    Control: Automatic

    Initial State: Solver.solver == CONCERNED

    Method: static void updateSolverType(std::string solverStr)

    Input: solverStr = ""

    Expected Output: invalidConfigurationValue exception is thrown and Solver.solver == CONCERNED

    Actual Output: invalidConfigurationValue exception is thrown and Solver.solver == CONCERNED

    Results Summary: Pass

7 Changes Due to Testing

The biweekly meetings with Dr Yang and her team have been wonderful for receiving feedback on the entire design process. Particularly once we had a working demo for Rev0. As mentioned in the Design Verification section we had a focused meeting with Dr Yang and her team regarding the Rev 0 demo and what improvements they would like to see for Rev1. One common thread that was exposed throughout the VnV process was our sore lack of **robustness**. This cost us dearly in the evaluation of Usability metrics that was performed with Dr Yang and co. This is also the primary shortfall identified in the unit and system tests. The highest priority and most sweeping change that is planned to be made between Rev0 to the final product is definitely the addition of proper error checking and handling for many methods and processes in the program.

There were also some less catastrophic but still important findings as a result of the process. For example, Dr. Yang pointed out that our bridge

visualization was inaccurate to real world standards, due to the relative thinness of the middle bridge support and the lack of side supports and a bridge deck. This is valuable information that would not have been easy to figure out without her feedback, and we will be changing this for Rev1.

Finally there were many small pieces of feedback regarding making the UI more intuitive/consistent/etc from Dr. Yang and her team. Things like wanting the concerned section and discretization length information visible on the solver tab as well as the bridge tab, or change the labels on certain buttons or tabs to be more in line with industry standards, to name a few.

8 Automated Testing

Automated testing was accomplished using the QTest framework and CTest. All of the unit testing performed for VnV was automated. The QTest framework was used to create the test suites as it was created for QT application and allows for easy execution of many tests while having a clear display of test results. CTest was used to augment the ease of use of QTest even further by integrating it with the CMake build system already in use. This combination allows for the straightforward and automated running of all unit tests.

9 Trace to Requirements

Table 1: Functional System Test Traceability

Requirement	**System Tests**
FR1	FR1.ST1, FR1.ST2
FR2	FR2.ST1, FR2.ST2
FR3	FR3.ST1, FR3.ST2
FR4	FR4.ST1
FR5	FR5.ST1
FR6	FR6.ST1, FR6.ST2
FR7	FR7.ST1, FR7.ST2
FR8	FR8.ST1

Requirement	System Tests
FR9	FR9.ST1, FR9.ST2
FR10	FR10.ST1
FR11	FR11.ST1
FR12	FR6.ST1
FR13	FR13.ST1
FR14	FR14.ST1
FR15	FR15.ST1
FR16	FR16.ST1

Table 2: Non-Functional System Test Traceability

Requirement	System Tests
NFR1	NFR1.ST1
NFR2	NFR2.ST1
NFR3	NFR3.ST1, NFR3.ST2, NFR3.ST3, NFR3.ST4, NFR3.ST5
NFR4	NFR4.ST1, NFR4.ST2, NFR4.ST3, NFR4.ST4
NFR5	NFR5.ST1
NFR6	NFR6.ST1
NFR7	NFR7.ST1
NFR8	NFR8.ST1
NFR9	NFR9.ST1
NFR10	NFR10.ST1
NFR11	FR3.ST1, FR7.ST1, FR10.ST1, FR11.ST1, FR12.ST1, FR13.ST1
NFR12	NFR9.ST1, NFR10.ST1
NFR13	NFR13.ST1, NFR13.ST2, NFR13.ST3, NFR13.ST4
NFR14	N/A
NFR15	N/A

10 Trace to Modules

The unit tests can be directly traced to modules by the name of the tests. For example, the test PlatoonConfiguration.UT7 is a unit test for the PlatoonConfiguration module.

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.)

Adham: I think the biggest difference between what we planned for the VnV and what we actually managed to do was the extent of the involvement of outside parties. I don't think it was just patently infeasible, but due to time constraints we weren't really able to get user feedback from anyone except our supervisor Dr. Yang and her team. Granted, their feedback is by far the most important of anyone, but it would have been interesting and valuable to get feedback from other engineering students or even complete layman for example. Realistically this happened because we saved things for the last minute, and it could have been anticipated and avoided, but oh well.

David: The VnV Plan and actual VnV were most different in terms of how the testing was done. Our plan of interviews and surveys changed as we didn't have access to MTO bridge engineers, so we instead relied on feedback from Dr. Yang and her team, as well as a few friends (and family) who are in the civil engineering field. These changes occurred as we didn't have the time (in terms of deadlines) to accomplish this properly, so we resorted to less ideal options. It's fairly difficult to anticipate changes like this, sometimes life/work just get busy. To avoid more of the risk, I would come up with realistic contingency plans beforehand, so that we get the best feedback we can, with the resources we have.

Pedram: The team had truly laid out a clear process toward an encompassing verification plan however the scope and the room for error were a slightly misallocated. The original plan included surveys as well as interviews that were not quite realistic when combined with the actual implementation

as well as other school work. In an ideal situation, there would be contingency plans in place that would allow quality work even if we were to miss internal milestones. Moving forward, there should be more frequent checkups within the team to make sure the tasks are being executed according to the plan.

Victor: Another way in which the activities conducted for VnV were different to what was planned in the VnV Plan is the system tests. In the VnV Plan, many of the system tests were automated, but in actual testing many of these tests were conducted manually. This was because we had initially believed that the many GUI interactions required for the tests could be fairly easily automated through the use of QTEST, but when we got to actually testing the system we found that automating these interactions was infeasible. This change could be anticipated in the future by doing extensive research to fully understanding the capabilities and limitations of the testing tools and libraries being used before writing the VnV Plan.

Farzad: Another way in which the activities conducted for VnV were different to what was planned in the VnV Plan is implementation verification plan. Granted, that prototype testing ad unit testing was performed but there was a third dimensions to implementation verification which is analysis and it includes code inspection. The original plan was to have a system in which every newly written code would be reviewed and approved by other team members before it gets merged to the product. However, as a result of time pressure before preparing the first prototype team members were working in parallel and merging their continuously. More progress checking meetings could have potentially alerted us of this time crunch earlier and get us to start things off earlier and work slower with higher quality.

Darren: We stated we would survey civil engineers while they used the program to confirm things such as ease of use and intuitiveness but in practice settled for presenting the program to the client and her team and asking for feedback. This was done because it turned out difficult to get in touch with MTO and arrange for a meeting with a bridge engineer. We would've also needed access to their computers to fully follow the original plan. In the future we would settle for specifying the client's team as the ones surveyed instead or make earlier arrangements for meeting with bridge engineers.