System Verification and Validation Plan for Bridge Corrosion: A Chloride Exposure Prediction Model

Cynthia Liu

April 15, 2024

Revision History

Date	Version	Notes
Feb 9, 2024	0.0	Initial release
Mar 7, 2024	0.1	Modify according to feedback from peer review
Mar 16, 2024	0.2	Modify according to feedback from Dr. Smith
Apr 12, 2024	1	Add unit test

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

1.1 Table of Symbols

symbol	unit	description
C_s	kg/m^3/vehicle	chloride on the bridge substructure
$C_{s_{air}}$	kg/m^3/vehicle	chloride sprayed and splashed per unit air volume per vehicle
h_{app}	m	daily water film thickness on the road
$longitude$	$^\circ$	longitude
$latitude$	$^\circ$	latitude
h_{total}	m	the total snowfall during a winter season
M_{app}	kg/m^2	deicing salts quantity applied per day
M_{total}	kg/m^2	total amount of deicing salts quantity over winter
SD_{total}	kg/m^3	spray density kicked up by each passing truck
SD_{total_cl}	kg/m^3	mass of chloride ions per unit air volume
t_1	days	number of days with snowfall
t_2	days	number of days with snow melting

1.2 Abbreviations and Acronyms

symbol	description
AADT	Annual Average Daily Traffic
AADTT	Annual Average Daily Truck Traffic
BC	Bridge Corrosion
CI	Continuous Integration
FR	Functional Requirement
MG	Module Guide
MIS	Module Interface Specification
NFR	Non-functional Requirement
R	Requirement
SRS	Software Requirements Specification
T	Test
VnV	Verification and Validation

2 General Information

This document provides the road-map of the verification and validation plan for Bridge Corrosion (BC), to ensure the requirements and goals of the program mentioned in the SRS. The organization of this document starts with the general information and verification plans, followed by the system test and unit test description for functional and non-functional requirements.

2.1 Summary

BC provides predictive trends for chloride exposure based on user-input coordinates within the province of Ontario. The software utilizes climate and traffic models to build a chloride exposure database, and search for the trends when a user inputs location data.

2.2 Objectives

The objective of this document is to build confidence in the software's correctness and increase its realiability. All functional requirements mentioned in SRS are tested. We cover a usability test and a maintainability test althought the number of participants is limited. We also assume the climate and traffic models we generate online have been verified by their implementation team so their accuracy is ensured.

2.3 Relevant Documentation

The relevant documentation includes Problem Statement which is the proposed idea, Software Requirements Specifications which outlines the requirement of the software. It would also be related to VnV Report which is a conclusion for validation and verification after the software is developed. The MG and MIS are also relevant.

3 Plan

The VnV plan begins by introducing the verification and validation team (section 3.1), followed by the verification plans for the SRS (section 3.2) and design (section 3.3). Subsequently, the verification plans for the VnV Plan

(section 3.4) and implementation (section 3.5) are presented. In the end, there are sections on automated testing and verification tools (Section 3.6) and the software validation plan (section 3.7).

3.1 Verification and Validation Team

Name	Document	Role	Description
Dr. Spencer Smith	All	Instructor/ Reviewer	Review all the documents.
Mingsai Xu	-	Domain Expert	Provide theory support for the software.
Cynthia Liu	All	Author	Create all the documents, implement the software, and verify the implementation according to the VnV plan.
Phil Du	All	Domain Expert Reviewer	Review all the documents.
Hunter Ceranic	SRS	Secondary Reviewer	Review the SRS document.
Fasil Cheema	VnV plan	Secondary Reviewer	Review the VnV plan.
Fatemeh Norouziani	MG + MIS	Secondary Reviewer	Review the MG and MIS document.

Table 1: Verification and Validation Team

3.2 SRS Verification Plan

The SRS verification is done by peer review with classmates (Phil Du and Hunter Ceranic), giving suggestions by creating issues in Github. The author (Cynthia Liu) is responsible for reviewing and addressing the issues.
The SRS verification is also reviewed by Dr. Spencer Smith to offer any suggestions and feedbacks.
There is a SRS checklist designed by Dr. Spencer Smith available to use.

3.3 Design Verification Plan

The design verification, including the module guide (MG) and module interface specification (MIS), will be verfied by Phil Du and Fatemeh Norouziani. Dr. Spencer Smith will also review both documents. Reviewers will give feedbacks through Github issues and the author need to address them. The MG checklist and MIS checklist designed by Dr. Spencer Smith could act as a help for reviewers.

3.4 Verification and Validation Plan Verification Plan

The VnV plan is first created and verfied by the author, then turns to the team members to give any suggests by Github issues. The reviewers will access the VnV checklist designed by Dr. Spencer Smith for reference.

3.5 Implementation Verification Plan

The implementation would be verified by testing the functional requirements and non-functional requirements in section 4. The unit tests will also be conducted as detailed in section 5. There will also be a code walkthrough with class during the presentation.

3.6 Automated Testing and Verification Tools

The software will be done in Python and so does the automated testing. During the coding, lintr will be used as a static code analysis. The Github Actions is used for continuous integration, that the CI work flow will run all the unit testing, and the package installation when new code is pushed to the repository.

3.7 Software Validation Plan

Software validation plan is beyond the scope for BC as it may require additional time, expertise, and resources that are not available within the scope of the project. The validation process, should be done by the domain expert, which include ensuring the models and formulas that this project are using are correct.

4 System Test Description

This section includes the system test that will be used for the functional requirements and non-functional requirements.

4.1 Tests for Functional Requirements

Functional requirements for BC are given in SRS section 5.1. There are three functional requirements for BC, R1 is related to the input, and R2 and R3 related to the output. Section 4.1.1 describes the input tests related to R1, and the output test is discussed in section 4.1.2.

4.1.1 Input Test - test_invalid_input

This section covers R1 of the SRS which includes the input coordinates check for the software. This section also cover the dataset inputs that developers input from models to the software. In most of the cases, the dataset in the models are valid as they have been verified by their developers, but the test cases here still include the scenarios where data may be missing or invalid.

Functional tests - Input tests - Coordinate

Input (°)		Output	
longitude	*latitude*	valid?	Error Message
123.45	abc	N	Input need to be a number
	-123.3656	N	Input need to be a number
277.4771	42.63238	N	Input position not inside Ontario
-79.07264	43.26259	N	Input position not inside Ontario
-79.07622	43.26204	Y	No error message, continue to calculate output
280.84538	44.550356	Y	No error message, continue to calculate output

Table 2: Coordinate input tests

1. test-input-coordinate
   Control: Automatic

Initial State: Uninitialized

Input: The input column in Table 2.

Output: The output column in Table 2.

Test Case Derivation: The input coordinate need to be a location inside Ontario. It does not allow nonnumerical input as well. The software produces error message for the invalid inputs, for the valid inputs, it will proceed to next step.

How test will be performed: The author will create test cases and run automatic test by Python.

Functional tests - Input tests - Models

Input			Output	
h_{total} (m)	t_1 (days)	t_2 (days)	valid?	Error Message
183.9371	109	76	Y	No error message, continue to import model data
194.6057	367	255	N	t_1 and t_2 must be less than 365
119.62	53	-30	N	t_1 and t_2 need to be positive
	109	76	N	Data missing for h_{total}
183.9371		76	N	Data missing for t_1
183.9371	109		N	Data missing for t_2

Table 3: Climate model input tests

1. test-input-climate

   Control: Automatic

   Initial State: Uninitialized

   Input: The input column in Table 3.

   Output: The output column in Table 3.

   Test Case Derivation: This test case is used when inputing the climate models. There are three parameters that would be used from the model and this test is trying to detect if any of them is missing. If so, it will

produce error message. It will also produce error message if there is such data but it is invalid.

How test will be performed: The author will pick the sample test case from the model (as in Table 3) and create automatic test by Python.

Input		Output	
AADT	*AADTT*	valid?	Error Message
20176	433	Y	No error message, continue to import model data
152	8095	N	*AADTT* should be less than *AADT*
	152	N	Data missing for *AADT*
8095		N	Data missing for *AADTT*

Table 4: Traffic model input tests

2. test-input-traffic

   Control: Automatic

   Initial State: Uninitialized

   Input: The input column in Table 4.

   Output: The output column in Table 4.

   Test Case Derivation: This test case is applicable when entering traffic models into the software. There are two parameters that would be used from the model and this test is trying to detect if any of them is missing. If so, it will produce error message. Similarly, if the data exists but is invalid, the software will also generate an error message.

   How test will be performed: The author will pick the sample test case from the model (as in Table 4) and create automatic test by Python.

4.1.2 Output Tests - test_output

This section covers R2, R3 of the SRS which includes the check for the final outputs.

Functional tests - Output tests - Chloride exposure

1. test-output-exposure

   Control: Automatic

   Initial State: Uninitialized

   Input: Coordinate

   Output: If the input coordinate is within Ontario, it will return a series (length of at lease ten) of data showing the chloride exposure at that location.

   Test Case Derivation: This test case cover the R2 and R3 of the SRS. The test will be done by comparing the actual result with the result from Matlab, which is also implemented by the author as a second way to ensure the accuracy.

   How test will be performed: The automatic test will be performed by Python.

4.2 Tests for Nonfunctional Requirements

Non-functional requirements for BC are given in SRS section 5.2. There are five non-functional requirements for BC, and four of them are described in the following sections correspondingly.

4.2.1 NFR: Reliability

Reliability The reliability of the software is tested through the test for functional requirements in section 4.1.2 and the unit testing in section 5.

4.2.2 NFR: Usability

Usability

1. test-usability

   Type: Manual with potential users

   Initial State: The software is set up and ready for testing.

   Input/Condition: None

Output/Result: Survey result about the user experience with the software.

How test will be performed: Observe real users as they perform tasks with the software. Ask them to fill out a survey on their experience, including ease of use, intuitiveness, and any challenges encountered. The survey is in Section 6.1

4.2.3 NFR: Maintainability

Maintainability

1. test-maintainability

   Type: Code walkthrough with potential developers

   Initial State: None

   Input/Condition: None

   Output/Result: Feedback from fellow classmates or team members in Table 1.

   How test will be performed: Show the code and the other documents to fellow classmates/potential developers, ask them to explain the code to the author and see how much they could understand. Gather any feedback or questions they have.

4.2.4 NFR: Portability

Portability

1. test-portability

   Type: Manual and automatic

   Initial State: None

   Input/Condition: None

   Output/Result: Result of compilation.

   How test will be performed: Use Github Actions to test if the software could be installed in multiple machines and environments. Also do manual test with potential users to install the software on their computers, see how good they are following the README procedure.

4.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	NFR1	NFR2	NFR3	NFR4
Test 4.1.1	X						
Test 4.1.2		X	X				
Test 4.2.1				X			
Test 4.2.2					X		
Test 4.2.3						X	
Test 4.2.4							X

Table 5: Traceability Between Test Cases and Requirements

5 Unit Test Description

The source code for BC has following modules, they are corresponding to M2 to M14 in MG and MIS:

- input_check.py (M2)
- main.py (M3)
- search.py (M4)
- visualization.py (M5)
- calculation_load.py (M6)
- constant.py (M7)
- deicing_salts_cal.py (M8)
- melted_water_thickness.py (M9)
- single_water_SAS_cal.py (M10)
- single_Cl_SAS_cal.py (M11)
- all_Cl_SAS_cal.py (M12)

- chloride_decomposition_cal.py (M13)
- calculation.py (M14)

5.1 Unit Testing Scope

Unit testing is performed for the following modules:

- calculation.py
- calculation_load.py
- input_check.py
- search.py
- visualization.py

These modules are high priority modules that can affect the whole system if not work properly. The calculation module is the module that will called and use M8 to M14, so M8 to M14 could be verified if the calculation module is working properly. Other modules such as constant are tested in the system as those have low priority static modules. The whole system follows [1] and Mingsai, the author, has validate in the paper that the formulas and models used in the paper is accurate and reliable.

5.2 Tests for Functional Requirements

All tests are automatic and performed by PyTest. The tests are classified by modules in this section, with the link to the unit test file. The explanations of test decisions are included in the comment of the unit test file, with short name introduction in this section.

5.2.1 Calculation Module

The test cases in this section not only cover the calculation module, but also cover M8 to M14, as those modules are used here. The tests here are checking the value for each step of the calculation module, and their pass indicates the functionality of the encompassed modules as well. The expected output is from another verification from Matlab. Each test case is named to clearly declare its purpose. The code could be found in test_calculation.py.

- test_deicing_salts_cal
- test_melted_water_thickness
- test_single_water_SAS_cal
- test_single_Cl_SAS_cal
- test_all_Cl_SAS_cal
- test_salts_decomposition_cal

5.2.2 Calculation_load Module

The test cases in this section check if the traffic model and climate model used to generate the database has any missing values. Ideally, there would not be empty values as those models have been verified by their developers, this module is doing the double check. The code could be found in test_model_input.py.

- test_longitude
- test_latitude
- test_AADT
- test_AADTT
- test_t1
- test_h_total
- test_t2

5.2.3 Input_check Module

This section covers the access programs in input_check module. It includes some boundary cases for checking if a coordinate is within ontario. The coordinate includes both negative cases and positive cases and the software could handle them both. One worth noticing point here is the Ontario boundary only include the land, not the water. For example, the points on Lake Erie is not considered as within Ontario. The code could be found in test_input_check.py.

- test_load_file
- test_value_error_nonnumerical
- test_value_error_empty
- test_on_ontario_boundary
- test_in_ontario_boundary
- test_outside_ontario_boundary
- test_lake_on_edge_of_ontario

5.2.4 Search Module

This section cover the access programs in search module. The main purpose for the test case is to demonstrate that the find_closest method can differentiate between coordinates up to four decimal places. It also shows the method could work fine on large dataset. The code could be find in test_search.py

- test_load_data
- test_find_closest_four_decimal
- test_find_closest_large_dataset

5.2.5 Visualization Module

The most essential function of visualization module, plotting, is from the plotly library, so the unit test on plotting is not tested. Instead, we focus on verifying whether this function successfully generates output. Also, the helper function for generating sublist is tested, as to plot the graph we want to exclude the coordinate information in the beginning. The code could be find in test_visualization.py

- test_generate_sub_keylist
- test_generate_sub_valuelist
- test_draw_graph

5.3 Tests for Nonfunctional Requirements

Unit testing the non-functional requirements is beyond the scope. Tests related to non-functional requirements of system are introduced in section 4.2.

5.4 Traceability Between Test Cases and Modules

	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
Test 5.2.1					X	X	X	X	X	X	X	X	X
Test **??**					X								
Test 5.2.2	X	X											
Test 5.2.3		X	X										
Test 5.2.4				X									

Table 6: Traceability Between Test Cases and Modules

Reference

1. Mingsai Xu, Yuxin Zheng, Cancan Yang (2024). Assessing Highway Bridge Chloride Exposure at a Provincial Scale: Mapping and Projecting Impacts of Climate Change. [Manuscript in preparation].

6 Appendix

This is where you can place additional information.

6.1 Usability Survey Questions

- From one to five and five being the most satisfied, how would you rate your overall experience using the software?
- Were the instructions clear and easy to understand. If no, please explain why.
- Did the software include all the features you expected? If no, what additional features would you like to see?
- From one to five and five being the most satisfied, how would you rate the speed and responsiveness of the software?