Software Requirements Specification for Bridge Corrosion: A Chloride Exposure Prediction Model

Cynthia Liu

February 15, 2024

Revision History

Date	Version	Notes
Feb 9, 2024	1.0	Initial release

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
BC	Bridge Corrosion
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

[symbols, abbreviations, or acronyms — you can simply reference the SRS (**?**) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document provides the road-map of the verification and validation plan for Bridge Corrosion (BC), and ensure the requirements and goals of the program mentioned in SRS. The organization of this document starts with the general information and verification plans, followed by the system test description for functional and non-functional requirements. Then, it includes the unit test which would not be filled in until after the MIS.

2 General Information

2.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS] The software 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 trend when user input 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 and non-functional requirements mentioned in SRS are tested. We will try to cover a usability test if we have time and resource. We also assume the climate and traffic models we generate online have been verified by their implementation team so their accuracy is ensured.

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your

system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

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.

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. —SS]

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

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).

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

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 if desired. [List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates, or you may plan for something more rigorous/systematic. —SS]

[Maybe create an SRS checklist? —SS]

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. [Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

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. [The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

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. If possible, a code walkthrough could happen. [You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

3.6 Automated Testing and Verification Tools

The first part of the software, generating the database through climate and traffic model, will be done in MATLAB. The second part of finding the data for the input location, will be done in R. The automated testing will be done in MATLAB and R correspondingly. During the coding, mlint and

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

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

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. [If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

4 System Test Description

4.1 Tests for Functional Requirements

Functional requirements for BC are given in SRS section 5.1. There are five functional requirements for BC, R1 is related to the input, R3 is related to the midway calculation, and the other requirements are corresponding to outputs. Section 5.1.1 describes the input tests related to R1, and section 5.1.2 describes the output test related to other requirements. The test for R3 will be covered in section 5.

R2: The output need to be a series of data showing the trend of chloride exposure in the past and future at the input location (By IM2).
R3: During the calculation, the software should be capable of handling situations where units do not match.
R4: The output from the previous year should be verifiable against real-world data.
R5: The output should be in two decimal points, showing the mass of chloride ions per unit air volume (By IM1)

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

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.

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Functional tests - Input tests - Coordinate

Input (°)		Output	
longitude	*latitude*	valid?	Error Message
0.039	0.046	Y	NONE
0.039	0.046	Y	NONE
0.039	0.046	Y	NONE
0.039	0.046	Y	Error Message
0.039	0.046	Y	Error Message
0.039	0.046	Y	Error Messagevsdgdfsgfas

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. [The expected result for the given inputs —SS]

   Test Case Derivation: The input coordinate need to be a location inside Ontario. The software produces error message for the invalid inputs, for the valid inputs, it will proceed to next step. [Justify the expected value given in the Output field —SS]

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

Functional tests - Input tests - Models

1. test-input-climate

   Control: Automatic

   Initial State: Uninitialized

Input (°)			Output	
h_{total}	t_1	t_2	valid?	Error Message
0.039	0.046	93	Y	fnjsdfnksas

Table 3: Climate model input tests

Input: The input column in Table 3.

Output: The output column in Table 3. [The expected result for the given inputs —SS]

Test Case Derivation: This test case is used when inputing the climate models, the software will try to find the data needed for the software. If there is no such data it will produce error message. It will also produce error message if there is such data but it is invalid. [Justify the expected value given in the Output field —SS]

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 R.

Input (°)		Output	
$AADT$	$AADTT$	valid?	Error Message
0.039	0.046	Y	fnjsdfnksas

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. [The expected result for the given inputs —SS]

Test Case Derivation: This test case is applicable when entering climate models into the software. The software will attempt to locate the necessary data. If the required data is not found, it will generate an error message. Similarly, if the data exists but is invalid, the software will also generate an error message. [Justify the expected value given in the Output field —SS]

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 R.

4.1.2 Intermediate Tests - test_model_calculation

This section covers R3 of the SRS which includes the check for the middle calculation of the software to generate the database of the software.

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Functional tests - Intermediate tests - Model Calculations

1. test-calculation

Control: Automatic

Initial State: None

Input: The input columns in Table 5

Output: The output columns in Table 5

Test Case Derivation:

How test will be performed:

Test case	Input	Output
test-deicing-salts-quantity	M_{total}, t_1 d-by-one-truck	M_{app}
test-melted-water-thickness		
test-water-sprayed-and-splashed		
test-chloride-ions-sprayed-and-splashed-by-one-truck		
test-chloride-ions-sprayed-and-splashed-by-all-vehicles		
test-deicing-salts-deposition-on-the-surface-of-the-bridge-substructure		

Table 5: Test Case for Model Calculation

4.1.3 Output Tests - test_output

This section covers R2, R4, R5 of the SRS which includes the check for the final outputs.

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

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 two decimal points data showing the chloride exposure at that location.

   Test Case Derivation: This test case cover the R2 and R5 of the SRS. The test will be done by checking the length of the output and the

precision after decimal points. The expected value should be within the data constraints in the 4.4.7 section of SRS. [Justify the expected value given in the Output field —SS]

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

2. test-output-verification

   Control: Manual

   Initial State: Uninitialized

   Input: Chloride exposure data

   Output: True or False.

   Test Case Derivation: This test case cover the R4 of the requirement. It compared the chloride exposure data that the software generates with the real world data for verification. The test result will be true if the derivation is within 5%.

   [Justify the expected value given in the Output field —SS]

   How test will be performed: Manually compare it with the real world data from previous year.

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, which are described in the following sections correspondingly. [The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

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 could be find in 6.2

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

   Initial State: None

   Input/Condition: None

   Output/Result: Result of compilation.

   How test will be performed: Manually run the system on multiple versions of browser and operating system to see if it is successful.

4.2.5 NFR: Scalability

Scalability

1. test-scalability

   Type: Automatic

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

   Input/Condition: Increase the workload by adding more data.

   Output/Result: Different response time on how the software handles the increased workload.

   How test will be performed: Record the response time for the software when given increasing amount of dataset, evaluate the different response time.

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

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

5.2 Tests for Functional Requirements

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

5.2.1 Module 1

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

1. test-id1

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

   Initial State:

Input:

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

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

How test will be performed:

2. test-id2

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

   Initial State:

   Input:

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

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

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

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

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

5.3.1 Module ?

1. test-id1

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

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

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

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

- 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?
- How would you rate the speed and responsiveness of the software?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

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

Appendix — Reflection

[This section is not required for CAS 741 —SS]

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

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?