Module Guide for Bridge Corrosion

Cynthia Liu

March 21, 2024

1 Revision History

Date	Version	Notes
March 6, 2024	1.0	Initial release

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
BC	Bridge Corrosion
DAG	Directed Acyclic Graph
GUI	Graphical User Interface
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team. We advocate a decomposition based on the principle of information hiding. This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section **??** lists the anticipated and unlikely changes of the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** gives a detailed description of the modules. Section **??** includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section **??** describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section **??**, and unlikely changes are listed in Section **??**.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Coordinate outside Ontario might be considered if the project becomes a larger scale. This is relevant to R1 in SRS.

AC2: The algorithm for calculating chloride exposure might change if the theory proceeds.

AC3: The input method could include clicking a point on the map as another way of inputing coordinate.

AC4: The output could have another format which is saving the search result to file.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The traffic data model and climate data model are inputed from developer and will be hidden to user.

UC2: The user input is only the coordinate.

UC3: The output is a list of chloride exposure data displayed in grid, visualized in histogram or line graph.

UC4: The database storing chloride exposure data will always be external.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table **??**. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware Hiding Module

M2: Input Parameter Module

M3: Input Verification Module

M4: Control Module

M5: Data Searching Module

M6: Output Visualization Module

M7: Chloride Exposure Calculation Model

M8: Sequence Data Structure Module

M9: Plotting Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Parameter Module Input Verification Module Control Module Data Searching Module Output Visualization Module Chloride Exposure Calculation Model
Software Decision Module	Sequence Data Structure Module Plotting Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table **??**.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding". The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *ProgName* means the module will be implemented by the ProgName software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M??)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 Input Parameter Module (M??)

Secrets: The format and structure of the input data.

Services: Get input from the user. Converts the input data into the data structure used by the Input Verification module.

Implemented By: BC

Type of Module: Function

7.2.2 Input Verification Module (M??)

Secrets: The kind of valid and invalid data.

Services: Verify if the input is within the physical and software constraint, throw warning message if it is not. Pass the valid input to the Data Searching module.

Implemented By: BC

Type of Module: Abstract Data Type

7.2.3 Control Module (M??)

Secrets: The algorithm for running the program.

Services: Provide the main program and the GUI.

Implemented By: BC

Type of Module: Abstract Data Type

7.2.4 Data Searching Module (M??)

Secrets: The irrelevant values in database.

Services: Find the data needed in the database by searching algorithm, stored it in the data structure that the Output Visualization module needs.

Implemented By: BC

Type of Module: Abstract Data Type

7.2.5 Output Visualization Module (M??)

Secrets: The format and structure of the output data.

Services: Provide the visualization of the resulting chloride exposure trend, using line graph or histogram.

Implemented By: BC

Type of Module: Abstract Data Type

7.2.6 Chloride Exposure Calculation Module (M??)

Secrets: Detail calculation formulas for chloride exposure trend.

Services: Get the climate and traffic data model from developer, use them to generate the chloride exposure database that is needed in the Data Searching Module.

Implemented By: BC

Type of Module: Abstract Data Type

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Sequence Data Structure Module (M??)

Secrets: The data structure for sequence data type.

Services: Provides different list operations including looping, adding and removing elements.

Implemented By: Python

Type of Module: Library

7.3.2 Plotting Result Module (M??)

Secrets: The data structure and algorithms for plotting data graphically.

Services: Provides plot function that can plot the results from Data Searching Module, used by Output Visualization module.

Implemented By: Plotly

Type of Module: Library

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
R??	M??, M??, M??, M??
R??	M??, M??, M??, M??, M??
R??	M??, M??
R??	M??, M??
NFR??	M??, M??
NFR??	M??, M??, M??
NFR??	all modules
NFR??	M??
NFR??	all modules

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC??	M??
AC??	M??
AC??	M??
AC??	M??
UC??	M??
UC??	M??
UC??	M??
UC??	M??

Table 3: Trace Between Anticipated Changes, Unlikely Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure **??** illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of

the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

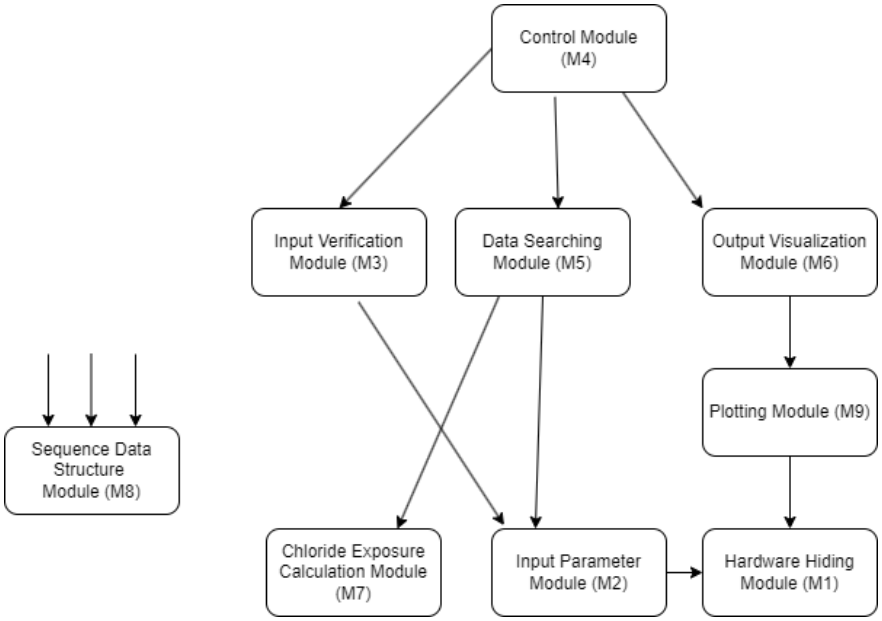

Figure 1: Use hierarchy among modules

10 User Interfaces

Figure 2: GUI illustration before inputing coordinate

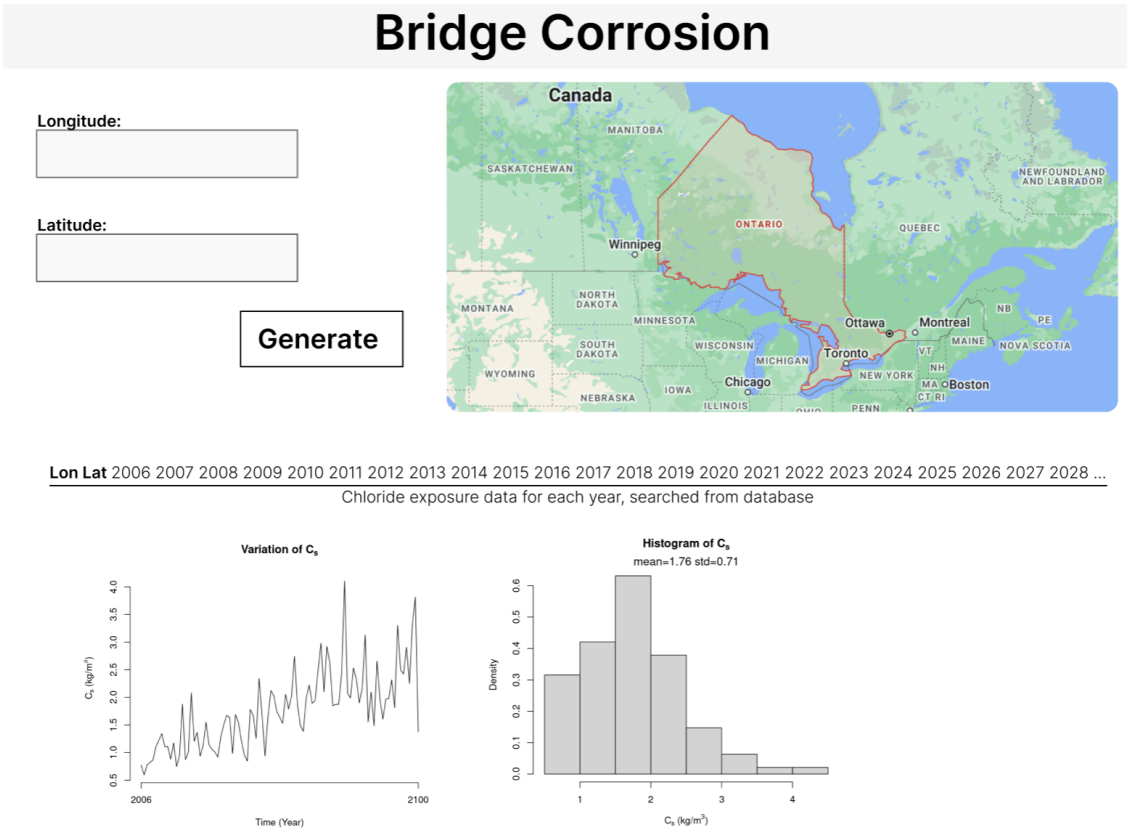

Figure 3: GUI illustration after inputing coordinate

11 Timeline

The specific timeline is shown in below table:

Modules	**Finish by**	**Responsible**
Chloride Exposure Calculation Module	February 25, 2024	Cynthia Liu
Input Verification Module	March 5, 2024	Cynthia Liu
Input Parameter Module	March 8, 2024	Cynthia Liu
Data Searching Module	March 20, 2024	Cynthia Liu
Output Visualization Module	March 27, 2024	Cynthia Liu
Control Module	April 5, 2024	Cynthia Liu

Table 4: Timeline

12 Appendix

This section shows the requirements in SRS for quick look up.

12.1 Functional Requirements

R1: The software should only accept input that is inside Ontario.

R2: The output need to be a list of data showing the trend of chloride exposure in the past and future at the input location.

R3: During the automatic calculation steps, the software should handle situations where units do not match.

R4: The output should be in two decimal points, showing the mass of chloride ions per unit air volume.

12.2 Nonfunctional Requirements

NFR1: **Reliability**: The predictions generated by the software should be accurate and reliable, reflecting real-world conditions and factors influencing chloride exposure.

NFR2: **Usability**: The software interface should be intuitive and user-friendly, allowing users to easily input coordinates and look at the predicted chloride exposure in the past and future.

NFR3: **Maintainability**: The code for this software should be designed and structured in a way that it could be easily comprehended and modified by other potential developers.

NFR4: **Portability**: This software should be able to run on recent versions of Google Chrome, Firefox, MS Edge and Safari. The operating system include Windows 7+ and Mac OS X 10.7+.

NFR5: **Scalability**: The software should be scalable to accommodate potential future expansions or updates, ensuring its continued usefulness as new data or techniques become available.