Module Guide for Greenway

Team #11, Roadkill
Priyansh Shah, shahp36
Utsharga Rozario, rozariou
Jash Mehta, mehtaj8
Bilal Shaikh, shaikb2
Pranay Kotian, kotianp
Sharjil Mohsin, mohsis2

January 19, 2023

1 Revision History

Date	Version	Notes
Jan 18, 2023	1.0	Revision 0
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Greenway	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

Contents

1 Revision History **i**

2 Reference Material **ii**

2.1 Abbreviations and Acronyms . . . ii

3 Introduction **1**

4 Anticipated and Unlikely Changes **2**

4.1 Anticipated Changes . . . 2

4.2 Unlikely Changes . . . 2

5 Module Hierarchy **3**

6 Connection Between Requirements and Design **3**

7 Module Decomposition **4**

7.1 Hardware Hiding Modules (M1) . . . 4

7.2 Behaviour-Hiding Module . . . 4

7.2.1 Location Input Module (M2) . . . 4

7.2.2 Car Input Module (M3) . . . 5

7.2.3 Map Display Module (M4) . . . 5

7.2.4 Fuel Information Module (M5) . . . 5

7.2.5 Elevation Module (M6) . . . 5

7.3 Software Decision Module . . . 5

7.3.1 Map Data Module (M7) . . . 6

7.3.2 Car Data Module (M8) . . . 6

7.3.3 Fuel Data Module (M9) . . . 6

8 Traceability Matrix **6**

9 Use Hierarchy Between Modules **7**

List of Tables

1 Module Hierarchy . . . 3

2 Trace Between Requirements and Modules . . . 7

3 Trace Between Anticipated Changes and Modules . . . 7

List of Figures

1 Use hierarchy among modules . . . 7

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 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 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 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 9 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 4.1, and unlikely changes are listed in Section 4.2.

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: The addition of the application being able to use mileage data as well as gas tank size to calculate the most efficient route on a long journey, allowing for the least amount of stops to fill up gas.

AC2: The addition of the application taking data of electric cars to allow the use of not only gas station but charging stations for electric car users.

AC3: The application will be able to be used in all of Ontario with elevation and gas data for all areas.

AC4: Future changes should allow the application to be used with other apps other than Google Maps, such as Wayz or Apple Maps.

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 application will not support other provinces or countries.

UC2: The application will not be able to update car catalog information on its own and will require an update to the application when new cars are introduced.

UC3: The application will not support hybrid vehicles as they require complex calculations to determine when the electric component of the vehicle is used and when the gas component is.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Location Input Module

M3: Car Input Module

M4: Map Display Module

M5: Fuel Information Module

M6: Elevation Module

M7: Map Data Module

M8: Car Data Module

M9: Fuel Data Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Location Input Module
	Car Input Module
	Map Display Module
	Fuel Information Module
	Elevation Module
Software Decision Module	Map Data Module
	Car Data Module
	Fuel Data 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 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). 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. *Greenway* means the module will be implemented by the Greenway 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 (M1)

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 Location Input Module (M2)

Secrets: Handles the user input of any location data.

Services: Converts the user input of location to coordinates with the assistance of Map Data Module. Stores any necessary information from the conversion and for future use in the app.

Implemented By: Greenway

7.2.2 Car Input Module (M3)

Secrets: Handles the user input of any car related data.

Services: Converts the user input of car details to store necessary information. And works with Car Data to validate car details to use with the application.

Implemented By: Greenway

7.2.3 Map Display Module (M4)

Secrets: Displays the Map and coordinates interactions related with it.

Services: Works with the Map Data Module to retrieve map details accurately update the display for it as necessary. Along with using the data provided by the Location Input Module to update any markers on the map.

Implemented By: Greenway

7.2.4 Fuel Information Module (M5)

Secrets: Handles any information related to mileage data and fuel prices. Respectively outputs or takes in put of any related data.

Services: Works with the Fuel Data Module to retrieve accurate fuel prices and mileage data respectively. Along with parsing information for use in any other module.

Implemented By: Greenway

7.2.5 Elevation Module (M6)

Secrets: Handles any information related to Elevation related information at any point on the map.

Services: Works with the Map Data Module to retrieve Elevation data to use for computations such as total cost and mileage changes respectively due to elevation gradients.

Implemented By: Greenway

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 Map Data Module (M7)

Secrets: Retrieves any map related data that is required for the application to function.

Services: Uses HTTPS get requests with Google Maps API's to retrieve necessary information for the getting map related data and stores it respectively.

Implemented By: Greenway

7.3.2 Car Data Module (M8)

Secrets: Retrieves any car related data that is required for the application to function.

Services: Uses HTTPS RESP API requests with existing MongoDB Database to retrieve any car related data for validation purposes. Similarily, store any information that is not found in the database.

Implemented By: Greenway

7.3.3 Fuel Data Module (M9)

Secrets: Retrieves any Fuel related data that is required for the application to function.

Services: Uses HTTPS get requests with existing Fuel Information API to retrieve any fuel related data for calculations. Similarily, store any information that is not from the API calls.

Implemented By: Greenway

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
FR1	M2, M7, M4
FR2	M2, M7, M4
FR3	M3, M8,
FR4	M5, M9
FR5	M3, M8
FR6	M4, M7
FR7	M4, M5, M9
FR8	M5, M9
FR9	M7, M8, M9
FR10	M6, M7
FR11	M6, M7, M5, M9
FR12	M7, M8, M9

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M3, M5, M8, M9
AC2	M3, M7, M8
AC3	M5, M6
AC4	M1

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that 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 1 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

References

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.