Module Guide for 4TB6 - Mechatronics Capstone

Team #5, Locked & Loaded
Abi Nevo, nevoa
Elsa Bassi, bassie
Steffi Ralph, ralphs1
Abdul Iqbal, iqbala18
Stephen De Jong, dejons1
Anthony Shenouda, shenoa2

January 16, 2023

1 Revision History

Date	Version	Developer	Notes
02-01-23	1.0	Elsa	Likely and Unlikely Changes
10-01-23	2.0	Elsa	Modules
15-01-23	3.0	Elsa	Remaining Contents
16-01-23	4.0	Elsa	References

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
SRS	Software Requirements Specification
4TB6 - Mechatronics Capstone	Explanation of program name
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 (Parnas et al, 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972a). 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 Parnas et Al (1984), 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, (Parnas et Al, 1984). 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 specific hardware on which the software is running.

AC2: The format of the initial input data and parameters on the App.

AC3: The format of the output data on the App.

AC4: The implementation of the transmission of the lock status signal (engaged status signal).

AC5: The implementation of the transmission of the 'disengage' signal.

AC6: How the battery status calculation is defined within the App.

AC7: How the Arduino allows/disallows power to the load (electromagnet) I.e., the implementation of the switch (transistor).

AC8: The implementation of the location determinant feature.

AC9: The size of the battery.

AC10: The size of the electromagnet.

AC11: The implementation of the locking mechanism.

AC12: The implementation of the lock frame.

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 I/O devices (iPhone or android smartphone).

UC2: The circuit devices (Arduino, battery and electromagnet).

UC3: Output data are displayed to the output device (App).

UC4: Output data can be verified by checking the lock frame.

UC5: The goal of the system (wirelessly and hands-free disengage the bike lock).

UC6: There will always be a source of input data external to the software.

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: Input Parameters Module

M3: Output Parameters Module

M4: Engage Status Signal Module

M5: Disengage Signal Module

M6: Battery Status Module

M7: Load Power Signal Module

M8: Location Module

M9: Battery Module

M10: Electromagnet Module

M11: Locking Mechanism Module

M12: Lock Frame Module

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.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Parameters Module
	Output Parameters Module
	Engage Status Signal Module
	Disengage Signal Module
	Battery Status Module
	Location Module
	Lock Frame Module
Software Decision Module	Load Power Signal Module
	Battery Module
	Electromagnet Module
	Locking Mechanism Module

Table 1: Module Hierarchy

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. *4TB6 - Mechatronics Capstone* means the module will be implemented by the 4TB6 - Mechatronics Capstone 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 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 accept inputs.

Implemented By: App 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 Parameters Module (M2)

Secrets: The data structure for input parameters and how the values are inputted and verified. The load and verify secrets are isolated to their own access programs, found in their respective submodules.

Services: Receives input from the user, stores input and verifies that the input parameters comply with physical and software constraints. Throws an error if a parameter violates a physical constraint. Throws a warning if a parameter violates a software constraint. Stored parameters can be read individually, but write access is only to redefine the entire set of inputs.

Implemented By: App

7.2.2 Output Parameters Module (M3)

Secrets: The format and structure of the output data.

Services: Outputs the results of the battery status calculations, the engaged status signal and the location data.

Implemented By: The App

7.2.3 Engage Status Signal Module (M4)

Secrets: The determination and transmission of the 'Engage Status' signal, which tells the user whether the locking mechanism is engaged or disengaged.

Services: The Arduino determines the status of the locking mechanism, then wirelessly transmits this data to the App as a signal so it can be displayed to the user as the 'Engage Status'.

Implemented By: The App and the Arduino

7.2.4 Disengage Signal Module (M5)

Secrets: The determination and transmission of the 'Disengage' signal, which disengages the locking mechanism.

Services: The user inputs the command to disengage the locking mechanism on the GUI. The App then wirelessly transits the 'Disengage' signal, which is received by the Arduino.

Implemented By: The App and the Arduino

7.2.5 Battery Status Module (M6)

Secrets: The determination of the current level of the battery to inform the user when replacement is required.

Services: The battery level is calculated within the App based on some inherent battery properties. It is then displayed on the GUI as 'Battery Status'.

Implemented By: The App

7.2.6 Location Module (M8)

Secrets: The implementation of the location determinant.

Services: The algorithm that governs the geocaching of the bike's location upon locking.

Implemented By: The App

7.2.7 Lock Frame Module (M12)

Secrets: The design of the physical 'lock frame' that connects the bike and the external frame it is being 'locked' to.

Services: Is the interface that connects the user, the App and the locking mechanism that allows the user to utilize the entire system.

Implemented By: The Mechanical System

7.3 Electrical/Mechanical/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 Load Power Signal Module (M7)

Secrets: How the Arduino allows and disallows power to the load.

Services: The implementation, logic and transmission of the power signal sent from the Arduino to the transistor, which acts as a switch to allow power to the electromagnet.

Implemented By: The Arduino

7.3.2 Battery Module (M9)

Secrets: The specifications of the circuit power supply.

Services: The type and size of the battery which supplies power to the Arduino and the electromagnet.

Implemented By: The Electrical Circuit

7.3.3 Electromagnet Module (M10)

Secrets: The specifications of the component of the circuit responsible for moving the locking mechanism.

Services: The type and size of the electromagnet, which must provide a magnetic force strong enough to move the locking mechanism into its desired position for disengagement.

Implemented By: The Electrical Circuit

7.3.4 Locking Mechanism Module (M11)

Secrets: The design of the moving part that engages and disengages the lock frame.

Services: Responds to the magnetic force supplied by the electromagnet in such a way that it engages and disengages the lock frame.

Implemented By: The Mechanical System

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	M12
FR2	M1, M2, M5, M7, M9, M10, M11
FR3	M1, M4, M9
FR4	M1, M4, M9
FR5	M1, M2, M8, M9
FR6	M9, M10, M11, M12
FR7	M2, M5
FR8	M12
FR9	M1, M3, M6, M9
FR10	M1, M3, M8
FR11	M1, M9, M10, M11

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M4
AC5	M5
AC6	M6
AC7	M7
AC8	M8
AC9	M9
AC10	M10
AC11	M11
AC12	M12

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.

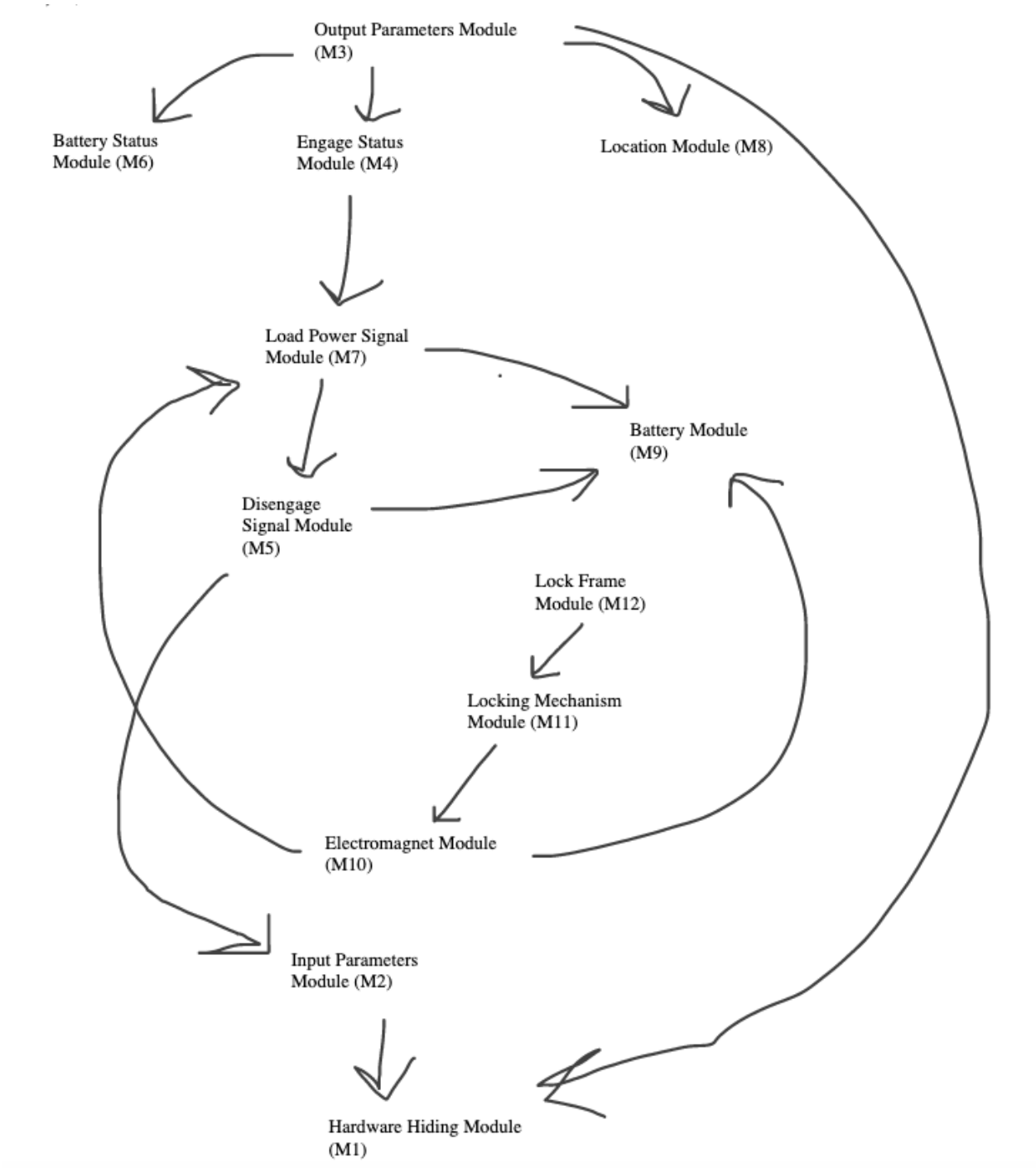

Figure 1: Use Hierarchy Among Modules