Module Guide for Mechatronics Engineering

Team #1, Back End Developers
Jessica Bae
Oliver Foote
Jonathan Hai
Anish Rangarajan
Nish Shah
Labeeb Zaker

January 16, 2023

1 Revision History

Date	**Version**	**Notes**
Date 1	1.0	Notes
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
Mechatronics Engineering	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

Contents

List of Tables

List of Figures

3 Introduction

This document serves as the module guide for the system currently in development by Back End Developers. It will decompose the system into modules; detailing the purpose of each individual module of the system, what resources are necessary to fulfill the requirements of each module, and the interactions between said modules. In addition, this document will list the anticipated and unlikely changes expected for the system, and the relevant traceability matrices between related components.

This document has been written to address the needs of a wide audience. It is intended to help on-board new members joining the project by providing them with a structured breakdown of the project in its intended state, and is intended to provide a reference document to the engineers who are responsible for maintenance of the system should they feel that they need to make any necessary changes. In addition, it is intended to be read by the designers of the system itself; to give them the ability to have a big-picture overview of the system that they are working on. After a thorough read-through of this document, a reader with a reasonable amount of background knowledge (See document from before) should be able to recreate the functionality of each module, their interactions with each other, and therefore the system as a whole.

For this document to fulfill all of these requirements specified above is a difficult task, and the content of it is likely unable to capture 100% of the information necessary to fulfill its goals to perfection. It should be obvious that engineers are by no means omniscient, not even regarding the systems that they design themselves. However, the authors of this document will still attempt to cover the relevant content in a digestible and comprehensive way, and thank the reader for their patience and good will.

All text in red is written by the professor:

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, 1972). 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.

...

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: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

...

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: Device Manager Module

M2: Data Storage Module

M3: Sensor Array Module

M4: Display System Module

M5: Prompt Generation Module

M6: Real Time Clock Module

M7: Moving Average Algorithm Module

M8: Graph Plotter 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	Device Manager
	Data Storage
	Sensor Array
Behaviour-Hiding Module	Display System
	Prompt Generation
	Real Time Clock
Software Decision Module	Moving Average Algorithm
	Graph Plotter

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. *Mechatronics Engineering* means the module will be implemented by the Mechatronics Engineering 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

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

7.1.1 Device Manager (M1)

Secrets: Queue to store the data

Services: This module provides the interface between the hardware and the software. The hardware will be copying the processed values into this module while the software reads from this module and produce plots for the researcher when prompted.

Implemented By: Firmware

7.1.2 Data Storage (M2)

Secrets: Static array to store the data

Services: This module will store the data through the entire monitoring period. The data from this module will be stored in the database, so that a history of past records are maintained.

Implemented By: Firmware

7.1.3 Sensor Array (M3)

Secrets: Static array to store the sensor data

Services: This module will store the raw data measured by the sensors. These are then processed on the hardware and copied to Device Manager to be read by the software.

Implemented By: Firmware

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 Display System Module (M4)

Secrets: The error codes and screen parameters.

Services: This module is used to display text at desired positions on the tft display. This module contains API functions that are called in the main executable to display text on the screen.

Implemented By: BED

Type of Module: Library

7.2.2 Prompt Generation Module (M5)

Secrets: The format and structure of the prompts.

Services: Converts the prompt messages into the data structure used by the display module. This module also stores all the prompts in the data structure specified. Multiple prompts are stored in a list that is accessed later in display module through API.

Implemented By: BED

Type of Module: Abstract Data Type

7.2.3 Real Time Clock Module (M6)

Secrets: The format and structure of the time data structure.

Services: Converts the real time from the board into the data structure used by the display module. The time will also be used alongside the data monitored from sensors to keep respective timestamps. It also includes API for the display module to access the time.

Implemented By: BED

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 Moving Average Algorithm (M7)

Secrets: An array of the last 20 recent beats measured from sensors

Services: This algorithm basically calculates average of 20 different beats per minute at any point in time. It is cumulative and helps in deciding whether the EMA should be triggered.

Implemented By: C

7.3.2 Graph Plotter (M8)

Secrets: The data structures and algorithms to plot a graph

Services: This algorithm basically provides a plot of 2 parameters of interest. This can help in identifying desired trends to deal with it.

Implemented By: Python

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**
R1	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M??
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

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. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

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.

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:

1.

2.