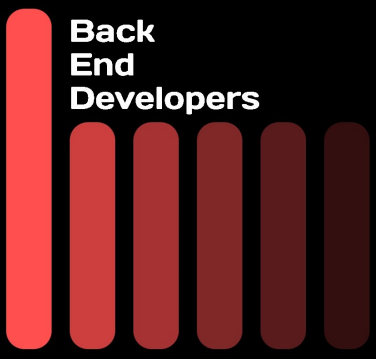
Module Guide for Mechatronics Engineering



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

April 5, 2023

1 Revision History

Date	Version	Notes
2023-01-18	1.0	Initial documentation
2023-03-15	2.0	Minor improvements and proof reading for revision 1
2023-04-03	2.1	Incorporated TA comments
2023-04-04	2.2	Added logo and style to the document

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

Contents

List of Tables

List of Figures

3 Introduction

This document serves as the module guide for the system currently in development by the 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 SRS) 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.

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.

AC3: The type of heartrate sensor used.

AC4: The use of gyroscope 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, the modules that will actually be implemented.

M1: Battery Module

M2: MicroSD Module

M3: Database Module

M4: Sensor Reading Module

M5: Sensor Data Processing Module

M6: Display System Module

M7: Prompt Generation Module

M8: Real Time Clock Module

M9: Graph Module

M10: Data Display Module

M11: Configuration Module

M12: Watch Straps and Case Module

Table 1: Module Hierarchy

Level 1	**Level 2**	**Level 3**
Hardware-Hiding Module	Battery Management	Battery
	Data Storage	microSD Database
	Sensor Array	Sensor Reading Sensor Data Processing
	Physical Design	Watch Straps and Case
Behaviour-Hiding Module	Display System	Display Screen
	Prompt Generation	Prompt Generation
	Real Time Clock	RTC
Software Decision Module	Parameter selection	Configuration
	Data Processing	Graph Data Display

6 Connection Between Requirements and Design

Please refer to section 6.4 in the System Design document .

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 Battery System (M1)

Secrets: Array to store data

Services: This module provides the interface between the hardware and the software for the battery status. The battery levels will be used to determine which systems are active. This module will also shut down inactive processes so that battery life can be saved.

Implemented By: Firmware

7.1.2 MicroSD (M2)

Secrets: Static array to store the data on device.

Services: This module will store the data through the entire monitoring period.

Implemented By: Firmware

7.1.3 Database (M6)

Secrets: Static array to store the data on local host computer.

Services: This module will store the data in the database, so that a history of past records are maintained.

Implemented By: Firmware

7.1.4 Sensor Reading (M4)

Secrets: Static array to store the sensor data

Services: This module will store the raw data measured by the sensors without determining the validity of those raw data.

Implemented By: Firmware

7.1.5 Sensor Data Processing (M5)

Secrets: Static array to store the sensor data

Services: This module will process the law data and smooth out noise for filtering, ready to be stored in memory.

Implemented By: Firmware

7.1.6 Watch Straps and case(M12)

Secrets: Not applicable.

Services: This module lays out the structural design of the device.

Implemented By: CAD, straps

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 (M6)

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 (M7)

Secrets: The format and structure of the prompts.

Services: This module converts the prompt messages into the data structure used by the display module. It also stores all the prompts of 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 (M8)

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

Services: This module converts the real time from the board into the data structure used by the display module. It will also be used alongside the data monitored from sensors to keep respective timestamps, and include 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 Graph (M9)

Secrets: The data structures and algorithms to plot a graph

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

Implemented By: Python

7.3.2 Data Display (M10)

Secrets: The data structures and algorithms to filter and display data on UI, drawn from the database.

Services: This module examines locate database and filters out existing data to display onto the UI as requested by the user.

Implemented By: Python, SQL

7.3.3 Configuration (M11)

Secrets: The data structures and algorithms to take user input and store parameter selection.

Services: This algorithm allows user to preset the device with their own choice of parameters.

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	M1, M4, M6, M8
R2	M4
R3	M4, M6, M7, M8,
R4	M12
R5	M11
R6	M2, M4, M7
R7	M2, M9

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M12
AC2	M2
AC3	M4, M5
AC4	M4, M5, 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.

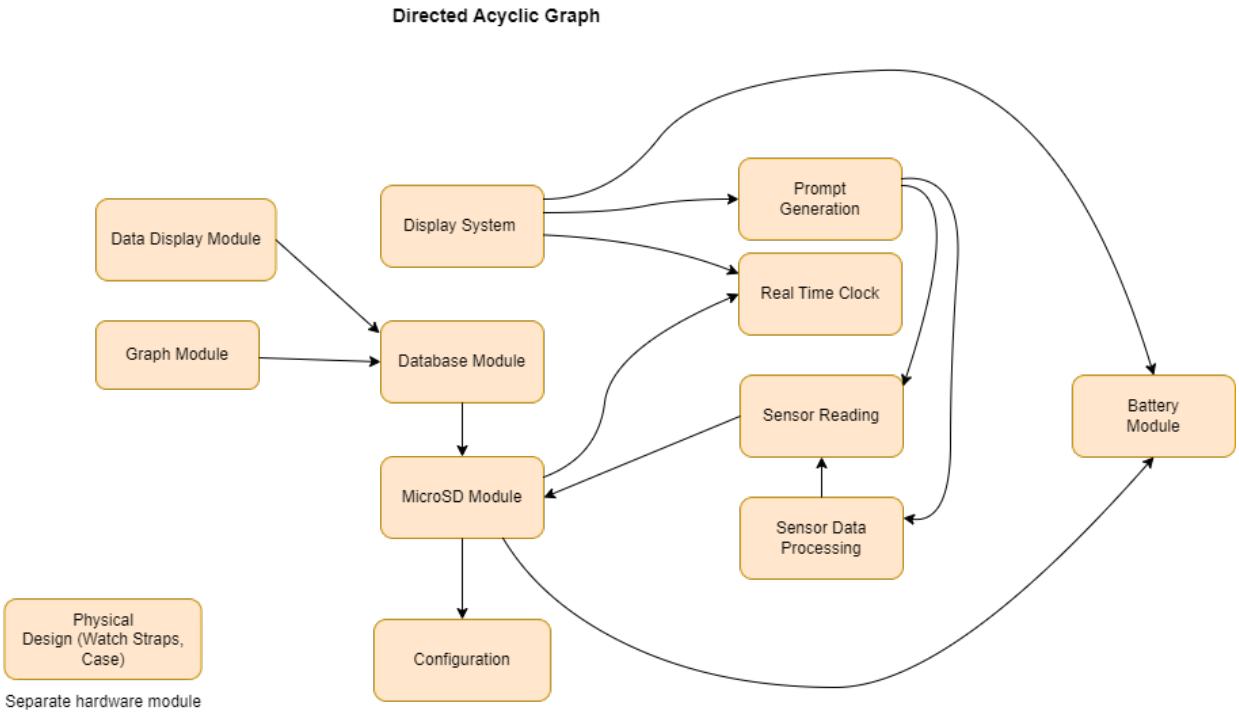


Figure 1: Use Hierarchy Between Modules

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.