Module Guide for RwaveDetection

Junwei Lin

May 8, 2025

1 Revision History

Date	Version	Notes
Mar 17, 2025	1.0	Creation
April 10, 2025	1.1	Modified the dependencies between two modules and added annotations to the Figure 1

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
RMSE	Root Mean Square Error
RwaveDetection	Explanation of program name
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 (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 types of specified IIR filters.

AC3: The input and output format.

AC4: Specific implementation methods of various mathematical functions.

AC5: The log format.

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 specific algorithm to implement R-wave detection task.

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: General Digital Filter Module

M3: Specified IIR Filter Module

M4: Math Module

M5: Pan Tompkins Algorithm Module

M6: Annotated Data RMSE Module

M7: Rwave Detect Module

M8: Input Output Processing Module

M9: Error Handler Module

M10: Logger Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	General Digital Filter Module Specified IIR Filter Module Pan Tompkins Algorithm Module Annotated Data RMSE Module Rwave Detect Module
Software Decision Module	Math Module Input Output Processing Module Error Handler Module Logger 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. *RwaveDetection* means the module will be implemented by the RwaveDetection 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 General Digital Filter Module (M2)

Secrets: The data structure and algorithm for implementing filter difference equations.

Services: Input the original signal and digital filter parameters, perform digital filtering, and return the filtered result.

Implemented By: RwaveDetection

Type of Module: Library

7.2.2 Specified IIR Filter Module (M3)

Secrets: Specific implementation details for different types of IIR filters such as Butterworth filter and Chebyshev filter.

Services: Input the desired filter type, order, cutoff frequency, and other parameters to output a set of usable filter parameters.

Implemented By: RwaveDetection

Type of Module: Library

7.2.3 Pan Tompkins Algorithm Module (M5)

Secrets: The specific implementation of the Pan-Tompkins Algorithm and some finely tuned parameters within the algorithm.

Services: Input a sequence of raw ECG data, process it through the algorithm, and output the index sequence of the predicted R-wave peaks.

Implemented By: RwaveDetection

Type of Module: Library

7.2.4 Annotated Data RMSE Module (M6)

Secrets: Point-to-point matching algorithm between predicted data and annotated data.

Services: Input a sequence of predicted R-wave indices and a sequence of annotated indices, and output the RMSE between two sequences.

Implemented By: RwaveDetection

Type of Module: Library

7.2.5 Rwave Detect Module (M7)

Secrets: The method and order for initializing and calling various internal required modules.

Services: Provides an entry point to the entire program.

Implemented By: RwaveDetection

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 Math Module (M4)

Secrets: The specific implementation methods and algorithms for various mathematical functions.

Services: Provide interfaces for mathematical functions such as square, thresholding, and RMSE calculation.

Implemented By: RwaveDetection

Type of Module: Library

7.3.2 Input Output Processing Module (M8)

Secrets: The specific process of user input and output, the process of converting input into a program-readable format, and the supported input and output formats.

Services: Obtain input readable by other modules, and provide an interface for output to other modules.

Implemented By: RwaveDetection

Type of Module: Library

7.3.3 Error Handler Module (M9)

Secrets: The detailed error handling process, error report format and path.

Services: Provide an interface for unified redirection after an error occurs. When an error is encountered, it will automatically generate a series of reports and terminate the program, facilitating issue reproduction.

Implemented By: RwaveDetection

Type of Module: Library

7.3.4 Logger Module (M10)

Secrets: Definition of log levels and required print format.

Services: Input a string, print it as a log with a timestamp, log level, and other relevant information. It can be saved in real-time.

Implemented By: RwaveDetection

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
R1	M2, M9, M10
R2	M3, M9, M10
R3	M3, M9, M10
R4	M4, M9, M10
R5	M4, M9, M10
R6	M4, M9, M10
R7	M1, M5, M7, M8, M9, M10
R8	M1, M6, M7, M8, M9, M10

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M3
AC3	M8
AC4	M4
AC5	M10

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). The Error Handler module and Logger module with dotted arrows in the Figure 1 indicate that they are referenced by all other modules except themselves and the Hardware-Hiding Module. 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.

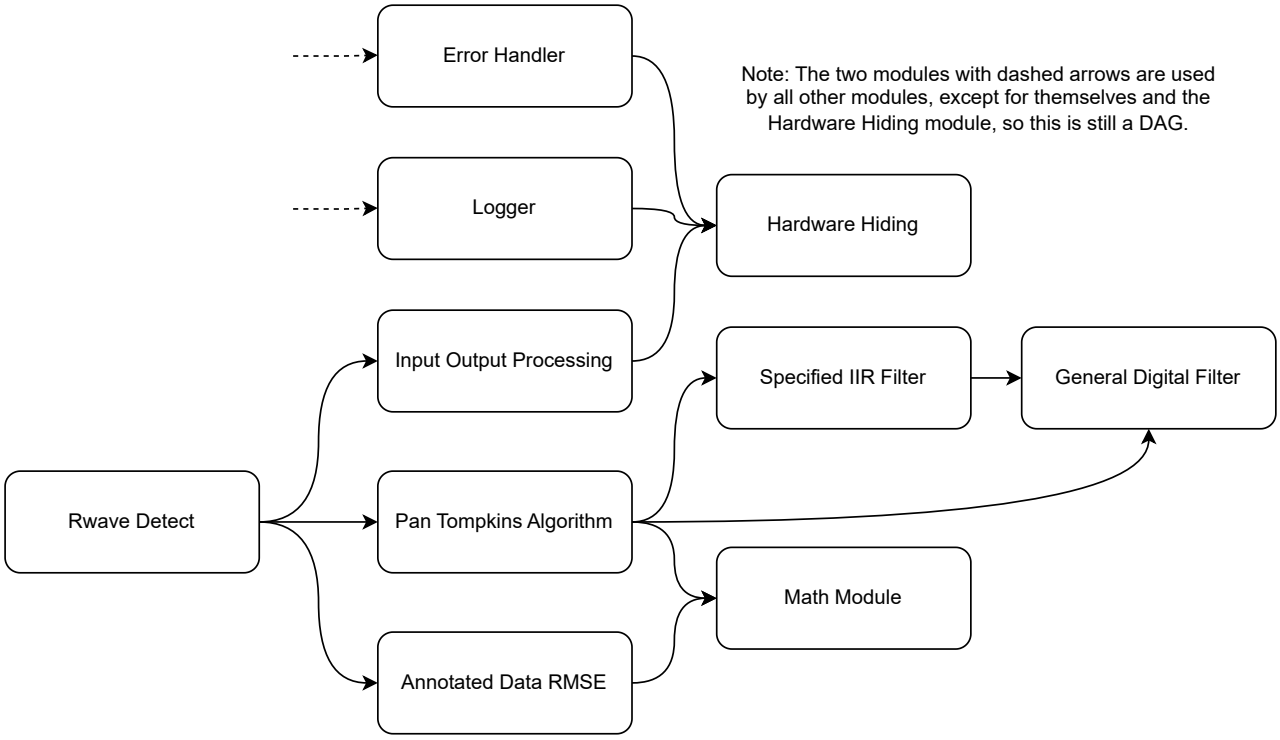

Figure 1: Use hierarchy among modules

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.