

Module Guide for 3D-H3C

Reyhaneh Norouziani

April 16, 2024

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
3D-H3C	Explanation of program name
UC	Unlikely Change

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	Input Format Module (M2)	4
7.2.2	Output Format Module (M3)	5
7.2.3	Output Verification Module (M4)	5
7.2.4	coilT module (M6)	5
7.2.5	Magnetic Core Module (M5)	5
7.2.6	Control Module (M7)	6
7.3	Software Decision Module	6
7.3.1	Vector Module (M8)	6
8	Traceability Matrix	6
9	Use Hierarchy Between Modules	7

List of Tables

1	Module Hierarchy	3
2	Trace Between Requirements and Modules	6
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 (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 laid 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 input data.

AC3: The constraints on the input.

AC4: The format of the result as the output.

AC5: The constraints on the output results.

AC6: How the required currents are calculated.

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: Methods of interaction and data presentation (Inputs via: Keyboard or File; Outputs via: Screen, File, or Memory storage).

UC2: The presence of an external source for the provision of input data to the system is a given.

UC3: The system ensures the visualization or saving of output data.

UC4: The primary objective of this system is to determine the specific currents needed by each coil for generating desired magnetic forces and torques at the center.

UC5: The equations for calculating the required currents can be defined using parameters defined in the input parameters module.

UC6: The magnetic field and force/torque equations can be defined using the parameters defined in the input parameters module.

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 Format Module

M3: Output Format Module

M4: Output Verification Module

M5: Magnetic Core Module

M6: CoilT Module

M7: Control Module

M8: Vector Module

Level 1	Level 2
Hardware-Hiding Module	
	Input Format Module
	Output Format Module
Behaviour-Hiding Module	Output Verification Module
	Magnetic Core Module
	CoilT Module
	Control Module
Software Decision Module	Vector 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. *3D-H3C* means the module will be implemented by the 3D-H3C 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 Input Format Module (M2)

Secrets: The format and structure of the input data, the methods for data entry, and how the data are varified. The processes for loading and verifying inputs are segregated into distinct access programs. The integrated nature of these services, which are executed collectively, implies that the principle of adhering strictly to a single secret per module can be made more flexible in this scenario.

Services: Converts the input data into the data structure used by the Magnetic Core module ([M5](#)).

Implemented By: 3D-H3C

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.2 Output Format Module (M3)

Secrets: The format and structure of the output data

Services: Outputs the currents needed in each pair of coils.

Implemented By: 3D-H3C

7.2.3 Output Verification Module (M4)

Secrets: The module hides the algorithm used for evaluating output data.

Services: It verifies the output result by first checking that the calculated currents remain within acceptable operational thresholds and, secondly, validates the accuracy of the output by comparing it with predefined accuracy benchmarks. If discrepancies are found, whether in terms of accuracy or current values exceeding safe limits, the module throws a warning.

Implemented By: 3D-H3C

7.2.4 coilT module (M6)

Secrets: The data structure of a pair of Coils

Services: Provides comprehensive data on the coil pair's configuration, using the detailed coil parameters defined in the input parameters module.

Implemented By: 3D-H3C

Type of Module: Abstract Object

7.2.5 Magnetic Core Module (M5)

Secrets: The physical constants and algorithm used to implement the magnetic equations to calculate the required currents. Given that there is only one physical constraint, segregating it into a separate module could introduce unnecessary complexity without substantial benefits. Therefore, accommodating two secrets within a single module is considered reasonable in this scenario.

Services: Define the current calculator equations.

Implemented By: 3D-H3C

7.2.6 Control Module (M7)

Secrets: The algorithm for coordinating the running of the program

Services: Provide the main program

Implemented By: 3D-H3C

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 Vector Module (M8)

Secrets: The algorithms for performing a wide range of vector operations

Services: Provides functions executing various vector operations.

Implemented By: NumPy

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

For a detailed overview of the 3D-H3C requirements please refer to the [System Requirements Specification \(SRS\)](#) document.

Req.	Modules
R1	M1, M2
R2	M5, M6, M7, M8
R3	M3
R4	M4, M3

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M2
AC4	M3
AC5	M4
AC6	M5

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.

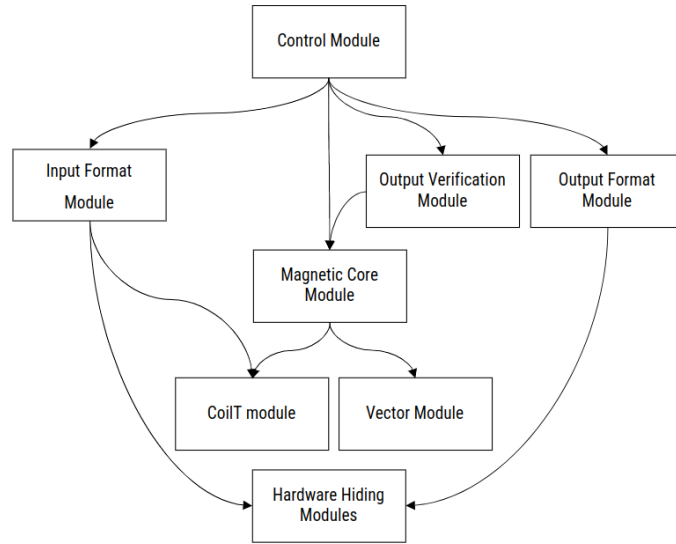


Figure 1: Use hierarchy among modules

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.