Module Guide for ROC

John M Ernsthausen

December 4, 2020

1 Revision History

Date	Version	Notes
22 November 2020	1.0	First submission

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
LC	Likely change
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
ROC	Explanation of program name
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

This document follows the MG Template for our course CSE 741, Development of Scientific Computing Software, taught in the Fall of 2020 by Prof. Spencer Smith. We are permitted to freely use the content of the MG Template.

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. Anticipated changes are labeled by **AC** followed by a number.

AC1: Handle input acquisition.

AC2: Handle output.

AC3: Handle parameter acquisition.

AC4: Algorithms.

AC5: Find the distance to the nearest real pole.

AC6: Find the distance to the nearest complex conjugate pair of poles.

AC7: Find the distance to the nearest pole in hard to resolve case.

AC8: Pole identification, distinguish a real pole from a complex conjugate pair of poles from a complicated situation.

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. Unlikely changes are labeled by **UC** followed by a number.

UC1: The hardware architecture is assumed to be a common personal computer running Ubuntu 20.04 or Windows 2010. The author does not have access to a personal computer running Mac OS, so Mac OS is excluded as well. However, the codes should not be written to exclude the Mac OS hardware architecture. We are not writing for graphic card architectures or other high performance computing clusters.

UC2: We assume that the inputs do not represent entire functions such as the exponential function.

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. Modules are labeled by **M** followed by a number.

M1: Input module.

M2: Output module.

M3: Parameter module.

M4: Solver module.

M5: Real pole module.

M6: Complex conjugate pair of poles module.

M7: Resolve nearest pole in hard to resolve case module.

M8: Pole identification module.

I'm surprised to see these. I thought your program was a library

those names do not match the names in the uses 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.

Level 1	Level 2
Hardware-Hiding Module	Input Format Module Output Format Module
Behaviour-Hiding Module	Input Format Module Output Format Module Parameters Module Real pole module Complex conjugate pair of poles module Resolve nearest pole in hard to resolve case module Pole identification module
Software Decision Module	Solver 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. *ROC* means the module will be implemented by the ROC 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

7.1.1 Input Format Module (M1)

Secrets: Handle input acquisition via hardware.

Services: Read files from hard drive. This module provides the interface between the hardware and the software. So, the system can use it to accept inputs.

Implemented By: OS

7.1.2 Output Format Module (M2)

Secrets: Handle output via hardware.

not phrased as a secret

Services: Write files to hard drive. This module provides the interface between the hardware and the software. So, the system can use it to display and save outputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

7.2.1 Input Format Module (M1)

Secrets: The format and structure of the input data.

Services: Validate input format. Validate input type. Handle input acquisition via software.

Implemented By: ROC

Type of Module: Library

7.2.2 Output Format Module (M2)

Secrets: The format and structure of the output.

Services: Handle output format. Handle output type. Handle output via software.

Implemented By: ROC

Type of Module: Library

7.2.3 Parameters Module (M3)

I'm looking at your code and I do not see a module for this.

Secrets: Handle all aspects of parameters.

Services: This module handles parameter acquisition, format, type, distribution, and constraints.

Is this really a module you intend to have?

Implemented By: ROC

Type of Module: Record

7.2.4 Real pole module (M5)

Secrets: Implements solver for real poles.

Services: Computes R_c and μ for real poles.

Implemented By: ROC

Type of Module: Library

7.2.5 Complex conjugate pair of poles module (M6)

Secrets: Implements solver for complex conjugate pair of poles.

Services: Computes R_c and μ for complex conjugate pair of poles.

Implemented By: ROC

Type of Module: Library

7.2.6 Resolve nearest pole in hard to resolve case module (M7)

Secrets: Implements solver to find the distance to the nearest pole in hard to resolve case.

Services: Computes R_c for hard to resolve case.

Implemented By: ROC

Type of Module: Library

7.2.7 Pole identification module (M8)

Secrets: Implements pole identification that distinguishes a real pole from a complex conjugate pair of poles from a complicated situation.

Services: Decides which solver is best suited to resolve the inputs.

Implemented By: ROC

Type of Module: Library

7.3 Software Decision Module (M4)

Secrets: Algorithms.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user. Algorithm to find the distance to the nearest real pole. Algorithm to find the distance to the nearest complex conjugate pair of poles. Algorithm to find the distance to the nearest pole in hard to resolve case.

Implemented By: ROC

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	M1
R2	M1
R3	M1
R4	M1
R5	M1
R6	M1
R7	M2
R8	M2
R9	M2
R10	M2
R11	M3
R12	M3
R13	M3
R14	M3
R15	M3
R16	M4
R17	M4
R18	M4
R19	M1, M2, M3, M4, M5, M8
R20	M1, M2, M3, M4, M6, M8
R21	M1, M2, M3, M4, M7, M8
R22	M1, M2, M8
R23	M1, M2,M3, M4, M5, M6, M7, M8
R24	M1, M2, M3, M4, M5, M8
R25	M1, M2, M3, M4, M6, M8
R26	M1, M2, M3, M4, M7, M8
R27	M4

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

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.

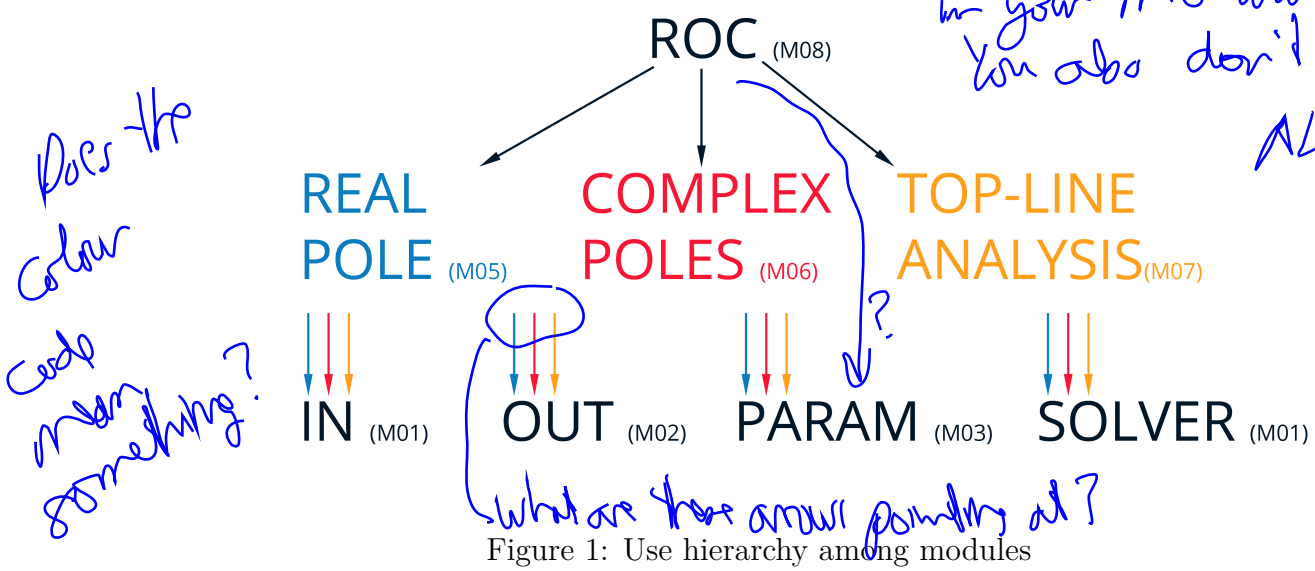

Figure 1: Use hierarchy among modules

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.