Module Guide for MPIR

Xunzhou (Joe) Ye

April 14, 2025

1 Revision History

Date	Version	Notes
19 March 2025	1.0	Initial draft
14 April 2025	1.1	Refine according to feedbacks

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
MP	Mixed-Precision
IR	Iterative Refinement
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 D. Parnas, Clement, and Weiss 1984. We advocate a decomposition based on the principle of information hiding D. L. 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 D. Parnas, Clement, and Weiss 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 D. Parnas, Clement, and Weiss 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 assumed properties that the input matrix holds, which further determines the solver/algorithm used to factorize the matrix.

AC3: The required properties of the user-defined mixed-precision configurations.

AC4: The implementation of common matrix-vector operations in arbitrary precisions.

AC5: The algorithm used for internal solves during the refinement steps.

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 underlying mathematical method of the solver follows the iterative refinement scheme.

UC2: The goal of the system is to solve a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

UC3: The input matrix is non singular. In another words, the problem given to the solver is meaningful in a mathematical sense.

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 [1]

M2: Factorization Module

M3: Floating Point Concepts Module

M4: Matrix Operations Module

M5: Iterative Solver Module

Level 1	**Level 2**
Hardware-Hiding Module	–
Behaviour-Hiding Module	Floating Point Concepts Module Matrix Operations Module Factorization Module Iterative Solver Module
Software Decision 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 D. Parnas, Clement, and Weiss 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. *MPIR* means the module will be implemented by the MPIR 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.

[1] MPIR is intended to run on any general purposed operation system (OS). M1 is assumed to be by implemented by the OS.

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 Factorization Module (M2)

Secrets: The structure of the input matrix, how to factorize the input matrix.

Services: Obtains factors of a given input matrix, provides routines to solve for different vectors given the same (factorized) input matrix.

Implemented By: MPIR

Type of Module: Library

7.2.2 Floating Point Concepts Module (M3)

Secrets: Internal details of the user-defined mixed-precision configuration, specifically the types selected for various stages of solver arithmetic.

Services: Validates the user-defined mixed-precision configuration.

Implemented By: MPIR

Type of Module: Generic Interface [2]

[2] The "mixed-precision configuration" refers to a set of type parameters used for arithmetic operations within the solver. This module provides meta-programming constructs to constrain and validate the allowable type combinations.

7.2.3 Matrix Operations Module (M4)

Secrets: N/A

Services: Provides routines for basic matrix operations, including matrix-vector multiplication, vector-vector multiplication, vector dot product, vector Euclidean norm, etc.

Implemented By: MPIR

Type of Module: Library

7.2.4 Iterative Solver Module (M5)

Secrets: The structure of the input matrix, content of the input matrix, factors of the input matrix

Services: Provides the main logics of the solver. Solves a given linear system with iterative refinement.

Implemented By: MPIR

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: –

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, M5
R2	M3, M4
R3	M2

Table 2: Trace Between Requirements and Modules

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

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. D. L. 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.

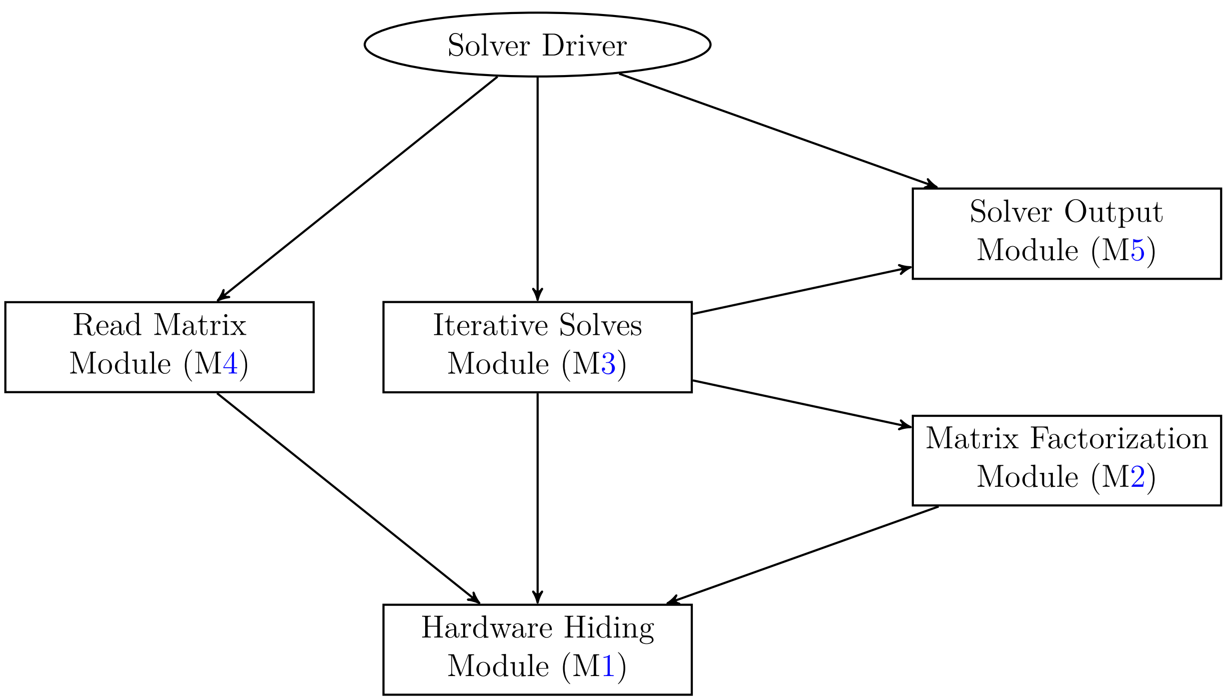

Figure 1: Use hierarchy among modules

10 User Interfaces

N/A

11 Design of Communication Protocols

N/A

12 Timeline

N/A

References

Parnas, D.L., P.C. Clement, and D. M. Weiss (1984). "The modular structure of complex systems". In: *International Conference on Software Engineering*, pp. 408–419.

Parnas, David L. (Dec. 1972). "On the Criteria To Be Used in Decomposing Systems into Modules". In: *Comm. ACM* 15.2, pp. 1053–1058.

— (1978). "Designing Software for Ease of Extension and Contraction". In: *ICSE '78: Proceedings of the 3rd international conference on Software engineering.* Atlanta, Georgia, United States: IEEE Press, pp. 264–277. ISBN: none.