Module Guide for Damped Harmonic Oscillator

Muhammad Waqar Ul Hassan Awan

March 19, 2024

1 Revision History

Date	Version	Notes
18 Mar, 2024	1.0	Initial MG documentation

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
SRS	Software Requirement Specification
ODE	Ordinary Differential Equation
DE	Differential Equation
DHO	Damped Harmonic Oscillator
SHM	Simple Harmonic Motion

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 format of the initial input data.

AC2: The format of the input parameters.

AC3: The constraints on the input parameters.

AC4: The format of the final output data.

AC5: As user feedback is collected and as the software finds use in broader contexts, adjustments to the user interface may be necessary to enhance usability, accessibility, and user experience.

AC6: How the plotting of the output is implemented.

AC7: The overall control of the calculation.

AC8: Future developments might necessitate the inclusion of more complex systems such as coupled oscillators or multidimensional oscillatory systems.

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).

UC2: The fundamental physics and mathematics governing harmonic oscillators are well-established. Significant changes to these principles are unlikely and would fundamentally alter the nature of the software.

UC3: The selection of the programming language and core libraries is based on their ability to handle the computational requirements and maintainability of the software. While updates and minor changes to dependencies are expected, a complete shift is unlikely due to the substantial effort involved in rewriting and revalidating the software.

UC4: The selection of the programming language and core libraries is based on their ability to handle the computational requirements and maintainability of the software. While updates and minor changes to dependencies are expected, a complete shift is unlikely due to the substantial effort involved in rewriting and revalidating the software.

UC5: The software is designed as a specific solution to modeling harmonic oscillators with damping. Changing its operational mode from a standalone application to an integrated module within a larger system, or vice versa, would require a significant architectural change.

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 numbered by M followed by a number.

M1: Hardware-Hiding Module

M2: Data Input Module

M3: Parameter Input Module

M4: Calculation Engine Module

M5: Output Module

M6: User Interface Module

M7: Plotting and Visualization Module

M8: ODE Solver Module

Note that M1 is a commonly used module and is already implemented by the operating system. It will not be reimplemented. Similarly, M7 and M8 are already available in Python/React and will not be reimplemented.

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	M1
Behaviour-Hiding Module	M2
	M3
	M4
	M5
	M6
Software Decision Module	M7
	M8

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. *Damped Harmonic Oscillator* means the module will be implemented by the Damped Harmonic Oscillator 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 Data Input Module (M2)

Secrets: The format and validation rules for the data entered by the user.

Services: Validates and transforms user input into a standardized format for processing by the calculation engine. Ensures that all inputs meet the constraints and formats expected by the system.

Implemented By: Damped Harmonic Oscillator

Type of Module: Library

7.2.2 Parameter Input Module (M3)

Secrets: The structure and validation criteria for simulation parameters.

Services: Processes and validates the parameters specific to the damped harmonic oscillator simulation, such as mass, spring constant, damping coefficients, etc.

Implemented By: Damped Harmonic Oscillator

Type of Module: Library

7.2.3 Calculation Engine Module (M4)

Secrets: The algorithms that implement the mathematical models of the damped harmonic oscillator.

Services: Performs the core calculations based on input parameters and models, generating the dynamics of the oscillator over time.

Implemented By: Damped Harmonic Oscillator

Type of Module: Library

7.2.4 Output Module (M5)

Secrets: The data structure and algorithms used to format simulation results for output.

Services: Transforms calculation results into various output formats (e.g., graphical, textual) for user interpretation.

Implemented By: Damped Harmonic Oscillator

Type of Module: Library

7.2.5 User Interface Module (M6)

Secrets: The design and logic for the software's user interface.

Services: Provides the graphical or command-line interface through which users interact with the software, including inputting parameters and viewing results.

Implemented By: Damped Harmonic Oscillator

Type of Module: Library

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 Plotting and Visualization Module (M7)

Secrets: The algorithms and libraries used to generate visual representations of the simulation results.

Services: Creates charts, graphs, and animations to visually depict the behavior of the damped harmonic oscillator based on the simulation outcomes.

Implemented By: ReactJS

Type of Module: Library

7.3.2 ODE Solver Module (M8)

Secrets: The numerical methods and algorithms for solving ordinary differential equations (ODEs) that describe the motion of the damped harmonic oscillator.

Services: Provides the computational backbone for accurately solving the equations of motion under various conditions and damping models.

Implemented By: Python

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, M3
R2	M6
R3	M4, M5
R4	M4
R5	M5, M7
NFR1	M4
NFR1	M6
NFR1	Depends on the nature of the change
NFR1	M1

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M2
AC2	M3
AC3	M3
AC4	M5
AC5	M6
AC6	M7
AC7	M4
AC8	M4

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.

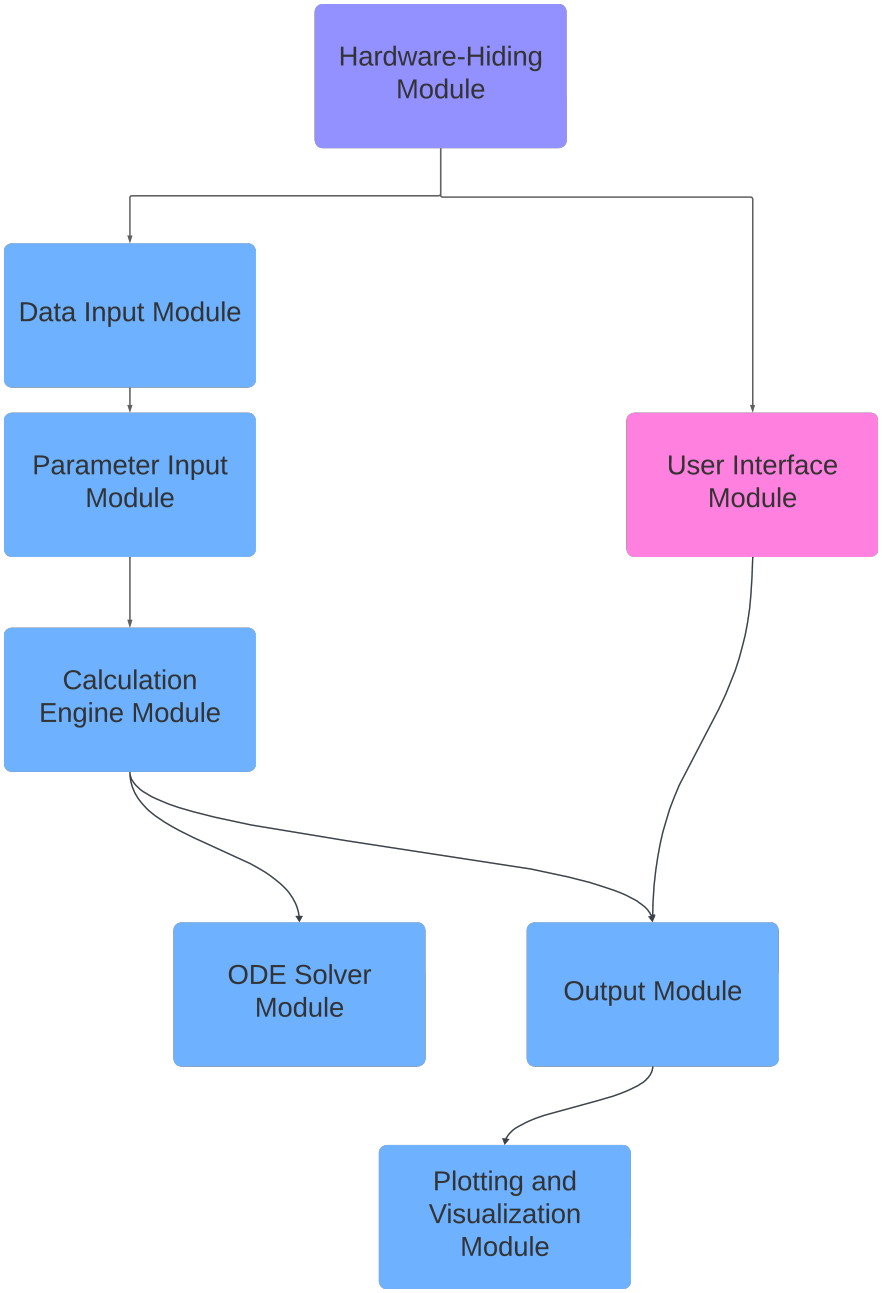

Figure 1: Use hierarchy among modules

10 User Interfaces

Following is a tentative design for the Damped harmonic oscillator simulation.

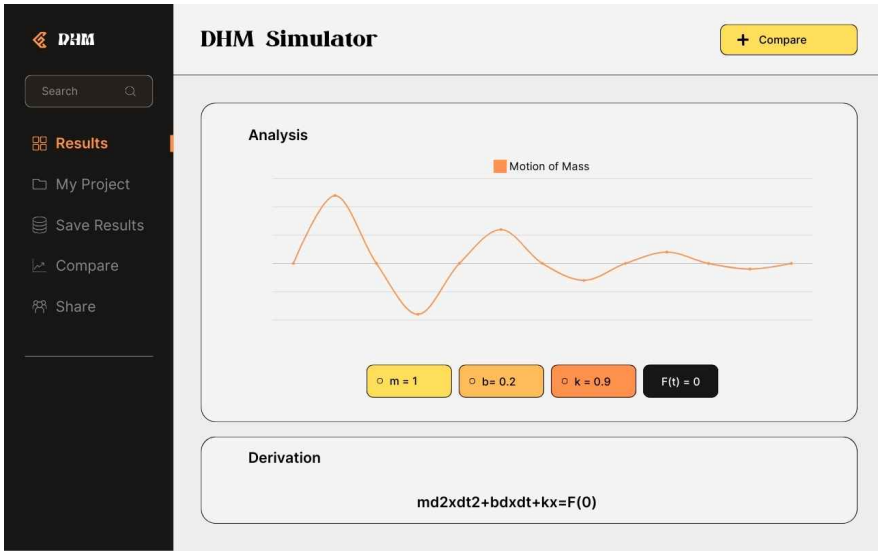

Figure 2: Tentative Design

11 Timeline

Table displays the timeline for the development of each module and its respective developer.

Modules	**Time**	**Responsible**
Frontend	20 March - 03 April	Waqar
Backend	24 March - 03 April	Waqar

Table 4: Timeline

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.