Module Guide for SynthEddy

Phil Du (Software)
Nikita Holyev (Theory)

April 15, 2024

1 Revision History

Date	Version	Notes
2024-03-18	1.0	Initial MG
2024-03-21	1.1	Feedbacks by domain expert addressed

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
CFD	Computational Fluid Dynamics
DAG	Directed Acyclic Graph
GUI	Graphical User Interface
I/O	Input/Output
M	Module
MG	Module Guide
NFR	Non-Functional Requirement
NumPy	Python Package for Scientific Computing
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
SynthEddy	Synthetic Turbulent Flow Generator
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: User can input some hyperparameters to generate realistic eddy profiles instead needing manual profile input. [SRS: LC1].

AC2: Allow generating internal flow instead of external [SRS: LC2].

AC3: Adding more query protocols/endpoints for integration with CFD software [SRS: R3].

AC4: Adding shape functions to choose from, potentially done by user [SRS: NFR3].

AC5: Output 3D or 2D cross-section visualization of flow field.

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: Work with 2D flow field instead of 3D [SRS: LC3].

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: Main Control Module

M3: Query Interface

M4: Eddy Profile Module

M5: Flow Field Module

M6: Eddy Module

M7: Shape Function Module

M8: File I/O Module

M9: Vector Module

M10: Visualization Module (Placeholder)

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Main Control Module Query Interface Eddy Profile Module Flow Field Module Eddy Module Shape Function Module
Software Decision Module	File I/O Module Vector Module Visualization 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. *SynthEddy* means the module will be implemented by the SynthEddy 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 Main Control Module (M2)

Secrets: Overall flow of the program

Services: Taking command line arguments, initializing the system, and calling appropriate modules.

Implemented By: SynthEddy

Type of Module: Abstract Object

7.2.2 Query Interface (M3)

Secrets: Query format and structure.

Services: Provide endpoints for manual and automated query. Unpack query and pass to Flow Field Module for processing. Serialize and return response.

Implemented By: SynthEddy

Type of Module: Abstract Object

7.2.3 Eddy Profile Module (M4)

Secrets: Algorithm to generate eddy profiles.

Services: Generate physically realistic eddy profile based on hyperparameters (not implemented currently) or load existing profile.

Implemented By: SynthEddy

Type of Module: Abstract Data Type

7.2.4 Flow Field Module (M5)

Secrets: Mechanisms to ensure conservation of mass within flow field.

Services: Initialize flow field. Compute velocity vector at any given point in the flow field at any given time.

Implemented By: SynthEddy

Type of Module: Abstract Data Type

7.2.5 Eddy Module (M6)

Secrets: Mathematical model of velocity around an eddy center, given a shape function.

Services: Compute the velocity vector at any given point relative to the eddy center.

Implemented By: SynthEddy

Type of Module: Abstract Data Type

7.2.6 Shape Function Module (M7)

Secrets: Shape function equations

Services: Providing a list of shape functions to choose from.

Implemented By: SynthEddy

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 File I/O Module (M8)

Secrets: File format and structure

Services: Read and write to file for persistent eddy profiles and flow fields between program runs.

Implemented By: SynthEddy

Type of Module: Abstract Object

7.3.2 Vector Module (M9)

Secrets: Algorithms for fast vector operations

Services: Common vector/matrix operations.

Implemented By: NumPy

Type of Module: Abstract Data Type

7.3.3 Visualization Module (M10)

Secrets: Placeholder, no consideration yet.

Services: Render visualization of flow field.

Implemented By: SynthEddy

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	M3, M5
R2	M5, M6
R3	M3
NFR1	M4, M5
NFR2	M2, M3
NFR3	M7
NFR4	M1

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M4
AC2	M5
AC3	M3
AC4	M7
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). 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.

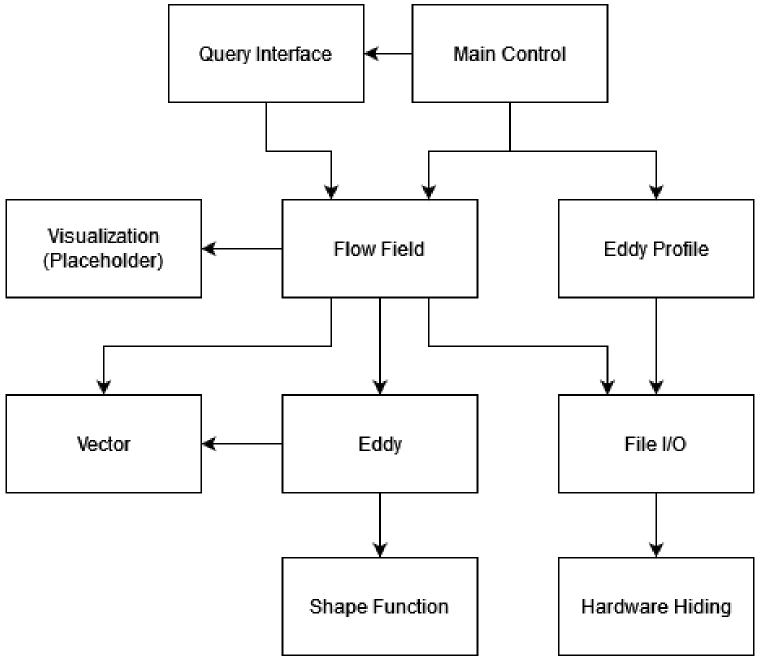

Figure 1: Use hierarchy among modules

10 User Interfaces

N/A (No GUI)

11 Design of Communication Protocols

Potentially use TCP socket or HTTP (RESTful API) for AC3. This will largely dependent on the CFD side.

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.