Module Guide for SFWRENG 4G06 - Capstone Design Process

Team 17, DomainX

Awurama Nyarko
Haniye Hamidizadeh
Fei Xie
Ghena Hatoum

November 5, 2025

1 Revision History

Table 1: Revision History

Date	**Developer(s)**	**Change**
Nov 3, 2025	Fei	Rev -1

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
SFWRENG 4G06 - Capstone Design Process	Software Engineering Capstone Project
UC	Unlikely Change
AHP	Analytical Hierarchy Process
BWM	Best-Worst Method
SSB	Skew-Symmetric Bilinear
Domain	Research Software Domain
Packages	Software Packages
API	Application Programming Interface
ADT	Abstract Data Type

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 **??** 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 8 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 metrics used to assess a domain's state of practice may be updated (e.g. Addition of new metrics, such as code coverage percentage)

AC2: The system should support extension to user authentication (e.g. Using two-factor authentication on top of username and password)

AC3: The ranking algorithm used within a domain for packages may be changed (e.g. Alternatives to AHP, such as BWM, SSB, etc)

AC4: The comparison algorithms used between domains may be changed

AC5: The user access roles available might be expanded (e.g. Admin, User, Contributor)

AC6: Additional languages might be supported (e.g. French, Simplified Chinese)

AC7: The APIs used to extract repository metrics

AC8: The visualization libraries used to display and export data

AC9: Additional export and import file formatting may be added (e.g. CSV, Excel)

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 schema structure of our domain and metrics systems are designed for long-term use and allows for the addition of new metric types

UC2: The platform of the tool will remain as a web application

UC3: The development stack of our web application will remain the same (React, Django)

UC4: Input types of existing metrics, as outlined in the Methodology paper

UC5: Popular browser compatibility is expected to remain unchanged (e.g. Chrome, Firefox, Edge)

UC6: Tool owner, Dr. Spencer Smith, is not expected to change for the duration of the tool's lifespan

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware Hiding Module

M2: Browser Module

M3: Domain View Module

M4: Data Edit Module

M5: User Authentication Module

M6: User Role Access Module

M7: Automated Metrics Module

M8: Localization Module

M9: Domain Management Module

M10: System API Gateway Module

M11: Ranking Algorithm Module

M12: Graphing Module

M13: File Import Module

M14: File Export Module

M15: Repository API Module

M16: Domain Comparison Module

M17: Database Persistence Module

Level 1	**Level 2**
Hardware-Hiding Module	Browser Module
Behaviour-Hiding Module	Domain View Module
	Data Edit Module
	User Authentication Module
	User Role Access Module
	Automated Metrics Module
	Localization Module
	Domain Management Module
Software Decision Module	System API Gateway Module
	Ranking Algorithm Module
	Graphing Module
	File Import Module
	File Export Module
	Repository API Module
	Domain Comparison Module
	Database Persistence Module

Table 2: 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 4.

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. *SFWRENG 4G06 - Capstone Design Process* means the module will be implemented by the SFWRENG 4G06 - Capstone Design Process 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

7.1.1 Browser Module (M2)

Secrets: The data structure and algorithm used to implement the browser, which is outside of the scope of the project.

Services: The browser allows all users of the product to view and retrieve the project through the internet, and displays the contents for use.

Implemented By: Web browser

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 Domain View Module (M3)

Secrets: The data structures and algorithms to retrieve and display domain metrics and packages.

Services: Displays domain content, including it's corresponding packages and metrics.

Implemented By: DomainX

Type of Module: ADT

7.2.2 Data Edit Module (M4)

Secrets: The data structures and algorithms to edit and update metrics of a domain.

Services: Handles and validates user inputs of domain/package/metric data. Provides user feedback on errors (invalid type, missing field, etc)

Implemented By: DomainX

Type of Module: ADT

7.2.3 User Authentication Module (M5)

Secrets: The data structures and algorithms used to securely store, validate and manage user credentials.

Services: Provides user registration, login, and session management services.

Implemented By: DomainX

Type of Module: Library

7.2.4 User Role Access Module (M6)

Secrets: The data structures and algorithms used to store and validate users access level within the system.

Services: Provides user role and capabilities related to the role.

Implemented By: DomainX

Type of Module: ADT

7.2.5 Automated Metrics Module (M7)

Secrets: The data structures and algorithms used for retrieving automatable metrics.

Services: Handles any automated data entry into system, uses Repository API module and File Import Module to automate data input.

Implemented By: DomainX

Type of Module: Library

7.2.6 Localization Module (M8)

Secrets: The language files, translation mappings, and locale formatting mechanisms.

Services: Provides translation and formatting of UI text and date/number formats.

Implemented By: react-i18next

Type of Module: Library

7.2.7 Domain Management Module (M9)

Secrets: The data structures and algorithms that handles the creation and publication of domains.

Services: Provides versioning and publication of completed domains.

Implemented By: DomainX

Type of Module: ADT

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 System API Gateway Module (M10)

Secrets: The business logic that coordinates the flow of communication between modules, and the frontend, backend, and database.

Services: Manages application state and serves as the central communication hub for the system

Implemented By: React, Django

Type of Module: Abstract Object

7.3.2 Ranking Algorithm Module (M11)

Secrets: Ranking algorithms and weights (AHP, BWM, SSB).

Services: Computes comparative rankings using configurable methods.

Implemented By: AHPy, DomainX

Type of Module: Abstract Object

7.3.3 Graphing Module (M12)

Secrets: The data structures and algorithms used for graphing data.

Services: Provides the functions related to graphing.

Implemented By: Matplotlib

Type of Module: Abstract Object

7.3.4 File Import Module (M13)

Secrets: The data structures and algorithms used to import data of varying formats (e.g. Excel, CSV).

Services: Facilitates the process of processing a inputted file into data for use by the rest of the system.

Implemented By: pandas

Type of Module: Abstract Object

7.3.5 File Export Module (M14)

Secrets: The data structures and algorithms used to export data in varying formats (e.g. Excel, CSV).

Services: Facilitates the process of transforming system data into an exportable format (e.g. Excel, CSV).

Implemented By: pandas

Type of Module: Abstract Object

7.3.6 Repository API Module (M15)

Secrets: API endpoints, tokens, and rate limit strategies.

Services: Fetches metrics and metadata from external repositories (e.g., GitHub, GitLab).

Implemented By: DomainX

Type of Module: Abstract Object

7.3.7 Domain Comparison Module (M16)

Secrets: The algorithms and parameters used for comparing domains.

Services: Computes and visualizes comparisons across multiple domains.

Implemented By: DomainX

Type of Module: Abstract Object

7.3.8 Database Persistence Module (M17)

Secrets: The algorithms and parameters used for interacting with the database.

Services: Provides database methods related to updating and querying the stored data. As well as the connection to the database itself.

Implemented By: MySQL

Type of Module: Abstract Object

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

Req.	**Modules**
FR1	M6
FR2	M5, M6
FR3	M5
FR4	M5
FR5	M4, M17
FR6	M9
FR7	M4, M6, M17
FR8	M4
FR9	M4, M6, M7, M13
FR10	M7, M13, M15
FR11	M3, M17
FR12	M11, M3
FR13	M12
FR14	M14
FR15	M12, M14

Table 3: Trace Between Functional Requirements and Modules

Req.	Modules
LF-AR1	M??, M??, M??, M??
LF-AR2	M??, M??
LF-AR3	M??
LF-AR4	M??, M??
LF-AR5	M??, M??, M??, M??, M??, M??
LF-AR6	M??, M??, M??, M??, M??, M??
LF-SR1	M??, M??, M??, M??, M??
LF-SR2	M??, M??, M??, M??, M??
LF-SR3	M??
UH-EU1	M??, M??, M??
UH-EU2	M??, M??, M??, M??
UH-EU3	M??, M??, M??, M??
UH-LR1	M??, M??, M??, M??
UH-LR2	M??, M??, M??, M??
UH-UP1	M??, M??, M??, M??
UH-UP2	M??, M??, M??, M??
UH-UP3	M??, M??, M??, M??
UH-AR1	M??, M??, M??, M??
UH-AR2	M??, M??, M??, M??
PR-SL1	M??, M??, M??, M??
PR-SC1	M??, M??, M??, M??
PR-SC2	M??, M??, M??, M??
PR-SC3	M??, M??, M??, M??
PR-PA1	M??, M??, M??, M??
PR-RFT1	M??, M??, M??, M??
PR-RFT2	M??, M??, M??, M??
PR-RFT3	M??, M??, M??, M??
PR-CR1	M??, M??, M??, M??
PR-CR2	M??, M??, M??, M??
PR-CR3	M??, M??, M??, M??
PR-SE1	M??, M??, M??, M??
PR-SE2	M??, M??, M??, M??
PR-LR1	M??, M??, M??, M??
PR-LR2	M??, M??, M??, M??
OE-EPE1	M??, M??, M??, M??
OE-EPE2	M??, M??, M??, M??
OE-EPE3	M??, M??, M??, M??
OE-WE1	M??, M??, M??, M??
OE-WE2	M??, M??, M??, M??
OE-WE3	M??, M??, M??, M??

AC	**Modules**
AC1	M3, M4, M17
AC2	M5
AC3	M11
AC4	M16
AC5	M6
AC6	M??
AC7	M15
AC8	M12
AC9	M13, M14

Table 5: Trace Between Anticipated Changes and Modules

8 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.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

Figure 1: Use hierarchy among modules

9 User Interfaces

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS]

10 Design of Communication Protocols

[If appropriate —SS]

11 Timeline

[Schedule of tasks and who is responsible —SS]

[You can point to GitHub if this information is included there —SS]

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.