Module Guide for Software Engineering

Team 15, ASLingo
Andrew Kil
Cassidy Baldin
Edward Zhuang
Jeremy Langner
Stanley Chan

January 10, 2024

1 Revision History

Date	Version	Notes
Jan. 9, 2024	1.0	Andrew and Edward; added module hierarchy
Jan. 10, 2024	1.1	Stanley; added some module decompositions for hardware hiding and behaviour hiding modules
Date 10, 2024	1.2	Andrew; rearranged Module Hierarchy and added decomposition descriptions for controller, hand sign recognition and verification modules
Date 2	1.1	Notes

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
Software Engineering	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

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 (**?**). We advocate a decomposition based on the principle of information hiding (**?**). 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 **?**, 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 (**?**). 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 format of the initial input data.

...

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).

...

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: Hand Sign Recognition Module

M2: Hand Sign Verification Module

M3: Controller Module

M4: Data Collection Module

M5: Data Processing Module

M6: Machine Learning Module

M7: Testing and Verification Module

M8: Video Input Module

Level 1	**Level 2**
Hardware-Hiding Module	Video Input Module
Behaviour-Hiding Module	Controller Module Data Processing Module Machine Learning Module Hand Sign Recognition Module
Software Decision Module	Hand Sign Verification Module Data Collection Module Testing and Verification 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 **?**. 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. *Software Engineering* means the module will be implemented by the Software Engineering 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 (M??)

7.1.1 Video Input Module (M8)

Secrets: Image processing algorithms, software interaction with webcam hardware.

Services: Outputs real-time video data through user's webcam.

Implemented By: OpenCV and OS

7.2 Behaviour-Hiding Module

7.2.1 Hand Sign Recognition Module (M1)

Secrets: Recognises hand signs are being made

Services: Relays the information that a hand sign is made

Implemented By: Python and Pytorch

7.2.2 Controller Module (M3)

Secrets: Able to relay necessary information from back-end component to the correct corresponding front-end component and vice versa

Services: Handles communication between front-end components and back-end components

Implemented By: Python and JavaScript

7.2.3 Data Processing Module (M5)

Secrets: Algorithm used to process collected data.

Services: Interprets collected data accordingly.

Implemented By: Python

7.2.4 Machine Learning Module (M6)

Secrets: Training data sets and structure of neural network

Services: Takes in hand coordinate data and interprets and processes the data to return the appropriate letter.

Implemented By: Python and Pytorch

7.3 Software Decision Module

7.3.1 Hand Sign Verification Module (M2)

Secrets: The handshake used to determine whether the sign interpreted by the Machine Learning Module matches with the sign requested by front-end components

Services: Returns a pass/fail to front-end based on result of match attempt

Implemented By: Python

7.3.2 Data Collection Module (M4)

Secrets: The data and structure of data collected.

Services: Stores collected data to be retrieved at a later time.

Implemented By: Python

7.3.3 Testing and Verification Module (M7)

Secrets: Unit test cases used to test the software system.

Services: Verifies the software works as intended by testing the system on various test cases which ensures robustness, accuracy, and reliability.

Implemented By: Python and Pytest

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	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M??
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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