Module Guide for Software Engineering

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

January 17, 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
Jan. 10, 2024	1.2	Andrew; rearranged Module Hierarchy and added decomposition descriptions for controller, hand sign recognition and verification modules
Jan. 11, 2024	1.3	Jeremy and Cassidy added front end anticipated changes, modules and traceability mapping to these modules
Jan. 12, 2024	1.4	Stanley; added some back end anticipated changes, more traceability mapping for functional requirements and modules
Jan 13, 2024	1.5	Andrew; Added Uses Hierarchy Diagram
Jan 15, 2024	1.6	Andrew; Cleaned up minor errors
Jan 17, 2024	1.7	Jeremy; Replaced Login/Signup module with Authentication module. Added in Andrews updates UsesHierarchy image
Jan 17, 2024	1.8	Cassidy; Added in Reference Material, Abbreviations, and Reflection Template

2 Reference Material

This section records information for easy reference.

The Development Plan outlines the roles of each team member and the areas that each member will focus on. This breakdown of team responsibilities allows the team to assign testing roles accordingly. This document also contains the tools that the team plans on using for testing.

The Software Requirements Specification lists the functional and non-functional requirements which will aid in testing by formulating a testing plan to meet each requirement. Non-functional requirements should be tested such that the fit criteria are met.

The Hazard Analysis identifies failure modes to determine the implementation strategies to mitigate them. These will be used as a part of the testing plan to ensure that the failures are covered.

The Module Interface Specification further decomposes the software's modules into specific access routines. The team will build the testing plan such that each function and routine works as intended.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
ASL	Shorthand for American Sign Language. It is a form of sign language primarily used in the US and in parts of Canada.
ASLingo	The commercial name for the project.
CV	Refers to Computer Vision, the field of technology that invloves processing visual input ot achieve various means.
HSR	Shorthand for "Health and Safety Requirements", a subsection of Non-Functional Requirements.
FR	Shorthand for Functional Requirements.
LR	Shorthand for "Legal Requirements", a subsection of Non-Functional Requirements.
LFR	Shorthand for "Look and Feel Requirements", a subsection of Non-Functional Requirements.
MSR	Shorthand for "Maintainability and Support Requirements", a subsection of Non-Functional Requirements.
OER	Shorthand for "Operational and Environmental Requirements", a subsection of Non-Functional Requirements.
OpenCV	Refers to the Open Computer Vision Library library available for free to developers in order to develop Computer Vision applications.
M	Module
MG	Module Guide
PR	Shorthand for "Performance Requirements", a subsection of Non-Functional Requirements.
SR	Shorthand for "Security Requirements", a subsection of Non-Functional Requirements.
SRS	Software Requirements Specification
UC	Unlikely Change
UHR	Shorthand for "Usuability and Humanity Requirements", a subsection of Non-Functional Requirements.

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 ASL vocabulary included in the program may be updated in the future as the language evolves.

AC2: The Landing Page Module may change if the developers want to change the formatting of the informational pages on the website (for example the home/resources page that use this module).

AC3: Exercise and Exercise Selection/History module may change if developers want to add in additional testing methods or material.

AC4: Account management module may change if developers change what information is stored about the users progress.

AC5: Structure of neural network in Machine Learning module may change to improve performance and accuracy.

AC6: Test cases in the Testing and Verification module may change throughout development to improve code coverage as well as to cover any previously mentioned anticipated changes.

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 Authentication Up module is unlikely to change as we will be using user authentication methods and standard login procedures that are unlikely to 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.

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

M9: Landing Page Module

M10: Exercise Module

M11: Exercise Selection/History Module

M12: Authentication Up Module

M13: Account Management Module

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

Level 1	Level 2
Hardware-Hiding Module	Video Input Module M8
Behaviour-Hiding Module	Hand Sign Recognition Module M1 Controller Module M3 Data Processing Module M5 Machine Learning Module M6 Landing Page Module M9 Exercise Module M10 Authentication Up Module M12
Software Decision Module	Hand Sign Verification Module M2 Data Collection Module M4 Testing and Verification Module M7 Exercise Selection/History Module M11 Account Management Module M13

Table 1: Module Hierarchy

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

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 Modules

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: Formats datasets into usable training data for Machine Learning Module.

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.

7.2.5 Landing Page Module (M9)

Secrets: Relevant information to shown to user about the program/website.

Services: Display necessary application information about ASLingo to the user on website.

Implemented By: Javascript.

7.2.6 Exercise Module (M10)

Secrets: Question selection process and question bank used to quiz user.

Services: Creates an exercise module with questions based on current users level of progress.

Implemented By: Javascript.

7.2.7 Authentication Up Module (M12)

Secrets: How user information is stored and accessed.

Services: Takes in user input to allow them to sign into their account.

Implemented By: Javascript.

7.3 Software Decision Modules

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 un-processed training 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.

7.3.4 Exercise Selection/History Module (M11)

Secrets: How historical data is stored for each user.

Services: Display users completed exercise progress and display current exercise options.

Implemented By: Javascript.

7.3.5 Account Management Module (M13)

Secrets: How user information is stored and processed.

Services: Will keep a record of users account history after signing in to their account.

Implemented By: Javascript.

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
FR1	M8
FR2	M1 M6
FR3	M12
FR4	M12
FR5	M13
FR6	M10
FR7	M11
FR8	M13 M4
FR10	M2
FR11	M2
FR12	M11
FR13	M12
LFR1	M9
LFR2	M11
LFR3	M10
UHR3	M13
SR1	M12
SR2	M6
SR3	M13
LR2	M6
PR2	M6 M2
OER2	M8
MSR3	M7

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1, M2, M3, M4, M5, M6, M7,
AC2	M9
AC3	M10, M11
AC4	M13
AC5	M6
AC6	M7

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.

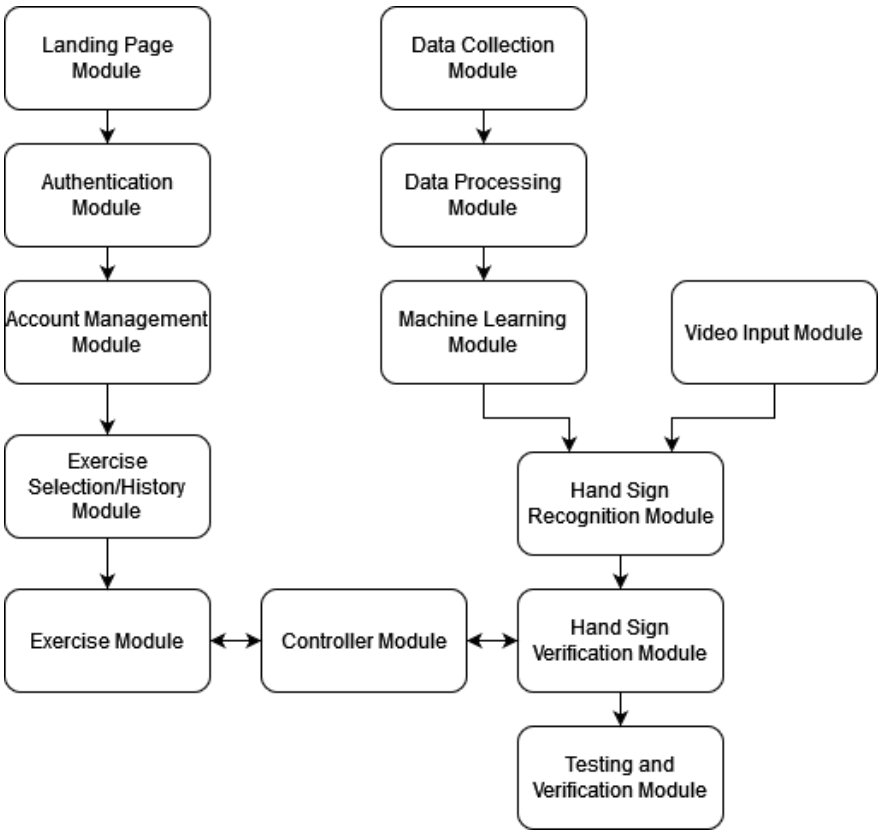

Figure 1: Use hierarchy among modules

10 Timeline

[Schedule of tasks and who is responsible —SS]

11 Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select documented design? (LO_Explores)

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.