Module Guide for Software Engineering

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

April 4, 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
Jan 17, 2024	1.9	Stanley; Answered reflection question 2
Jan 23, 2024	2.0	Cassidy; Addressed git issue #69 regarding AC1 module
Jan 23, 2024	2.1	Cassidy; Addressed git issue #49 regarding NFRT2 - PT3 and PR3
Apr 3, 2024	2.2	Cassidy; Removed unnecessary modules and cleaned up others
Apr 4, 2024	2.3	Jeremy and Cassidy; Added in Quiz Creation Module and updates UsesHierarchy figure

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. 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 laid out by Parnas, 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 **??** lists the anticipated and unlikely changes of the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** 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 **??** 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 **??**, and unlikely changes are listed in Section **??**.

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 module may change if developers want to add in additional testing methods or material.

AC4: Structure of neural network in Machine Learning module may change to improve performance and accuracy.

AC5: 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).

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table **??**. 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 Module

M12: Quiz Generation Module

Level 1	**Level 2**
Hardware-Hiding Module	Video Input Module M??
Behaviour-Hiding Module	Hand Sign Recognition Module M?? Controller Module M?? Data Processing Module M?? Machine Learning Module M?? Landing Page Module M?? Exercise Module M?? Exercise Selection Module M??
Software Decision Module	Hand Sign Verification Module M?? Data Collection Module M?? Testing and Verification Module M?? Quiz Creation Module M??

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 **??**.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas. 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

7.1.1 Video Input Module (M??)

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 (M??)

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 (M??)

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 (M??)

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 (M??)

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 (M??)

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 and Typescript.

7.2.6 Exercise Module (M??)

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

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

Implemented By: Javascript and Typescript.

7.2.7 Exercise Selection Module (M??)

Secrets: How exercise questions are chosen for each user.

Services: Display users current exercise options.

Implemented By: Javascript.

7.3 Software Decision Modules

7.3.1 Hand Sign Verification Module (M??)

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 (M??)

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 (M??)

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 Quiz Creation Module (M??)

Secrets: ASL Letter difficulty and quiz formality constants.

Services: Generates random quizzes of sign language letters or words of different difficulty and size.

Implemented By: Typescript.

8 Traceability Matrix

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

In Table **??**, the anticipated change AC**??** maps to multiple modules in our system. This change refers to the fact that ASL just like many other languages is always evolving and changing over time. Because of this, if this system were to update the words or phrases that are used in this application, it would affect the hand recognition models that were built on older data, as well as all the other modules that are affected by this model change.

Req.	Modules
FR1	M??
FR2	M?? M??
FR3	M?? M??
FR4	M??
FR5	M??
FR6	M??
FR7	M?? M?? M??
LFR1	M??
LFR2	M?? M??
LFR3	M??
UHR3	M?? M??
SR1	M??
LR2	M??
PR2	M?? M?? M??
OER2	M??
MSR3	M??

Table 2: Trace Between Requirements and Modules

AC	Modules
AC??	M??, M??, M??, M??, M??, M??, M??,
AC??	M??
AC??	M??, 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. Parnas 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 **??** 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.

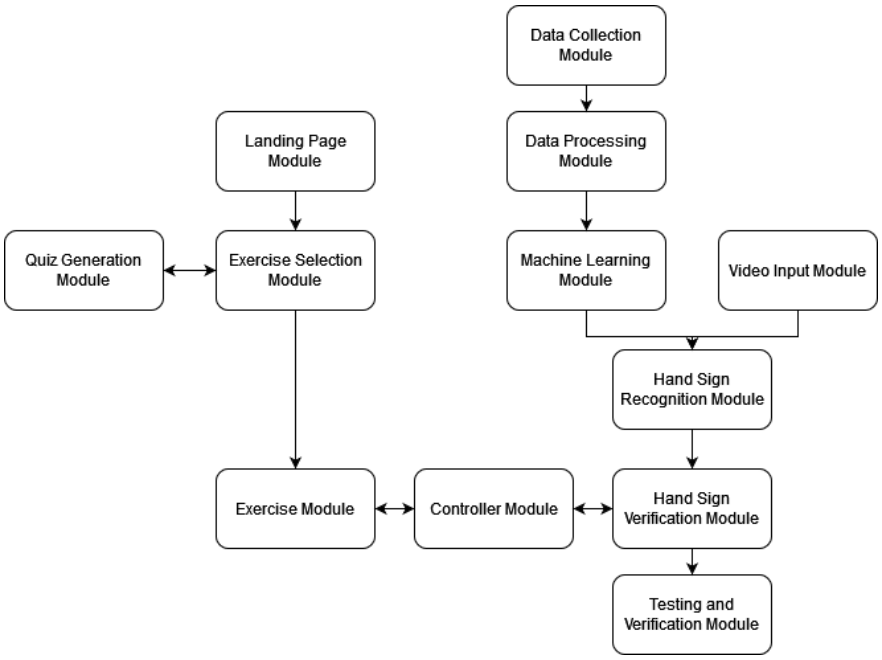

Figure 1: Use hierarchy among modules

10 Timeline

This timeline is under the assumption that this is all implemented for a single learning exercise which is what we projected in our expectations.

- **Monday, Jan 22**
  - Back-end : Data Collection and Data Processing Modules Completion
  - Front-end : Landing Page Module
- **Friday, Jan 26**
  - Back-end : Machine Learning Module Completion
  - Front-end : Work on Exercise and Exercise Selection Module
- **Monday, Jan 29**
  - Back-end : Video Input Module Integration
  - Front-end : Continue Improving Exercise and Exercise Selection Module

- **Friday, Feb 2**
  - Back-end : Hand Sign Recognition + Verification Module Completion
  - Front-end : Exercise and Exercise Selection Module Completion
- **Revision 0 Presentation : February 6**
- **Verification and Testing : Feb 7 - Mar 23**
  - While verifying and testing the application, we will be getting user feedback from stakeholders and representative users
  - The team will incorporate this feedback into adjustments to the specified modules to increase usability for our users in response to testing
- **Final Capstone Presentation (Rev 1) : March 24 at 2:00 pm**

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)
   There are a few improvements we could implement given more time and resources. One of the main limitations of our design is that we do not have separate modules for dealing with static and dynamic hand signs. A perfect solution would recognize the expected type of hand sign and deal with the corresponding type individually. Static and dynamic hand signs are dealt with differently as for static signs, individual frames are what the machine learning model looks at, and for dynamic signs, the model should look at groupings of frames.
   Another limitation of our design is having one single controller module to connect the frontend with the backend. While this simplifies our work in design, given unlimited time, it would be ideal to have data sent through multiple controller modules, wherever a frontend/backend connection needs to be facilitated.

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)
   One particular design solution we had in mind for the transfer of webcam frame data from the frontend to the backend was to only read the user's hand signs after a specified countdown timer, where the system would then send the current frame to the backend. This would avoid the need of sending each frame to the backend, improving performance. Although this would work with static hand signs (most letters of the alphabet), dynamic hand signs which involve the movement of hands to be tracked,

would fail to be recognized. As such, we decided to continue with the original plan of sending each frame to the backend, and dealing with any potential performance issues during implementation.
We also initially assumed that the middleman module between the frontend and the backend would only use the frontend and backend modules, and not vice versa where the frontend/backend modules would use the controller module. This was not the case, as we ran into scenarios where the hand sign verification module and the exercise module would require the use of certain functions in the controller module.