Module Guide for Bridging Gaps: AI for Diagram Accessibility

Team 22, Reading4All
Nawaal Fatima
Dhruv Sardana
Fiza Sehar
Moly Mikhail
Casey Francine Bulaclac

November 5, 2025

1 Revision History

Date	**Version**	**Notes**
Date 1	1.0	Notes
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
Bridging Gaps: AI for Diagram Accessibility	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[xxx —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 (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 specific hardware on which the software is running.

AC2: The format of the initial input data.

[Anticipated changes relate to changes that would be made in requirements, design or implementation choices. They are not related to changes that are made at run-time, like the values of parameters. —SS]

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: Hardware-Hiding Module

M2: Behaviour-Hiding Module

M3: Software Decision Module

Level 1	Level 2
Hardware-Hiding Module (M1)	N/A
Behaviour-Hiding Module (M2)	DataPreprocess Module (M7.2.1) WCAGCompliance Module (M7.2.2) ModelOutput Module (M7.2.3)
Software Decision Module (M3)	AIModelTraining Module (M7.3.1) CaptionGeneration Module (M7.3.2) FeedbackMetrics Module (M7.3.3) FeedbackIntegration Module (M7.3.4)

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.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. —SS]

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. *Bridging Gaps: AI for Diagram Accessibility* means the module will be implemented by the Bridging Gaps: AI for Diagram Accessibility 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 (M2)

This section is written in the following format for each module as defined in Table 1. It is heavily inspired from the template given by Dr. Spencer Smith for the course SE 4G06.

Secrets: This section describes the essential behavioral components that define a module's intended functionality.

Services: This section defines the module's programs that implement the externally visible behaviors of the system, as outlined in the Software Requirements Specification (SRS). Any modifications to the SRS will likely require corresponding updates to the programs within this module.

Implemented By: This section mentions which part of the system will implement the specific module.

Type of Module: Is the module a Record, Library, Abstract Object, or Abstract Data Type? This section also includes information for leaf modules in the decomposition by secrets tree.

7.2.1 DataPreprocess Module

Secrets: Input data is the image given by the end-user via the relevant backend modules.

Services: Filters out or accepts the input data into the data structure used by the model training module.

Implemented By: [what do i put here?]

Type of Module: [Library] [Information to include for leaf modules in the decomposition by secrets tree???]

7.2.2 WCAGCompliance Module

Secrets: Takes input from the CaptionGeneration Module.

Services: Checks generated text against certain parameters to ensure WCAG 2.1 AA compliance.

Implemented By: [Your Program Name Here]

Type of Module: [Abstract Data Type][Part of a Generic Module?] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.3 ModelOutput Module

Secrets: Takes in input from either WCAGCompliance or FeedbackMetrics module.

Services: Produces final alternative text to user via relevant back-end modules.

Implemented By: [what do i put here?]

Type of Module: [Record] [Information to include for leaf modules in the decomposition by secrets tree???]

7.3 Software Decision Module (M3)

This section is written in the following format for each module as defined in Table 1. It is heavily inspired from the template given by Dr. Spencer Smith for the course SE 4G06.

Secrets: This module encapsulates design decisions derived from mathematical theorems, facts, or programming logic. These internal details are *not* documented in the Software Requirements Specification (SRS).

Services: This section defines the module's data structures and algorithms that support system functionality without directly interacting with the user. Modifications in this module are typically driven by performance optimization rather than external requirements. As a side note, changes in these modules are more likely to be motivated by a desire to improve performance than by externally imposed changes.

Implemented By: Mentions which part of the system will implement the specific module.

Type of Module: Is the module a Record, Library, Abstract Object, or Abstract Data Type? This section also includes information for leaf modules in the decomposition by secrets tree.

7.3.1 AIModelTraining Module

Secrets: How to generate alternative text for a given image using various Python libraries.

Services: Provides a trained AI model that can take in new images and output corresponding text based on various relevant details in the image.

Implemented By: [Your Program Name Here]

Type of Module: Abstract Object

7.3.2 CaptionGeneration Module

Secrets: Predicts the alternative text for the image, based on certain features of the image.

Services: Generates a couple sentences that describe the image.

Implemented By: [Your Program Name Here]

Type of Module: [Abstract Data Type][Part of a Generic Module?] [Information to include for leaf modules in the decomposition by secrets tree.]

7.3.3 FeedbackMetrics Module

Secrets: Consists of predefined metrics to enhance generated alt text.

Services: Compares currently generated text's scores against thresholds and returns results for further optimization.

Implemented By: [what do i put here?]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree???]

7.3.4 FeedbackIntegration Module

Secrets: Consists of features used to further enhance the AI Model.

Services: Uses results from FeedbackMetrics module to optimize model on certain parameters.

Implemented By: [what do i put here?]

Type of Module: [Abstract Object] [Information to include for leaf modules in the decomposition by secrets tree???]

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	M7.2.1, M7.2.2
R2	M7.2.2, M7.2.3
R3	M7.3.1
R4	M7.3.2
R5	M7.3.3, M7.3.4
R6	M7.2.3, M7.3.3
R7	M7.2.1, M7.3.4
R8	M7.2.2, M7.3.3
R9	M7.3.2, M7.2.3
R10	M7.3.1, M7.3.4
R11	M7.3.3, M7.3.4

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M7.2.1
AC2	M7.2.1
AC??	M7.3.1
AC??	M7.2.2
AC??	M7.2.3
AC??	M7.3.4
AC??	M7.3.3
AC??	M7.3.2

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.

[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

10 User Interfaces

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

11 Design of Communication Protocols

[If appropriate —SS]

12 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.