Module Guide for Software Engineering

Team 2, SyntaxSentinals
Mohammad Mohsin Khan
Lucas Chen
Dennis Fong
Julian Cecchini
Luigi Quattrociocchi

January 11, 2025

1 Revision History

Date	Version	Notes
January 7, 2025	1.0	Initial document

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	SyntaxSentinals Code Plagiarism Detector
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 (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 modifications that are likely to occur during the development or maintenance of the system. These changes are identified based on the project's goals, stakeholder feedback, and potential future requirements. By isolating these changes within specific modules, we ensure that the system remains flexible and maintainable.

The following are the anticipated changes for the SyntaxSentinals Code Plagiarism Detector:

AC1: Changes in the input format of code snippets, such as supporting additional programming languages (e.g., C++, JavaScript) or new file formats.

AC2: Upgrades to the NLP model to improve semantic understanding, such as incorporating newer machine learning techniques or larger training datasets.

AC3: Changes in the user interface, such as adding new features (e.g., online learning, language-agnostic support) or improving usability (e.g., better navigation, accessibility features).

AC4: Adjustments to the similarity threshold for plagiarism detection, allowing users to customize sensitivity levels based on their specific needs.

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: Switching from an NLP-based approach to a non-NLP-based approach.

UC2: Storing user data beyond the immediate task (violating the zero data retention policy).

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: User Authentication Module: Will authenticate the user with Auth0 and grant them access to the application.

M2: Code Upload Module: Responsible for the handling of user's code upload. Acts as the entry point for the code similarity detection process.

M3: NLP Model Module: Processes code snippets and tokenizes them in order to create machine-readable representations that can be used for similarity detection.

M4: Similarity Scoring Module: Utilizes algorithm to compare tokenized representation of code snippets in order to compare code similarities

M5: Threshold Adjustment Module: Enables user to adjust the similarity threshold that constitutes plagiarism. This parameter directly modifies the algorithm.

M6: Report Generation Module: Module that generates plagiarism reports. These reports summarize code similarity scores as well as flagged pairs of code snippets.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	User Authentication Module Code Upload Module Report Generation Module
Software Decision Module	NLP Model Module Similarity Scoring Module Threshold Adjustment 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 3.

Req.	Modules
FR-1	M2
FR-2	M3 M4
FR-3	M6
FR-4	M5
FR-5	M3 M4
FR-6	M6
FR-7	M1
FR-8	M1
FR-9	M6
FR-10	M6

Table 2: Trace Between Requirements and Modules

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. *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??)

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

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 Code Upload Module (M2)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the NLP model module.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

7.2.2 Report Generation Module (M6)

Secrets: The format and structure of the output reports.

Services: Generates plagiarism reports, including similarity scores and flagged cases.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

7.2.3 User Authentication Module (M1)

Secrets: The authentication and authorization mechanisms.

Services: Handles user account creation, login, and access control.

Implemented By: Auth0

Type of Module: Abstract Data Type

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 NLP Model Module (M3)

Secrets: The NLP-based plagiarism detection logic.

Services: Processes code snippets and generates semantic representations for similarity scoring.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

7.3.2 Similarity Scoring Module (M4)

Secrets: The algorithm for calculating similarity scores.

Services: Compares semantic representations of code snippets and generates similarity scores.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

7.3.3 Threshold Adjustment Module (M5)

Secrets: The logic for adjusting plagiarism detection thresholds.

Services: Allows users to customize the similarity threshold for plagiarism detection.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

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
FR-1	M2
FR-2	M4
FR-3	M6
FR-4	M5
FR-5	M3
FR-6	M6
FR-7	M1
FR-8	M1
FR-9	M6
FR-10	M6

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M2
AC2	M3
AC3	M6, M1
AC4	M5

Table 4: 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.