# Module Guide for TTE RecSys

Yinying Huo

March 19, 2025

# 1 Revision History

| Date | Version | Notes |
|---|---|---|
| Mar 2 2025 | 1.0 | First Draft |

# 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 |
| TTE RecSys | Explanation of program name |
| UC | Unlikely Change |
| ANN | Approximate Nearest Neighbor |

# Contents


1 Revision History i

2 Reference Material ii
2.1 Abbreviations and Acronyms . . . ii

3 Introduction 1

4 Anticipated and Unlikely Changes 2
4.1 Anticipated Changes . . . 2
4.2 Unlikely Changes . . . 2

5 Module Hierarchy 3

6 Connection Between Requirements and Design 3

7 Module Decomposition 4
7.1 Hardware Hiding Modules (M1) . . . 4
7.2 Behaviour-Hiding Module . . . 4
7.2.1 Data Processing Module (M2) . . . 4
7.2.2 Model Training Module (M3) . . . 5
7.2.3 Embedding Generation Module (M4) . . . 5
7.2.4 Recommendation Module (M5) . . . 5
7.3 Software Decision Module . . . 5
7.3.1 Neural Network Architecture Module (M6) . . . 5
7.3.2 ANN Search Module (M7) . . . 6
7.3.3 Vector Operations Module (M8) . . . 6

8 Traceability Matrix 6

9 Use Hierarchy Between Modules 7

10 User Interfaces 8

11 Design of Communication Protocols 8

12 Timeline 8


# List of Tables


1 Module Hierarchy . . . 3
2 Trace Between Requirements and Modules . . . 6
3 Trace Between Anticipated Changes and Modules . . . 7

# List of Figures

1 Use hierarchy among modules . . . . . . . . . . . . . . . . . . . . . . . . . 7

# 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.

**AC3:** The neural network architecture for the user tower.

**AC4:** The neural network architecture for the item tower.

**AC5:** The algorithm used for Approximate Nearest Neighbor (ANN) search.

**AC6:** The loss function and regularization techniques used for training.

**AC7:** The similarity function used for ranking items.

**AC8:** The format of the output recommendations.

## 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:** The use of Python as the implementation language.

**UC2:** The use of deep learning for embedding generation.

**UC3:** The two-tower architecture for the recommendation system.

# 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:** Data Processing Module

**M3:** Model Training Module

**M4:** Embedding Generation Module

**M5:** Recommendation Module

**M6:** Neural Network Architecture Module

**M7:** ANN Search Module

**M8:** Vector Operations Module

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | Data Processing Module<br>Model Training Module<br>Embedding Generation Module<br>Recommendation Module |
| Software Decision Module | Neural Network Architecture Module<br>ANN Search Module<br>Vector Operations 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 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. *TTE RecSys* means the module will be implemented by the TTE RecSys 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

**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 Data Processing Module (M2)

**Secrets:** The format and structure of the input data.

**Services:** Loads, validates, and preprocesses user and item data for training and inference.

**Implemented By:** TTE RecSys

**Type of Module:** Library

#### 7.2.2 Model Training Module (M3)

**Secrets:** The training process, including loss function and optimization technique.

**Services:** Trains the embedding functions using the provided data.

**Implemented By:** TTE RecSys

**Type of Module:** Abstract Object

#### 7.2.3 Embedding Generation Module (M4)

**Secrets:** How embeddings are created from user and item features.

**Services:** Generates embeddings for users and items using the trained models.

**Implemented By:** TTE RecSys

**Type of Module:** Library

#### 7.2.4 Recommendation Module (M5)

**Secrets:** How recommendations are generated and ranked.

**Services:** Retrieves and ranks items for a given user based on embedding similarity.

**Implemented By:** TTE RecSys

**Type of Module:** Library

### 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 Neural Network Architecture Module (M6)

**Secrets:** The structure and parameters of the neural networks used for embedding.

**Services:** Defines the architecture of the user and item towers.

**Implemented By:** TTE RecSys

**Type of Module:** Abstract Object

#### 7.3.2 ANN Search Module (M7)

**Secrets:** The algorithm used for approximate nearest neighbor search.

**Services:** Efficiently retrieves nearest neighbors in the embedding space.

**Implemented By:** TTE RecSys(using FAISS library)

**Type of Module:** Abstract Object

#### 7.3.3 Vector Operations Module (M8)

**Secrets:** The implementation of vector operations such as dot product and normalization.

**Services:** Provides efficient vector operations for embeddings.

**Implemented By:** TTE RecSys

**Type of Module:** Library

# 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 | M2 |
| R2 | M3, M6 |
| R3 | M1, M3 |
| R4 | M4, M6 |
| R5 | M5, M8 |
| R6 | M5, M7 |

Table 2: Trace Between Requirements and Modules

| AC | Modules |
|---|---|
| AC1 | M1 |
| AC2 | M2 |
| AC3 | M6 |
| AC4 | M6 |
| AC5 | M7 |
| AC6 | M3 |
| AC7 | M8 |
| AC8 | M5 |

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.

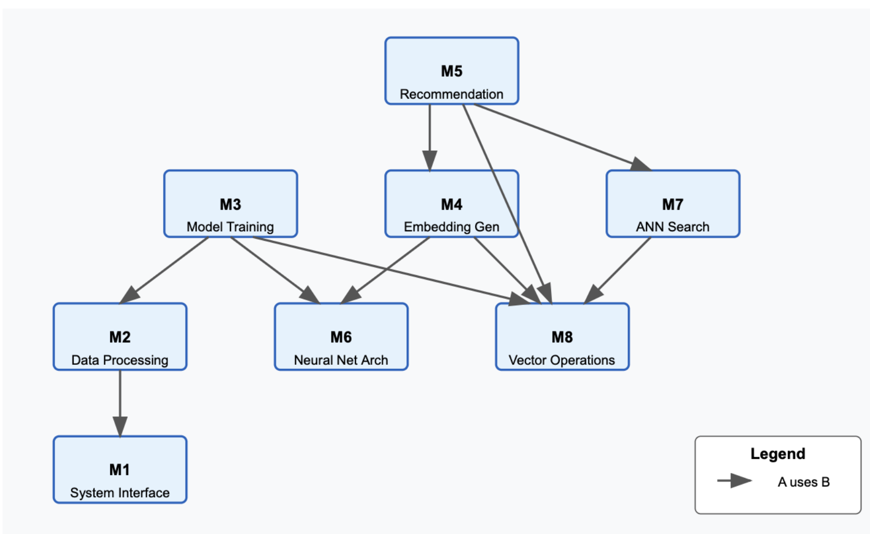


Figure 1: Use hierarchy among modules

# 10 User Interfaces

The user interface for the TTE RecSyswill be a simple terminal-based application that collects user information, processes it through the recommendation model, and displays a ranked list of recommended items.

# 11 Design of Communication Protocols

Not Applicable

# 12 Timeline

Please refer to the Github.

# 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.