Module Guide for ANN (Artificial Neural Newcounter)

Tanya Djavaherpour

March 19, 2024

1 Revision History

Date	Version	Notes
March 19	1.0	Initial Draft

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
ANN	Artificial Neural Network
DAG	Directed Acyclic Graph
GUI	Graphical User Interface
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
UC	Unlikely Change

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) Djavaherpour (2024), 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 Djavaherpour (2024). 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: Changes in the algorithms and techniques used for image classification.

AC4: Modifications in neural network training algorithms and parameter adjustments.

AC5: Updates in the scheduling and methodology for model training and updating.

AC6: ~~Evolution in the~~ format and structure of the output data, including classification results, ~~by developing~~ GUI.

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 source of input is always external to the software.

UC3: The classification results (outputs) are always displayed on the output software.

UC4: The nature of outputs as image classifications is a fixed aspect of the 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: ANN Control Module

M3: Saved ANN Model Module

M4: Output Module

M5: Input Classifier Module

M6: Input Image Module

M7: Training Model Module

M8: Input Preparing and Preprocessing Module

M9: Data Preparing and Preprocessing Module

M10: Training Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	ANN Control Module
	Saved ANN Model Module
	Output Module
	Input Classifier Module
	Input Image Module
	Training Model Module
Software Decision Module	Input Preparing and Preprocessing Module
	Data Preparing and Preprocessing Module
	Training 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 Djavaherpour (2024). 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. *ANN* means the module will be implemented by the ANN 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 Djavaherpour (2024). 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 Djavaherpour (2024).

Implemented By: –

7.2.1 ANN Control Module (M2)

Secrets: The algorithm for coordinating the running of the program.

Services: Provides the main program.

Implemented By: ANN

Type of Module: Abstract Data Type

7.2.2 Saved ANN Model Module (M3)

Secrets: The weights and biases that make up the ANN classification model for th 10 classes of CIFAR-10 dataset Krizhevsky (2009).

Services: Saves the data of the highest performance model trained by ANN in a .txt file.

Implemented By: ANN

Type of Module: Record

7.2.3 Output Module (M4)

Secrets: The format and structure of the output data.

Services: Outputs the class of the uploaded image and receives the end user's feedback.

Implemented By: ANN

Type of Module: Abstract Object

7.2.4 Input Classifier Module(M5)

Secrets: The algorithm for classification given the input image and using the saved ANN model.

Services: Predicts the class of the uploaded image by getting the saved model from M3.

Implemented By: ANN

Type of Module: Abstract Data Type

7.2.5 Input Image Module(M6)

Secrets: Receiving the input data.

Services: Prompts user for input image and checks its size and format.

Implemented By: ANN

Type of Module: Abstract Data Type

7.2.6 Training Model Module(M7)

Secrets: The algorithm for implementing the Artificial Neural Network.

Services: Provides the architecture and the performance of Artificial Neural Network. This module includes the number of neurons, the connection between them, feedforarding and backpropagassion algorithm.

Implemented By: ANN

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 Djavaherpour (2024).

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Input Preparing and Preprocessing Module(M8)

Secrets: Preparing the input data in a format that can be used by model for classifying.

Services: Changes the input image form a 3D matrix to an 1D one by grayscaling it and put all the cells next to each other as a vector.

Implemented By: ANN

7.3.2 Data Preparing and Preprocessing Module(M9)

Secrets: Preparing training dataset in a format that can be used by model for training and testing.

Services: For the better accurate training and testing results, shuffles the training and testing dataset. Changes the traing and test images form 3D matrices to 1D ones by grayscaling them and put all the cells of every image next to each other as vectors.

Implemented By: ANN

7.3.3 Training and Testing Module(M10)

Secrets: Finding weights and biases of the model.

Services: Applies the training process on prepared trining data from M9, using the implemted model from M7 to find the best weights between neurons and each neuron's biase. At the same time, tests the model to make sure it is being trained.

Implemented By: ANN

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	M1, M2, M6, M8
R2	M2, M4, M6
R3	M2, M3, M5
R4	M1, M2, M3, M4
R5	M2, M4

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M6
AC3	M5
AC4	M10
AC5	M7
AC6	M4

Table 3: Trace Between Anticipated Changes and Modules

why have M2, M3 etc. if they aren't related to anticipated changes? Based on my reviewer's comment, I fixed this. Issue #31

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.

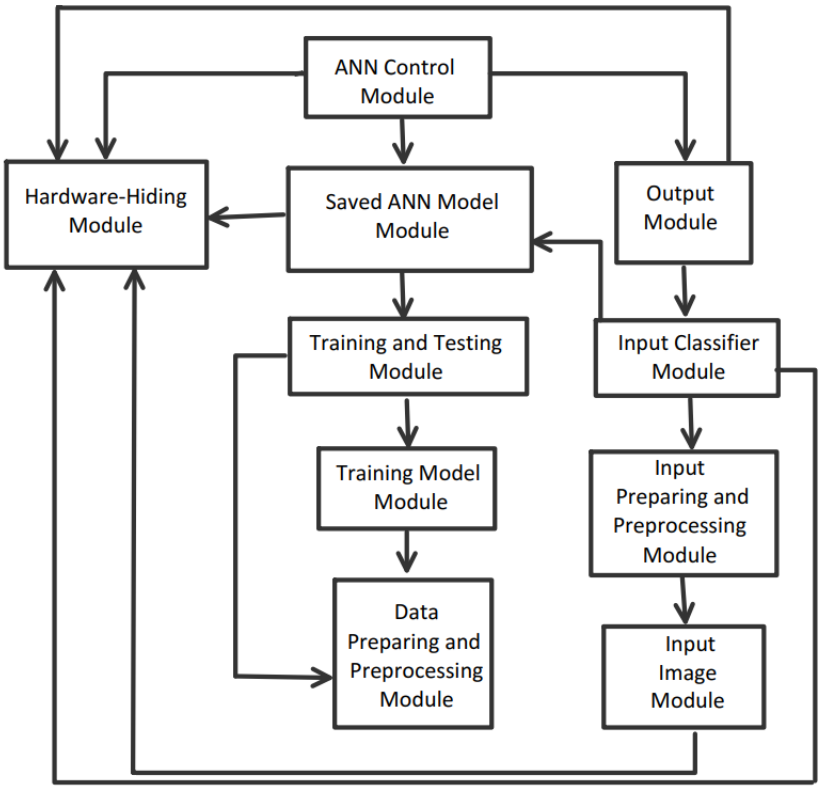

Figure 1: Use hierarchy among modules

These are easier to follow if all arrows are down (would involve moving modules like the hardware hiding to lower levels)

I have changed this a little based on my implementation

References

Tanya Djavaherpour. System requirements specification. https://github.com/tanya-jp/ANN-CAS741/blob/main/docs/SRS/SRS.pdf, 2024.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical Report, 2009.

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.