SE 3XA3: Module Guide: Rev 1
Plagiarism Check

Team 310, MXQ Gang
Qifeng Xu, xuq14
Xu Wang, wangx147
Mrinal Tiwari, tiwarim

April 7, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Mar 12, 2020	1.0	added the Module Guide
April 6, 2020	1.1	Updated the Module Guide

1 Introduction

1.1 Overview

This project is to redevelop a plagiarism checker that can detect the similarity ratio of two documents. The purpose of this software is to avoid plagiarism for writing essays and documentations. It is able to assist students and researchers to accomplish better documents.

1.2 Context

This document is the Module Guide, which is created along with Module Interface Specification after the Software Requirements Specification. The purpose of the SRS document is to illustrate and explain all the functional and non-functional requirements for the project. Based on the SRS document, the Module Guide provides a way to decompose the system into multiple modules, and describes the modular structure. The Module Guide also documents the connection between requirements and modules in this system. Module Interface Specification will be created along with the Module Guide in order to describe the detailed semantics and syntax of each module in the system.

1.3 Design Principles

Decomposition needs to be guided by principles and patterns. The design principles used for decomposition of this system are information hiding, principle of low coupling and high cohesion and "fan-in" in uses relation. The information hiding principle makes sure that each expected change is encapsulated and design interface will not change even when the module secret changes. Low coupling and high cohesion principle ensures that modules are independent and the elements in each module are strongly related.

1.4 Document Structure

- Section 2 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 2.1, and unlikely changes are listed in Section 2.2.
- Section 3 provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in a table.
- Section 4 provides the Connection Between Requirements and Design, which describes how the the design of the system is intended to satisfy the requirements developed in the SRS.
- Section 5 provides the detailed Module Decomposition information.
- Section 6 shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

- Section 7 provides the Uses Hierarchy between modules.

2 Anticipated and Unlikely Changes

2.1 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 operating system on which the software is running.

AC3: The browser on which the software is running.

AC4: The language characters of text that users input.

AC5: The length of texts that users input.

AC6: The payment method for tokens.

AC7: UI style that might vary for different operating systems.

2.2 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 format of the initial input data.

UC3: The format of the final output data.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware Hiding Module

M2: Register Module

M3: Detect Module

M4: Refill Module

M5: Integration Module

M6: Application Module

M7: Sysutils Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Register Module Detect Module Refill Module
Software Decision Module	Application Module Sysutils Module Integration Module

Table 2: **Module Hierarchy**

4 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. For the FR3 that user's password shall not be stored in database as the original password, it will be satisfied through Register Module by an encryption algorithm. This encryption mechanism also satisfies the security requirements in order to protect users' privacy. The Detect Module realizes the functional requirements 4-6 by implementing input functions and a texts comparing algorithm. The Refill Module makes sure that users can refill their tokens with a certain amount when they run out of it.

Application Module checks which resource is getting called by the Integration Module and directs to either Register, Detect and Refill Modules. Sysutils Module provides helper functions to reduce code duplication and improve code reusability. Integration Module also realizes the look and feel requirements by implementing a user-friendly GUI. It shall make it easier for users to operate our software, therefore, it satisfies the ease of use requirements as well.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. 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. Also indicate if the module will be implemented specifically for the 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. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules (M1)

Secrets: The implementation of virtual machine

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: Python virtual machine and Operating System

5.2 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: N/A

5.2.1 Register Module (M2)

Secrets: Password

Services: Register a account by a username and password

Implemented By: Python Libraries

5.2.2 Detect Module (M3)

Secrets: Text input

Services: Check if username and password matched, calculate the text similarity ratio.

Implemented By: Python Libraries

5.2.3 Refill Module (M4)

Secrets: Tokens

Services: Refill the tokens.

Implemented By: Python Libraries

5.3 Software Decision Module

Secrets: Data Structures

Services: Provides the data structures to store the information from the application.

Implemented By: N/A

5.3.1 Integration Module (M5)

Secrets: User interface

Services: UI contains the code to call backend.

Implemented By: Python Libraries

5.3.2 Application Module (M6)

Secrets: Resource

Services: Check which resource is getting called by the integration module and directs to either Register, Detect, Refill

Implemented By: Python Libraries

5.3.3 Sysutils Module (M7)

Secrets: Helper functions

Services: Provide helper functions to reduce code duplication and code reusability

Implemented By: Python Libraries

6 Traceability Matrix

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

Req.	**Modules**
FR1	M2
FR2	M2,M3
FR3	M3
FR4	M3
FR5	M3,M5
FR6	M3
FR7	M4
FR8	M1

Table 3: **Trace Between Functional Requirements and Modules**

Req.	**Modules**
NF1	M5
NF2	M1,M5
NF3	M2,M3
NF4	M1
NF5	M1
NF6	M1
NF7	M1,M2,M3,M4,M5,M6,M7
NF8	M2,M3
NF9	M1,M2,M3,M4,M5,M6,M7
NF10	M1
NF11	M1
NF12	M1
NF13	M2,M3,M4
NF14	M1
NF15	M1
NF16	M1
NF17	M1
NF18	M2,M3,M6,M7
NF19	M2,M3
NF20	M1,M2,M3,M4,M5,M6,M7
NF21	M1,M2,M3,M4,M5,M6,M7

Table 4: **Trace Between Non-Functional Requirements and Modules**

AC	**Modules**
AC1	M1
AC2	M1
AC3	M1
AC4	M2,M3
AC5	M2,M3
AC6	M4
AC7	M5

Table 5: **Trace Between Anticipated Changes and Modules**

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas1978 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.

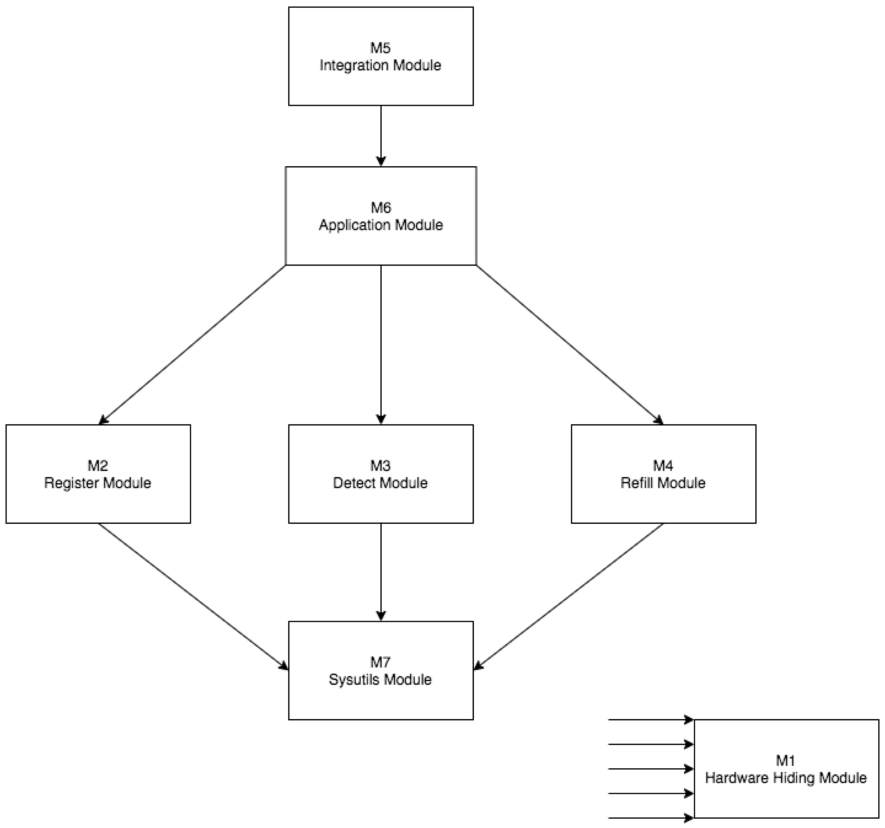

Figure 1: **Use hierarchy among modules**