SE 3XA3: Software Requirements Specification PasswordProtectionProgram

Team 28, Tuples1
Suhavi Sandhu (sandhs11)
Shabana Dhayananth (dhayanas)
Joseph Lu (luy89)

November 11, 2017

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2017-11-10	0.0	Creation

1 Introduction

1.1 Brief Project Overview

The purpose of this project is to implement an encrypted password manager, PasswordProtectionProgram (PPP) wherein a person can safely store and access all of the passwords they use with a single master password. Through working on this project the software team also hopes to learn about encryption methods and the development process.

1.2 Document Overview

This document specifies the modular structure of the system, the PasswordProtectionProgram, and is intended to allow both designers and maintainers to easily identify the parts of the software. The document shall act as a guide for new project members that want to understand the system, a hierarchical structure for maintainers that want to improve the system, and a source for designers to verify the systems consistency, feasibility and flexibility.

The document is organized such that the anticipated and unlikely changes of the software requirements are listed, following a summarization of the module decomposition that has been constructed keeping the likely changes in mind. Next, the connections between software requirements and modules are given along with a detailed description of the modules. The document also includes 2 traceability matrices, one that checks for completeness of the design against the requirements from the SRS and another that shows the relation between anticipated changes and the modules. Lastly there is the use relation between modules.

2 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 2.1, and unlikely changes are listed in Section 2.2.

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 operating system that the application is being run on.

AC2: The framework for the graphical user interface.

AC3: The database management system that is ued to store sensitive information.

AC4: The encryption library that secures user information.

AC5: Customization functionality for the user (ex. change the inactivity time before the application logs out).

AC6: The attributes of the database.

AC7: Additional functionality from password generator (ex. constraining the random password to have only numbers, special characters).

AC8: The portability of the application (ex. allow the application to be used online).

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 used by the system.

UC2: The input to the system will always be external.

UC3: The use of a database to store the user data.

UC4: Individual encryption of each password input to the system.

UC5: The goal of the system is provide a secure storage of the user's passwords.

AC8: The development language that is used.

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: Input Module

M2: Password Generator Module

M3: Encryption Handling Module

M4: Password Checking Upon Creation Module

M5: Password Checking Upon Login Module

M6: GUI Module

M7: Copy Module

M8: Inactivity Module

M9: Database Module

M10: Constants Module

Note: The implentation of PasswordProtectionProgram is completely software based, therefore there are no relevant hardware hiding modules.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Module Password Generator Module Encryption Handling Module GUI Module Copy Module Database Module
Software Decision Module	Password Checking Upon Creation Password Checking Upon Login Module Inactivity Module Constants Module

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The system was designed to satisfy the requirements that were developed in the Software Requirement Specification. The GUI module will satisfy all the visibility and usability requirements as it is the module to deal with the users ability to see and use the application. This module also satisfies many functional requirements of the SRS. The Encryption module satisfies security requirements. The database module satisfies many functional as well as security requirements. The connection between requirements and modules is listed in Table 4.

5 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. 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

Not applicable as the system is software-based and the lowest level that the software interfaces with is the Operating System..

5.2 Behaviour-Hiding Module

5.2.1 Input Module (M1)

Secrets: The format and structure of the input data.

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

Implemented By: PasswordProtectionProgram

5.2.2 Password Generator Module (M2)

Secrets: The algorithm used to generate random password.

Services: Creates a random 8 character password consisting of uppercase letters, lowercase letters and numerical values.

Implemented By: PasswordProtectionProgram and Python Libraries

5.2.3 Encrytion Handling Module (M3)

Secrets: The algorithm used to generate the key to encrypt the password.

Services: Creates a key using PBKDF2HMAC and Python's cryptography library, Fernet. That key is then used to encrypt and decrypt inputs.

Implemented By: PasswordProtectionProgram and Python Libraries (Fernet)

5.2.4 GUI Module (M6)

Secrets: The structure and processing of user input.

Services: Takes in user input data and verifies it if necessary. Stores data into and retrieves data from the database to display.

Implemented By: PasswordProtectionProgram and Python Libraries (tkinter)

5.2.5 Copy Module (M7)

Secrets: The cross platform to copy text to clipboard.

Services: Copies user's selected username and password as text to clipboard.

Implemented By: PasswordProtectionProgram and Python Libraries (pyperclip)

5.2.6 Database Module (M9)

Secrets: Database Management System structure used to store and retrieve data.

Services: Stores and retrieves entry information such as account name, account type, username, hash value, hash key.

Implemented By: PasswordProtectionProgram and Python Libraries (peewee)

5.3 Software Decision Module

5.3.1 Password Check Upon Creation (M4)

Secrets: The algorithm used to check the created master password.

Services: Informs user if the password is missing key elements, such as upper and lower case letters, a number, a special character, and a minimum length of 8.

Implemented By: PasswordProtectionProgram and Python Libraries (re)

5.3.2 Password Check Upon Login (M5)

Secrets: The algorithm used to verify that the user entered the correct master password.

Services: Verifies input password against one stored in database.

Implemented By: PasswordProtectionProgram and Python Libraries (re)

5.3.3 Inactivity Module (M8)

Secrets: The algorithm used to detect user inactivity.

Services: Logs user out of account after a certain period of inactivity.

Implemented By: PasswordProtectionProgram

5.3.4 Constants Module (M10)

Secrets: The style parameters used to characterize the physical display of the application.

Services: Constants defining the window size, colour scheme, and fonts.

Implemented By: PasswordProtectionProgram

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	M10
FR2	M9
FR3	M4
FR4	M5
FR5	M1, M6
FR6	M2
FR7	M6
FR8	M3, M9
FR9	M7
FR10	M4
FR11	M3

Table 3: Trace Between Functional Requirements and Modules

Req.	Modules
NFR1	M6
NFR2	M1, M6
NFR3	M1, M6
NFR7	M??
NFR9	M1, M6
NFR11	M10, M6
NFR12	M10, M6
NFR13	M1
NFR15	M3, M9
NFR16	M3, M5
NFR17	M8

Table 4: Trace Between Non-Functional Requirements and Modules

AC	Modules
AC1	M1
AC2	M6
AC3	M9
AC4	M3
AC5	M10
AC6	M9
AC7	M2
AC8	M6

Table 5: Trace Between Anticipated Changes and Modules

7 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.

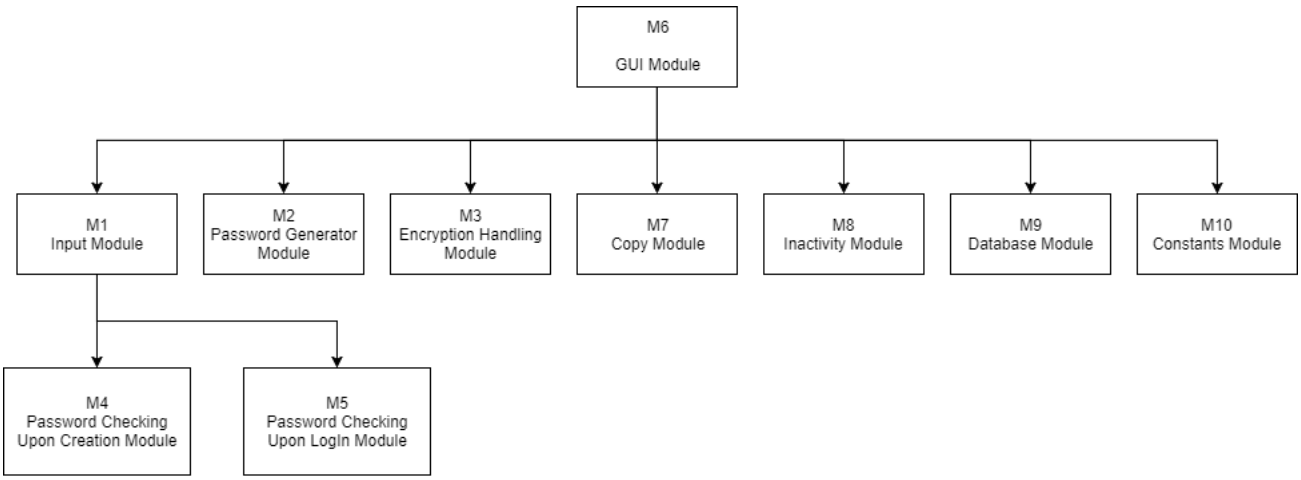

Figure 1: Use hierarchy among modules

References

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.