Software Requirements Specification (SRS)

Members: Huang Tianji (21099573d)[1], Zhang Zhiyuan (21096414d)[1],
Zheng Shouwei (21097982d)[1], Zhou Taiqi (21106717d)[1]
Team: 20 *1. Hong Kong Polytechnic University*

Table of Contents

1. Preface

This document serves as a comprehensive guide that outlines the specific requirements and expectations for the successful development and implementation of the software. It is intended to be used by a wide range of individuals involved in the software development process, including project managers, developers, testers, and end-users. It provides essential information to guide the development team in designing and implementing the software to meet the identified needs and requirements.

Throughout the document, you will find a comprehensive glossary that defines key terms and concepts to ensure consistent understanding and effective communication among all stakeholders. Additionally, the SRS includes user requirements, system architecture, and system requirements specifications, which collectively provide a holistic view of the software and its functionalities.

We believe that a well-defined software requirements specification is crucial for the success of any software project. By clearly articulating the goals, constraints, and functionality of the software, we can minimize misunderstandings and ensure that the final product meets the expectations of all stakeholders.

We encourage all readers to thoroughly review this document and provide feedback or suggestions for improvement. Your input is essential in ensuring that the software development process is aligned with the needs and expectations of our organization.

Thank you for your time and commitment to this project. We are confident that with your support, we will deliver a high-quality software solution that meets the requirements of our users.

Version History

Version	Date	Description of Changes	Author(s)
1	10/29/23	Finish the draft of SRS.	HUANG Tianji
1.1	11/1/23	Updated functional requirements based on feedback from review. Added non-functional requirements and expanded the glossary.	ZHANG Zhiyuan
1.2	11/5/23	Revised user stories and system models for clarity after team meeting.	ZHOU Taiqi
1.3	11/9/23	Added details in system architecture and interface requirements. Updated glossary.	ZHENG Shouwen
1.4	11/12/23	Finalized all sections after review. Corrections made to align with project guidelines.	HUANG Tianji
2	11/20/23	Updated functional requirements.	ZHANG Zhiyuan

2. Introduction

2.1. System Overview

The Personal Information Manager (PIM) is a command-line interface (CLI) application crafted to assist users with organizing personal data, encompassing contacts, events, tasks, and notes.

2.2. Objectives

The main objectives of this deliverable are the following:

- Furnish an efficacious tool, optimized for ease of use, that streamlines the management of personal information.
- Exemplify the practical implementation of software engineering (SE) methodologies and principles within a tangible project.
- Engineer a system that not only adheres to but also exemplifies the highest standards of software development and maintainability.

2.3. Scope

The PIM's capabilities will span creating, viewing, deleting, and searching for user-generated records. As a standalone application operating via the CLI, its design is laser-focused on simplicity and enhancing user experience.

2.4. Abbreviations

PIM: Personal Information Manager.
CLI: Command Line Interface.
SRS: Software Requirements Specification.
UI: User Interface.
FR: Functional requirement
NFR: Non-functional requirement

3. Glossary

Command-Line Interface (CLI): A text-based user interface used to view and manage computer files.
Functional Requirement: A requirement that defines a specific function of the software system.
Non-Functional Requirement: A requirement that specifies criteria for judging the operation of a system, rather than specific behaviors.
User Stories (US): A simple description of a software feature from the perspective of end users.
System Architecture: The conceptual model that defines the structure, and behaviors of a system.
Module: A distinct part of the software system responsible for specific functionalities.

4. User Requirements

US1: Management of Personal Information Records (PIRs)

The PIM is designed to facilitate the creation of various PIRs, consolidating the management of critical personal data into a singular hub.

US2: Creation of Plain Texts as PIRs

Within the PIM, users are empowered to craft plain text notes swiftly, leveraging them as PIRs for efficient note-taking.

US3: Task Management

The system enables users to establish tasks as PIRs, replete with detailed descriptions and set deadlines, thereby streamlining task management.

US4: Event Scheduling

Users can schedule and define events as PIRs, including descriptive details, initiation times, and alarm settings, to assist in meticulous schedule management.

US5: Contact Information Management

The PIM is designed to permit users to generate contacts as PIRs, cataloging critical details such as names, addresses, and phone numbers.

US6: Modification of Existing PIRs

Users are granted the capability to alter existing PIR data, ensuring the retention of up-to-date and precise information.

US7: Searching for PIRs

The PIM must support a robust search feature, allowing users to locate PIRs through a variety of filters, including type-specific data and temporal criteria, bolstered by logical operators.

US8: Detailed Information Display

There is a provision within the PIM to exhaustively display information pertaining to individual or collective PIRs stored within the system.

US9: Deletion of PIRs

Users retain the discretion to expunge selected PIRs from the system, as necessitated.

US10: Storage of PIRs in $.pim$ Files

The system will facilitate the storage of PIRs in $.pim$ files, promoting ease of future access and ensuring data continuity.

US11: Loading PIRs from $.pim$ Files

Users will be able to retrieve and resume work with PIRs from previously stored $.pim$ files.

US12: Automatic Email Notifications for Alarms

The system must autonomously dispatch email notifications to specified addresses upon the approach of scheduled event alarms, guaranteeing that users are reminded of imminent engagements.

5. System Architecture

5.1. Overview

The Personal Information Manager (PIM) application is structured following the Model-View-Controller (MVC) architectural pattern. This architecture divides the application into three interconnected components, enabling efficient code organization and separation of concerns.

5.2. Components

1. Model
Defines data structures and handles data management. It includes classes for different types of Personal Information Records (PIRs) like Contacts, Events, Tasks, and QuickNotes, inheriting from a base Item class in pim_model. This layer is responsible for operations such as saving, viewing, updating, and deleting items.

2. View
Responsible for the Command Line Interface (CLI) presentation layer. The Printer class in pim_view manages user interactions, displaying menus and options, and gathering user inputs.

3. Controller
Acts as an intermediary between the View and Model. The Controller class in pim_control handles the application logic, connecting user inputs from the View to the corresponding Model operations.

5.3. Flow of Control

1. The main function starts the application, starting with a daemon process for background tasks and instantiating the Controller.
2. The Printer class presents the main menu and user options.
3. User inputs are captured and sent to the Controller.
4. The Controller processes these inputs and calls appropriate methods in the Model classes (such as Contact, Event, Task, and QuickNotes), doing the requested operations.
5. Model classes handle data manipulation, storage, and retrieval, interacting with $.pim$ files for persistent storage.
6. The View reflects the results of these operations, giving feedback to the user.

5.4. Additional Features

In the background, $Thread2.py$ runs a daemon process, diligently monitoring for tasks such as alarm triggers. The Controller further enriches the system's capabilities with advanced search functionalities, accommodating keyword and logic-based queries.

5.5. Design Considerations

1.Modularity
Independence is a hallmark of each system component, fostering an environment where maintenance and updates are streamlined.

2. Scalability
The architecture's flexible design readily accommodates the infusion of new features and the expansion of PIR categories, all without necessitating significant codebase alterations.

3. Usability
With a focus on user experience, the CLI is intuitively crafted to serve those accustomed to command-line interfaces, prioritizing straightforward interactions.

4. Testing Strategy

An integral part of the architecture now includes a methodical testing protocol. This framework is vital for assuring that each module, such as the Item class, operates reliably across varying scenarios.

5. Mocking and Patching

The system's testing phase leverages techniques like mocking and patching, effectively demonstrated in the *TestItem* class's *test_init* method. These techniques allow for the replication and management of behaviors from external dependencies.

6. System Requirements Specification

6.1. Functional Requirements

FR1: User Interface Management

A command-line interface is integral for user interactions. Users can engage with various functionalities by selecting options via numbered menu items. To exit, type a specific character, for instance ‘q’.

FR2: Contact Management

Users will have the capability to create, view, edit, and remove contact details. Each contact record must encompass a name, phone number, email, and physical address. The system will prompt users when inputs deviate from expected data formats.

FR3: Event Management

The system enables users to manage events, including scheduling, viewing, updating, and deletion. Vital for each event record are details like a description, start time, and alarm settings. Alerts are issued for inputs that mismatch the required data types.

FR4: Task Management

Users can add, observe, modify, and delete tasks within the system. Necessary details for each task include a description and a deadline. The system will notify users if they do not type in the correct data format.

FR5: Note Management

Facilitating the creation and storage of quick notes is a key function. These notes are to be maintained as simple text. Any non-conforming user inputs are to be flagged by the system.

FR6: Persistent Data Storage

Persistent storage of PIRs in '.pim' files within the 'PIM_dbs' folder is mandatory. Loading of data from these files should be seamless upon user request. The system ensures uniformity in file format, rejecting any non-conforming new file creations.

FR7: Search Functionality

Users should have the ability to locate PIRs using keywords and date parameters. The search mechanism is to include logical operators like '!', '||', and '&&'.

FR8: Alarm and Notification System

A background process will constantly monitor for upcoming event alarms. Email notifications are automatically triggered at the designated alarm times.

FR9: System Testing

Comprehensive testing is essential to confirm the functionality of each component. These tests will encompass critical areas such as object creation, data handling, and persistent storage reliability.

6.2. Non-Functional Requirements

NFR1: Performance
The system should respond to user inputs quickly. Background processes should not significantly influence the system's overall performance.
NFR2: Usability
The CLI should be user-friendly. Instructions and error messages should be clear and informative.
NFR3: Reliability
The system should maintain a high level of accuracy in data management, with less error rate in data storage and retrieval.
NFR4: Scalability
The system architecture should support adding new features and PIR types without the majority's re-design.
NFR5: Security
User data stored in the system should be protected from illegal outer access. The system should include authority checks to prevent data corruption.
NFR6: Maintainability
The system should be structured modular to facilitate easy maintenance and updates. The code should be well-documented to support future development efforts.
NFR7: Testability
The system's architecture should help ease the creation and execution of unit tests. Components should be designed to support mocking and patching for unit testing.

6.3. System Constraints

1. Operating Environment
The system is designed to run with Python on standard operating systems such as Windows, Linux, and MacOS.
2. Development Constraints
The system should be developed using Python, following best practices in SE methods.

6.4. Data Storage and Management

All data should be stored in $.pim$ files within the PIM_dbs folder. The system should ensure data integrity during reading and writing operations.

6.5. Quality Assurance

1. Testing Protocol
Pay attention to the importance of unit testing in the development lifecycle to make sure that the system meets all FR and NFR.
2. Continuous Testing
Encourage the practice of continuously running the test program during development to identify issues earlier.