STUDENT SUCCESS PLANNER FINAL REPORT

Alexander Sukennyk

12/2/2024

1. Introduction

1.1. Purpose and Scope

The purpose of this software was to create a success planner to be used by students who want an organized, focused helper with no outside dependencies.

1.2. Product Overview

The software stores student information, including basic information, as well as information regarding their current semester and due dates. These can be cross referenced to a notepad section to keep their work and lecture notes organized.

1.3. Structure of the Document

This document will go over the management, specifications, architecture, design, and testing of the Student Success Planner software. A User Manual is available in the appendix for users.

1.4. Terms, Acronyms, and Abbreviations

SSP – Student Success Planner

GUI – Graphical User Interface

2. Project Management Plan

2.1. Project Organization

The project was developed in a GitHub repository, hopping between two computers both owned by the sole developer.

2.2. Lifecycle Model Used

The Waterfall Model was used to develop this software. Agile methods would not have worked well given the constraints of this being a course assignment, and a one-man project.

2.3. Risk Analysis

This software does not include any connection to cloud services of any kind and exists entirely within the user's system. As such, there are no security risks associated with access outside of physical access to the host system.

Risks in development include the possibility of failure. In this case, I did initially promise myself more content than I was able to complete. With no color labels, and lack of cross-referencing flags, some key content to make this a production-ready software is missing.

2.4. Hardware and Software Resource Requirements

Python, and a computer which can run it, are required for this software. No non-standard libraries need to be installed.

2.5. Deliverables and schedule

- Work was continued throughout each Saturday from the start day.
- The final software version was completed on 12.1.2024.
- The report and user manual were completed on 12.2.2024.
- They will be submitted before my presentation on 12.3.2024.

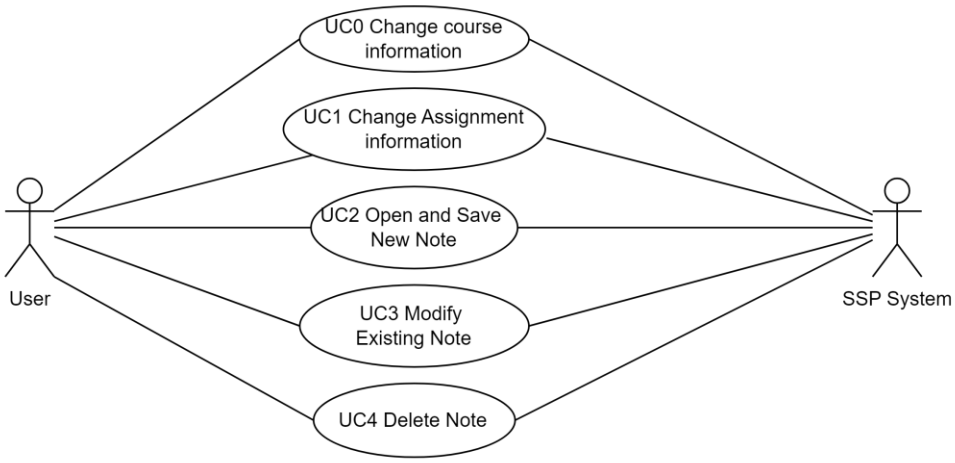
3. Requirement Specifications

3.1. Stakeholders for the system

Myself – The developer, user, and sponsor of the project.

3.2. Use cases

3.2.1. Graphic use case model



3.2.2. Textual Description for each use case

- UC0: User enters or updates course information.
- UC1: User enters or updates assignment information.
- UC2: User creates a new note and saves it.
- UC3: User opens an existing note, modifies it, and saves it.

- UC4: User deletes an existing note.

3.3. Rationale for your use case model

These use cases describe each operation that can be taken by the user using this software. Assured function w.r.t. each use case is critical for this software to be functional.

3.4. Non-functional requirements

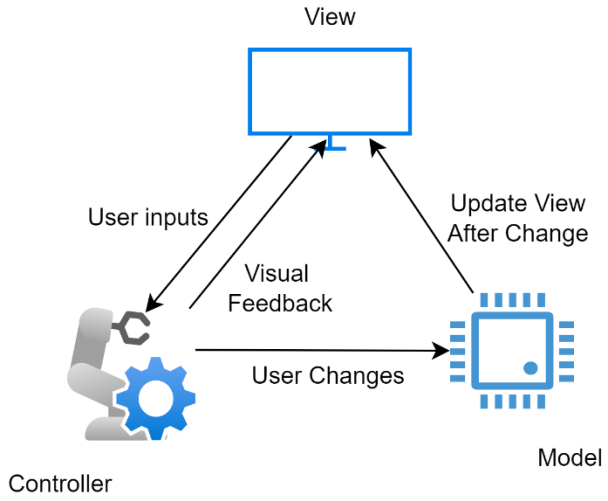
The features in the software should be clear and readable, each text instance should be large enough to be readable to the average user.

4. Architecture

4.1. Architectural style(s) used

A Model-View-Controller architecture was used, with the GUI elements in the view being most of the written code.

4.2. Architectural model (includes components and their interactions)



Model – The IO methods control the flow of data. The data is sent to the View component to be displayed to the user to fulfill the entered commands.

View – Tkinter's GUI frames control the view, with simple text boxes populating the GUI for user view and interaction.

Controller – Tkinter widgets read user inputs and manipulate the model appropriately. The cycle of input → access → display is the key her with the MVC architecture.

4.3. Technology, software, and hardware used

Python with the tkinter library was used for this entire software development cycle. The software was developed on a Windows 10 computer and tested on Win7, Ubuntu, and Win10 operating systems.

4.4. Rationale for your architectural style and model

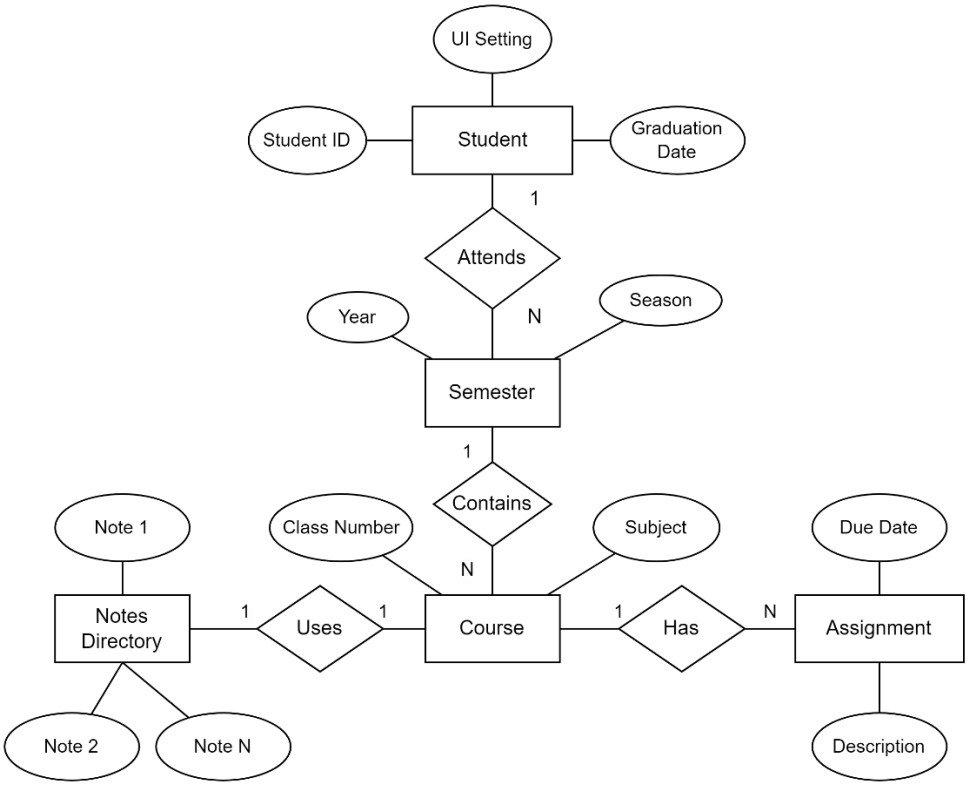
MVC was a simple choice for this work, as the overall complexity of control was not complex. For simple user intractability with no cloud saving functionality, the model is sufficient to conform to the needs of users.

5. Design

5.1. User Interface design

The user interface is a simple multi-frame swap GUI with tkinter. Several frames are initiated to act as tabs for the display. They are updated accordingly as changes are made to the user information.

5.2. Components design (static and dynamic models of each component)



5.3. Database design

The database in this circumstance is saving to 2 local folders, made during the initialization phase – Files, and Files/Notes. These

5.4. Rationale for your detailed design models

This association diagram was the best way to demonstrate the references between each

5.5. Traceability from requirements to detailed design models

6. Test Management

6.1. A complete list of system test cases

- [ID:01] Saving on user clicking any “save” step.
- [ID:02] Loading successfully on launch from unmodified files.
- [ID:03] Live updating of information as it is entered by user.
- [ID:04] Successful deletion of file from Notes Directory Display on Button Press
- [ID:05] Successful save of new and existing files from Notepad Window

6.2. Traceability of test cases to use cases

- Test case 01 is present in each frame with information to update. This is traced to each use case.
- Test 02 traces to each use case where existing data is accessed, UCs 0,1,2,3.
- Test 03 tests appropriate functions in UCs 0,1,2,3, as all data is updated in the display live afterwards.
- Test 04 traces to UC4
- Test 05 traces to UC3

6.3. Techniques used for test case generation

Test cases were created based on the actions that can be taken by users. By ensuring that each action is consistently repeatable, there should be no issues with user control in the SSP system.

6.4. Test results and assessments (how good are your test cases? How good is your software?)

Each of the tests passes except test case 03. This is described below. The tests sufficiently validate the key functions of the software and ensure that it is bug free by encouraging the developer to consider multiple failure possibilities.

6.5. Defects reports

A single defect can be noted – the introduction text does not appear after student info is entered for the first time. This is a failure of text case 03. It is resolved when the application is restarted and is due to the lack of a reference pass to the profile frame from the home frame. A quick solution could not be found, so this defect remains in the released version.

7. Conclusions

7.1. Outcomes of the project (are all goals achieved?)

Apart from the cross-referencing for due dates vs. documents, all other goals were achieved with this project. The software is functional and contains no errors during standard operation. Deliberate attacks may cause issues to occur, but are unlikely for offline, single user software.

7.2. Lessons learned

GUI design has a lot of depth and difficulty in terms of development. There are a lot of things to consider, and entering without a plan makes things much harder to recover in the long run. Better yet, entering with a bad or uninformed plan is much worse than just making fluid components that can be reused. I will consider, in the future, such lessons before tackling other projects.

7.3. Future development

This project can be extended by implementing sorting, a visualizer for the user's schedule, and other QoL features that could be expected from a planner app. I hope that in the future someone will make one that is up to these standards, so that other students may benefit from something better than I have made.