Four-Year Plan Tracker
Software Requirements Specification

River Veek (riverv) – 3-9-2021 – v2.4

Table of Contents

1. SRS Revision History

This lists every modification to the document. Entries are ordered chronologically.

Date	Author	Description
2-12-2021	ezekielp	Created initial document
2-13-2021	noahk	Worked on Section 2
2-13-2021	ezekielp	Edited ConOps
2-16-2021	riverv	Worked on Sections 3.3, 3.5
2-16-2021	noahk	Worked on Sections 2.4, 2.5, 2.6
2-17-2021	noahk	Worked on Sections 3.1, 3.2
2-17-2021	riverv	Peer-reviewed Sections 3.1, 3.2; made revisions
2-18-2021	ezekielp	Edited grammar in Section 3, added 3.4
3-05-2021	llevitre	Revised 3.1, 3.2, 3.3
3-08-2021	jkashuba	Proofreading, edited typos and grammar
3-08-2021	ezekielp	Revised use cases, grammar check for full document
3-09-2021	riverv	Edited grammar, fixed formatting inconsistencies

2. The Concept of Operations (ConOps)

2.1. Current System or Situation

Once a student begins college one of the first things they need to do after determining their declared major is to draft a four-year plan. This four-year plan is a theoretical guide for the student of the classes that they will need to take and when these classes should be taken. Currently, when a student wishes to plan out their degree path, they have two main options. The first option is to contact their advisor for their major and schedule a meeting to construct their four-year plan. The second option is to search for the major requirements on their own and use those to construct their plan.

2.2. Justification for a New System

While there is nothing functionally wrong with either of the approaches stated in section 2.1, each method comes with some distinct drawbacks. Regarding the first option, there is the issue that both students and advisors can become very busy over the course of a typical college term. Considering the many responsibilities already assigned to the typical college student, finding time to schedule and attend an advising meeting can be difficult and may be deprioritized. Regarding the second option, information may be lost due to student inexperience and cause the plan to be flawed. While all the information a student needs to plan out their time in college is usually available on university web pages, it is quite easy for some crucial information to be missed. To construct a robust degree-plan, the student needs to know what courses are required, what the prerequisites are, and in what terms they are offered (it should be noted that it is also useful to know the difficulty level of the course).

Additionally, both options possess the same major problem; plans change over time. While it is convenient to have an initial four-year-plan to help make short term decisions, these plans are almost certain to change before graduation. This could be due to a variety of reasons, such as having to retake a course, course conflicts with other required or desired courses, or perhaps a desire to adjust rate of major progress. When any of these possible causes for adjustment occurs, they result in the student needing to repeat the degree-planning process.

By creating a system that can store a student's major progress state, along with the knowledge of what is required for the major, we effectively remove the need to regularly repeat the degree-planning process by both the student and their academic advisor when a previously unforeseen change occurs.

2.3. Operational Features of the Proposed System

- Store course list for a major, course requirements for the UO CIS major
- Store student course information in a "database"
- Use the above information to construct a degree plan for the selected student based on the major requirements and the courses they have already taken
- Web interface for user
  - Interface for selecting/adding a new student
  - Once student is selected
    - Interface for adding classes taken (along with term and year taken)
    - Interface for removing classes prior to being saved (to mitigate errors)
    - Interface displaying a visual representation of the degree-plan for the selected student.

2.4. User Classes

There are two user classes for this system: academic advisors and students. These users will both want to use the system to organize/store degree progress and to generate a possible four-year plan based off said progress and degree requirements.

2.5. Modes of Operation

The difference between the users is academic advisors will utilize the functionality of our system to store multiple students. Specifically, a student user will only need to create a student object to store their own personal progress. Meanwhile, an academic advisor will create a student object for each student that they interact with. Then when a student comes back for additional advising later in their college career, the academic advisor will use the drop-down menu of saved student objects displayed on our system home page to select the student object that corresponds to the student they are currently helping. Once the student object is selected, the user's functional use of our system will be the same for both user classes.

2.6. Operational Scenarios (Also Known as "Use Cases")

Use Case 1: Add new courses taken to student data.

Brief description:

This use case describes how a user (either academic advisor or student) would go about updating a selected student's data to note that the student had completed some course.

Actors: Academic Advisor / Student

Preconditions:

1. User selects which student to update (either through selecting pre-created students or adding new student into the system).

Steps to Complete Task:

1. User selects course they want to add from provided drop-down menu.
2. User selects year for which they want to add the course to from provided drop-down menu.
3. User selects term for which they want to add the course to from provided drop-down menu.
4. User clicks "Add Course" button.

Postconditions:

The webpage will reflect the added course by displaying it in a table.

Use Case 2: Remove courses taken from student data.

Brief description:

This use case describes how a user (either academic advisor or student) would go about updating a selected student's data to remove a course (potentially one erroneously added).

Actors: Academic Advisor / Student

Preconditions:

1. The user is already logged in and one or more courses are currently staged in the table of added courses.

Steps to Complete Task:

1. User selects course they want to remove from provided drop-down menu of courses.
2. User selects year for which they want to remove the course from provided drop-down menu.
3. User selects term for which they want to remove the course from provided drop-down menu.
4. User clicks "Remove Course" button.

Postconditions:

The table of added courses will reflect the course removal by removing the associated line.

Use Case 3: Save student data state.

Brief description:

This use case describes how a user (either academic advisor or student) would go about saving a selected student's data after changes have been applied.

Actors: Academic Advisor / Student

Preconditions:

1. The staged change table has zero or more courses added via Use Case 1.

Steps to Complete Task:

1. User clicks the "Save" button on the webpage.

Postconditions:

A pickle file will be created/updated to store the selected student class object. This pickle file will be stored within the pickle directory of the Docker container and the staged change table will be cleared.

Use Case 4: Display possible degree plan for student.

Brief description:

This use case describes how a user (either academic advisor or student) would go about saving and displaying a selected student's possible degree plan based on the major requirements and the student data.

Actors: Academic Advisor / Student

Preconditions:

1. The staged change table has zero or more courses added via Use Case 1.

Steps to Complete Task:

1. User clicks "Save and Display" button on the webpage.

Postconditions:

The user will be redirected to a new webpage in which a forecasted degree plan will be displayed in a matrix format. The selected student class object will be saved in a pickle file in the Docker container.

3. Specific Requirements

3.1. External Interfaces (Inputs and Outputs)

3.1.1. Add New Student

- Inputs
  - identifier
    - A string input by the user to use as an identifier for the new student
- Outputs
  - student
    - A new Student class object that possesses the unique information inputted by the user (identifier)

3.1.2. Add / Remove Course

- Inputs:
  - course name
    - A string inputted by the user from a drop-down menu on the student webpage.
  - year
    - A string inputted by the user from a drop-down menu on the student webpage ("1st", "2nd", "3rd", "4th", "5th").
  - term

        - A string inputted by the user from a drop-down menu on the student webpage (“Fall”, “Winter”, “Spring”, “Summer”).
- Outputs:
    - The Student class object that was initially selected by the user will have staged changes that reflect the edits given by the add or remove course function call.

3.1.3. Save

- Side-effects:
    - Modifies the Student class object corresponding to the current student ID based on the current staged changes.
- Outputs:
    - None.

3.1.4. Save and Display

- Side-effects:
    - Modifies the Student class object corresponding to the current student ID based on the current staged changes.
    - Redirects to a forecast display page with information based on the current forecast calculation.
- Outputs:
    - None.

3.2. Functions

3.2.1. Add New Student

To add a new student to the system, the user will first start by clicking the textbox displaying “Enter 95 Number” and manually enter their student ID number, then click the “Create” button. This creates a new student object pertaining only to that student ID. The page will then rebuild a new webpage over the original, displaying their student ID and the options to select what classes they have already taken.

3.2.2. Add Course to Student Plan

To add a course to a student's course plan, the user will start by inputting the course information for the targeted course. To do this, the user will select the appropriate course name, year, and term from the provided drop-down menus. If the user forgets to select one of the appropriate choices and leaves it at the default setting, it will raise a warning saying that is not an option and alert the user to double check what they have currently chosen. With the correct information selected, the user will then click the “Add Course” button. Our system will then update the staged changes shown in the adjacent table to include the course provided by the user’s inputs and the year/term provided by the user’s inputs.
Note: Since all the user inputs are given via drop-down menus, there is no need to check the validity of these inputs.

3.2.3. Remove Course from Student Plan

To remove a course from a student's current list of courses taken, the user will start by inputting the course information for the targeted course. To do this the user will select the appropriate course name, year, and term from the provided drop-down menu. If the validity check does not find the course within the current table upon clicking “Remove Course”, an error message will be displayed to the user saying, “Error Removing Course: could not find given course at specified year, term.” Otherwise, if valid information selected, the course will be removed from the table of staged changes.

3.2.4 Save Current Course Plan

Once the user has entered all courses that have previously been completed by a student, the “Save” button can be selected, this will update and locally save the student object corresponding with the current student ID.

3.2.5 Save and Display Current Course Plan

Similar to the previous section (3.2.4 Save Current Course Plan), selecting the “Save and Display” button will update and locally save the student object corresponding to the current student ID. Additionally, this button will display the current course forecast in a sequence of tables, each corresponding to one school year.

3.3. Usability Requirements

The Four-Year Plan Tracker (FYPT) should allow a student user or an advisor user to create, manipulate, and view a CIS class schedule in a straightforward manner; these three functionalities are the main pillars of the FYPT.

While accessing a specific schedule, the user should only need to use the existing user dropdown menu containing the student IDs (the unique ID number assigned to each student by the University of Oregon). If a student has not yet been entered into the system, then the system must also grant the option to do so. Once the user has been added or verified to already exist, the user should then be able to view a previously created schedule (if applicable), alter a previously created schedule (by adding or removing classes; if applicable), or create an entirely new schedule. Each of these options should be clearly presented by the FYPT application in a user-friendly manner; the interactions between each functionality should be conducted in a clear, seamless fashion to preserve flow throughout the FYPT application.

After creating or making changes to a class schedule, the user should be able to view the schedule in the form of a matrix. This visualization should ensure that the FYPT application is both user-friendly and accessible.

3.4. Performance Requirements

When requesting a new plan from the system, it will redirect and display in less than 1 second.

Saving a student object will not take more than 1 second.

Any other operation (adding a course, removing a course, selecting a student to load, etc.) will not take more than 1 second.

3.5. Software System Attributes

The FYTP must be designed with simplicity, learnability, and understandability in mind.

To recall, one of the main justifications for this new system is to increase the efficiency of the course planning process for both student and advisor users. Conventionally, a student must find the time to meet with an advisor or be forced to research the courses they want to take and the possible prerequisites that may accompany such courses. As for advisors, they are often overloaded with the sheer number of students who require their assistance at a given time. For these reasons, simplicity must be at the forefront of this application. By constructing a simplistic system, all interested parties will be able to construct and view CIS schedules in a timely manner. Ensuring that the FYPT application is designed with this attribute in mind will flatten the learning curve and lessen the amount of understanding needed to create a schedule.

Currently, the process of planning and building a CIS schedule often involves a steep amount of prior research and learning. This holds true for students and advisors alike; for example, an incoming or returning student may not be knowledgeable about the many prerequisites associated with CIS courses. At the other end of the spectrum, an advisor who is not directly associated with the CIS department may not know about which courses are required for the CIS major. In these scenarios, both types of users would then be forced to research the possible courses as well as their requirements. Building a system with learnability in mind will ensure that either type of user will be able to construct or add to a schedule with little to no prior knowledge of how the system operates. Additionally, the FYPT application would store prerequisites for all classes, preventing users from needing to conduct any further research.

Lastly, the FYPT application must be designed with understandability in mind. This attribute would ensure that the application is able to transform the more abstract CIS degree requirements to visual outputs. Understandability would offer all users a way to more easily understand what classes need to be taken as well as how to move forward in the major. Visual representations of course data will help to better explain both long-term (year-to-year) and short-term (term-to-term) schedule planning and will ultimately lead to better ease of use for both types of users.

4. Acknowledgements

The SRS template was provided by Juan Flores, who updated it from Anthony Hornof for CIS 422.