**ELEC 3225: Applied Programming Concepts**

**Process Model**

### **System Overview**

* **Objective:** Design a scheduling system for a university similar to LeopardWeb.

#### **Requirements**

* **User Database:**
  + Support for 100 students, 10 instructors, and 1 admin (initial test with fewer users).
* **Course Database:**
  + Store information such as CRN, course name, times, and instructor.
* **User Types and Functionalities:**
  + **Student:**
    - Register for courses.
    - View available courses and their own schedule.
  + **Instructor:**
    - View available courses and their course roster.
  + **Admin:**
    - View and edit all courses, users, and schedules.
* **Additional Features:**
  + Support multiple semesters.
  + Print schedules.
  + Allow scheduling preferences.

#### **Class Structure**

* **Base Class: User**
  + **Attributes:** first name, last name, ID.
  + **Methods:**
    - Set functions for each attribute.
    - Function to print all information for the object.
* **Derived Classes:**
  + **Student:**
    - Additional attributes as needed.
    - Functions to search courses, add/drop courses, and print schedule.
  + **Instructor:**
    - Additional attributes as needed.
    - Functions to print schedule, print class list, and search courses.
  + **Admin:**
    - Additional attributes as needed.
    - Functions to add/remove courses, add/remove users, manage course enrollments, search and print rosters and courses.

**Waterfall Model**

|  |
| --- |
| **Software Specification:**  **Feasibility Study:**  Objective: Assess if the project is achievable with current technology and budget.   * Evaluate available technologies (databases, coding languages, UI frameworks). * Estimate project costs and budget allocation. * Conduct a risk analysis to identify potential challenges. * Evaluate UI frameworks: Assess UI frameworks and libraries suitable for creating a GUI for this project.     **Requirements Elicitation:**  Objective: Gather detailed and specific requirements for the system.   * Conduct interviews with students, instructors, and admin to understand needs. * Analyze existing systems like LeopardWeb to identify useful features. * Conduct surveys and focus groups with potential users. * Document functional and non-functional requirements.     **Requirements Specification:**  Objective: Document the requirements in a detailed and structured manner.   * Create use case diagrams and flowcharts to illustrate system functionality. * All users should be able to see all courses available in the school schedule database.   + Display a table with name, DoB, ID#, gender, race, date of semester last taken (Ex: Spring 2023 or Fall 2023, date of next semester registered for (Ex: Summer 2024), home city location, Major, credits#, and class standing (Freshman, sophomore, junior, senior, & graduate).   + Display all users of a certain major.   + Display all users from a certain class standing.   + … * Students should be able to register for classes, search classes, add/drop courses, print their schedule.   + View date of class registration   + View and set classes   + Set an alert 1 week before the classes start.   + … * Instructors should be able to print their schedule, print their class list and search for courses.   + Check and see who is registered for their courses.   + Check and update assignments.   + Set an alert if a student is added/dropped from the course.   + … * Admin roles should be able to add courses to the system, remove courses from the system, add/remove users, add/remove student from a course, search and print rosters and courses.   + Display a list a students and instructorsname, ID, role (student or instructor), scheduled days of the week, students, and corresponding major…   + Create a fair schedule (should not be more than 5 days a week).   + …   Functional Requirements:   * All users can view available courses. * Display detailed student profiles (name, DoB, ID, etc.). * Allow students to register, search, add/drop courses, and print schedules. * Instructors can print schedules, class lists, and search courses. * Admin can manage courses, users, and generate schedules. * Non-Functional Requirements: * The system should be user-friendly and accessible. * Data security and privacy must be ensured. * System performance should handle concurrent users efficiently.   **System and Software Design (Week 4-6)**  **Architectural Design:**  Objective: Define the overall structure and architecture of the system.   * Identify major system components (modules) and their interactions. * Define the architecture (ex: client-server). * Create high-level design diagrams (e.g., system architecture diagram).     **Interface Design:**  Objective: Design how users will interact with the system. I would like to create a user-friendly and intuitive interface designs   * Design user interfaces (UIs) for different user roles (students, instructors, admin). * Develop wireframes and prototypes for key screens. * Define navigation flows and user interactions.     **Component Design:**  Objective: Detail the design of individual system components.   * Identify reusable components and libraries. * Define class and object relationships. * Specify methods and attributes for each class.     **Database Design:**  Objective: Design the structure of the database.   * Identify database tables and their relationships. * Define table schemas (columns, data types, constraints). * Create ER diagrams to represent database structure. * Student Table: name, role DoB, ID#, gender, race, date of semester taken, date of next semester, home location, Major, # of credits, and class standing. * Instructor Table: name, ID, role(instructor), preferred days of the week, classes they like/prefer teaching. * Admin Table: name, ID, role(instructor) functions they serve: Functions to add/remove courses, add/remove users, manage course enrollments, search and print rosters and courses.   **Implementation and Unit Testing (Week 7-10):**  **Implementation:**  Objective: Develop system components based on design specifications.   * Write code for user classes (student, instructor, admin) with specified methods and attributes. * Implement functions for course registration, schedule management, and user administration. * Develop the database and integrate it with the application.     **Unit Testing:**  Objective: Test each component individually to ensure functionality.   * Write and execute unit tests for each function and class. * Validate individual components against requirements. * Debug and resolve issues identified during testing. * Verify and validate system components.     **Integration and System Testing (Week 11-13):**  **Integration:**  Objective: Integrate all system components into a cohesive system.   * Combine user interfaces, business logic, and database components. * Ensure seamless data flow and interaction between components and user functions. * Make sure the menu flows seamlessly to ensure smooth transition for the user. * Resolve any integration issues.     **System Testing:**  Objective: Validate the entire system through comprehensive testing.   * Perform functional testing to ensure all requirements are met.   + Each user function works completely and run with no errors. * Conduct performance testing to ensure the system handles expected loads. * Perform security testing to ensure data protection. * Conduct usability testing with end-users. * Verify and validate the system so it is ready for deployment.     **Operation and Maintenance (Week 14 onwards):**  **Deployment:**  Objective: Deploy the system for user testing and feedback.   * Prepare deployment environment. * Deploy the system on the target servers. * Conduct initial user training sessions.     **Maintenance:**  Objective: Provide ongoing support and address any issues.   * Monitor system performance and user feedback. * Fix any bugs or issues reported by users. * Implement updates and improvements based on user feedback.   **Detailed Timeline for Each Phase**  **Requirements Definition (Week 1-3):**  Week 1: Conduct feasibility study and begin requirements elicitation.  Week 2: Continue requirements elicitation and start requirements specification.  Week 3: Finalize requirements specification document.  **System and Software Design (Week 4-6):**  Week 4: Develop architectural design and interface design.  Week 5: Continue interface design and start component design.  Week 6: Finalize component design and complete database design.  **Implementation and Unit Testing (Week 7-10):**  Week 7-8: Develop user classes and implement key functionalities.  Week 9: Complete implementation and start unit testing.  Week 10: Finalize unit testing and prepare for integration.  **Integration and System Testing (Week 11-13):**  Week 11: Integrate system components.  Week 12: Conduct system testing.  Week 13: Finalize system testing and prepare for deployment.  **Operation and Maintenance (Week 14 onwards):**  Week 14: Deploy the system and conduct initial user training.  Ongoing: Provide maintenance and support. |

**Incremental Model**

|  |
| --- |
| **Version 1 (Week 1-2): Basic Functionality and Initial Data Handling**    Objectives: Establish basic functionality for listing instructors, students, and courses.  Implement initial user registration and course registration functionalities.   * Admin can list all instructors:   + Define data structure for storing instructor details.   + Implement functionality to display a list of all instructors. * Instructors can list all students in their courses:   + Define data structure for storing course and enrollment details.   + Implement functionality for instructors to view students registered in their courses. * Students upload profile information:   + Define user profile schema (name, DOB, hometown, major).   + Implement functionality for students to input and update their profile information. * Students view all courses:   + Define course schema (CRN, course name, times, instructor).   + Implement functionality to display all available courses. * Students register for classes:   + Implement course registration mechanism.   + Implement functionality to display student’s registered courses (schedule). * Basic user interfaces for listing instructors, viewing student profiles, listing courses, and course registration. * Initial database setup for users (students, instructors, admin) and courses.     **Version 2 (Week 3-4):** **Enhanced User and Course Management**    Objectives: Implement functionalities for viewing and updating user and course information. Add functionality for course dropping.   * Admin can view and update course/user information:   + Implement interfaces for admin to modify course details and user profiles. * Students view and update user/class information:   + Allow students to update their profiles and view registered courses.   + Implement functionality to display student schedules in a user-friendly format. * Instructors view and update schedule/class information:   + Allow instructors to update their teaching schedules.   + Implement functionality to display instructor schedules and class rosters. * Students drop courses:   + Implement functionality for students to drop courses.   + Ensure schedule updates dynamically upon dropping a course. * Enhanced user and course management interfaces. * Functionalities for updating and dropping courses.     **Version 3 (Week 5-6): Notification System**    Objectives: Implement a notification system for critical events like registration periods and unassigned instructors.   * Alerts one week before class registration opens:   + Implement notification mechanism for students, instructors, and admin. * Alerts for admin if students have not registered or instructors unassigned:   + Implement alerts for admin to monitor registration status and instructor assignments.      * Notification system integrated with the existing functionalities. * Alerts configured for registration periods and admin notifications.     **Final Version (Week 7-8):** **Automated Scheduling and Finalization of GUI Interface**    Objectives: Implement advanced features like automated instructor scheduling based on preferences. Finalize the system with comprehensive GUI testing and maintenance.  Automated instructor schedule generation:   * + Develop algorithm for generating instructor schedules based on availability and preferences.   + Integrate this functionality into the admin interface.   Operation and Maintenance:   * Finalize any remaining features. * Ensure all GUI elements (interface, buttons, menus, text) are implemented correctly and all functionality works seamlessly. * Perform comprehensive system testing (unit tests, integration tests, user acceptance tests). * Fix any bugs or errors.     **Proposed Timeline for Each Stage/Iteration**  Version 1 (Week 1-2): Focus on basic data handling and user interface setup. Initial functionalities for listing and registration.  Version 2 (Week 3-4): Enhance the system with updating capabilities and course dropping functionality.  Version 3 (Week 5-6): Introduce a notification system to keep users informed about critical events.  Final Version (Week 7-8): Implement advanced features and finalize the system with rigorous testing and maintenance. |

**Integrate and Configure Model**

|  |
| --- |
| **Database Integration (Week 1-2):**   * Integrate and Configure: Set up a database management system such as MySQL * Database Design: Design database tables for various factors such as users, courses, schedules, etc.   **User Interface Configuration (Week 3-4):**   * Integrate and Configure: Choose a UI framework such as React for a graphical interface or Flask & Django for a text-based interface. Then set up the necessary tools and dependencies for the chosen framework. * Interface Design: Design UI components for different functionalities such as user authentication, course registration, schedule viewing, etc. We will create mockups, or prototypes to visualize the layout and interactions.     **Component Integration (Week 5-6):**   * Integrate and Configure: Integrate backend components (database, functions) with the UI. This involves connecting the front-end and back-end parts of the application to work together seamlessly. This will make sure this runs smoothly. * Component Design: Implement APIs or controllers to handle user requests and database interactions. This includes defining endpoints, request-response formats, and data validation rules for the API. We might need an API, it will allow us to have a set of rules and protocols that allows different software applications to communicate with each other. We can use something like FastAPI.     **Testing and Validation (Week 7-8):**   * Component Testing: Hold unit tests for individual components such as API endpoints, UI components, and database functions. This make sure that each component functions correctly in isolation. * System Testing: Perform system testing to ensure proper integration and functionality of the entire system. We will need to test the end-to-end scenarios and interactions between different components. * Acceptance Testing: Validate the system against requirements and user feedback. We will do this by testing the system's usability, performance, and compliance with the project requirements. |