CPSC-6177
Design Documentation
(Version: 1.0)

By:
Ashlesha Singh
Khushi Jani
Utku Binkanat
Reid Roberts

Design Documentation for Smart Class Planning Tool

This design documentation provides a comprehensive blueprint for implementing the Smart Class Planning Tool, a Python-based standalone GUI application for academic advising. It enables developers to code the software without referencing the original requirements. The design is guided by the 13 user stories (e.g., Plan Setup Wizard, Track Selection, Export to Excel), the architecture, the data flow diagram (processes P1-P10), the program structure diagram (hierarchical components), and the five UML diagrams (Use Case, Activity, Sequence, Class, and State) that model the tool's key components and behaviors.

Key design priorities include modularity, configurability, and extensibility. Inputs such as DegreeWorks PDFs, study plans, and program maps are separated from the core software and parsed into structured repositories, allowing future tracks or input formats to be supported without code rewrites. Parsers are directly connected to repositories, ensuring extracted data (courses, prerequisites, offerings, sequences) is centralized for use by planning and validation modules.

The design emphasizes user-guided workflows and data-driven planning, supporting both students and faculty. The application will reduce advising time, prevent prerequisite errors, and ensure students can graduate on schedule. The GUI will be implemented using Tkinter for its simplicity, portability, and alignment with the standalone requirement.

Table of Contents

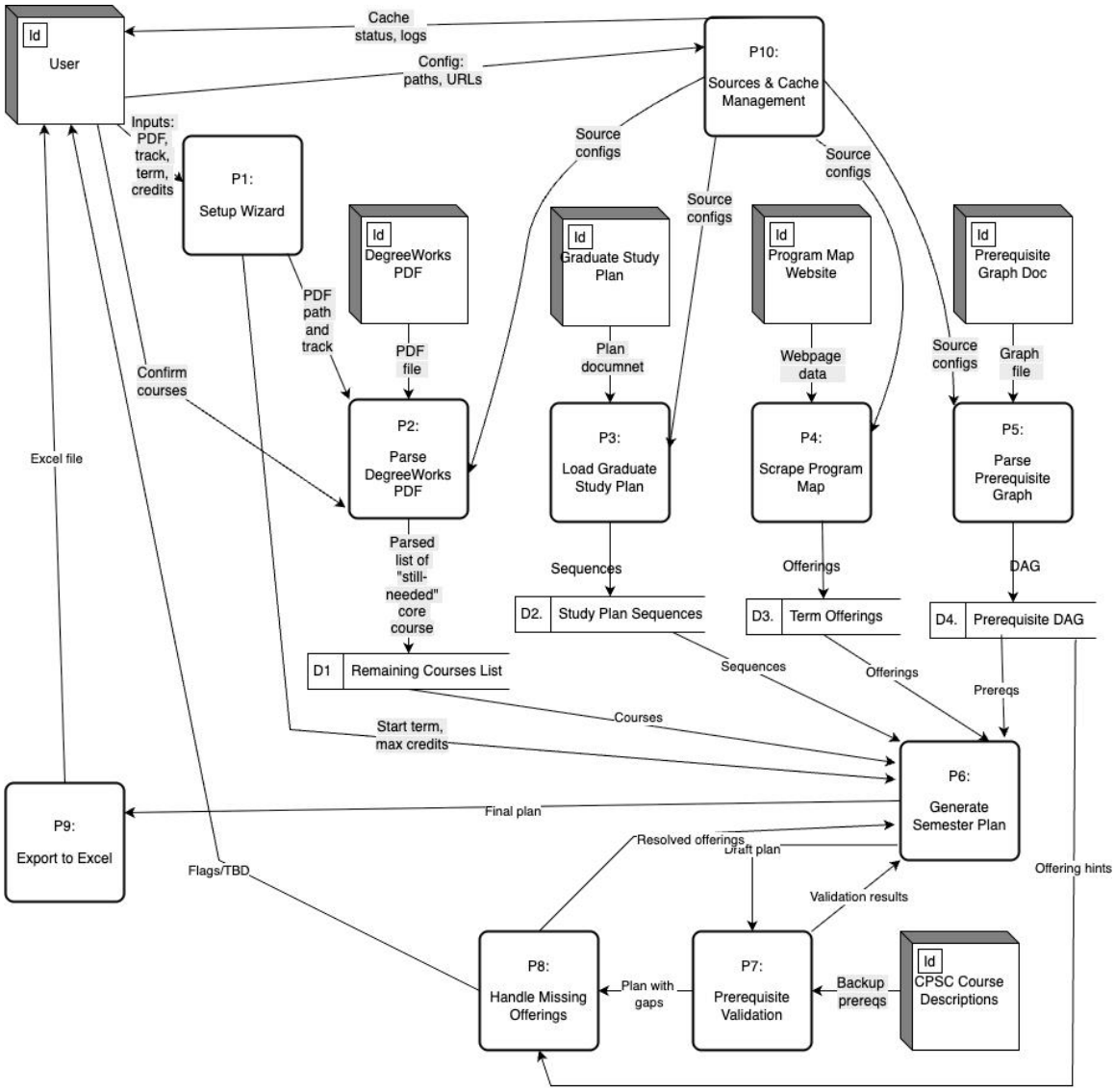
User Stories

The design is driven by the following user stories, which define the functionality and user interactions for the Smart Class Planning Tool:

- **User Story 1: Plan Setup Wizard** - As a user, I want a wizard to input a DegreeWorks PDF, select a track, choose a start term, and set max credits, so I can initiate a personalized class plan.
- **User Story 2: Track Selection** - As a user, I want to select a track (e.g., Software Systems), so the tool limits planning to core courses for that track.
- **User Story 3: Student Info Form** - As a user, I want to enter my student ID and name, so the exported plan includes these details as labels.
- **User Story 4: DegreeWorks PDF Parser** - As a user, I want the tool to parse my DegreeWorks PDF, so it identifies courses still needed for graduation.
- **User Story 5: Graduate Study Plan Loader** - As a user, I want the tool to load a graduate study plan document, so it provides course sequencing guidance.
- **User Story 6: Program Map Scraper** - As a user, I want the tool to scrape a program map URL, so it retrieves term-specific course offerings.
- **User Story 7: Prerequisite Graph Parser** - As a user, I want the tool to parse a prerequisite graph file, so it builds a dependency structure for course planning.
- **User Story 8: Prerequisite Validator** - As a user, I want the tool to validate prerequisites, so it ensures the plan respects course dependencies with a backup web crawl if needed.
- **User Story 9: Semester Plan Generator** - As a user, I want the tool to generate a semester-by-semester plan, so I can see a complete schedule based on inputs.
- **User Story 10: Missing Offerings Handler** - As a user, I want the tool to handle missing offerings, so it flags or skips unavailable courses with hints from the graph.
- **User Story 11: Excel Exporter** - As a user, I want the tool to export the plan to an Excel file, so I can share or archive it with (Term, Courses, Credits, Total).
- **User Story 12: Sources & Cache Panel** - As a user, I want a panel to manage config files and cache, so I can set paths/URLs and view last fetch details.
- **User Story 13: Track Config** - As a user, I want the tool to support new tracks via configuration, so I can plan for different programs without code changes.

These stories guide the design's focus on interactive setup, data parsing, plan generation, and export functionality.

Data Flow Diagram



The Data Flow Diagram illustrates how data moves through the Smart Class Planning Tool and how the different modules interact with one another.

- **User Interaction (Id: User):**
  The system begins with the user providing necessary inputs, such as the DegreeWorks PDF, selected track, start term, and maximum credits. The user also confirms course selections and eventually receives the final Excel file output.
- **Setup Wizard (P1):**
  The wizard serves as the entry point for user inputs. It collects configuration details (PDF path, track selection, term, credits) and passes them to the relevant parsers.

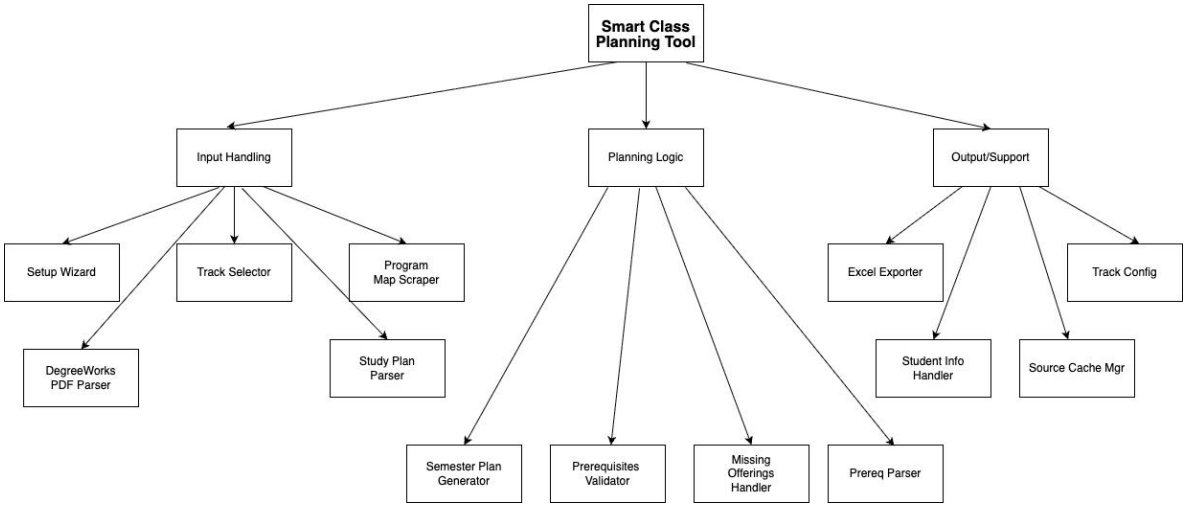
- **PDF and Plan Parsers (P2, P3, P4, P5):**
    - **P2: Parse DegreeWorks PDF** extracts the list of "still-needed" courses.
    - **P3: Load Graduate Study Plan** loads predefined sequences from the official graduate plan.
    - **P4: Scrape Program Map** gathers term offerings by crawling the CSU program map website.
    - **P5: Parse Prerequisite Graph** builds a Directed Acyclic Graph (DAG) of prerequisites.
- **Supporting Data Stores (D1–D4):**
    - D1: Remaining Courses List.
    - D2: Study Plan Sequences.
    - D3: Term Offerings.
    - D4: Prerequisite DAG.
- **Core Logic (P6: Generate Semester Plan):**
  This module merges all sources — remaining courses, study plan sequences, term offerings, and prerequisite DAG — to generate a semester-by-semester plan. It outputs a "draft plan," validates prerequisites, and adjusts as needed.
- **Validation and Exception Handling (P7, P8):**
    - **P7: Prerequisite Validation** ensures no course is scheduled before its prerequisites are completed, pulling backup data from course descriptions if needed.
    - **P8: Handle Missing Offerings** resolves scheduling gaps by marking TBD terms or suggesting alternatives.
- **Final Output (P9: Export to Excel):**
  Produces an Excel file with the recommended class plan by semester, including courses and credit totals.
- **Support Module (P10: Sources & Cache Management):**
  Maintains paths, configurations, URLs, logs, and caches for reusable inputs, ensuring extensibility and consistency across runs.

Although the DFD appears pipeline-like, this does not imply a pipe-and-filter architecture. Instead, these processes are implemented within a layered architecture:

- Parsers and scrapers (Infrastructure Layer).
- Data stores (Domain Layer).
- Planning and validation (Application Layer).
- Export and interaction (Presentation Layer).

Thus, the DFD provides a process view, while the layered architecture provides a structural view of the system.

Program Structure Diagram



The Program Structure Diagram depicts the logical grouping of system modules into three main categories: Input Handling, Planning Logic, and Output/Support.

- **Input Handling:**
  This group is responsible for ingesting and preparing all the raw input data.
  - **Setup Wizard**: Guides users to provide DegreeWorks PDF, track selection, start term, and credits.
  - **Track Selector**: Restricts scope to the appropriate degree track.
  - **DegreeWorks PDF Parser**: Extracts remaining required courses.
  - **Program Map Scraper**: Collects course offering details from the website.
  - **Study Plan Parser**: Loads sequencing guidance from the graduate study plan.
- **Planning Logic:**
  This group contains the core computational and validation modules.
  - **Semester Plan Generator**: Creates the term-by-term plan by merging prerequisites, offerings, and study plans.
  - **Prerequisites Validator**: Ensures that prerequisite constraints are respected.
  - **Missing Offerings Handler**: Flags or manages unavailable courses.
  - **Prereq Parser**: Translates prerequisite documents into a usable DAG format.
- **Output/Support:**
  This group ensures persistence, configurability, and presentation of results.
  - **Excel Exporter**: Produces the final recommended class plan in spreadsheet format.
  - **Student Info Handler**: Manages metadata such as student name/ID for labeling.
  - **Track Config**: Stores reusable configurations for tracks, enabling extensibility.
  - **Source Cache Manager**: Manages input sources, caches data, and provides logs for traceability.

This structure complements the layered architecture by showing how modules are grouped functionally. It ensures **modularity** and **clear responsibilities**.

Architecture

Overview

The system is structured using a **Layered Architecture with MVC principles**. This pattern was chosen because it ensures clear separation of concerns, enhances maintainability, and allows for future extensibility (e.g., new tracks or input formats).

- **Presentation Layer (View):** Provides user interaction via a Tkinter GUI (Setup Wizard, Track Selector, Student Info Form) and exports plans to Excel.
- **Application Layer (Controller):** Implements the core business logic, including the Semester Plan Generator, Prerequisite Validator, and Missing Offerings Handler.
- **Domain Layer (Model):** Maintains structured data models such as Course Repository, Study Plan Sequences, Term Offerings, and the Prerequisite DAG.
- **Infrastructure Layer:** Contains parsers and scrapers (DegreeWorks PDF Parser, Study Plan Parser, Program Map Scraper, Prerequisite Graph Parser) as well as cache management.

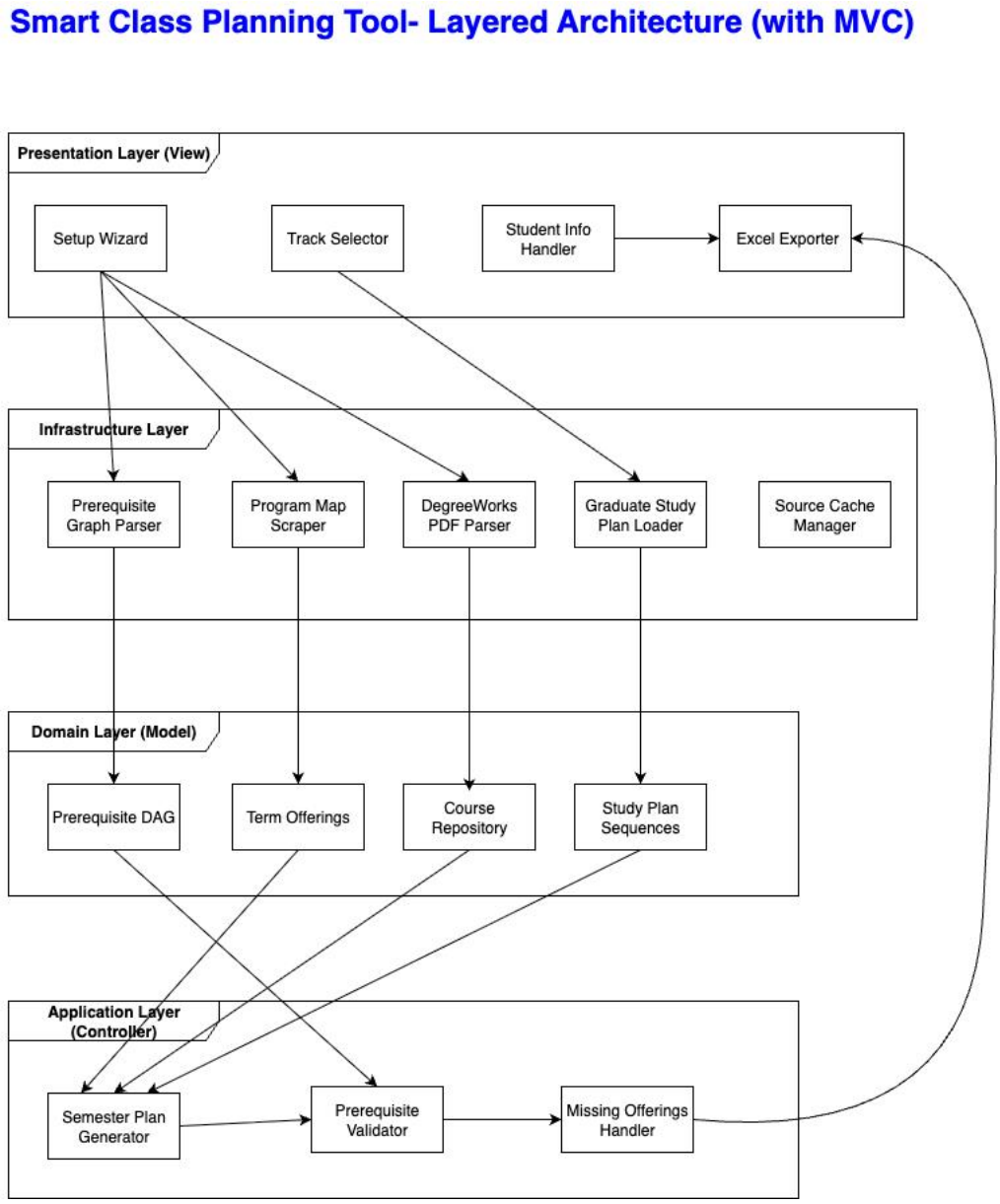
Decisions:
The layered design isolates GUI logic from data parsing and validation, making the system modular. Adding a new parser affects only the infrastructure layer, while adding a new track involves updating configuration in the domain layer.

Trade-offs:
The separation of layers requires well-defined interfaces, which adds some upfront complexity. However, the long-term benefits in testability, extensibility, and maintainability outweigh the initial cost.

Alternatives Considered:

- **MVC:** Useful for GUI-heavy apps, but Tkinter's event model already blends view/controller behavior.
- **Pipe-and-Filter:** Matches the DFD's sequential data flow but does not reflect user interaction loops.
- **Repository & Client-Server:** Rejected because the tool processes diverse local files and runs as a standalone application.

Architecture Diagram

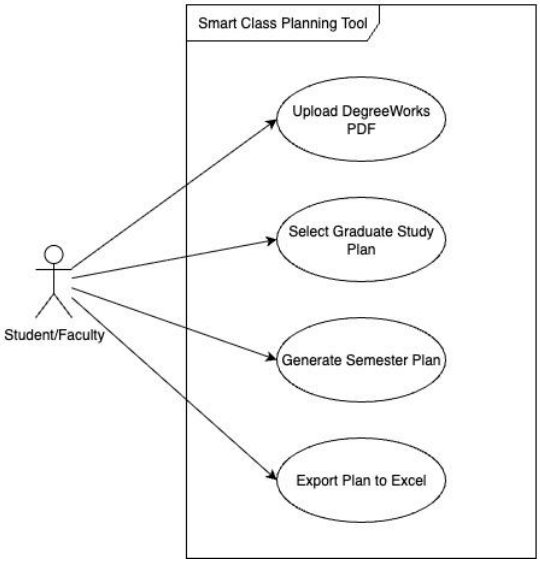


The Architecture Diagram highlights the four layers and their dependencies. The Presentation Layer interacts with the Infrastructure Layer for input parsing. Parsed data is stored in the Domain Layer, which the Application Layer consumes to generate validated plans. Finally, the Presentation Layer delivers the result to the user as an Excel file.

UML Diagrams

1. Use Case Diagram

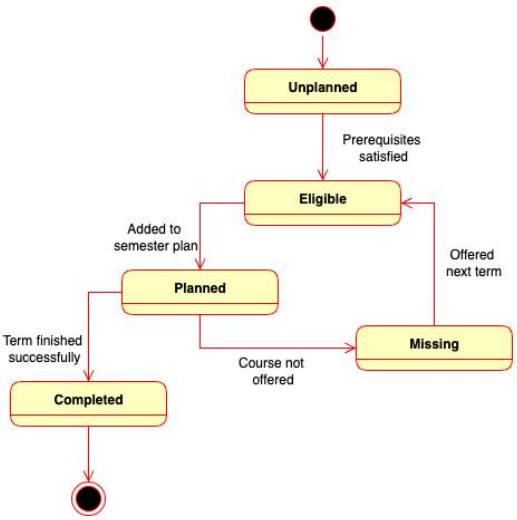
Use Case Diagram - Smart Class Planning Tool



The Use Case Diagram models external interactions. Students/Faculty interact with the system to upload DegreeWorks reports, select study plans, generate schedules, and export results.

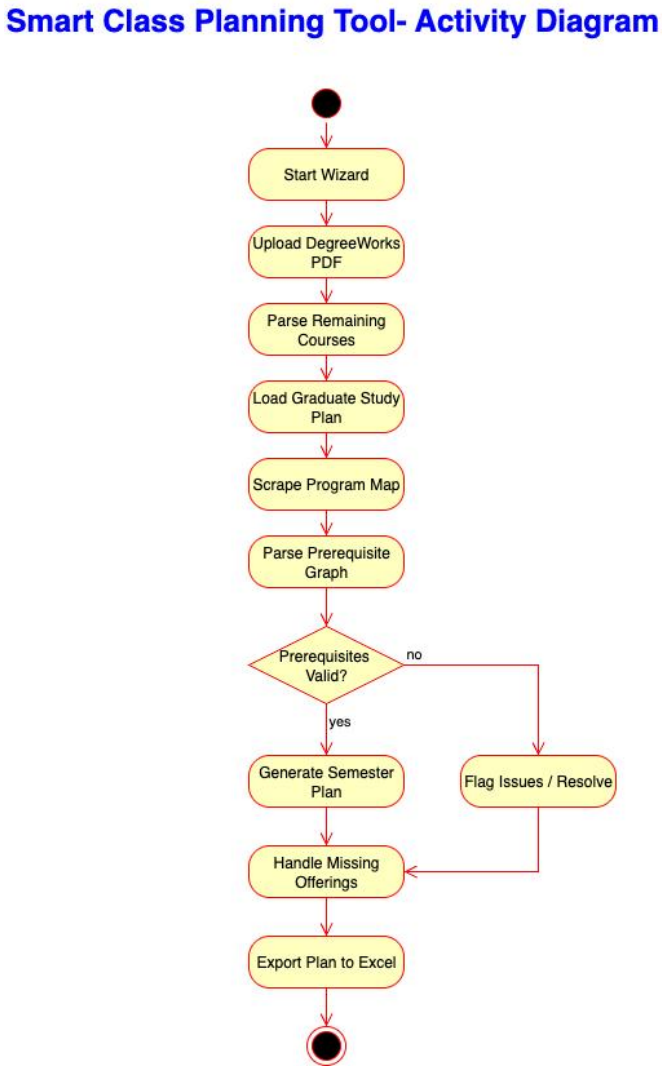
2. State Diagram

State Diagram - Course Lifecycle in Planning



The State Diagram models the lifecycle of a course. Courses begin as Unplanned, become Eligible once prerequisites are satisfied, and transition to Planned when scheduled. Completed courses transition to Completed. If a course is unavailable in a term, it moves to Missing, looping back to Eligible when offered again. This model ensures realistic scheduling behavior.

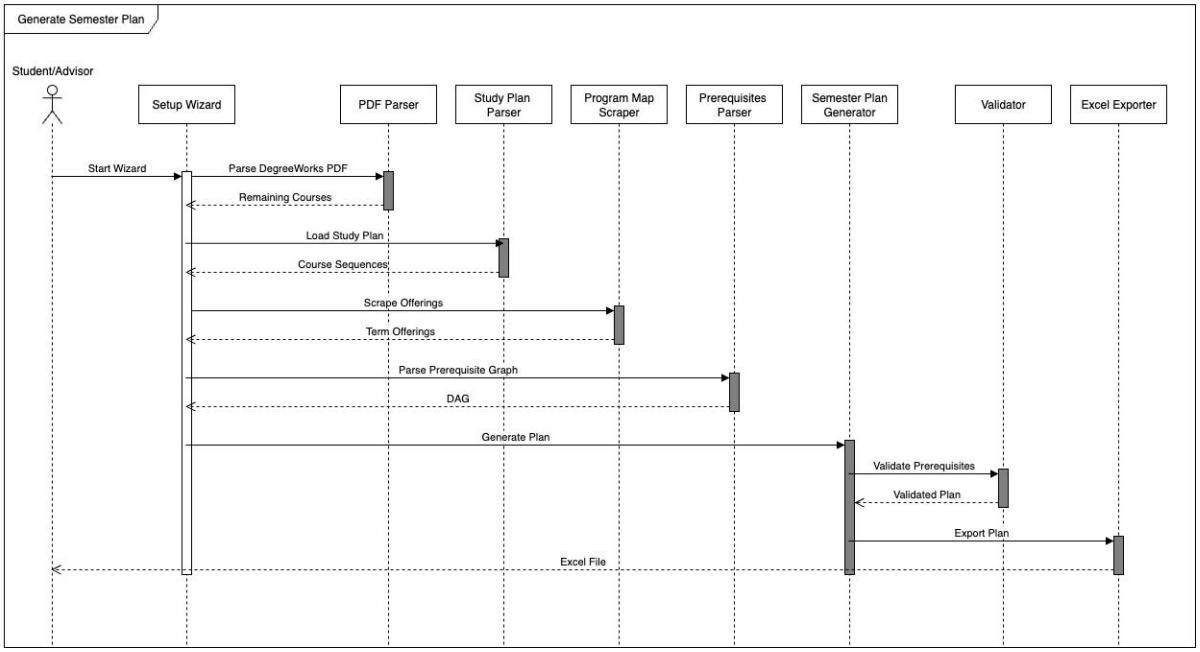
3. Activity Diagram



The Activity Diagram illustrates the workflow of the tool. After uploading and parsing inputs, prerequisites are validated. If valid, the plan is generated; otherwise, issues are flagged. Missing course offerings are handled gracefully, and the final plan is exported. This model emphasizes the control flow and decision points in the system.

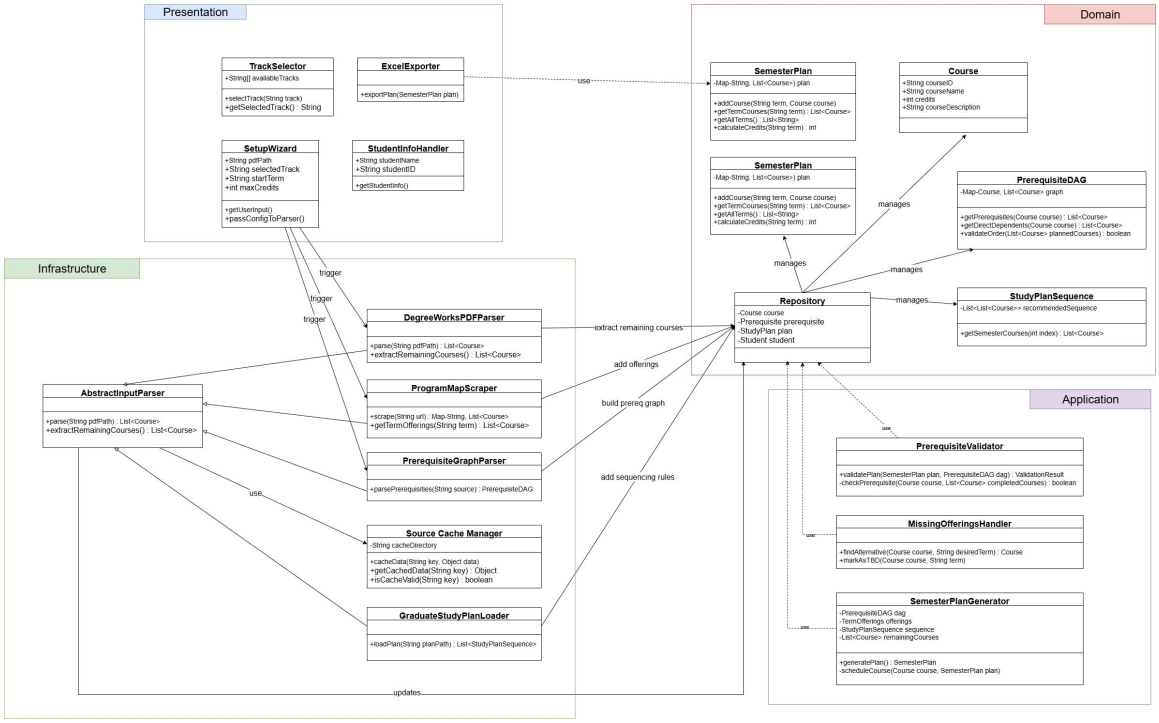
4. Sequence Diagram

Sequence Diagram - Generate Semester Plan



The Sequence Diagram describes the time-ordered interactions during semester plan generation. The Setup Wizard orchestrates calls to parsing modules, which return structured data. The Semester Plan Generator uses this data, interacts with the Validator, and passes the completed plan to the Excel Exporter for final output. This diagram clarifies module dependencies and runtime behavior.

5. Class Diagram



The Class Diagram gives a static structural view of the system's architecture and defines the key entities, their attributes, operations, and relations. Classes are organized into Presentation, Infrastructure, Domain, and Application layers to reflect the layered architectural design. The Domain layer contains core business objects like Course, Semester Plan, and Student Plan. It is centered around the Repository class, which is used to build and manage other components (Prerequisite DAG, Term Offerings, Semester Plan). The Infrastructure layer contains parsers and scrapers, which all inherit from the common class Abstract Input Parsers. The Application layer contains the Semester Plan Generator, Missing Offerings Handler, and the Prerequisite Validator, which all get data from the Repository for their planning and validating operations. The Presentation layer handles all user interactions and provides the final Excel output. The infrastructure services populate domain objects, which are used by the application logic to plan and validate. This diagram serves as a blueprint to implement clean, modular code so all system components have well-defined operations and interactions.

References

- Lecture resources
- Sommerville, I. (2016). *Software Engineering* (10th Ed.). Pearson.
- IBM. *Data Flow Diagram – Topic Guide*. https://www.ibm.com/think/topics/data-flow-diagram
- https://www.atlassian.com/work-management/knowledge-sharing/documentation/software-design-document

Acknowledgement of AI Assistance

This document was refined with the assistance of **ChatGPT**, an AI language model developed by OpenAI. The tool was used to improve grammar, clarity, and structure, and to provide suggestions for more professional phrasing where appropriate. All AI-generated content was carefully reviewed, edited, and adapted to ensure accuracy, originality, and alignment with the objectives of this work.