Software Design

January 27, 2025

Lumberjack Balancing

Project Sponsor: Dr. Scot Raab

Project Mentor: Paul Deasy

Team Members: Riley Burke, Cristian Marrufo, Sergio Rabadan, Braden Wendt

Table of Contents

Introduction

Faculty workload management is a critical administrative task in higher education, ensuring teaching responsibilities are equitably distributed in compliance with institutional policies. At Northern Arizona University (NAU), this process is currently performed manually by associate deans, requiring extensive time and effort while being prone to errors due to the complexity and volume of data involved. The Lumberjack Balancing project aims to automate faculty workload calculations through a software solution that improves efficiency, accuracy, and adaptability. Our proposed system is a Python-based desktop application designed to streamline the workload assessment process. It will automate data extraction from Microsoft Excel files, apply workload policies dynamically, and generate accurate workload reports with minimal manual intervention. The application will feature a user-friendly interface, enabling non-technical administrative staff to easily upload, process, and review faculty workload data. A core innovation of this system is its customizable algorithm, which allows associate deans to update workload policies directly within an Excel sheet without requiring code modifications. This ensures the system remains flexible and adaptable to changes in university policies over time.

Key User-Level Requirements

- **Automated Workload Calculation:** Faculty workload assignments will be computed based on predefined university policies.
- **User-Friendly Interface:** The system will offer intuitive file upload, calculation execution, and result generation functionalities.
- **Customizable Workload Rules:** Workload distribution parameters will be **configurable through an Excel sheet**, allowing for seamless adjustments.
- **Real-Time Error Checking:** The system will validate data inputs to prevent inconsistencies and errors in workload distribution.
- **Comprehensive Reporting:** Workload summaries will be automatically generated and available for export in Excel format.

The Lumberjack Balancing project presents an innovative and necessary solution to a longstanding administrative challenge at NAU. By automating faculty workload calculations, reducing errors, and improving efficiency, this system will enhance the workload assignment process, allowing associate deans to allocate their time more effectively. With its customizable framework, intuitive interface, and robust error validation, this solution will provide long-term benefits to NAU’s faculty workload management, fostering a more balanced and transparent academic environment.

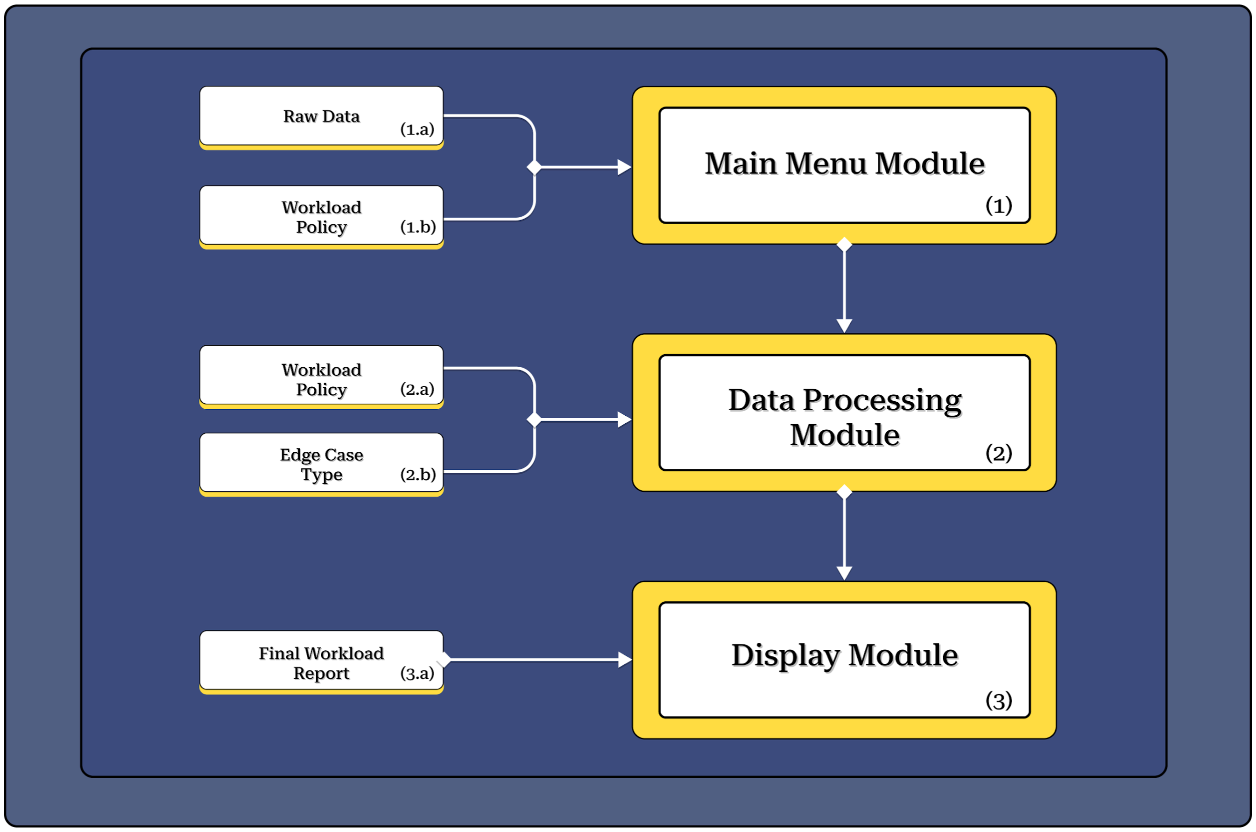
Implementation Overview

The Lumberjack Balancing software application has been designed with the ultimate purpose of automating and expediting the faculty workload management process, circumventing the extensive manual efforts currently in effect. We have designated Python as the primary programming language that will serve as the computational foundation for the application as it provides many benefits including ease of use, versatility, and most importantly, a vast selection of data processing libraries and overall capabilities. The application will implement a custom-tailored algorithm which consists of several components that work in congruence in order to process the input data and report the necessary information before it can be displayed or exported. The format of the data that is extracted from the university's database systems is expected to be in Microsoft Excel files and should include relevant information pertaining to each faculty member such as academic responsibilities, class curriculum, and expected credit hours based on the type of class the given faculty member is required to teach. The first step of our algorithm is tasked with aggregating the primary constants critical for the workload calculation process. As part of this procedure, data is first parsed and collected, discarding any irregular or unnecessary values that would hinder the efficiency of the operational calculations. To aid in this step, the pandas python library will be employed to its full potential, allowing us to manipulate and analyze data while remaining flexible enough to account for dynamic changes in the workload policy algorithm. The next component of this process will then identify certain predetermined edge cases based on the raw input data, prompting special operations that will calculate the workload percentage according to specific workload policies that can be dynamically altered as desired. The final component of this algorithm will store the critical variables in their respective data structures while also delegating specific markers that indicate if the final workload percentage falls below or above the designated threshold. Throughout this sequence of steps, data is periodically examined and compared to ensure it remains in accordance with the provided workload policies and to reduce any potential errors that could ruin the integrity of the workload distribution process. In our efforts to improve the efficiency and usability of the software, we have concluded that a simple and comprehensive user interface is to be implemented. An accessible design that follows inherent human-computer interaction principles will allow any user to quickly and easily understand the software’s functionality and ultimately reduce any potential technical skills requirements that might be introduced otherwise. In order to achieve this, the tkinter python library has been identified as the optimal tool for the development of the user interface structure.

Architectural Overview

The software architecture of the Lumberjack Balancing application is divided into three main modules or layers, ensuring modularity and some degree of abstraction from the previous layers. Each module is intended to conduct a specific function that will then support or inform the next module in the architecture while remaining independent for the most part. The layers primarily communicate at the point where the main functionality of the system is activated, allowing the transfer of raw data selected by the user at the user interface level, eventually computing and transforming the data into a workload report which only displays what is necessary. The modular architecture design will prevent any given issues that might occur at one layer of the system from propagating further, this in turn simplifies the error detection and software modification process associated with software maintenance.

Software High-Level Architecture Diagram:



1. **Main Menu Module:**
   The initial module or layer of the architecture which serves as the primary method of accessing the software's functionality. The settings sub-menu and the data processing feature can be accessed from here.

   1.a) Raw Data:
   The raw data, composed in the excel file format, is input by the user in this module. The data is validated to ensure the correct file format is in place before further processing.

   1.b) Workload Policy:
   This component assumes the excel file format and details necessary variables that dictate the mathematical and computational processes of the data processing module. The user is able to upload this file into the software's system as indicated by the user interface before moving on to the next module.

2. **Data Processing Module:**
   The most critical component of the system where the raw data is filtered and processed in accordance to the workload policies previously defined. Certain edge cases that will modify the computational algorithm are accounted for in this module. This module will be essentially invisible to the user after the adequate input data has been received.

   2.a) Workload Policy:
   The workload policy variables are applied and measured against in this module. The validation process for this component is designed to be dynamic, accounting for future changes in the workload policy.

   2.b) Edge Case Type:
   Edge cases determine how the algorithm calculates and processes the data. The type of edge case possessed by the data is identified in this module.

3. **Display Module:**
   The final layer of the modular software architecture. This component is in charge of displaying the final data values presented in a workload report format. Visual representations of workload thresholds can be observed in this module. Additionally, the user is able to export the workload report in the excel file format here.

   3.a) Final Workload Report:

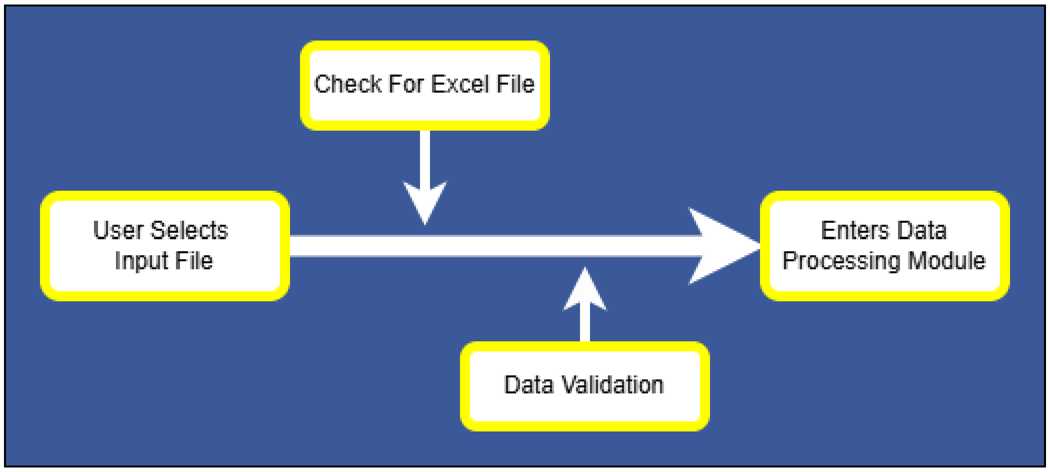
The workload report which contains the finalized workload distribution data points. The visual representation is able to be sorted as requested by the user.

Module and Interface Descriptions

Main Menu Module

a) The Main Menu Module serves as the primary interaction point for users. It facilitates the importation of raw data and workload settings into the system. The module handles user input, initiates data validation, and routes the selected input files to the Data Processing Module. It ensures that the data is correctly formatted before processing.

b) X



c) Public Interface - Methods:

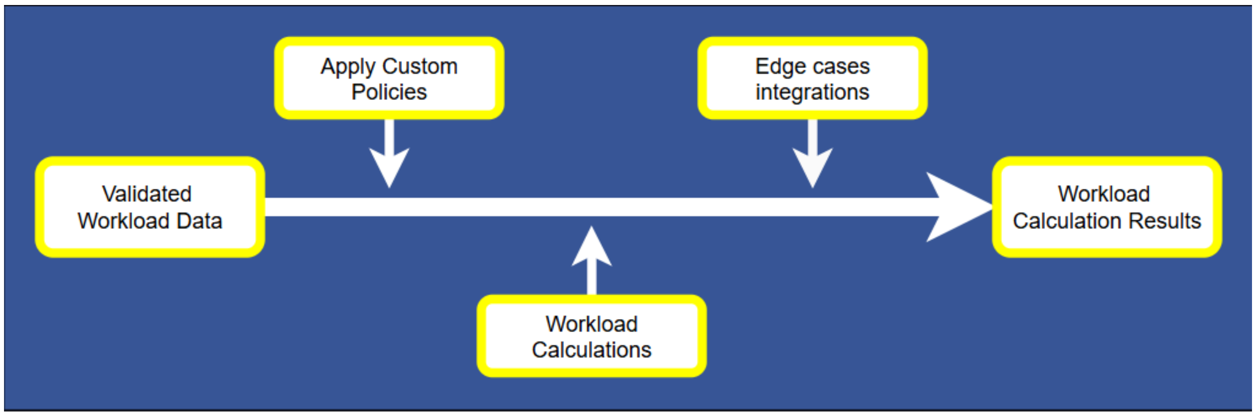
- uploadRawData(file_path: str) -> bool
  Uploads faculty workload data from an Excel file.
  Parameters:
  - file_path (str): Path to the input Excel file.
    Returns:
  - bool: True if the file was successfully uploaded, False otherwise.
- uploadCalculationTable(file_path: str) -> bool
  Uploads the Calculations Table for the algorithm
  Parameters:

    - file_path (str): Path to the input workload report file.
    Returns:
    - bool: True if the file was successfully uploaded, False otherwise.
- validateData() -> dict
  Performs initial validation on uploaded files to ensure proper format.
  Returns:
    - dict: Dictionary containing validation status and errors, if any.
- proceedToProcessing() -> None
  Sends validated data to the Data Processing Module for further calculations.

Data Processing Module

a) The Data Processing Module is the core computational engine of the system. It takes validated workload data, applies custom workload policies, and accounts for edge case types to compute faculty workload allocations. This module runs the workload calculation algorithm, ensuring that each faculty member's assigned workload adheres to NAU's institutional policies. Once processing is complete, the module prepares the Final Workload Report to be displayed.

b)

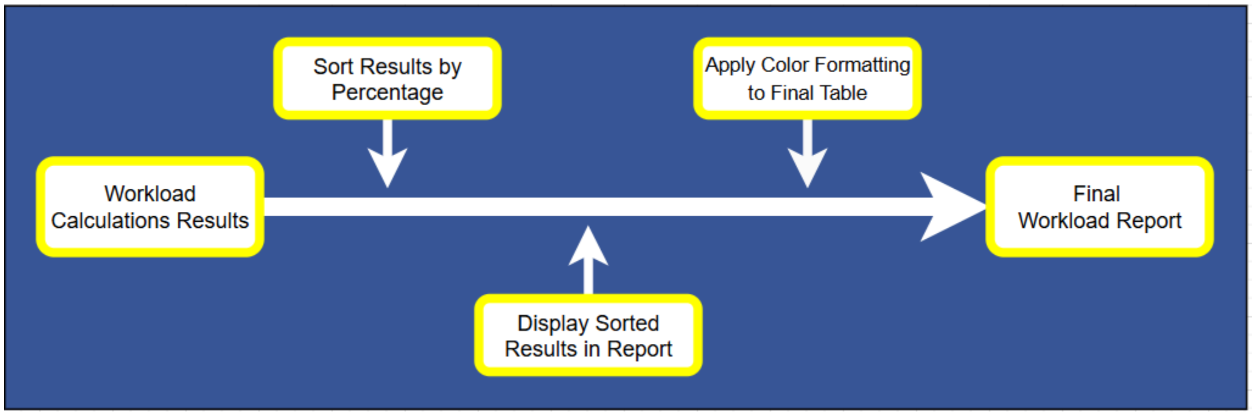


c) Public Interface - Methods:

- loadWorkloadPolicy(file_path: str) -> bool
  Loads an Excel file containing workload policies defining calculation rules.
  Parameters:
  - file_path (str): Path to the policy file.
    Returns:
  - bool: True if policies were successfully loaded, False otherwise.
- loadEdgeCases(file_path: str) -> bool
  Loads a dataset of predefined edge cases for special workload conditions.
  Parameters:
  - file_path (str): Path to the edge case file.
    Returns:
  - bool: True if edge cases were successfully loaded, False otherwise.
- computeWorkload() -> dict
  Executes the faculty workload calculation algorithm.
  Returns:
  - dict: Dictionary containing faculty workload assignments, warnings, and discrepancies.
- exportFinalWorkloadReport(file_path: str) -> bool
  Saves the final calculated workload report in an Excel format.
  Parameters:
  - file_path (str): Path where the report should be saved.
    Returns:
  - bool: True if the report was successfully saved, False otherwise.

Display Module

a) The Display Module is responsible for presenting the final workload report to users. It enables visualization of workload summaries, including graphical indicators for faculty workload distribution (e.g., underloaded, balanced, or overloaded). This module also provides export functionalities for generating Excel reports.

b) 
Sort Results by Percentage
Apply Color Formatting to Final Table
Workload Calculations Results
Final Workload Report
Display Sorted Results in Report

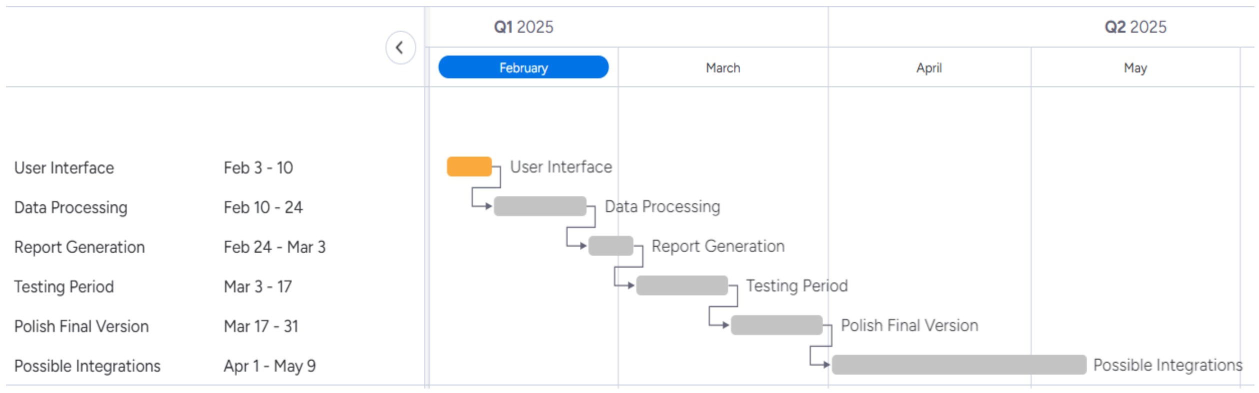
c) Public Interface - Methods:

- loadFinalReport(file_path: str) -> bool
  Loads the final computed workload report for display.
  Parameters:
  - file_path (str): Path to the generated report file.
    Returns:
  - bool: True if the report was successfully loaded, False otherwise.
- displayWorkloadSummary() -> None
  Visualizes key workload insights in a tabular or graphical format.
- highlightDiscrepancies() -> list
  Identifies and lists discrepancies in workload distribution.
  Returns:
  - list: List of faculty members with discrepancies.
- exportReport(file_path: str) -> bool
  Allows users to export the report in a preferred format (e.g., Excel, PDF).
  Parameters:
  - file_path (str): Destination path for the exported report.
    Returns:
  - bool: True if export was successful, False otherwise.

This modular architecture ensures separation of concerns, allowing each component to specialize in a specific function while integrating seamlessly with others. The Main Menu Module handles user interactions and data validation, the Data Processing Module manages computations and policy application, and the Display Module presents results in an understandable manner. This design supports scalability, customizability, and ease of use, aligning with the project's objectives. This also allows for easy updating for the future stretch goals we plan to implement.

Implementation Plan

Our implementation timeline focuses on developing each core module of the Lumberjack Balancing application in stages, with room for testing and integration at each step. Figure 1 shows our Gantt chart, which spans from early February through mid-April. The chart identifies five main phases of development, each reflecting major modules or milestones in the software architecture:



User Interface	Feb 3 - 10
Data Processing	Feb 10 - 24
Report Generation	Feb 24 - Mar 3
Testing Period	Mar 3 - 17
Polish Final Version	Mar 17 - 31
Possible Integrations	Apr 1 - May 9

Phase 1: User Interface (Feb 3 - Feb 10)

Objectives:

- Implement a basic but functional GUI
- Ensure simple navigation and file-upload functionality
- Validate that the interface can eventually communicate with the backend

Phase 2: Data Processing (Feb 10 - Feb 24)

Objectives:

- Develop the parsing logic that reads Excel data and applies preliminary workload calculation steps
- Integrate PANDAS and related data libraries and handle large datasets efficiently
- Implement consistent error handling for missing/corrupted data

Phase 3: Report Generation (Feb 24 - Mar 3)

Objections:

- Implement functionality to produce comprehensive workload reports in Exel format
- Design adaptable report templates that reflect adjustable policies
- Create a user-triggered workflow, so that the UI can request a final report (or directly generate it)

Phase 4: Testing Period (Mar 3- Mar 17)

Objectives:

- Conduct unit, integration, and user acceptance testing across all modules

- Verify that the data algorithms follow the policies accurately
- Validate the UI ability to handle various edge cases

Phase 5: Polishing Final Version (Mar 17 - Mar 31)

Objectives:

- Address bugs and feedback uncovered during testing
- Refine UI for better usability and clarity
- Finalize documentation

Phase 6: Possible Integrations (Apr 1 - May 9)

Objectives:

- Identify and evaluate additional feature requests from the client
- Implement enhancements based on user feedback (if feasible)
- Conduct targeted testing on new integrations to ensure stability

Conclusion

The Lumberjack Balancing project represents a significant step forward in automating faculty workload management at Northern Arizona University (NAU). By replacing the existing manual process with a Python-based software solution, we aim to enhance efficiency, reduce errors, and provide a scalable, adaptable system for workload calculation and reporting. Through its modular architecture, the application ensures that faculty workload data is accurately processed, dynamically adjusted according to institutional policies, and presented in an intuitive, user-friendly format. Each component of the system—the Main Menu Module, Data Processing Module, and Display Module—has been carefully designed to separate concerns, allowing for easy maintenance and future scalability. The Main Menu Module serves as the primary interface for users, ensuring seamless data input and validation. The Data Processing Module executes the core workload calculation algorithm, dynamically adjusting assignments based on workload policies and special conditions. Finally, the Display Module presents the computed workload distributions in a structured format, providing faculty and administrators with actionable insights. The implementation plan outlines a phased development approach, ensuring that each component undergoes rigorous testing and refinement before full integration. With milestones set for user interface development, data processing logic, report generation, and extensive testing, we are committed to delivering a robust and user-friendly solution that meets the needs of NAU's administrative staff. Additionally, by designing the system with flexibility in mind, future integrations and enhancements can be incorporated to address evolving university policies and administrative needs.

Ultimately, Lumberjack Balancing will transform faculty workload management by automating calculations, reducing administrative burdens, and improving accuracy. This initiative not only streamlines faculty workload distribution but also fosters greater transparency and efficiency, ensuring that NAU's faculty members receive equitable workload assignments that align with institutional policies. With its emphasis on customizability, usability, and scalability, this system is poised to become an indispensable tool for faculty workload management at NAU.