Tech Feasibility

October 15, 2024

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

The associate deans of each college at Northern Arizona University oversee faculty workload assignments to ensure the effective alignment with university policies. This crucial task involves calculating workload percentages of hundreds of faculty members based off of an even greater number of classes. Our client, Dr. Scot Raab, along with other associate deans, manually calculate the workloads, and the sheer volume of data makes this process highly complex for manual calculation. The current manual method not only demands considerable time and effort but also increases the likelihood of errors, given the intricate nature of faculty and class assignments. To address these challenges, we propose developing an application designed to automate the workload calculations, thereby significantly reducing the time and effort required by the associate deans. This solution will not only enhance efficiency but also improve accuracy in workload assessment, allowing the deans to focus on other critical administrative duties. The application will be user-friendly, requiring only the input of raw data and a reference sheet—both in Microsoft Excel format—and will generate a detailed, annotated workload assessment report. This report will provide a comprehensive and accurate overview, ensuring that the faculty workload is managed in a streamlined and effective manner.

To overcome the complexities of manual workload calculations and provide a streamlined, automated solution for Northern Arizona University, several technological challenges must be addressed. Each of these challenges requires a tailored approach to ensure the system is efficient, flexible, and user-friendly. The following sections outline these challenges and the strategies we propose to solve them.

Technological Challenges

We will need to:

Determine what data points are crucial for calculations:

- This involves identifying, extracting, and structuring various types of information, such as classes taught, and the assigned credits and amount of enrolled students, and their faculty track (career or tenure). We must also consider the diversity of classes, including lab sessions, lectures, seminars, and other formats, as each carry different weights and impact workload calculations differently. The challenge lies in ensuring that we gather all relevant data while filtering out any extraneous information that could skew the results or complicate the process. Additionally, we need to pay close attention to the instructor role, as we only need to consider the PI (primary instructor).

Determine the best method for parsing and condensing the data:

- These documents contain numerous extraneous columns that must be filtered out to focus on relevant information. The process will begin with an automated filtering mechanism to remove non-essential data, such as unrelated administrative details. This will streamline the dataset for subsequent processing. Next, the application will address special cases, in which the assignment values are pre-determined and loaded into the application by the user. These cases will require customized parsing rules and conditional logic to ensure accurate workload calculations. After handling these complexities, the system will move on to processing standard classes using predefined formulas, factoring in course type, credit hours, and faculty status.

Create a highly customizable algorithm to ensure maintainability:

- The algorithm must be designed to dynamically determine class workload weights based on an external Excel sheet that contains a table of conditions and corresponding weights. This approach will allow for flexibility, as associate deans can update or modify the conditions and weights directly in the Excel sheet without needing to alter the underlying code, which is highly valuable to our client. The algorithm will read the Excel sheet, parse the conditions (e.g., class type, credit hours), and match them with the associated weights. This modular design will ensure that any updates to the conditions or weights are immediately reflected in the calculations, minimizing the need for future code revisions. We will also provide guidelines for formatting the Excel sheet to ensure consistency and minimize errors when updating the conditions or weights.

Design a simple user interface to promote a seamless workflow:

- The UI will include basic functions such as file upload buttons for the Excel files, a settings button to check the table for calculations, and a run calculation button. Clear labels, minimal menus, and a status indicator will guide users, making the process straightforward and efficient. Tooltips and brief instructions will be incorporated to provide quick guidance, minimizing the need for training. This minimalistic design will allow for easy updates and modifications without complicating the user experience, keeping the focus on efficiency and usability.

Having outlined the key technological challenges, it is essential to evaluate various approaches to address each issue effectively. The following technology analysis examines potential solutions, comparing their strengths and weaknesses to determine the most suitable methods for integrating these components into a cohesive and efficient system.

Technology Analysis

Data Identification

Issue Introduction:

- The primary function of the product we are attempting to develop is to conduct a series of what are essentially mathematical calculations provided a pre-organized data set, however, although the operations themselves might be relatively simple, one of the challenges we expect to encounter as we manipulate and analyze the data involves identifying the adequate data points that will be necessary to produce the results we want. More specifically, we need to develop an algorithm that is able to parse through all the data provided in order to effectively distinguish between any unnecessary information and locate the specified data points that will be utilized in the program's primary operations.

Desired Characteristics:

- To resolve this particular issue, the ideal solution needs to satisfy various characteristics we have identified and deemed crucial for the purposes of our project. As our program is expected to not only extract but also analyze a vast amount of data, irrespective of size and uniformity, we require the tools to **filter through and extract certain variables** from the data set. These crucial data points will then need to be stored into data structures that should optimally allow for easy access while also maintaining readability. Additionally, this process must be able to be conducted through its entirety **without being hindered by irregular data**, such as specific edge cases, or the size of the data set, since we assume the information given will change frequently as workload is assigned or altered in various ways.

Alternatives:

- We were able to find a variety of different programming languages and their respective utility libraries which we considered as potential solutions to the issues of analyzing and manipulating data, particularly as it pertains to parsing through large data sets. The following are the most promising alternatives we identified:

    - **SQL:** SQL (Structured Query Language) is a domain-specific language designed to manage and query relational databases. It remains essential for extracting and manipulating data stored in databases. Developed in the early 1970s by Donald D. Chamberlin and Raymond F. Boyce at IBM, SQL was based on Edgar F. Codd's relational model. By the late 1980s, SQL was standardized by ANSI and ISO.
    - **R:** R is a programming language and software environment specifically designed for statistical computing and graphics. It is widely used in academia, research, and industries that require heavy statistical analysis. Created by **Ross Ihaka** and Robert Gentleman in 1993 at the University of Auckland, New Zealand, R was designed as a free, open-source alternative to the proprietary S language, used for statistical analysis.
    - **Java:** Java is a general-purpose, object-oriented programming language known for its portability, performance, and scalability. It's a language of choice for many enterprise-level data applications. Developed by James Gosling at Sun Microsystems (released in 1995), Java was designed to have as few implementation dependencies as possible, allowing it to run on any system capable of running the Java Virtual Machine (JVM).
    - **Python:** Python is a high-level interpreted programming language, notorious for being exceptionally simple and understandable. It's open-source and has extensive libraries that support various forms of data manipulation, analysis, and visualization. Created by Guido van Rossum and first released in 1991, Python was initially designed as a general-purpose language, focusing on code readability. Its simplicity made it quickly adaptable for data science tasks.

Analysis:

- **SQL**
    - **Efficient for querying large datasets:** can be incredibly useful and effective when querying and aggregating data from relational databases in a speedy fashion.
    - **Standardized language:** widely used and supported across various database systems including MySQL, PostgreSQL, and Microsoft SQL.

    - **Limited for complex analysis:** effective for basic data manipulation, however, it seems to encounter some limitations using more complex data analysis algorithms.
    - **Issues handling unstructured data:** lacks compatibility with unstructured data types such as text, images, or JSON for web implementation purposes.
- **R**
    - **Designed for data analysis:** a free open source programming language specifically built for statistical analysis and data visualization.
    - **Extensive libraries:** has a wide range of packages useful for a variety of purposes, such as statistical computing, visualization, and even machine learning.
    - **Slow performance:** compared to other languages, R is comparably slower when working with larger datasets.
    - **Lacks flexibility:** due to being specifically designed for data analysis applications, it lacks the ability to be implemented in more versatile use cases compared to other languages.
- **Java**
    - **High performance and scalability:** notoriously fast when handling larger datasets or any application that necessitates high performance.
    - **Strong type system:** strongly typed languages enforce strict rules that help in preventing errors when compiling.
    - **Limited libraries:** when compared to other languages, java appears to possess a lower amount of libraries for data analysis, which could be detrimental to the functionality of our application.
    - **Memory management:** requires active memory management and garbage collection, which could inevitably lead to complication when managing data.
- **Python**
    - **Easy to learn and use:** syntax is clear, concise, and easy to understand irrespective of experience with programming.
    - **Large set of libraries:** possess a vast array of libraries specifically designed for data analysis, in particular pandas.
    - **Versatile:** general purpose programming language, meaning the applications of the languages are not restrictive.

    - **Handling large datasets:** can handle large datasets efficiently and effectively.

Chosen Approach:

- Through the process of extensive deliberation we identified several candidates for which programming language would best be suited for the application of data manipulation and analysis, by comparing the alternatives and their effectiveness for implementation in our use case. We considered various characteristics that would be most relevant for our purposes and our current experience with different technologies, ultimately leading us to choose **Python** as the best option. The following table illustrates our decision process by comparing all the options we examined.

Alternative Technology	Performance	Versatility	User-friendly	Constraints	Average
Python	4/5	5/5	5/5	4.5/5	**4.625**
R	2/5	1/5	2/5	2/5	**1.75**
Java	5/5	4/5	4/5	4/5	**4.25**
SQL	4/5	3/5	3/5	3/5	**3.25**

Feasibility:

- Python is an incredibly powerful language as it not only provides a vast amount of resources with its many libraries, in particular those pertaining to data analysis and manipulation, but it is also fairly easy to use and understand regardless of skill level. In the future we intend to perform several tests to validate the language's feasibility when parsing through data and extracting the functional variables we specify while discarding irrelevant data values.

Data Filtration & Condensing

Issue Introduction:

- One of the main challenges we face is the design of an algorithm that can dynamically calculate faculty workload while maintaining a substantial level of flexibility and adaptability. There is a considerable diversity in credit hours, class types, and faculty roles resulting in numerous factors that need to be considered in order to accurately calculate the faculty's workload. Our client will need the ability to modify workload conditions and related weights without requiring code changes. This flexibility is essential as there might be university policies or faculty changes over time, ensuring the algorithm remains efficient, user-friendly, and capable of being utilized regardless of any challenges that might arise.

Desired Characteristics:

To achieve the goals of flexibility and adaptability, the algorithm must have the following characteristics:

- **Flexibility:** The algorithm must be able to dynamically apply workload weights based on the Excel sheet. This ensures that our client can modify workload conditions and weights without requiring code changes, making the program responsive to policy updates or faculty changes as needed.
- **Modularity:** The algorithm should be broken down into independent modules, such that each aspect (e.g., credit hours, class type, faculty role) can be updated without affecting the entire system. This will allow for easier maintenance and targeted adjustments over time.
- **Maintainability:** The goal is to minimize code changes. By reading workload conditions and weights from an Excel file, updates can be made without developer intervention. This ensures the program remains adaptable to evolving needs while minimizing the need for future development work.
- **User-Friendliness:** The Excel sheet must be clearly formatted with guidelines to ensure that our client can update conditions and weights without the risk of formatting errors. This reduces potential mistakes and makes the system more accessible for non-technical users.
- **Scalability:** The algorithm must handle datasets of any size, provided the format is respected. This ensures that the program will remain efficient and maintain its performance as the needs of NAU and our client potentially grow over time.

Alternatives:

- A possible alternative solution is the following:
    - **Hardcoded Workload Conditions**: This approach offers low flexibility, as every update to workload policies would require the

intervention of a developer, increasing long-term maintenance costs. While it might be more efficient in terms of speed, as it does not have to analyze numerous possibilities, it lacks the adaptability we are aiming to achieve. Any changes to workload conditions or policies would need to be manually coded into the system, making it impractical for a project where frequent updates or changes, especially to address edge cases, may be necessary

Analysis:

- **Hardcoded Workload Conditions:**
    - **Flexibility:** Low - any policy changes requires code updates
    - **Modularity:** Low - all aspects are tightly integrated
    - **Maintainability:** Low - future code changes are required
    - **User-Friendliness:** Medium - Client will be very limited, but the program itself will be easy to use
    - **Scalability:** Medium - As long as the current policies are respected the program has the possibility to scale. However, it might not cover edge cases

Chosen Approach:

Alternative Technology	Performance	Versatility	User-friendly	Constraints	Average
Dynamic Algorithm	5/5	5/5	4.5/5	4.5/5	**4.75**
Hardcoding Conditions and Weights	4/5	3/5	3/5	2/5	**3**

Feasibility:

- Using a dynamic algorithm is the best choice because it allows faculty workload to be adjusted easily without modifying the code. By pulling data from the Excel file and passing it through the algorithm, users can directly modify workload conditions and weights. This approach ensures the system remains flexible and adaptable to policy changes, minimizing the need for developer intervention. Resulting in a more practical and scalable solution.

Customizable Algorithm

Issue Introduction:

- The challenge is to create an algorithm capable of dynamically determining class workload weights using an external sheet that holds condition and corresponding weights. This system needs to be flexible, allowing associate deans to make updates directly in the Excel sheet without requiring code modification. The algorithm must be able to read and interpret these conditions, apply the correct weights, and immediately reflect changes in calculations, reducing the need for ongoing code maintenance.

Desired Characteristics:

- **Flexibility**: The algorithm should adapt to updates in the Excel sheet without requiring code changes, allowing for dynamic policy adjustments.
- **Modularity**: The design must be modular, allowing each component (condition reading, weight matching, and calculation) to operate independently and be easily modified if needed.
- **Efficiency**: The algorithm should parse and apply weights quickly, minimizing processing time, even as the size or complexity of the dataset grows.
- **Reliability**: It must include error-checking mechanisms to ensure the conditions and weights in the Excel sheet are formatted correctly, preventing miscalculations.
- **User-Friendliness**: The process for updating the Excel sheet must be straightforward, with clear guidelines to minimize errors and maintain consistency.

Alternatives:

- Hardcoding Conditions and Weights
    - **Pros**: The initial setup is simple and quick, as rules are directly embedded in the code, making it suitable for small-scale or short-term implementations. It avoids the need for external file management and minimizes the risk of user errors.

    - **Cons**: Lacks flexibility; any policy changes require code modifications, leading to higher maintenance time and costs. Frequent updates increase the risk of bugs, affecting stability and accuracy.
- Storing Conditions and Weights in a Database
    - **Pros**: Centralized data management and allows for dynamic querying, ensuring the system always uses the latest information. Offers robust access control and is ideal for complex or frequently changing workload policies.
    - **Cons**: Adds complexity with database setup, schema design, and user management. It also introduces dependencies beyond Excel, which may not align with the client's preference for simplicity and familiarity.

Analysis:

- The Excel sheet approach provides the most flexibility, allowing users to update weights and conditions directly without requiring code changes, which ensures that the algorithm can adapt easily to evolving policies. This method also supports user adoption since Excel is familiar to most administrative staff, minimizing the learning curve and promoting efficient use. While relying on Excel introduces potential risks, such as inconsistent formatting, these can be mitigated through built-in error-checking mechanisms and clear guidelines for users. Additionally, given that Excel files are lightweight and can be processed efficiently with existing libraries, this approach is expected to meet performance requirements without adding significant overhead.

Chosen Approach:

- The table below compares different methods for implementing the workload calculation algorithm, focusing on three approaches: hardcoding conditions and weights, storing them in a database, and using an external Excel sheet.

Alternative Technology	Performance	Versatility	User-Friendly	Constraints	Average
Using an External Excel Sheet	5/5	5/5	5/5	4/5	**4.75**

Storing Conditions and Weights in a Database	4/5	4/5	3/5	3/5	**3.5**
Hardcoding Conditions and Weights	3/5	2/5	4/5	2/5	**2.75**

Feasibility:

- The proposed solution, using Python's pandas library, is highly feasible both technically and operationally. Pandas provides efficient methods for reading, filtering, and processing large Excel files, allowing the algorithm to handle complex and diverse datasets with ease. By leveraging pandas alongside libraries like openpyxl, we can implement robust error-checking mechanisms to validate data integrity, column consistency, and overall file format before processing. This ensures the system remains adaptable and reliable, aligning with the requirement for flexibility as the associate deans can adjust conditions directly in Excel without code changes. Python and pandas are open-source and well-supported, making this approach cost-effective, scalable, and maintainable, ultimately meeting Northern Arizona University’s needs for a streamlined and automated workload calculation system.

Simple Graphical User Interface

Issue Introduction:

- One of the main technological challenges we are facing is a proper user friendly interface that enables the associate dean to navigate this with ease. The Interface should support the key functionality of the product with ease such as file uploads, executing calculations and providing reports. It can be assumed that many people who will be using this software are not tech savvy, so the UI should be minimalistic and easy to navigate. This will minimize user errors and improve efficiency.

Desired Characteristics:

- Minimalist design: The interface will focus on key functionality.
- Clarity: Everything will be clearly labeled and users will be guided through the process of the input file selection.

- Status feedback: The UI will give clear feedback as to what is happening in the process. In the case that there are error messages it will be shown, as well as when it is complete.
- Flexibility: The interface will allow users to upload excel files that contain the faculty workload data.

Alternatives:

- Command-Line-Interface: This is a more technical approach that would require users to enter commands. It is overly complicated and not user friendly.
- Web based Interface: This would provide a more scalable solution, but would require more overhead. For a project of this size it would be overkill.

Analysis:

- Ease of use: The Tkinter library provides a simple yet powerful way to create the necessary UI elements.
- Lightweight: Tkinter is lightweight and runs locally.
- Cross Platform: Tkinter will run on all platforms, but it is highly likely that it will be run on Windows.
- Integration with Excel: Since Tkinter is a Python library it is easily integrated with Excel.

Chosen Approach:

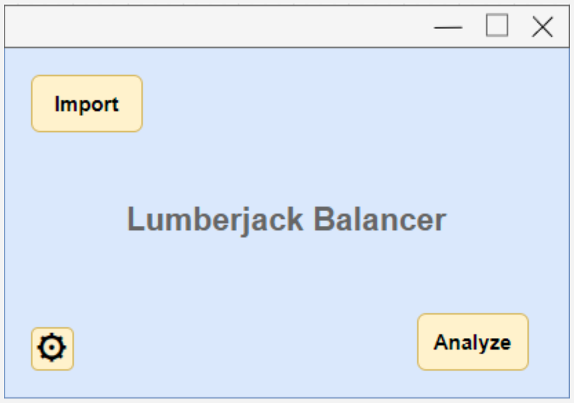
Alternative Technology	Performance	Versatility	User-friendly	Constraints	Average
Tkinter	5/5	4/5	5/5	4/5	**4.5**
Web-Based Interface	4/5	5/5	3/5	/5	**3.75**
Command Line Interface	3/5	3/5	2/5	4/5	**3**

Feasibility:

This approach is feasible because of its simplicity and ease of implementation. Tkinter is a well known and widely used library. The Python ecosystem offers extensive support for excel file manipulation which ensures a smooth integration between the UI and workload calculation logic. The UI can be expanded with additional functionalities as this is developed and will provide advanced options for handling edge cases.

With the analysis complete and the most effective solutions identified, the next step is to integrate these components into a unified system architecture. The following section outlines how these individual elements come together within a coherent design, ensuring that each part functions seamlessly to meet the overall objectives of the application.

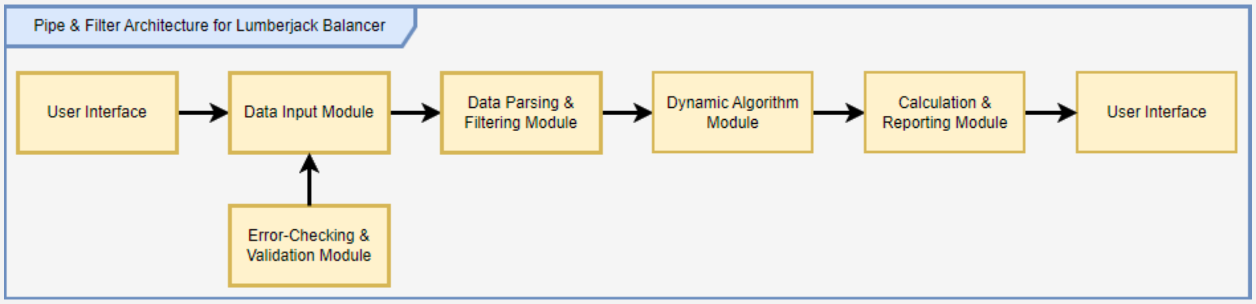
Technology Integration



To effectively automate the faculty workload calculation process, we must integrate each micro-solution into a cohesive system architecture. The pipe-and-filter architecture, depicted in the diagram above, represents how the major components work together to achieve our product requirements. This architecture allows data to flow sequentially through various modules (filters), each performing specific operations before passing the data to the next stage. This approach ensures modularity, flexibility, and maintainability.

System Overview

The diagram outlines the following key components of our system:


Pipe & Filter Architecture for Lumberjack Balancer
User Interface
Data Input Module
Data Parsing & Filtering Module
Dynamic Algorithm Module
Calculation & Reporting Module
User Interface
Error-Checking & Validation Module

1. **User Interface (UI)**:
    - The UI is both the starting point and the endpoint of the data flow. Users interact with this interface to upload Excel files containing

workload data and conditions, and to initiate the calculations. The minimalistic design of the prototype on the side promotes ease of use and ensures that even non-technical users can navigate the system efficiently.

2. **Data Input Module:**
   - This module receives the Excel files uploaded via the UI and validates them to ensure they meet the required format. It acts as the gatekeeper, ensuring only correctly formatted data is passed to the next module. If any discrepancies are detected, they are flagged by the Error-Checking and Validation Module, which is integrated into this process.
3. **Data Parsing and Filtering Module:**
   - After validation, this module extracts relevant data from the Excel sheets, filters out extraneous information, and structures the data in a format suitable for further processing. It also applies specific rules for handling special cases, ensuring that the data is both accurate and ready for the dynamic algorithm. This module is essential for streamlining and organizing the data before it moves to the calculation stage.
4. **Dynamic Algorithm Module:**
   - The core of the system, this module applies the conditions and weights defined in the external Excel sheet dynamically. It calculates the workload based on the parsed data, adjusting automatically to changes in the conditions sheet without requiring code modifications. This flexibility allows for quick adaptation to new policies or changes in faculty roles.
5. **Calculation and Reporting Module:**
   - This module finalizes the workload calculations and generates a comprehensive report, which is then formatted for output. The report includes annotated details that align with the university’s policies, providing a clear and actionable overview for the associate deans.
6. **Error-Checking and Validation Module:**
   - This auxiliary module interacts with multiple stages to validate data consistency and ensure accuracy throughout the pipeline. It monitors for errors and discrepancies, providing feedback to users through the UI when issues arise. This component is crucial for maintaining the integrity of the system and preventing miscalculations.

Data Flow

The diagram demonstrates how each module is connected, showcasing the linear flow of data:

- Data Entry and Processing: Users upload the necessary files through the UI, which directs the data to the Data Input Module. If validated, the data is passed through the Data Parsing and Filtering Module, where it is condensed and organized.
- Algorithm Execution: The structured data moves to the Dynamic Algorithm Module, where conditions and weights are applied dynamically. The processed data then flows into the Calculation and Reporting Module for final analysis and report generation.
- Output and Feedback: The UI displays the final report for download and provides error messages or status updates throughout the process to keep users informed.

By integrating these components into a cohesive system architecture, we establish a robust framework that meets the application's functional requirements and addresses the identified challenges. The following conclusion summarizes how this integrated approach effectively automates the faculty workload calculations, providing a flexible and scalable solution for Northern Arizona University.

Conclusion

In summary, the development of an automated faculty workload calculation application for Northern Arizona University addresses the complex and time-consuming task currently faced by associate deans. By leveraging an intuitive, Excel-based system, we aim to enhance both the accuracy and efficiency of workload assessments. This solution integrates dynamically customizable algorithms that respond to Excel sheet updates, reducing the need for code modifications and ensuring the application adapts to evolving policies and data requirements. The technological challenges outlined, including identifying crucial data points, parsing and condensing data, and creating a customizable algorithm, are tackled with strategies that balance flexibility and maintainability. By utilizing Excel as a familiar interface and adopting a modular approach, we minimize user training and optimize adoption while ensuring data integrity through error-checking mechanisms. Furthermore, the choice of a simple, minimalist user interface ensures ease of use, allowing administrative staff to seamlessly interact with the application. This comprehensive approach, combined with a focus on user-friendliness and adaptability, ensures that the application will effectively support associate deans in their workload management duties. It will also provide a scalable and maintainable solution that aligns with Northern Arizona University’s needs, ultimately enabling more accurate and efficient workload planning across all colleges.