Predictive Analytics for Software Project Management

Authors:
Mattia Tritto
Rossella Tritto

Project Report:
Software Architecture and Pattern Design

Contents

1 Introduction

Predictive analytics has emerged as a powerful tool to assist project managers in estimating outcome and enabling better decision-making.
This project aims to develop a predictive analytics system to support planning and management in software projects. By leveraging historical data on project attributes, this system will generate values for completion times, costs and adjusted function points.

The project focuses on building predictive models using various machine learning techniques, including regression, decision trees, neural network and upport vector machines. The predictive models will be integrated into a user-friendly web application, where users can input project details and receive key project metrics, such as estimated cost and completion date. Additionally, the application provides visualizations of historical project data, helping users understand trends and make informed comparisons.

The report is organized as follows. Chapter 2 provides a **Literature Review** of existing methods and frameworks in predictive analytics for software project management. Chapter 3 details the **Methodology** employed. Chapter 4 focuses on the **Machine Learning Model** development, including initial trials, feature engineering, model building and analysis of performance metrics. In Chapter 5, the **Software Architecture** of the system is described, covering functional and non-functional requirements, as well as the tech stack used, including all frameworks to enables application deployment. Chapter 6 shows a use case of the developed application. Finally, Chapters 6 and 7 present the **Conclusions** and potential **Future Work** to further enhance the system further.

2 Background and Literature Review

Accurate estimation of effort, cost and duration is crucial for the success of software projects. Numerous studies have explored techniques and models to improve prediction accuracy in these areas.

Traditional approaches, such as those outlined by Kitchenham and Taylor [4], focused on **cost estimation** models for software development and they set a foundation for this type of analysis. Similarly, Borade [1] reviewed general effort and cost estimation techniques, highlighting the need for **robust methods** to address the uncertainties in software projects. Also Kitchenham et al. [3] conducted a comprehensive study on estimation accuracy in software maintenance and development, identifying key factors influencing prediction reliability.
In more recent work, Molokken and Jorgensen [5] provided an extensive review of software effort estimation surveys, underlining common challenges and biases that affect estimation outcomes.

With the spread of **machine learning techniques**, research has shifted toward leveraging predictive models for more accurate estimates. Zakaria et al. [8] explored the application of machine learning to software project estimation, finding that these techniques offer improvements over traditional models. Rahman, Goncalves, and Sarwar [6] reviewed available datasets for software effort estimation, emphasizing the importance of high-quality data for training accurate models. Based on this, Rahman et al. [7] conducted an empirical analysis of machine learning methods for estimation, demonstrating the effectiveness of models such as neural networks and regression.

Recent developments in **large language models** have further expanded predictive capabilities. Carpenter, Wu, and Eisty [2] studied the use of these models for predicting software project costs and durations, suggesting that they offer new possibilities to improve estimation accuracy in complex projects.

To sum up, these studies highlight the evolution of estimation techniques from traditional statistical models to advanced machine learning approaches, showing the potential for predictive analytics to enhance software project management.

3 Methodology

This chapter describes the process followed to develop the system for software project estimation. The methodology consists of six key phases, a visual representation of this process is shown in Figure 1.

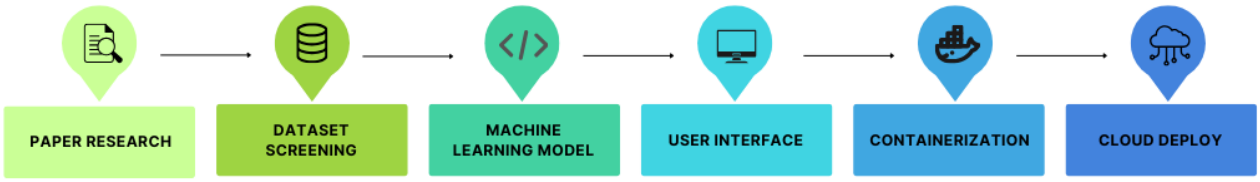

Figure 1: Project Process

1. **Paper Research**
   The project began with a review of existing literature to understand the current state of software project estimation techniques. This research phase aimed to identify established methods, common challenges and recent advancements in predictive analytics with machine learning. The findings from this literature review are detailed in Section 2.

2. **Dataset Screening**
   We searched for datasets related to software project estimation from both academic sources, such as research papers and public repositories like GitHub. Initially, we collected ten datasets. However, not all datasets met the criteria needed for our analysis. We applied exclusion criteria to ensure data quality and relevance. For example, one dataset contained only six projects, which was too small. Two datasets lacked the AFP metric, which was a critical feature for our analysis, and another dataset did not include "Duration," a core feature for estimating project timelines. Additionally, two datasets contained outdated features that were no longer relevant to modern software development (for example CPU time constraints, which were specific to programming practices in the 1990s). After this screening process, we used four datasets that met all criteria and were suitable for our analysis: *China, Desharnais, Kemerer and Kitchenham*. Figure 2 illustrates the dataset screening process and the filtering criteria used.

3. **Machine Learning Model**
   After obtaining the datasets, we developed and trained the machine learning models used for project prediction. We implemented a variety of models, including regression models, random forest, and other algorithms to identify the most effective approach. The details of this phase, with the model selection and evaluation metrics, are discussed in depth in Section 5

4. **User Interface**
   To provide a user-friendly way to interact with the predictive models, we developed a web application using Streamlit. This application enables users to input

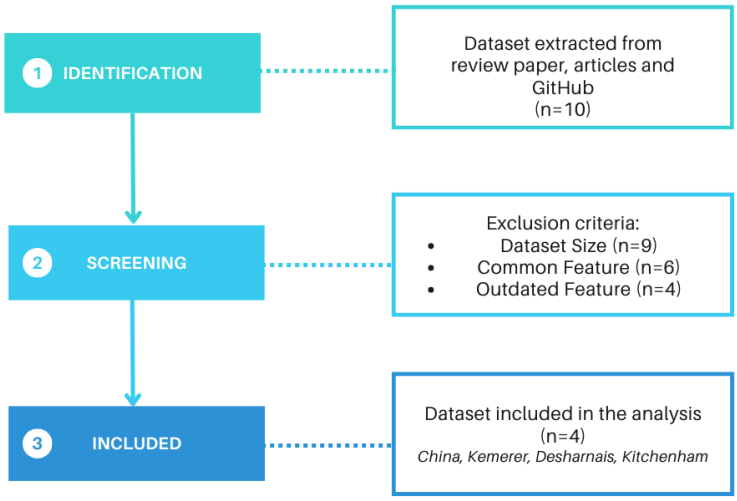

Figure 2: Dataset Screening

project details and receive real-time predictions on key metrics, including project duration, cost, AFP and estimated end date. Additionally, the application includes a historical data section, allowing users to view insights from past projects. This feature includes visualizations of important features from previous projects, helping users understand trends and make informed decisions. The web interface design prioritizes simplicity and clarity to ensure ease of use.

5. **Containerization**
   To facilitate deployment and maintain consistency across different environments, we encapsulated the application and its dependencies in a Docker container. This approach ensures that the application can be easily replicated, with all necessary libraries and configurations packaged together. This choice make the system more portable and resilient to environmental changes.

6. **Cloud Deployment** Finally, we deployed the application on Google Cloud Platform (GCP) to make it cloud-native, enhancing its accessibility and scalability. Deploying on GCP allows the application to be accessed remotely by users and ensures easy updates and maintenance.

3.1 Team Organization

Our team organized the work by dividing responsibilities between frontend and backend development. Initial meetings were held to plan the workflow and define the division of tasks. Each team member focused on their respective part: one developed the user interface, while the other worked on the machine learning model. Regular checkpoints were conducted to review each other's progress, ensuring alignment and addressing potential issues. In the end, both team members reviewed and refined each other's work. This collaborative process was facilitated by the container-based architecture of the project, which allowed each service to be independently developed and modified.

4 Machine Learning Model

4.1 Generalities

In order to make **duration** and **costs** predictions, we developed a machine learning model aimed at predicting the hours required to complete a software project (and indirectly, also total costs). The model uses various input variables, particularly **Adjusted Function Points (AFP)**, as predictors. Our initial approach involved combining multiple datasets to increase the robustness of the model, followed by experimenting with several machine learning techniques to achieve the best predictive accuracy.

4.2 Data Collection and Initial Trials

The initial step involved **unifying datasets** based on their common columns: AFP, Effort, and Duration. This resulted in a combined dataset featuring these three variables. However, after testing several machine learning models on the unified dataset, the highest R^2 score achieved was 0.24, indicating **limited predictive accuracy**.
We hypothesized that this low performance was due to **high heterogeneity across datasets**, meaning that the datasets differed significantly in characteristics and therefore may not have been suitable for direct combination. This insight led us to pivot our approach and focus on each dataset individually.
Following the initial trials, we identified the **CHINA dataset** as the most promising, as it yielded the highest R^2 score among all datasets tested. Subsequently, we centered our efforts on developing a model specifically for this dataset.

4.3 Feature Engineering and Preprocessing

To tailor the dataset for cost prediction, we performed **feature selection**, retaining only columns most relevant to the task: AFP, Effort, Input, Output, Enquiry, and File Interface. These features were directly related to the goal of predicting software project costs.
In the preprocessing phase, we **normalized data** using the median and interquartile range (IQR), as this method proved to be more effective than using mean and standard deviation. Normalization was essential to ensure that all features contributed proportionally to model training.

4.4 Model Development

We tested a range of regression models, including:

- **Linear Regression with Regularization**
- **Random Forests**
- **Neural Networks**

- **Support Vector Machines**

Among these, the Random Forest model delivered the best results, with an R^2 score of 0.67—a considerable improvement over earlier attempts, though still not ideal. We hypothesized that the limited dataset size (500 rows) may have contributed to this relatively low score.

4.5 Data Augmentation and Further Improvements

To address data limitations, we explored data augmentation techniques. The most effective approach involved **adding Gaussian noise to each row**, generating new data points that were similar to, but not identical to, actual software projects. This method helped expand the training dataset.
After augmenting the dataset, we conducted a grid search to explore a range of models and hyperparameters.

Model	Hyperparameter	Values Tested
Linear Regression with L2 reg (Ridge)	alpha	[10, 100, 1000]
Support Vector Machine (SVR)	C	[1, 0.1, 0.01]
	epsilon	[0.5, 1, 10]
	kernel	[rbf]
Random Forest Regressor	n_estimators	[110, 120, 130, 140]
	max_depth	[10, 20, None]
	min_samples_split	[2, 5, 7, 10]
	min_samples_leaf	[1, 2, 5]
Neural Network (MLPRegressor)	hidden_layer_sizes	[(64, 32, 16), (64, 32, 16, 8), (32, 64, 32), (64, 128, 64), (64, 32, 64), (64, 32, 16, 32, 64)]
	activation	[relu, tanh]
	solver	[adam]
	alpha	[0.01]
	learning_rate_init	[0.1]

Table 1: Hyperparameters and values tested in Grid Search for different models

This systematic search identified an optimal Random Forest configuration with the following parameters:

- **Tree Depth**: Limited to 20
- **Minimum Samples per Leaf**: 1
- **Number of Trees**: 130

Model	Best Parameters	Test MSE	Test RMSE	Test R^2 Score
Linear Regression with L2 reg (Ridge)	{'alpha': 10}	3004216.32	1733.27	0.21
Support Vector Machine (SVR)	{'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}	4372954.41	2091.16	0.15
Random Forest Regressor	{'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 130}	414774.87	644.03	0.89
Neural Network (MLPRegressor)	{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (64, 128, 64), 'learning_rate_init': 0.1, 'solver': 'adam'}	2558901.99	1599.66	0.32

Table 2: Grid Search Results: Best Parameters and Evaluation Metrics for Each Model

In reality, we also identified a Random Forest model with an R^2 score of 0.94. However, this model did not have limited tree depth, increasing the risk of overfitting. Consequently, we selected the model with a 20-depth limit, which achieved a reliable **balance between accuracy and generalizability**.

4.6 Model Deployment

Once the optimal Random Forest model was finalized, we **saved it using the Joblib library**, enabling easy loading for future use. This allows the server to make real-time predictions based on new input data. Additionally, we **stored the median and IQR values** used during normalization to ensure consistent preprocessing of any new input data.

5 Software Architecture

5.1 Overall Architecture

From the beginning, we adopted a **top-down approach**, first defining the overall system architecture and the interactions between components, and then developing each part individually.
The system architecture is as a **cloud-native solution**, consisting of **three distinct microservices** that communicate with one another with **HTTP REST APIs**. All microservices were dockerized and deployed on Google Cloud Platform using Google Cloud Run, Google Cloud Build, and the Artifacts Registry API.
The first component is the **frontend service**, which is developed as a Streamlit application. This service serves as the sole interface exposed to users, allowing them to interact with the system. When necessary, it sends requests to the backend service to retrieve either predictions or a generated DOCX report.
The **backend service** operates as a server, responsible for handling incoming HTTP requests. If a user requests a report, the backend acts as a mediator, forwarding this request to a specialized internal service known as the report service.
The **report service** is dedicated solely to processing requests related to report generation. It handles a single type of HTTP request: when it receives a request from the backend, it generates the DOCX report and sends it back to the backend. Subsequently, the backend forwards the report to the frontend service, completing the cycle of interaction and ensuring that the user receives the requested information.

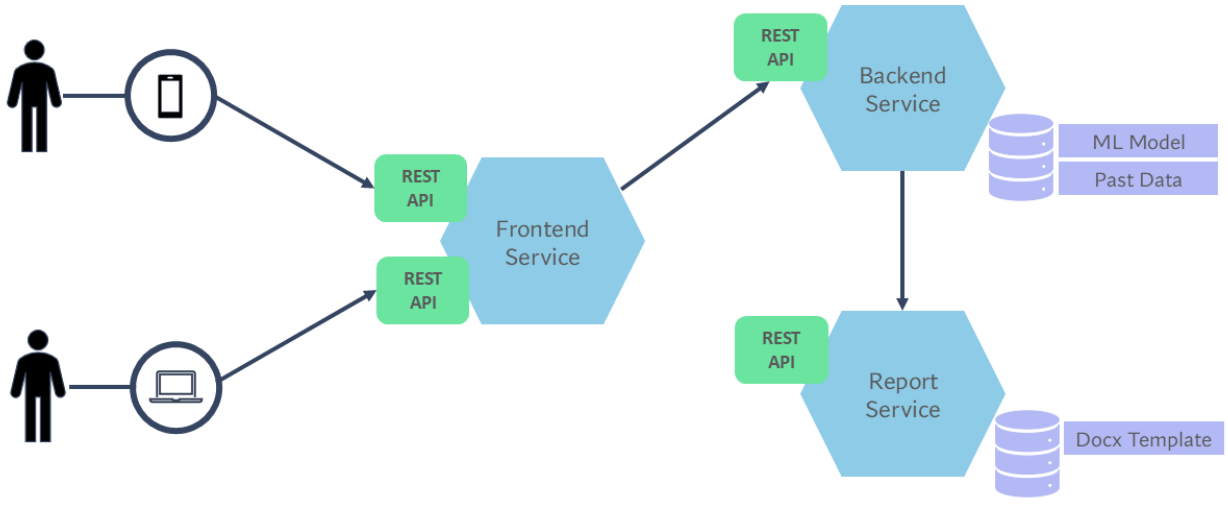

Figure 3: Software Architecture

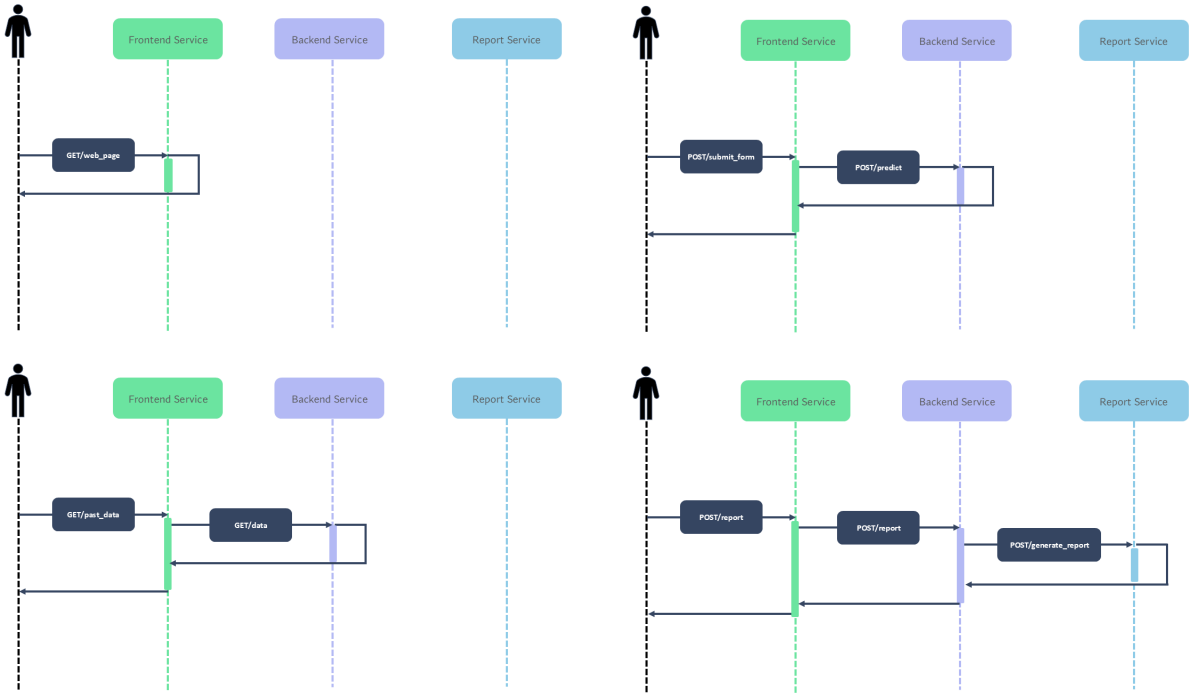

Figure 4: Sequence Diagram for different use cases

5.2 Functional Requirements

Data Input

Users shall input various data, including input and output values, enquiries, internal and external logic files, GSC values for AFP calculation, and the number of developers.

Data Processing

The system shall calculate AFP and use a machine learning model for predictions based on the provided data.

Result Display and Export

Results shall be displayed in graphs and tables. Users shall be able to generate and export customizable reports in DOCX format, which can be further adjusted before converting to PDF for stakeholder sharing.

5.3 Non-Functional Requirements

Performance

Objective: **Minimize latency** to ensure a smooth user experience.

- All services are hosted on a Local Area Network (LAN) within the Google Cloud Platform (GCP) to reduce network latency for HTTP requests.
- The frontend, built using the Streamlit framework, enhances performance with its efficient Document Object Model (DOM) rendering.
- Service is available with a **latency less than 0.5 seconds**.

Scalability

Objective: Handle varying user loads by scaling resources efficiently.

- A cloud-native, microservices architecture supports both vertical and horizontal scaling, adjusting automatically based on request volume.

Security

Objective: Protect system integrity and prevent unauthorized access or data vulnerabilities.

- The server **logs every request** to monitor access.
- **Inputs are validated** using the Pydantic library to guard against injection attacks and other input-related vulnerabilities.
- A **package dependency audit** was conducted to ensure the absence of security flaws in third-party libraries.

Usability

Objective: Provide an **intuitive user experience** that requires minimal learning.

- A streamlined and straightforward interface guides users directly to core functionalities without unnecessary steps or complexity.

Reliability

Objective: Ensure consistent performance and accuracy in various scenarios.

- Each component undergoes automated **unit testing**, verifying that it behaves as expected across all inputs.

Maintainability

Objective: Simplify system maintenance and minimize dependencies between components.

- The cloud-native architecture with loosely coupled microservices allows each component to be updated or modified independently without affecting the others.

Portability

Objective: Allow flexibility in deployment across different environments and cloud providers.

- **Docker containers** were used to package all components, making it straightforward to migrate the system between cloud providers if necessary.

5.4 Tech Stack

5.4.1 FastAPI

FastAPI is a modern, fast (high-performance) web framework for building **RESTful APIs** with Python. It is designed to create RESTful APIs quickly and efficiently. Key features include:

- **Asynchronous support**: Allows for handling many requests concurrently, suitable for high-performance applications.
- **Automatic OpenAPI documentation**: Simplifies API testing and exploration.
- **Data validation**: Built-in support for validating request data using Pydantic models.

5.4.2 Uvicorn

Uvicorn is an **ASGI server** that runs Python web applications. It is a lightning-fast server designed for asynchronous frameworks like FastAPI.

5.4.3 Pydantic

Pydantic is a **data validation** library for Python that enables developers to define data models and automatically validates data against those models. Notable features include:

- **Data validation**: Ensures input data adheres to specified types and formats, reducing runtime errors.
- **Ease of use**: Straightforward model definition and data validation, allowing for rapid development.
- **Integration with FastAPI**: Pydantic models can be seamlessly integrated with FastAPI to validate request and response data.

5.4.4 Pandas, NumPy, and SciPy

These powerful libraries are used for **data manipulation and analysis** in Python, facilitating data manipulation on datasets.

5.4.5 Scikit-learn (sklearn)

Scikit-learn is a widely used **machine learning library** in Python that provides simple and efficient tools for data mining and analysis. Key features include:

- **Wide range of algorithms**: Includes various supervised and unsupervised learning algorithms, such as regression, random forests, support vector machines, and neural networks.
- **Model evaluation**: Provides utilities for model selection and evaluation, including cross-validation and metrics.
- **Hyperparameter tuning**: Supports GridSearchCV for systematic testing of hyperparameter combinations to optimize model performance.

5.4.6 Joblib

Joblib is a lightweight library for **saving and loading Python objects**, particularly useful in machine learning workflows involving large data.

5.4.7 Streamlit

Streamlit is an open-source **app framework for machine learning and data science projects** that simplifies the creation of interactive web applications. Key features include:

- **Rapid prototyping**: Allows developers to turn data scripts into shareable web apps for quick iteration and testing.
- **Interactive visualizations**: Supports the creation of interactive graphs and visualizations for effective communication of insights.

5.4.8 Docker

Docker is a platform for developing, shipping, and running applications in **containers**. Key benefits include:

- **Isolation**: Each container runs independently, ensuring applications do not interfere with each other.
- **Portability**: Containers can be moved easily between environments without compatibility issues.
- **Scalability**: Facilitates horizontal scaling by deploying multiple containers.

5.4.9 Google Cloud Platform (GCP)

Google Cloud Platform is a suite of **cloud computing services** offered by Google, providing various tools for deploying, managing, and scaling applications.

5.5 Frontend Service Structure

The frontend of the application, built using **Streamlit**, provides an interactive interface to users. The application is structured across three main pages: Home, Predictive Analysis and Project History, accessible through a sidebar menu to make easy for users navigate the application. Below, we define each page's features and functionality.

Home Page

The Home page serves as the landing page for the application, introducing users to the platform, named "**CodeAnalytics**". The page displays a brief description of the application's purpose and it includes the application logo. The home page layout can be seen in Figure 5.

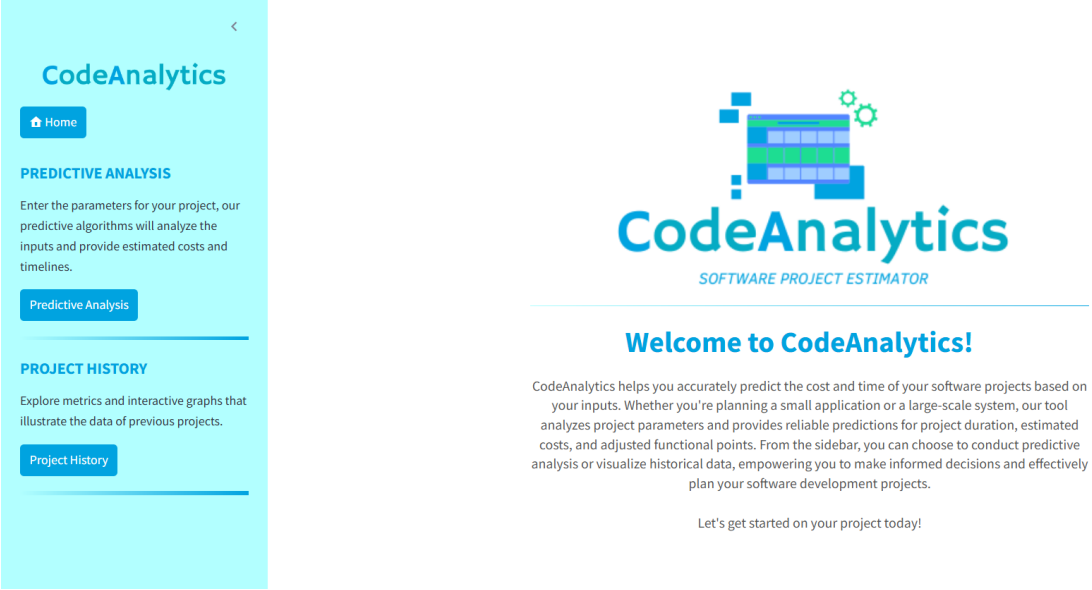

Figure 5: Home Page

Predictive Analysis Page

The Predictive Analysis page allows users to input project-specific details and obtain predictions on key metrics such as cost, duration, AFP and estimated end date.
Users are required to enter values for the functional components:

- **External Inputs** (EI): These refer to user inputs that the system must validate or process, such as data entered through forms
- **External Outputs** (EO): These represent outputs produced by the system, such as reports or other data that the system sends out to the user
- **External Inquiries** (EQ): These are interactive inputs that query the system and expect a response, like prompt

- **Internal Logical Files** (ILF): These files or data structures are maintained within the system for processing, such as internal databases
- **External Interface Files** (EIF): These are files managed by external systems but used by the application.

Then, there are some general characteristics about the project:

- **Number of Developers**
- **Hourly Wage**
- **Start Date**

To calculate AFP, the model considers some adjustment factors that impact project complexity. Each factor can be rated on a scale from 0 to 5, where 0 indicates "Not Present/No Influence" and 5 indicates "Strong, Generalized Influence." The adjustment factors are:

- **Data Communication**: Indicates the complexity involved in data transfer and communication across components.
- **Distributed Processing**: Refers to the need for processing across multiple systems, increasing project complexity.
- **Performance**: indicates the importance of performance in the project
- **Use of Configuration**: Indicates the degree of configuration required.
- **Transaction Rate**: Measures the frequency and complexity of system transactions.
- **Online Data Entry**: Relates to the amount of direct data input required.
- **User Efficiency**: Indicate the importance of user efficiency in the application
- **Online Update**: Refers to requirements for real-time updates.
- **Computational Complexity**: Reflects the need for complex computations.
- **Reusability**: Indicates how much the code and components can be reused.
- **Installation Ease**: Reflects the effort required to install the system.
- **Operation Ease**: Indicates the ease of system operation.
- **Multiple Sites**: Measures the need to support multiple locations.
- **Ease of Modification**: Reflects how easy it is to make changes post-deployment.

After submitting the form, the application displays the following predicted metrics:

- **Cost**: The estimated cost of the project based on the inputs provided
- **Duration**: The predicted time required to complete the project

- **Adjusted Function Point**s: A measure of the project's functional size
- **Estimated End Date**: The end date of the project based on the start date and duration.

Additionally, a timeline graph visualizes the project schedule from the start date to the predicted end date.

Project History Page

The Project History page provides an overview of key metrics from past projects, helping users to understand historical trends. This page is divided into several sections, as follows:

- **Data Highlights** displays summary statistics (mean and standard deviation) for the functional components of past projects, specifically EI, EO, EIF, ILF and EQ. These statistics provide users with a quick overview of typical values for functional components across completed projects.
- **Duration Overview** shows the minimum, maximum, and average duration values of past projects, giving users an idea of project lengths.
- **Graphs**: To help users explore historical data further, the Project History page includes the following graphs:
  - AFP vs. Duration Pair Plot: This scatter plot shows the relationship between AFP and project duration, allowing users to observe potential correlations between project size and time requirements
  - AFP Histogram: This histogram displays the distribution of AFP values across past projects, illustrating the range and frequency of project sizes
  - Duration Histogram: This histogram shows the distribution of project durations, allowing users to see typical project timelines and identify any outliers.

5.6 Backend Service Structure

The backend service was built using FastAPI, with the main entry points defined in `app.py`. This file initializes the FastAPI application.
The backend features three primary endpoints:

- The `/predict` endpoint, which accepts input data via a POST request and returns predictions for project duration, costs and AFP.
- The `/data` endpoint, which serves a CSV dataset file useful for the Streamlit application for building interactive graphs on the client side.
- The `/report` endpoint, which forwards requests to the report service to generate a DOCX report.

The application uses Pydantic models defined in `models.py` for input validation. The `AFPModel` establishes the structure used for calculating the AFP, while the `InputModel` extends it with fields for hourly pay and effort. This ensures that all incoming data adheres to specified constraints, such as non-negative integers and valid weight ranges.
In `utils.py`, utility functions are implemented to perform essential calculations. The `calculate_afp` function computes AFP using the input counts and weights, while the `predict_duration_and_costs` function loads a pre-trained machine learning model (saved as a pickle file) to predict duration based on the calculated AFP. This function constructs a DataFrame from the input data and returns the predicted duration, cost and AFP.

5.7 Report Service Structure

The Report service is designed to handle a **single POST request** for generating DOCX reports. The main entry point for this service is defined in `app.py`, which initializes the service and manages incoming requests.
This service features one primary endpoint: the `generate_report` endpoint, which accepts parameters necessary for constructing the report. The input model is defined using Pydantic in `schemas/ReportData.py`, ensuring that all required fields are validated. This model includes fields for project details, cost estimates, and other relevant information, enforcing constraints such as valid formats and mandatory entries.
Upon receiving a request, the service processes the input data, performs any necessary calculations, and generates a DOCX report using the validated parameters. The report is then sent back as a response to the backend service, ready to be forwarded to the frontend.

5.8 DevSecOps Principles Used

In this project, several key DevSecOps principles were applied to ensure a secure, efficient, and standardized development process.
Testing played a crucial role, with **unit tests** implemented to validate both the functionality of the prediction endpoint and the correct serving of static files.
Security compliance was enforced through multiple measures:
logging was used to track requests and responses for **auditing purposes**, and **data validation** using Pydantic models ensured that only properly structured and safe inputs were processed by the API.
The project was **containerized**, which enabled seamless deployment across different environments (development, testing, production), ensuring consistency and isolation.
To maintain a uniform code style, the **Black auto formatter** was integrated to run on every commit, normalizing the code regardless of contributions from multiple developers.

6 Use Case Example

In this section, we provide a practical example of using the application developed. For this test, we input the parameters of the current project (developing CodeAnalytics itself) into the application's form and analyze the results obtained.

6.1 Input Parameters

The parameters entered are:

- **External Inputs: 3**
  These components contribute directly to user interaction and the application's functionality. CodeAnalytics includes three primary external inputs:
  1. Function Components Section
  2. Project-Specific Inputs, which includes inputs like the number of developers, hourly rate, and start date
  3. Adjustment Factor Section, comprising 14 inputs for the adjustment factor that refine the estimates based on project complexity.
- **External Queries: 2**
  These queries enable users to retrieve and visualize data, so we assign a count of two external queries:
  1. Report Generation to generate a summary of calculated metrics.
  2. Graphical Outputs, which provides visual representations of data, such as distribution of duration and AFP.
- **External Outputs: 2**
  External outputs involve information generated by the application to display or share with the user. In this project, we have:
  1. Form output, which provides immediate results based on the user's inputs.
  2. Dataset statistics, which summarize important feature of the dataset.
- **Internal Logical Files: 1**
  We have one main internal logical file: a dataset that contains all the information necessary to estimate project parameters.
- **External Interface Files: 0**
  Since all data is processed within the application and there are no dependencies on external applications or files, this count remains zero.
- **Hourly Rate: 10€**
  We assume an hourly rate of 10 euros, which is the cost assigned to each developer's work hour.

- **Number of Developers: 2**
  For this project, we assume a small development team of two individuals who collaboratively work on developing the system.

For calculating the AFP, we employed predefined weights for each of the functional components (EI, ILF, EIF, EQ and EO). The default weights, illustrated in Figure 6, standardize these calculations to provide consistent and comparable results.

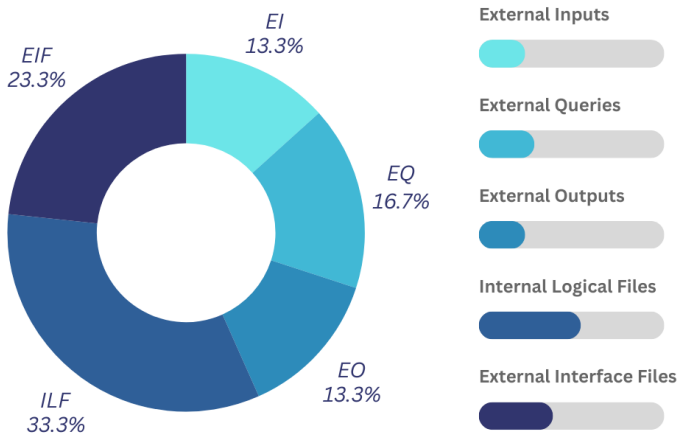

Figure 6: Default Weights for Functional Components

The application includes a series of adjustment factors, each rated on a scale based on the influence on this specific project:

- **Data Communication: 4**
  Rated as 4 due to the presence of multiple microservices in the architecture that need to communicate to fulfill the application's functionality. Each interaction between services adds complexity, making data communication a key factor for the project.
- **Distributed Data Processing: 0**
  Rated as 0 because all services run within a single environment. Even though there are multiple microservices, most of the processing remains centralized.
- **Performance: 4**
  Performance is rated as 4, given that interactions between microservices can introduce latency, impacting response time, making performance a critical project factor.
- **Heavily Use of Configuration: 0**
  No additional configuration exists for this application, resulting in a score of 0.
- **Transaction Rate: 1**
  Rated as 1 since the system does not require constant real-time updates; it processes requests as they come, with minimal transaction demand.

- **Online Data Entry: 2**
  Rated as 2 due to the presence of online forms that allow users to input data directly.
- **End User Efficiency: 4**
  Rated 4 due to the application's design, which provides essential data, improving user experience and efficiency.
- **Online Update: 0**
  No interface changes occur dynamically, so this factor is rated as 0.
- **Computational Complexity: 2**
  Rated as 2, based on the complexity of using the Random Forest algorithm with numerous trees for estimation.
- **Reusability: 5**
  With a highly modular design based on microservices, the application's code is reusable, resulting in a high rating.
- **Installation Ease: 0**
  Rated 0 as it is a web application requiring no installation.
- **Operation Ease: 3**
  Rated 3, as the application is straightforward to operate and manage.
- **Multiple Sites: 1**
  With only three main pages, this application scores a 1 in this category.
- **Facility Change: 4**
  A modular, cloud-native design allows easy adaptation and scaling, resulting in a high facility in modification.

6.2 Results of the Estimation

Based on the inputs and adjustment factors described, the web application generated the following estimates for the development of this project:

- **Total Cost of Project:** 5,055.00€
- **Total Duration:** 337 hours
- **Adjusted Function Points (AFP):** 38
- **Projected Start Date:** 14/12/2024
- **Projected End Date:** 25/01/2025

This case study demonstrates the practical application of the estimation tool for software project planning. The model takes into account various project characteristics, resulting in realistic estimations of cost, duration and function points. These estimates closely match the actual work requirements for this type of project, showing a high accuracy.

7 Conclusions

In summary, the application has **effectively fulfilled all specified functional and non-functional requirements**. This successful achievement reflects the comprehensive planning and execution that guided the development process.

The top-down approach that was used throughout the development lifecycle proved to be exceptionally efficient. By breaking down the project into manageable components, this strategy not only facilitated a clear understanding of the overall objectives but also resulted in a significant savings of time. This efficiency is particularly notable in how it allowed team members to focus on specific elements of the application without losing sight of the larger project goals.

Furthermore, the implementation of a microservices architecture played a critical role in enhancing the development process. This architectural choice enabled the division of coding tasks among multiple developers, fostering a collaborative environment that allowed team members to work simultaneously on different components of the application. As a result, the integration of these components was streamlined, leading to a more cohesive final product.

Regarding the predictive capabilities of the application, the Random Forest model demonstrated an impressive **R^2 score of 0.89**. This high score signifies that the predictions generated by the application are reliable.

8 Future Work

Looking ahead, the next crucial step for this project is to **integrate the software into a real-world scenario**. This integration will allow the application to be utilized by an actual organization, providing a practical context in which its capabilities can be thoroughly evaluated and refined.
One of the key advantages of implementing this framework within a company that uses its own dataset of previous projects is the potential for significantly enhancing the reliability of the predictions made by the application. Currently, the dataset utilized during the development of this project was sourced from an unknown company, which may follow distinct methodologies and processes for executing their projects. As a result, the insights derived from this external dataset might not fully align with the operational practices or challenges faced by other organizations.
By tailoring the predictive model to specific organizational data, we anticipate that the application will yield more accurate and relevant forecasting outcomes. This customization process will involve not only the integration of historical project data unique to the company but also an iterative evaluation of the model's performance based on real-world feedback.

Moreover, future work may also explore the incorporation of additional features and variables that are relevant to the specific industry or project types of the organization. This could involve analyzing qualitative factors that influence project outcomes, thereby providing a more holistic view of the elements that contribute to project success.

In conclusion, the successful integration of the software into a real-world scenario represents a pivotal next step that has the potential to transform theoretical predictions into actionable insights, driving more effective decision-making within organizations.

References

[1] Borade. Software Project Effort and Cost Estimation Techniques. *IJARCSSE*.

[2] J. Carpenter, C.-Y. Wu, and N. U. Eisty. Leveraging Large Language Models for Predicting Cost and Duration in Software Engineering Projects, Sept. 2024. arXiv:2409.09617 [cs].

[3] B. Kitchenham, S. Lawrence Pfleeger, B. McColl, and S. Eagan. An empirical study of maintenance and development estimation accuracy. *Journal of Systems and Software*, 64(1):57–77, Oct. 2002.

[4] B. A. Kitchenham and N. Taylor. Software project development cost estimation. *Journal of Systems and Software*, 5(4):267–278, Nov. 1985.

[5] K. Molokken and M. Jorgensen. A review of software surveys on software effort estimation. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, pages 223–230, Rome, Italy, 2003. IEEE Comput. Soc.

[6] M. Rahman, T. Goncalves, and H. Sarwar. Review of Existing Datasets Used for Software Effort Estimation. *IJACSA*, 14(7), 2023.

[7] M. Rahman, H. Sarwar, M. A. Kader, T. Gonçalves, and T. T. Tin. Review and Empirical Analysis of Machine Learning-Based Software Effort Estimation. *IEEE Access*, 12:85661–85680, 2024.

[8] N. A. Zakaria, A. R. Ismail, A. Y. Ali, N. H. M. Khalid, and N. Z. Abidin. Software Project Estimation with Machine Learning. *IJACSA*, 12(6), 2021.