Software Design Plan

WGU Student ID	011917160

A. Business Case

1. Problem Statement

Endothon Finance is a loan servicing company that has recently launched a new client-facing web application to streamline the loan application process for small businesses. Information gathered during this process is used to build a loan profile, which is then sent to potential lending vendors, helping to streamline the loan approval process. One of the intended functions of the application is to request the previous five years of financial data to determine qualifications. If the company has not been in business for at least five years, the application will request the applicant to enter the remaining years as forecasted data. A support ticket has been submitted to report that the app is not functioning as intended. The problem is that the application returns the first five years, instead of the expected five most recent years of financial data.

2. Business requirements

Requirement 1: Recent Years for Established Companies

Expected: The application is expected to request the five most recently completed years of financial data. This requirement applies to companies that have been established for more than five years.

Example: The year is 2025. A company established in 2010 is expected to request financial data for 2020, 2021, 2022, 2023, and 2024.

Failure: The application returns the first five completed years of financial data.

Example: The year is 2025. A company established in 2010 requests financial data for 2010, 2011, 2012, 2013, and 2014.

Requirement 2: Forecasting Years for Newer Companies

Expected: The application is expected to request the most recent completed and remaining forecasted years of financial data to reach a total of five years. This requirement applies to companies that have been established for less than five years.

Example: The year is 2025. A company established in 2023 is expected to request financial data for 2023 and 2024, along with forecasted data for 2025, 2026, and 2027.

Failure: Although not stated in the ticket, because the application failed to select the five most recently completed years in the first scenario, the logic needs to be verified to confirm that forecasting is present and the correct forecasted years are requested.

Example: The year is 2025. A company established in 2023 is requesting financial data for 2023 and 2024, along with forecasted data requests for 2022, 2021, and 2020.

Example: The year is 2025. A company established in 2023 is requesting financial data for only 2023 and 2024, with no forecasted data requested.

3. In-scope action items

Action 1: Recent Years for Established Companies

The application's logic needs to be updated so that it selects the five most recently completed years instead of the first five years. The code responsible for calculating the selected years needs to be reviewed and corrected to request years based on the current year minus one, accounting for only complete years.

Action 2: Forecasting Years for Newer Companies

The application's logic needs to be verified to ensure that forecasting is present and that the correct future years are requested. This includes confirming that the system is using the correct baseline year and that the combination of completed and forecasted years results in a total of five years.

4. Out-of-scope action items

Implementing Importable Forecasting Data for Newer Companies

This action would involve implementing a system to accept imported forecasting data generated by the applicant or a third party. It aligns with AC2 of the ticket by fulfilling the requirement of requesting forecasted future years for companies in business for less than five years. This action would be out of scope because it would not address the primary issue of incorrect year selection. It also does not verify that forecasted years are being correctly generated.

Adding a Confirmation Prompt for Selected Years of Financial Data

This action would involve implementing a confirmation prompt during the application process, providing the user with a preview of the fiscal years that have been entered and added to their loan profile. This aligns with the ticket because it would inform users if the requested years were incorrect. This action would be out of scope since it does not solve the underlying issue of the application selecting the wrong years.

B. Requirements

1. Functional requirements

Functional Requirement 1

The system must correctly request the five most recent completed years of financial data for companies that have been in business for more than five years. The ticket describes that the application is currently requesting the first five years instead of the most recent ones. The application's logic regarding year selection must be reworked to meet this requirement.

Functional Requirement 2

The system must correctly identify when a company is less than five years old and identify and request the correct mix of completed and forecasted years. To meet this requirement, the application's logic must be verified to ensure that forecasting is present and that the correct number of forecasted years is requested.

Functional Requirement 3

The system must disregard the current year when determining completed years for a business. Incomplete years must not be included when generating loan profiles to ensure accurate and valid results.

2. Non-functional requirements

Non-Functional Requirement 1

The system must apply robust security measures to protect the client's financial data. Access to data must be restricted to authorized users to prevent external entities from viewing or modifying sensitive information. Utilizing role-based controls ensures that only admins and verified applicants are permitted to access or update loan profiles.

Non-functional Requirement 2

The system must be able to scale as Endothon Finance grows and expands its operations. The application must be able to handle an increase in loan applications without compromising performance.

C. Software Design

1. Software behavior

One input would be a date selector that applicants use to indicate when their business was established. If the date is valid, the application is expected to calculate the company's age. If the date is invalid, the application is expected to display a warning message. The data is intended to be submitted using a text box or, potentially, a pop-up calendar tool. This input will be used to determine whether a company needs to forecast some fiscal years.

Constraints:

- The date submitted must not be in the future
- The date submitted must use numeric traditional American notation (MM/DD/YYYY) and must not use textual notation (January 20, 2025) or European notation (DD/MM/YYYY)

Another input would be when a user is prompted to enter financial data for completed or forecasted years. The expected behavior of the application is to accept the entered data and generate a corresponding loan profile. If the data is invalid, the application is expected to display a warning message. The input is intended to be entered by the applicant using a text box. This input will be used to generate a loan profile for the applicant, which will then be sent to potential lending vendors.

Constraints:

- The data entered must be in numeric form
- The data entered must use no more than two decimal places
- Data must be present for all five years

2. Software structure

Fiscal Year Module

Functionality: This module determines the company's age, which fiscal years need to be requested, and how many years must be forecasted

Methods:

- companyAge(establishDate): returns how many years the company has been in business
- getFiscalYears(companyAge): uses a company's age to determine which fiscal years should be requested. If a company is older than five years, a list of the five most recent complete years is returned. If a company is less than five years old, a list of the most recent complete years and the remaining forecasted years is returned

Input Validation Module

Functionality: This module handles user-submitted input, such as establishment date and financial data. It checks that inputs are valid, formatted, and usable by the fiscal year module.

Methods:

- validateEstablishDate(establishDate): checks that the user enters a date that is valid, not null, not in the future, and is in MM/DD/YYYY format. If the entry is valid, a date object is returned. If the entry is invalid, an error message is returned indicating the specific failure.
- validateFiscalData(fiscalData): checks that the user enters fiscal data usable for calculations. The method checks that the entry is not null, only numeric, and uses no more than two decimal places. If the entry is valid, a float is returned to use for calculations. If the entry is invalid, an error message is returned indicating the specific failure.

Error Handling Module

Functionality: This module handles errors and internal logging when invalid input is detected

Methods:

- displayWarning(message): displays a warning message. Used to give the user a textual description of the error
- highlightField(field): highlights the field in which the error has occurred. Used to help the user quickly identify the error
- logError(event, timestamp): records the error event in a log file for later review. Used for debugging

D. Development Approach

1. Planned deliverables

Deliverable 1: Fiscal Year Module

This module calculates a company's age and returns the five fiscal years that should be requested. The fiscal years can be all completed years or a mix of both completed and forecasted, depending on the company's age.

Steps to Create

1. Design: Establish the methods, inputs, and outputs required for calculating a business's age and determining the correct set of fiscal years to be requested.

2. Implement companyAge(establishDate) Method: Develop the method to calculate the company's age by comparing the establishment date to the current date. This method returns an int to represent the company's age.

3. Implement getFiscalYears(companyAge) Method: Develop the method to determine which fiscal years should be requested. Compare companyAge with requiredYears to determine if the company is more than or less than five years old. This method returns a list of the latest five complete fiscal years for companies that are more than five years old, or a mixed list of completed and forecasted fiscal years for companies that are younger than five years old.

4. Unit Testing: Design and run tests to verify the functionality of both methods. Test different scenarios, such as null or future established date, as well as the company age being greater than, less than, and exactly five years old.

Deliverable 2: Input Validation Module

This module ensures that user-submitted inputs are valid, properly formatted, and usable by other modules.

Steps to Create

1. Design: Define validation rules for each type of input. Define what qualifies as a valid and formatted date or fiscal data, and what errors to produce if validation fails.

2. Implement validateEstablishDate(establishDate) Method: Develop the method to confirm that the establishment date is not null, follows the numeric traditional American formatting (MM/DD/YYYY), represents a real calendar date, and is not in the future. If the entry is valid, return a parsed date object. If the entry is invalid, return an error message specifying the validation failure.

3. Implement validateFiscalData(fiscalData) Method: Develop the method to confirm that the fiscal data is not null, contains only numeric values, and uses no more than two decimal places. If the entry is valid, parse the input and return it as a float value. If the entry is invalid, return an error message specifying the validation failure.

4. Unit Testing: Create and test each validation rule to confirm correct handling. Include tests for different inputs, such as valid inputs, improper formatting, null fields, and alphabetic characters.

Deliverable 3: Error Handling Module

This module provides feedback to the user when input or logic errors occur and records errors for debugging purposes.

Steps to Create

1. Design: Define the method and properties needed for displaying warnings, highlighting invalid fields, and logging errors. Determine how each method interacts with input validation and fiscal year modules.

2. Implement displayWarning(message) Method: Develop the method to display a descriptive textual warning message to the user when an input or logic error is detected.

3. Implement highlightField(field) Method: Develop the method to visually highlight the specific field causing the input error. This helps the user quickly identify and solve the issue.

4. Implement logError(event, timestamp) Method: Develop the method to record error events with timestamps in a log to later review for debugging purposes.

5. Unit Testing: Create and test each method to ensure proper error handling, displaying of warnings, and accurate logging. Include tests for various scenarios of proper and improper input validation, like invalid values, missing fields, and incorrect formatting.

Deliverable 4: Documentation

This deliverable provides detailed documentation for developers, including modules, methods, testing methods, and results. It ensures that future developers can understand, maintain, and add to the existing application.

Steps to Create

1. Design: Define sections of the outline such as module overviews, method descriptions, constraints, properties, testing procedures, and results.

2. Document Modules: For each module, describe its purpose, methods, and expected behavior.

3. Document Tests and Results: Define procedures used for testing. Include different scenarios used, expected results, and actual results.

4. Review and Finalize: Ensure that the documentation is clear and accurate. Finalize the document and push it to the repository storage.

2. Sequence of deliverables

Sequence	Deliverable	Justification
1	Fiscal Year Module	This module is first because it contains the core logic of the application. Implementing this module first enables the calculation of a company's age and determines which fiscal years to request. This module handles the ticket resolution.
2	Input Validation Module	This module is second because once the core logic has been established, validation is needed to ensure that the user-entered data is usable. This module relies on knowing what outputs the fiscal year module expects.
3	Error Handling Module	This module is third because once the core logic and validation are in place, error handling can be implemented to provide the user with meaningful feedback. This module relies on the two other modules to detect and handle errors.
4	Documentation	This module is the last one, allowing it to include finalized descriptions of all modules, methods, testing methods, and results. This module relies on the completion of all other modules to ensure accuracy and completeness of documentation.

3. Development environment

Programming Language: Python & JavaScript

Purpose: Python will be used in the backend to implement core logic and process fiscal data. It is a versatile language that has extensive libraries and supports modular code organizations. JavaScript will be used in the frontend to handle displaying error messages and interactive features.

Environment: Visual Studio Code

Purpose: Visual Studio Code will be used for both frontend and backend development because it supports multiple languages. It provides highlighting, IntelliSense, and debugging tools. It has extensions that can assist in writing, testing, and maintaining modules. It also allows for easy management of modular files and project management.

Version Control: Git

Purpose: Git will be used for version control to track changes in the codebase, manage multiple development branches, and collaborate with other developers. It also allows for rollbacks in the event of a substantial error.

Database: SQL Server

Purpose: SQL Server will be used as the database to store the fiscal data required for generating loan profiles.

4. Development Methodology

The project will utilize the Agile methodology, which focuses on iterative development, frequent testing, and continuous feedback. This approach informed the planning process by allowing deliverables to be developed in small sprints. It also allows each module to be reviewed and tested before proceeding. Agile was selected because the scope of the issue is relatively small, and it provides more flexibility than methods like Waterfall, which might not reveal additional bugs until later in the process.