CS6.401 Software Engineering

Spring 2024

Project - 1

Welcome to the first project of the CS6.401. Software Engineering course. Your objective in this project is to apply the concepts you have learned (and will learn) in the class to improve an existing software system. You will reverse engineer the design of the system and refactor it following grounded software engineering principles. You will use some tools to analyze the code and collect metrics to aid this process.

Books, a web-based book server will serve as the testing ground for the project. The app can be used to organize your books collection. You can find the repository here. We will use a class-specific fork which has been slightly streamlined for ease of use.

Note: Should you choose to employ Large Language Models (LLMs) like chatGPT, Bard etc. in completing this assignment, kindly include screenshots delineating the specific prompts employed and the corresponding generated responses. Additionally, provide an evaluation of the accuracy of these responses, highlighting instances of correctness or errors. Furthermore, mention any novel contributions or modifications introduced by you during the utilization of LLMs in addressing the task.

Task 1 (30)

Welcome to the world of Book Management! Books are organized in several components, and to make meaningful improvements and contributions to it, we need to become familiar with its structure and organization. You have been provided with the code base of an existing Book Management App, a literary

haven that strives to make the reading experience delightful. Now, let's dive into the code and uncover the story it holds.

The major subsystems of the app are as follows:

1. *Book Addition & Display Subsystem*:

   This is where new characters (books) enter our literary world. Picture a multi-modal approach, barcode scanners on Android and iOS – like magical portals, and the use of ISBNs. Google and OpenLibrary databases are our trusted encyclopedias, and we can even import data from Goodreads for a richer narrative.

   This system also displays the books' cover art, detailed descriptions, author biographies – it's like opening a treasure chest of literary information. Users can even interact with the books, marking them as read or completed.

2. *Bookshelf Management Subsystem*:

   In the vast library of our app, users can create their own bookshelves, like personal book genres. Imagine organizing books into these virtual bookshelves, making it easy to navigate through the literary landscape.

3. *User Management Subsystem*:

   Every literary kingdom needs its guardians. This subsystem empowers admins to add users, change passwords, and maintain order by clearing other opened sessions.

Task Description:

- **Identify Relevant Classes**: Examine the provided Book Management App codebase and discern the classes responsible for the specified features (Book Addition & Display, Bookshelf Management, and User Management).

- **Document Functionality and Behavior**: For each identified class, create detailed documentation elucidating its functionality and behavior in the context of the corresponding subsystem.

- **Create UML Diagrams**: Utilize Object-Oriented Programming (OOP) concepts such as inheritance, association, composition and aggregation to model the relationships and interactions between the identified classes accurately. Employ UML class diagrams to visually represent the structure, relationships, and attributes of the identified classes. Ensure that the level of abstraction chosen is appropriate for conveying a clear and concise representation of the system. ***Please note that you have to use plantUML or starUML as the tool for creating UML diagrams. Hand-drawn diagrams or diagrams drawn using tools such as draw.io, lucidchart etc. will not be considered for grading.***

- **Observations and Comments**: Include observations and comments in your documentation, highlighting points of strength and weakness within the system.

- **Assumptions**: If simplifying assumptions are made during the documentation process, explicitly state these assumptions in your report.

Task 2a (30)

Design smells are structures and patterns in code that while not incorrect, are indicative of violating fundamental design principles. These can hinder development by leading to recurring problems down the line and should be avoided.

Many automated tools exist to identify these. Sonarqube is one of those tools. We also have Designite Java and other plugins are available on different IDEs. We encourage you to use multiple tools as well as manually inspect the code, but Sonarqube is a must. To get started with Sonarqube, follow this documentation.

Sonarqube identifies *code* smells in a given repository. Code smells are closely related to design smells, but are more specific in nature. Make sure that you list design smells, not code smells.
Note: Sonarqube or any automated tool is not perfect, so use your own judgment.

Task Description:

Detect 5-7 design smells using the tools mentioned above, especially Sonarqube, and support your analysis for each design smell with either the code smells provided by Sonarqube or the analysis performed by using other tools.

PS - Sonarqube and the project itself require different Java versions to run. Instructions for handling this are provided in the project README.

Task 2b

The goal of this task is to conduct a comprehensive code metrics analysis for the initial state of the project, utilizing appropriate tools such as CodeMR, Checkstyle, PMD, or any other tools deemed suitable. The analysis should provide insights into up to 6 key code metrics, and the implications of these metrics should be discussed. Additionally, consider how these metrics might guide decision-making processes for potential refactoring.

Task Description:

- **Code Metrics Analysis**: Employ code analysis tools (e.g., CodeMR, Checkstyle, PMD) to extract up to 6 relevant code metrics for the project. These metrics may include but are not limited to: Cyclomatic Complexity, Code Duplication, NPathComplexity, Lines of Code and so on.

- **Tools Used**: Clearly state the tools used for the code metrics analysis and ensure that the selected tools provide a reliable and accurate representation of the project's codebase.

- **Implications Discussion**: For each identified metric, discuss its implications in the context of software quality, maintainability, and potential performance issues. Provide insights into how each metric reflects on the project's current state.

Task 3 Refactoring (40)

Having gained a comprehensive understanding of our system's organization and performance, it's time to roll up our sleeves. In the preceding task, we have pinpointed various metrics and design smells within the repository, shedding light on code quality and functionality. With ample room for enhancement, the final step in this phase involves addressing these issues.

Task 3a Design Smells

You previously identified 5-7 design smells. Now, the goal is to rectify these issues through code refactoring without fundamentally altering the existing codebase. It's crucial to emphasize that we're not discarding the previous code but rather enhancing its design. ***The refactoring should go beyond trivial changes, and your evaluation will be based on the correctness and significance of the improvements made.***

Important Note: Whenever you're tweaking the code to fix a design issue, make it a habit to kick off the process by opening a new issue on GitHub. This helps us keep a neat record of all the changes we're making in the codebase. ***Your performance will be evaluated only based on the issues you create****, so it's a crucial step before diving into the problem-solving part. Think of it as maintaining a clear roadmap for the improvements we're making.*

Task 3b Code Metrics

Following the refactoring process, it's time to reassess the metrics measured earlier. Have they seen improvements or deterioration? Is the trend consistent across all

metrics, and does it align with your expectations? Additionally, attempt to analyze the factors contributing to these changes.

Task 3c Leveraging Large Language Models for Refactoring

In the era of advanced language models like GPT-3.5 and bard, we have powerful tools at our disposal that can assist in the refactoring process. As an additional step to this assignment, you are encouraged to leverage these Language Models (LLMs) to generate alternative refactored versions of the identified code snippets.

Task Description:

- **Identify Code Snippets**: Select specific code snippets corresponding to the design smells identified in Task 2a.
- **Apply Manual Refactoring**: Before using LLMs, you should manually refactor the selected code snippets based on your understanding of design principles and best practices.
- **LLM Refactoring**: Use an LLM (e.g., GPT-3.5 or bard) to generate alternative refactored versions of the same code snippets. Provide screenshots or links to the conversation to show the LLM's output. Compare the LLM-generated refactoring with the manually refactored version in terms of clarity, conciseness, and adherence to best practices.
- **Evaluate and Document**: Document the differences between the manual and LLM-generated refactored versions.
  Evaluate the strengths and weaknesses of each approach, considering factors such as code readability, maintainability, and efficiency. Explore scenarios where LLMs excel and where manual intervention is still preferable.

Note: Keep in mind, you only have to document the changes in the code suggested by LLMs and not apply it in your actual code base. The code base should only contain changes made by you.

Bonus Task: Automated Refactoring Pipeline

This bonus task challenges you to design and implement a full pipeline that utilizes Language Models (LLMs) to periodically detect design smells in a GitHub repository, automatically refactor the identified issues, and generate a pull request for the changes.
You are expected to use openAI APIs for this task. OpenAI provides 5$ worth of free API usage to every user.

Task Description:

- **Automated Design Smell Detection**: Develop a script or tool that periodically scans the GitHub repository for design smells. This can include metrics analysis, code pattern recognition, or any other relevant approach. Utilize the capabilities of LLMs to assist in identifying potential design issues in the codebase.

- **Automated Refactoring**: Implement an automated refactoring module that takes the identified design smells and generates refactored code using LLMs. Ensure that the refactoring process is robust, preserving the functionality of the code while enhancing its design.

- **Pull Request Generation**: Design a mechanism to automatically create a pull request with the refactored code changes.
  Include detailed information in the pull request, such as the detected design smells, the applied refactoring techniques, and any relevant metrics.

You can also provide additional diagrams such as flowcharts to explain your approach better.

Please note that you can evaluate your implemented pipeline on a specific file instead of the entire repository considering the limitations of API usage (for GPT-3.50) imposed by openAI. **This bonus contributes to 2% of the overall course weightage.**

Submission Instructions

The submission for this phase will be through the GitHub classroom. The codebase will be automatically downloaded at the deadline, so ensure that everything is up in time. No exceptions will be granted.

In addition to the code changes required, you also need to submit a report containing your responses for each task. The report should be present in the docs directory, titled project1_<team_number>.pdf. If you are attempting the bonus, submit a separate document titled project1_bonus_<team_number>.pdf containing the corresponding responses.

Accurately report the contribution of each team member at the end of the report. This will be considered when evaluating the project.

Soft deadline - 16th February, 2024

Hard deadline - 23rd February, 2024

Note: Bonus can be submitted till the hard deadline without any penalty. Late days are not applicable for bonus components
The course policy mentioned on this website will be followed for late submissions and associated penalties/late days.

Best of Luck!