COM3523/6523 Group Project

Reengineering Open-Source Software Systems

Version 1.1 (26/02/2024)

Change History

- GenAI-related instructions are added to Unfair means.
- More guidelines for the draft submission are added.

Overview

In this group assignment, you are expected to effectively reengineer a legacy software system based on what you have learned from the module. You can choose an open-source software system to reengineer from GitHub, but there are certain selection criteria to consider.

The project consists of two main parts:

- Part 1: Understand the evolution of the system.
- Part 2: Restructure the system to satisfy certain requirements.

Part 1 is about analysing and understanding the system. You should investigate not only the current snapshot of the system but also how the system has evolved over time.

Part 2 is about restructuring the system to satisfy certain requirements, such as improving maintainability or introducing new features.

In the end, you must submit a *report* summarising your reengineering efforts (possibly including unsuccessful yet worthy of mentioning for future work) and the *reengineered software*. You are expected to show evidence that you did *effective* reengineering within a limited time.

A few weeks prior to the final submission, you will be provided with an opportunity to receive constructive feedback on your draft from your peers, i.e., other groups. Note that the draft submission and feedback will *not* be assessed, but it would be a valuable chance for you to reflect on your work.

Subject Selection

You are free to select the subject software system, but considering the learning objectives and the scope of this project, the subject system must satisfy all the following criteria:

- It is an open-source software system available on GitHub.
- Its main programming language is Python (more than 80%).
- More than a year has passed since the first commit.
- The total number of commits is more than 20.
- The total number of source code lines (except comments) is more than 1K.
- A reasonably comprehensive-looking test-set.
- You must be able to run it (e.g. by running test cases).

Your first task is to nominate your choice of system. For this, a nominated member of the group (and just one member) must fill in the nomination Google form:

https://forms.gle/deBrJ1bDDLgMad1WA

Once you have received confirmation from the teaching team, you can proceed to work on this system.

If two groups nominate the same system, the first group to have submitted the nomination will be granted permission to work on the system, and the second group will need to choose a different system.

Initial Setup

We will use GitHub Classroom. The link for this assignment:

https://classroom.github.com/a/GUpwTmL4

Each group will have one shared group repository. Since GitHub Classroom does not allow the creation of pre-formed groups, you must follow these steps:

(1) One of your group members must create a "team" in GitHub Classroom using the link below. The team name must be the same as the group name on the Blackboard (e.g., UG01, PGT01).

(2) Once the team is created in GitHub Classroom, the other group members must join the correct team using the same link below. This will provide one shared group repository for all your group members to access.

Once your group repository is ready, you should first download the snapshot of the open-source legacy software system you selected and push it as the first commit of your group repository.

Please note:

- In the initial commit message, please specify the original repository link and the commit ID of the snapshot you downloaded.
- Do *not* start reengineering before pushing the initial code to your group repository. All *your reengineering efforts* must be in your group repository.

Instructions

The goal is to put all the skills and knowledge we have covered in this module. Note that completely reengineering a legacy software system may require more time and effort than is feasible for this project. Your submission (report and commits) will be assessed in terms of (a) the techniques that you have successfully demonstrated and (b) their appropriateness, including the rationales you provided. More details will be provided later in this document.

Your tasks are as follows (your report can have more sections than below):

0. Front Page

Please include the following information in the front page:

- Group ID and members.
- Group repository address (hyperlink).
- Short description of the responsibilities of each member.

1. Introduction (300-500 words[1])

Introduce the open-source software you choose and explain why you chose it.

Things to consider:

- What is the original legacy software system? What is it for? How does it work?
- Which version (commit ID) of the system did you download as your basis?

2. Understanding the System (1500-2000 words)

First, investigate the **evolution of the system** and the driving forces behind it. For example, you might plot the total number of lines over time and identify significant increases within a short time. You can then study what made such significant increases and what are the implications of them.

Second, identify **existing reengineering efforts** made by the original developers and discuss whether the efforts were successful or not. For example, you could check the commit history of the original repository and find specific commits relevant to refactoring or restructuring rather than introducing new features.

Third, discuss the **remaining code smells** in the base snapshot (i.e., the downloaded version) of the system. For example, you can use techniques you have learned from the module to identify potential code smells either automatically or manually.

(COM6523 Students Only) During the above analyses, you should additionally utilise relevant **advanced techniques** introduced in the module.

Things to consider:

- How have the system's size and complexity changed over time?
- How have the system's architecture, design, and functionality changed over time?
- What factors have influenced the system's changes?
- What reengineering efforts have been undertaken? For what reasons?
- Are the existing test cases enough for regression testing? If not, why?
- What were the outcomes of these reengineering efforts? Did they achieve their intended goals?
- What code smells persist in the current snapshot of the system?
- How significant are the identified code smells in terms of their impact on the system?

3. Restructuring the System (1500-2000 words)

First, define a set of **requirements** for restructuring. For example, you may want to focus on improving the modularity of the system. Or, you may want to remove outdated functionalities of the system, making it more compact and concise. The requirements will set the aims and objectives of the reengineering.

Second, reengineer the system using both **system-level restructuring** and **unit-level refactoring**. You might have a strategy for distributing time and resources to satisfy the requirements on time.

[1] The word counts for each section are provided as a ***guideline***. Tables, diagrams (including associated legends), appendices, references, and footnotes are ***excluded*** from the word count.

Third, **evaluate and discuss** your reengineering efforts. For example, you can check if you accidentally introduced any unintended changes using regression testing. You can also use static and dynamic analysis techniques to compare before and after the changes and demonstrate the achievements of the requirements.

(COM6523 Students Only) During the above analyses, you should additionally utilise relevant **advanced techniques** introduced in the module.

Things to consider:

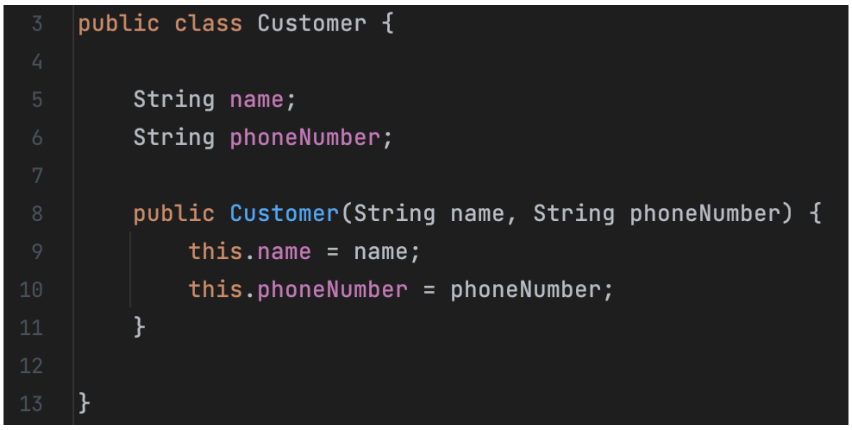
- What are the desired outcomes in terms of improved maintainability, performance, extensibility, or other quality attributes?
- What are the potential conflicts or compromises that may arise when addressing multiple requirements simultaneously?
- Are the identified requirements achievable within the given timeframe and resources? What potential challenges or limitations should be considered?
- Are there any modules that are overloaded or overburdened, or where unnecessary code exists?
- How might changes to the system affect the quality attributes you care about?
- How will you ensure that the reengineering efforts do not introduce unintended behaviours?
- What specific tests will you design to verify that the reengineered system functions as intended?
- How will you compare the system's behaviour before and after the reengineering to demonstrate the impact of the changes?

4. **Conclusion** (500-1000 words)

Reflect on what you as a group have accomplished throughout the project. Aside from the shared conclusion of the project, each member should answer the following question separately: "What were the most challenging aspects of reengineering the system, and how did you address them?"

Code in the Report

When you include code fragments as exemplar code in the report, you must use source code colouring, line numbers and name each listing. You can take a screenshot as follows:



```java
public class Customer {

    String name;
    String phoneNumber;

    public Customer(String name, String phoneNumber) {
        this.name = name;
        this.phoneNumber = phoneNumber;
    }

}
```

Listing 1. src/main/java/example/project/example/Customer.java

Please note:

- The Caption must include the folder and filename from the root directory to allow the marker to see more detail in your repository.
- You should not (typically) include a whole file listing.
- Standard code quality applies — for example, appropriate comments should be included.

Commits in the Repository

In your repository, you must make fine-grained commits with clear commit messages (e.g., “initial analysis”, “new tests added for X”, and “refactoring Y”). Note that the commits in your repository will be used for assessment. Do *not* commit everything in a single go. The point is that you make regular, atomic commits as you work on this.

Assessment Rubric

The assessment of your submission will be conducted according to the schema below.

Aspect	**0-39%**	**40-54%**	**55-70%**	**70-100%**
Introduction (15%)	Missing or incomplete.	The legacy software system under reengineering is described, but not clear nor comprehensive.	The legacy software system under reengineering is described, including some initial analysis results.	The legacy software system under reengineering is well described, including rich initial analysis results. Extensive references to evidence corroborated by the source code in the commits.
Understanding the System (30%)	Missing or incomplete.	The understanding of the system, including its evolution, existing reengineering efforts, and remaining code smells, is described with essential evidence. A few relevant techniques, such as static/dynamic analysis and software metrics, are tried. (COM6523 Only) Limited efforts using advanced techniques are described.	The understanding of the system, including its evolution, existing reengineering efforts, and remaining code smells, is described with some evidence. Some relevant techniques, such as static/dynamic analysis and software metrics, are used appropriately. (COM6523 Only) Some efforts using advanced techniques are well described.	The understanding of the system, including its evolution, existing reengineering efforts, and remaining code smells, is well described with rich evidence. Various relevant techniques, such as static/dynamic analysis and software metrics, are used appropriately. (COM6523 Only) Sufficient efforts using advanced techniques are well described.
Restructuring the System (30%)	Missing or incomplete.	The reengineering efforts, including the definition of requirements, the system-level restructuring and unit-level refactoring, and the evaluation of the results, are described with essential evidence. A few relevant techniques, such as regression testing, refactoring, and restructuring, are tried. (COM6523 Only) Limited efforts using advanced techniques are described.	The reengineering efforts, including the definition of requirements, the system-level restructuring and unit-level refactoring, and the evaluation of the results, are well described with some evidence. Some relevant techniques, such as regression testing, refactoring, and restructuring, are used appropriately. (COM6523 Only) Some efforts using advanced techniques are well described.	The reengineering efforts, including the definition of requirements, the system-level restructuring and unit-level refactoring, and the evaluation of the results, are well described with rich evidence. Various relevant techniques, such as regression testing, refactoring, and restructuring, are used appropriately. (COM6523 Only) Sufficient efforts using advanced techniques are well described.
Conclusion (15%)	Missing or incomplete.	The conclusion of the project is summarised. Individual reflections on the reengineering challenges are discussed.	The conclusion of the project is clearly summarised. Individual reflections on the reengineering challenges are discussed.	The conclusion of the project is comprehensively summarised. Individual reflections on the reengineering challenges are sufficiently discussed.

Aspect	0-39%	40-54%	55-70%	70-100%
Overall presentation (10%)	Some attempt, but unfocused with high content of irrelevant material. Inadequate style and/or grammar. Commit messages are unclear, not individual commits are separated enough.	Subdivided into sections and subsections, but not particularly logical in flow of material within divisions. Style and grammar are poor. Commit messages are unclear, yet reasonably separated among different commits.	Well-organised with a sensible subdivision of material into sections and subsections. Figures and tables are relevant and reasonably presented. Clearly written overall, but may be ambiguous in places and show lapses of grammar. Commit messages are reasonable to link the corresponding tasks.	Sensible subdivision of material into sections and subsections to produce a coherent and well-balanced report. Figures and tables are relevant and clearly presented. Unambiguous and grammatically correct English. Commit messages are clear enough to understand what each commit is about.
Peer assessment (adjustment factor)	This assesses your individual contribution to the group project. This will be executed using the Blackboard Buddycheck system by your group members. (Updated in v1.1) Notice that your peer assessment must include **evidence-based justifications** (e.g., based on the meeting minutes, task distributions on your GitHub Projects, etc.)			

Draft Submission

Your draft submission will be:

- A written **report (draft)** submitted to the Blackboard as a PDF (this is strict).

(Updated in v1.1) To receive useful and concrete feedback, we recommend that you have **Part 1** (i.e., Chapters 1 and 2) **nearly complete** when submitting your draft.

The deadline for the draft submission is **17:00 GMT+1 on 22 March 2024**. Although the draft submission is not subject to assessment, it is *mandatory* to submit a draft to get peer feedback.

Draft Feedback

Once all groups submit their drafts, each group will receive two reports to review and provide constructive feedback. Please use the **above assessment rubric** to mock the final assessment.

The deadline for the draft feedback is **17:00 GMT+1 on 26 April 2024**. Note that this peer feedback is not subject to assessment, but it is *mandatory* to provide constructive feedback.

Final Submission

Your final submission for assessment will be in two parts.

- A written **report** submitted to the Blackboard as a PDF (this is strict).
- **Commits** in your GitHub Classroom Repository.

If you have separate data files for the individual activities (e.g., csv files), if any, please create a new directory called "*Submission*" in your repository and add them to the directory.

The deadline for the final submission is **17:00 GMT+1 on 17 May 2024**.

Support

We will only respond to queries on the Blackboard forum dedicated to this assignment to ensure fairness. As usual, please read other queries before you post your own to ensure that your query has not already been answered. Also, please give your query a descriptive title to assist your colleagues when checking the forum.

You must under no circumstances post your own solutions (or parts thereof) to the forum - so please be mindful when you are posing questions.

Unfair Means

It is essential to bear in mind the departmental rules on unfair means. Activities such as plagiarism or collusion will be treated as serious academic offences. This could lead to the award of a grade of zero for this assignment.

We will be closely scrutinising submissions to detect such practices because it is important that this assessment is a genuine reflection of your own understanding of the module.

It is especially important that you do not discuss your solutions with other groups to avoid potential accidental wrongdoings. If you have questions about this assignment, please use the discussion forum.

Generative AI: We acknowledge that responsible and ethical use of GenAI (e.g., Google Bard) is an essential skill for you. Therefore, you can use GenAI in your work (i.e., reports and code commits to your team repository). However, you must follow the Acknowledge / Describe / Evidence model for doing this. Specifically, you must provide the information by completing a template and submitting it as an appendix to your report. Attempts to pass off AI-generated content as your own work are counted as unfair means and may lead to action being taken against the student.