Assignment 04

SOFTWARE REENGINEERING

SE-4001

Course Instructor:

Dr. Isma Ul Hassan

Group Members:

Zeeshan Ali	20i-2465
Ans Zeshan	20i-0543
Saad Shafiq	20i-1793
Hammad Aslam	20i-1777
Dayyan Shahzad	20i-2393

Due Date:

Dec 03, 2023

Doctor Appointment Booking System

Project Overview:

We selected an open-source project to re-engineer the project following all the strategies that we have studied in the course that how to reengineer a project code. We start evaluating the current state of the project, identifying pain points, inefficiencies, and areas that need improvement.

In first Iteration we do the following things

- ✓ Identifying Features.
- ✓ Perform analysis using legacy techniques.
- ✓ Identify strength and weaknesses.
- ✓ Suggest Improvement areas.
- ✓ Understanding of code by AST.
- ✓ Architecture using Mental Model.

In Second Iteration, we do the following things.

- ✓ Mapping of Re-Engineering Strategies.
- ✓ Reverse Engineering part
    - Control Flow graph
    - Control dependency graph
    - Data Dependency graph
    - Program Dependency graph
- ✓ Architecture & Design Information.
- ✓ Database Refactoring and restructuring.
- ✓ Software Quality Assessment
    - Qualitative Code Metrics using Function Point Analysis.
    - Maintainability Metrics.

In third Iteration, we do the following things.

- ✓ Identifying the code smells
- ✓ Suggested Refactoring Techniques
- ✓ Restructure the code.
- ✓ Improvements (Quality Assurance)

In forth Iteration, we have done with these things.

- ✓ Organizational chart (Our Work)
- ✓ Risk Analysis
- ✓ Identifying the Re-engineering Patterns
- ✓ Quality Assurance (Testing NFR)

The combination of all the four phases describes how the project is reengineer using all the tools and techniques involved in these phases. The context of the project is reengineering an outdated/unknown product.

The Problem at Hand:

The project aims to enhance various aspects of the existing system to improve its usability, security, error handling, code quality, and dependency management. The interpretation of the problem is derived from an in-depth analysis of the current state of the system, focusing on multiple dimensions such as user experience, security vulnerabilities, exception handling, code documentation, and dependency management.

User Experience Enhancement:

- ✓ **Core Problem:**
  The user experience is suboptimal due to styling issues and a lack of responsiveness in the user interface. This affects user engagement and satisfaction, leading to a need for interface improvements.
- ✓ **Complexities:**
  The complexities involved include addressing development challenges associated with correcting styling components. Additionally, compatibility issues may arise with different devices, browsers, and user adaptation to the redesigned interface.
- ✓ **Goals:**
  The goal is to not only fix the existing styling problems but also to ensure that the system provides a seamless and responsive user experience across various platforms.

Security Improvement:

- ✓ **Core Problem:**
  The system lacks robust security measures, leaving it vulnerable to SQL injection and other security threats. This poses a significant risk to sensitive data and user privacy.
- ✓ **Complexities:**
  Implementing security measures introduces complexities in terms of development effort and potential impacts on user experience. Striking a balance between security and usability is crucial.
- ✓ **Goals:**
  The primary goal is to implement security measures such as input validation, encryption, and authentication to fortify the system against potential security vulnerabilities.

Exception Handling:

- ✓ **Core Problem:**
  The system currently lacks robust exception handling, leading to unpredictable behaviour and inadequate user feedback during errors.
- ✓ **Complexities:**
  The challenge lies in implementing comprehensive exception handling without overly complicating the code. Poorly implemented exception handling can lead to error propagation, making the system harder to maintain.

- ✓ **Goals:**

  The goal is to implement robust exception handling to ensure predictable behaviour and meaningful user feedback, thereby improving the overall quality of the system.

Comments for Code:

- ✓ **Core Problem:**

  The codebase lacks comprehensive and consistently formatted comments, making it challenging for developers to understand the purpose and logic of the code.
- ✓ **Complexities:**

  Maintaining comments over time presents challenges, as code changes may not always be accompanied by updates to associated comments.
- ✓ **Goals:**

  The goal is to add clear and consistent comments throughout the codebase to enhance code readability and provide valuable documentation for developers.

In summary, the core problem involves multiple dimensions of the software system that require attention and improvement. Addressing these challenges necessitates a strategic re-engineering effort that balances usability, security, code quality, and system stability. The project's scope encompasses various tasks, including interface enhancements, security implementations, exception handling improvements, code documentation, and dependency management updates.

---------- A Mapping of Reengineering Patterns ----------

1- Setting Direction:

- ✓ Establish a clear direction to identify improvement areas.
- ✓ Define a roadmap for refactoring and improvements.
- ✓ Prioritizing tasks such as security, usability, and performance.

2- First Contact:

- ✓ Identifying the Strengths and weaknesses.
- ✓ Perform analysis using Legacy code techniques.
- ✓ Read all the Code (code review, coding styles)
- ✓ Identify the risks readability to establish a foundation for further improvements.

3- Initial Understanding

- ✓ Speculate about Design Patterns
- ✓ Speculate about Architecture
- ✓ Visualizing Metrics
  - Control Flow graph
  - Control dependency graph
  - Data Dependency graph
  - Program Dependency graph

4- Detailed Model Capture

- ✓ Tie Code and Questions (Annotate the code)
- ✓ Refactor to Understand
- ✓ Step Through the Execution (run-time architecture)

5- Test: Your Life Insurance

- ✓ Write Tests to Enable Evolution
- ✓ Use a Testing Framework (page speed)
- ✓ Test the Interface, Not the Implementation

6- Migration Strategies

- ✓ Plan for a smooth transition from the current state to the desired state.
- ✓ Css styling code extraction from php code.

7- Detecting Duplicated Code

- ✓ Management operations are the same in each module.

8- Redistribute Responsibilities

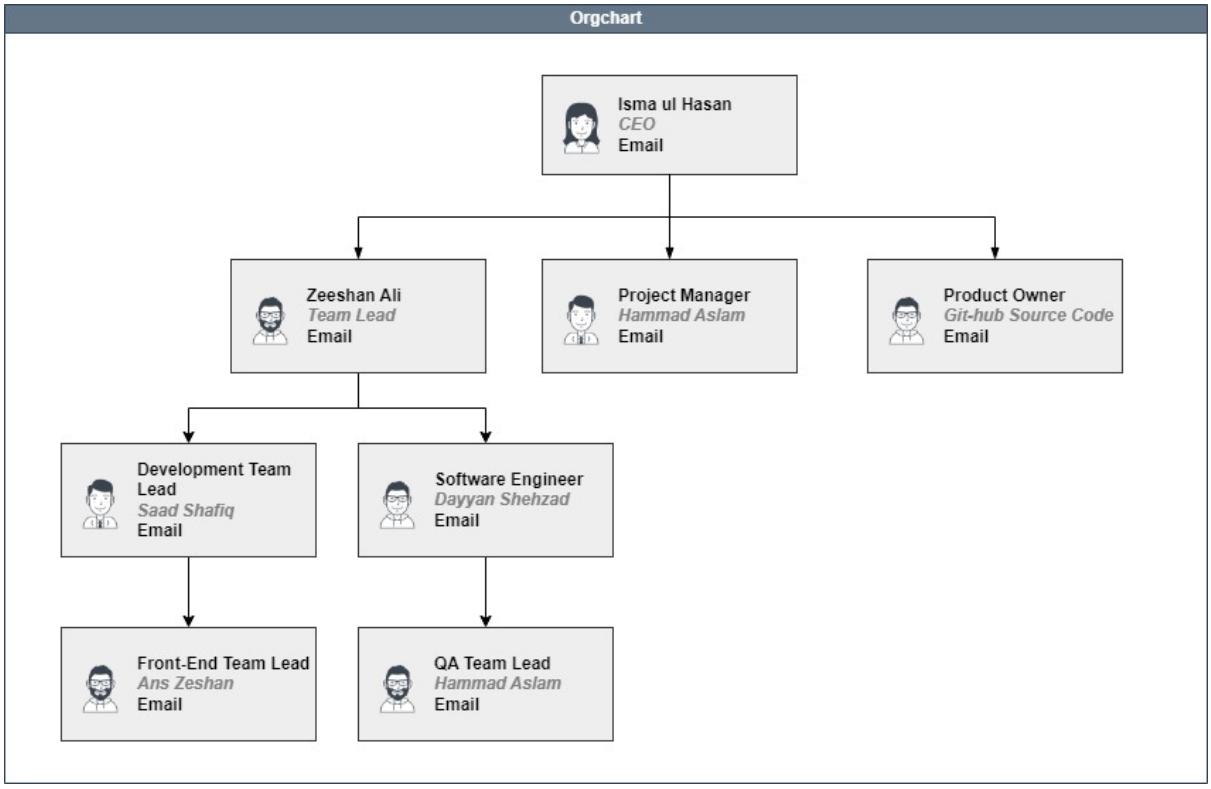
- ✓ Move Behavior Close to Data
- ✓ Transformation
- ✓ Detection strategy
- ✓ Reevaluate modular and maintainable architecture.
- ✓ Refactor code to encapsulate functionalities into functions or classes.

9- Transform Conditionals to Polymorphism

- ✓ Transform Conditional into Registration
  - ✓ Long methods which tools to invoke based on (file extension)
  - ✓ Transformation

---------- Project Management ----------

Organizational structure:



Scope:

In Scope:

1. **User Experience Enhancement:**
   - ✓ Corrections to styling components of the interface.
   - ✓ Implementation of responsive design for better usability on various devices.
2. **Security Improvement:**
   - ✓ Implementation of security measures, including input validation, encryption, and authentication.
   - ✓ Addressing SQL injection vulnerabilities and enhancing overall system security.
3. **Exception Handling:**
   - ✓ Robust implementation of exception handling to validate input and provide meaningful error messages.
   - ✓ Improved predictability of system behavior during errors.
4. **Comments for Code:**
   - ✓ Addition of comments throughout the codebase to document functionality and improve code understandability.
5. **Dependency Mapping:**

- ✓ Review and update dependencies to ensure compatibility with the latest versions.
- ✓ Implementation of dependency management tools.

Out of Scope:

1. **Major Functionality Changes:**
   - ✓ The project does not include major changes to the core functionality of the application.
2. **Complete System Rewrite:**
   - ✓ The project does not entail a complete rewrite of the existing system.
3. **Introduction of New Features:**
   - ✓ The scope does not include the introduction of new features do not present in the current system.

Risk Register:

Risk ID	Risk Description	Impact	Likelihood	Priority	Mitigation Strategy	Alternative Action
R1	Development Complexity	High	Moderate	High	Divide the project into manageable phases, prioritize tasks, and conduct thorough testing at each stage.	Adjust the project timeline and scope if complexity exceeds expectations.
R2	Compatibility Issues	Moderate	Moderate	Medium	Gradual implementation of changes, rigorous testing, and monitoring compatibility closely during and after updates.	Maintain a rollback plan to revert to previous versions in case of critical issues.
R3	User Adaptation	Moderate	Low	Medium	Implement a gradual rollout of interface changes, accompanied by user training and communication.	Provide additional user support channels and documentation.
R4	User Experience Impact	Moderate	Low	Medium	Conduct user testing throughout the	Implement an effective communication

					development process and gather feedback to address usability concerns promptly.	strategy to explain changes to users.
R5	Testing Overhead for Dependencies	High	Moderate	High	Implement automated testing where possible and conduct thorough manual testing for critical components.	Adjust testing plans based on the criticality of dependencies.
R6	Code Complexity	Moderate	Moderate	Medium	Balance comprehensive exception handling without introducing unnecessary complexity through regular code reviews.	Adjust the development plan and seek additional expertise if code complexity increases unexpectedly.
R7	Error Propagation	High	Low	High	Implement a clear and standardized approach to exception handling to minimize the risk of error propagation.	Regularly review and update the exception handling strategy.
R8	Dependency Issues	High	Low	High	Maintain a rollback plan to revert to the previous dependency versions in case of critical issues and dependencies on external factors.	Seek alternative dependencies or solutions if issues persist.
R9	Unforeseen Security Vulnerabilities	High	Low	High	Implement continuous security monitoring and	Rapid response plan for addressing newly

					conduct periodic security audits.	discovered vulnerabilities.
R10	User Resistance to Interface Changes	Moderate	Low	Medium	Implement an effective communication strategy to explain the benefits of the changes to users.	Provide additional support channels and documentation to address user concerns.
R11	Unexpected Codebase Challenges	High	Moderate	High	Include a buffer in the project timeline to accommodate unexpected challenges.	Maintain a flexible development plan that can adapt to changing circumstances.

---------- Software Reengineering Aspects ----------

Tests:

Verification Methods:

Unit Testing:

- ✓ Objective: Validate individual components or functions in isolation.
- ✓ Rationale: Ensures that each unit of code works as intended and helps catch any isolated issues early in development.

Performance Testing:

- ✓ Objective: Evaluate system performance under various conditions.
- ✓ Rationale: Ensures the system can handle expected loads and performs optimally under different scenarios.

Testing Strategy:

Test-Driven Development (TDD):

- ✓ Rationale: Writing tests before writing the actual code ensure that the code meets the specified requirements. It provides a clear set of criteria for successful implementation.

Confidence Level Assessment:

Performance and Security Testing: Moderate Confidence

- ✓ Performance and security testing contribute to moderate confidence, but real-world scenarios may reveal additional challenges.

DIAGNOSTICS

▲ Ensure text remains visible during webfont load

Leverage the `font-display` CSS feature to ensure text is user-visible while webfonts are loading. Learn more about `font-display`. FCP LCP

URL	Potential Savings
GitHub Utility 1st Party	**430 ms**
…inter/Inter-Bold.woff (zeeshanali10771.github.io)	130 ms
…inter/Inter-Regular.woff (zeeshanali10771.github.io)	150 ms
…inter/Inter-Medium.woff (zeeshanali10771.github.io)	150 ms

▲ Serve static assets with an efficient cache policy — 8 resources found

A long cache lifetime can speed up repeat visits to your page. Learn more about efficient cache policies.

URL	Cache TTL	Transfer Size
GitHub Utility 1st Party		**575 KiB**
…img/bg01.jpg (zeeshanali10771.github.io)	10m	203 KiB
…inter/Inter-Bold.woff (zeeshanali10771.github.io)	10m	126 KiB
…inter/Inter-Medium.woff (zeeshanali10771.github.io)	10m	125 KiB
…inter/Inter-Regular.woff (zeeshanali10771.github.io)	10m	114 KiB
…css/main.css (zeeshanali10771.github.io)	10m	3 KiB
…css/index.css (zeeshanali10771.github.io)	10m	1 KiB

○ Avoid chaining critical requests — 5 chains found

The Critical Request Chains below show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load. Learn how to avoid chaining critical requests. FCP LCP

Maximum critical path latency: **436.977 ms**

Initial Navigation

- /reengineering-code/ (zeeshanali10771.github.io)
  - …css/animations.css (zeeshanali10771.github.io) - **133.97 ms, 0.90 KiB**
  - …css/main.css (zeeshanali10771.github.io)
    - …css/font-inter.css (zeeshanali10771.github.io)
      - …inter/Inter-Medium.woff (zeeshanali10771.github.io) - **149.841 ms, 124.67 KiB**
      - …inter/Inter-Bold.woff (zeeshanali10771.github.io) - **129.919 ms, 126.07 KiB**
      - …inter/Inter-Regular.woff (zeeshanali10771.github.io) - **146.287 ms, 114.46 KiB**
  - …css/index.css (zeeshanali10771.github.io) - **129.423 ms, 1.24 KiB**

○ Largest Contentful Paint element — 800 ms

This is the largest contentful element painted within the viewport. Learn more about the Largest Contentful Paint element LCP

Element
 How is health today, Sounds like not good! Don't worry. Find your doctor online… `<p class="sub-text2">`

Phase	% of LCP	Timing
TTFB	35%	280 ms
Load Delay	0%	0 ms

Refactoring Applied:

1. **Separate PHP Logic from HTML:**

   Role in Enhancement:

   - ✓ This refactoring involves moving PHP code that handles session checks, database connections, and queries to separate PHP files or classes.
   - ✓ Enhances code readability by separating concerns, making it easier to maintain and understand.

2. **Create Reusable Functions:**

   Role in Enhancement:

   - ✓ Introduces reusable functions for repetitive tasks like generating table rows or menu items.
   - ✓ Reduces code duplication, making the codebase more maintainable and scalable.

3. **Externalize CSS Styles:**

   Role in Enhancement:

   - ✓ Moves inline CSS styles to an external stylesheet, improving code organization.
   - ✓ Enhances the separation of concerns by isolating styling from HTML content.

4. **Refactor HTML Structure:**

   Role in Enhancement:

   - ✓ Simplifies HTML by removing unnecessary nested tables and inline styles.
   - ✓ Promotes the use of semantic HTML tags and CSS classes for improved styling.

Overall Impact on Support of Newly Intended Features:

Code Readability and Maintenance:

- ✓ The refactored codebase is more readable, modular, and maintainable, providing a solid foundation for the incorporation of new features without introducing complexity.

Modular Components:

- ✓ The creation of reusable functions and the use of templates result in modular components that can be easily integrated into new features, promoting code reusability.

Flexibility and Adaptability:

- ✓ Externalizing styles, using configuration files, and separating concerns contribute to the system's flexibility, making it easier to adapt to new requirements and design elements.

Improved Collaboration:

- ✓ Clear separation of PHP logic from HTML and the use of external files for configuration and styles enhance collaboration among team members, enabling efficient development and integration of new features.

Overall, these refactoring efforts aim to create a more robust, flexible, and maintainable codebase, providing a solid foundation for the support and integration of newly intended features in the future development of the project.

---------- The End ----------