Software Engineering: Reflective Report

Student No: 100571823

The objective of this project was to develop a passenger management system for an airline company. Throughout the development process, various software engineering practices were employed to ensure the system was efficient, maintainable, and scalable. This reflective report documents my experiences, evaluates the effectiveness of the practices applied, and discusses the lessons learned.

One of the key successes of the project was the implementation of the Singleton pattern in managing the database connection. By using the Singleton pattern, I ensured that only one instance of the database connection was created, which improved the efficiency and resource management of the application (Stencel and Węgrzynowicz, 2008). Additionally, the use of the Factory pattern for creating Passenger objects greatly simplified object creation and encapsulated the instantiation logic. This abstraction made the code more modular and easier to extend (Ellis, Stylos, and Myers, 2007).

High cohesion was achieved in the Passenger class, as it solely focused on representing passenger data and providing methods to access this data. By ensuring that the Passenger class had a single responsibility, the code became more understandable and easier to maintain. Furthermore, the RESTful service classes were designed with low coupling in mind. Each class had well-defined responsibilities and interacted with other classes through well-defined interfaces. This separation of concerns made the system more flexible, easier to modify, and more effective in diagnosing issues (Candela et al., 2016).

Code documentation was another critical area for improvement. Initially, the code lacked comprehensive comments and documentation, making it difficult to understand and maintain. To rectify this, I added detailed comments and documentation throughout the codebase. This effort improved code readability and would facilitate collaboration if more developers were involved, as they could more easily understand the purpose and functionality of the code.

Another critical area for improvement was my initial designs for the system not matching the first version of my implementation. During the early stages of the project, I created detailed plans and diagrams to guide the development process. However, as I started coding, I quickly realized that several aspects of my design were impractical or incomplete. The architecture I had envisioned did not fully account for certain complexities and edge cases that emerged during development. This discrepancy between the design and implementation led to significant refactoring and restructuring of the code. By revising my designs and conducting regular design reviews, I was able to better align the implementation with the project goals, ultimately leading to a more cohesive and functional system.

If tasked with a similar project in the future, I would prioritize early and comprehensive documentation to save significant time and effort in the long run, making the code more accessible and easier to maintain. Additionally, adopting a more modular approach by breaking down large functions into smaller, reusable components would enhance code readability and maintainability. Implementing a comprehensive error-handling strategy from the beginning would also be a priority. This includes standardizing server responses and ensuring that client-side functions handle errors better. Effective error handling not only improves the user experience but also simplifies debugging and maintenance.

The use of design patterns such as Singleton and Factory was guided by their ability to promote code reusability and maintainability. Emphasizing low coupling and high cohesion was crucial as it facilitated easier modification and extension of the system. Regular refactoring sessions helped maintain a clean and efficient codebase by addressing technical debt and improving code quality.

In conclusion, this project underscored the importance of flexible and iterative design, comprehensive documentation, and the strategic use of design patterns to achieve a robust and maintainable system. Moving forward, I will apply these lessons to ensure better planning, execution, and maintenance in future projects, ultimately leading to higher quality outcomes.

**Reference list**

Candela, I., Bavota, G., Russo, B. and Oliveto, R. (2016). Using Cohesion and Coupling for Software Remodularization. *ACM Transactions on Software Engineering and Methodology*, 25(3), pp.1–28. doi:https://doi.org/10.1145/2928268.

Ellis, B., Stylos, J. and Myers, B. (2007). *The Factory Pattern in API Design: A Usability Evaluation*. [online] ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org/abstract/document/4222592.

Stencel, K. and Węgrzynowicz, P. (2008). Implementation Variants of the Singleton Design Pattern. *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*, 5333, pp.396–406. doi:https://doi.org/10.1007/978-3-540-88875-8\_61.