**CS-499 2-1 Journal**

I wrote this journal as a reflection of the video code review that I compiled. As a result, I will be speaking in past tense as I think through why I did what I did.

**What is Code Review?**

Code review is a systematic examination of computer source code, conducted by one or more people other than the original author. Its primary aim is to find mistakes overlooked in the initial development phase, ensuring the overall quality of the software. This process encourages a collaborative working environment where team members can share knowledge and collectively improve the code base.

**Why is it an Important Practice for Computer Science Professionals?**

Code review is crucial for maintaining a high standard of code quality and functionality. It helps in identifying bugs and errors that might not be evident to the original developer. Through peer scrutiny, it ensures code efficiency, readability, and maintainability. Furthermore, code review fosters learning and knowledge sharing among developers, encouraging a culture of collective code ownership and continuous improvement. This practice reduces the cost of future bug fixes, enhances security, and accelerates the development process by ensuring that issues are addressed early on.

**What are some code review best practices that you read about in the resources that are crucial to include in a code review?**

*Early and Frequent Reviews:* Code review should occur early in the development process and continue frequently. Early reviews help in identifying potential issues before they become deeply integrated into the project, making them easier and less costly to address.

*Keep Changes Small:* Reviews should be conducted on small changes. Large changes are harder to understand and evaluate, increasing the likelihood of oversight. Smaller, more manageable pieces of code ensure thorough examination and understanding.

*Clear Goals and Checklists:* Having clear objectives and a checklist can guide reviewers to focus on the critical aspects of the code, such as functionality, readability, performance, and security. This structured approach ensures a comprehensive review process.

*Automate Where Possible:* Automating mundane tasks (e.g., coding standard checks) can save time and allow reviewers to focus on more significant issues. Automated tools can catch common errors, ensuring that human reviewers can concentrate on logic, design, and more complex problems.

*Constructive Feedback:* The feedback during a code review should be constructive and focused on the code, not the individual. This promotes a positive culture of learning and improvement, encouraging developers to be receptive to feedback.

*Foster a Learning Environment:* Code reviews are an excellent opportunity for knowledge sharing and learning. Reviewers should share not only what issues exist but also why they are issues and how they can be improved or resolved.

Integrating code review early and throughout the software development process is critical. It should ideally occur after a piece of code is completed but before it merges into the main codebase. This timing ensures that any issues can be addressed before they impact the broader project, reducing the technical debt and preventing the accumulation of bugs. Early reviews facilitate easier and less costly corrections, promoting a more agile and efficient development process. By making code review a fundamental part of the development lifecycle, teams can maintain high code quality, reduce future debugging efforts, and enhance software reliability and performance.

In summary, code review is an indispensable practice for ensuring high-quality software development. By adhering to best practices and integrating reviews into the early stages of development, computer science professionals can enhance code quality, foster a collaborative and learning-oriented environment, and ultimately deliver superior software products.

**What software have you chosen to use to record your code review?**

I chose Microsoft Clipchamp as my software for recording code reviews. This decision was guided by several key factors that align with the objectives for improving code quality, fostering collaboration, and enhancing learning opportunities within the team. Microsoft Clipchamp offers an intuitive user interface that ensured I was able to quickly learn to utilize it, having little-to-no experience with video editing software.

**Describe your approach to creating an outline or writing a script for your code review for each of the three categories that you will be reviewing based on the rubric as well as the code review checklist.**

I chose to craft a script, so that my thoughts were conveyed clearly, concisely, and aligned with course objectives. To craft a script for the code review covering the three distinct categories—Software Design and Engineering, Algorithms and Data Structures, and Databases—based on the rubric and the provided code review checklist, I followed a structured approach. This approach ensured clarity, thoroughness, and relevance throughout the review process. Here are the objectives I covered in my narrative, captured in a loose outline:

1. Introduction to the Category

Objective: Start with a clear statement of what the category focuses on and its importance in the context of the project. This sets the stage for the audience, providing a lens through which they can view the upcoming analysis.

Example: For Software Design and Engineering, introduce the significance of secure, efficient, and maintainable code structure.

2. Overview of Review Findings

Objective: Summarize the key findings from the initial review, based on the checklist. Highlight both the strengths and areas needing improvement in the existing code related to the category at hand.

Example: In Algorithms and Data Structures, discuss how the current linear search approach impacts performance and propose more efficient data structures.

3. Detailed Analysis Using the Checklist

Objective: Break down the checklist items applicable to the category, providing a detailed analysis of how the existing code measures up against each point. This includes examples from the code where improvements are needed or where the code already excels.

Example: For Databases, analyze the current file-based storage system's limitations and discuss how database integration can enhance security and scalability.

4. Specific Examples and Enhancement Proposals

Objective: Offer concrete examples of code improvements or rewrites that address the identified issues. This includes pseudo-code or real code snippets to illustrate the enhancements clearly.

Example: Show how to replace MD5 hashing with bcrypt in Python, including sample code snippets.

5. Rationale for Enhancements

Objective: Explain why these enhancements are recommended, focusing on the benefits they bring, such as increased security, performance improvements, or better maintainability. This helps justify the proposed changes to stakeholders.

Example: Discuss the security vulnerabilities of MD5 and how bcrypt offers a more secure alternative that guards against brute-force attacks.

6. Conclusion and Future Directions

Objective: Wrap up the category review with a summary of the key points and how the proposed enhancements align with best practices and project goals. Also, suggest areas for future development or further investigation.

Example: Conclude with how the enhancements in data handling and security pave the way for future features, such as multi-factor authentication or scalability to handle more users.

7. Integration with Course Outcomes

Objective: Connect the enhancements and findings back to the course outcomes, demonstrating how the proposed changes support broader learning goals, such as understanding of algorithms, mastery of software engineering principles, or proficiency in database management.

Example: Highlight how the transition to a database-supported system exemplifies practical application of database management skills and principles.

By following this structured approach for each category, I was able to provide a comprehensive, insightful, and actionable code review that not only identifies areas for improvement but also highlights the best practices and rationale behind each enhancement.

***References:***

Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code

review. In Proceedings of the 2013 International Conference on Software Engineering

(pp. 712-721). IEEE Press.

Bosu, A., Carver, J. C., Bird, C., Orbeck, J., & Chockley, C. (2017). Characteristics of useful

code reviews: An empirical study at Microsoft. In Proceedings of the 39th International

Conference on Software Engineering (pp. 582-593). IEEE/ACM.