Module Two Journal

Hector Maldonado

Southern New Hampshire University

CS-499: Computer Science Capstone

Professor Ramsey Kraya

7/9/2025

Part 1: Code Review Fundamentals

What is code review?

A code review is a systematic process where one or more developers examine another developer's code before it is merged into the main codebase. The primary goal is to ensure code quality, correctness, maintainability, and adherence to project standards. Code reviews can be performed in various ways, such as pair programming, over-the-shoulder reviews, or using dedicated code review tools.

Why is it an important practice for computer science professionals?

Code review is essential for several reasons:

- **Improves Code Quality:** It helps catch bugs, logic errors, and security vulnerabilities early, before code reaches production.
- **Knowledge Sharing:** Reviewing code exposes team members to different parts of the codebase, spreading knowledge and reducing the risk of information silos.
- **Mentorship/Learning:** Junior developers learn from more experienced peers, and even senior developers can pick up new techniques or spot bad habits.
- **Consistency/Standards:** Code reviews enforce coding standards and style guides, ensuring a consistent codebase.
- **Collaboration/Team Ownership:** They foster a collaborative environment where everyone feels responsible for the quality of the code.
- **Security/Compliance:** Reviews help identify and mitigate security issues and ensure compliance with organizational or industry standards.

What are some code review best practices that are crucial to include?

Some best practices from the resources and the code review checklist

include:

- **Review for Overall Design**: Ensure the code integrates well with the system, is not overly complex, and solves the intended problem without over-engineering.

- **Check for Correctness**: Look for logic errors, edge cases, and potential bugs. Make sure all requirements are met and that the code does what it's supposed to do.
- **Test Coverage**: Verify that appropriate unit, integration, or end-to-end tests are included and that they are meaningful and maintainable.
- **Naming/Readability**: Ensure variables, functions, and classes have clear, descriptive names. Code should be easy to read and understand.
- **Documentation/Comments:** Comments should explain why code exists, not what it does. Code should be self-explanatory where possible, with documentation for complex logic.
- **Style/Formatting**: Code should follow the project's style guide. Minor style issues not covered by the guide can be suggested as "nits" but shouldn't block merging.
- **Avoid Large Reviews**: Limit code reviews to manageable chunks (ideally 200-400 lines per session) to maintain reviewer focus and effectiveness.
- **Use Checklists**: Employ a code review checklist to ensure common issues are not missed and to standardize the review process.
- **Collaborative/Respectful Tone**: Reviews should be constructive, focusing on the code, not the coder. Encourage learning and improvement.
- **Timely Reviews**: Reviews should be performed promptly to avoid blocking progress and to keep context fresh.

When should a code review occur in the development process, and why?

Code reviews should occur after code is written and tested, but before it is merged into the main branch. This timing ensures that:

- The code is as complete and correct as possible, so the reviewer's time is spent on things machines can't catch (like design, readability, and maintainability).
- Bugs and issues are caught before they reach production, reducing the cost and risk of fixing them later.
- The codebase remains stable and high-quality, as only reviewed and approved code is merged.

In agile or continuous integration workflows, code reviews are typically integrated into the pull request or merge request process, right before merging.

Part 2: My Code Review Recording Approach

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

For my code review screencast, I am using OBS Studio. OBS is a free, open-source tool that allows me to record my screen, capture audio, and switch between different windows or scenes as needed. It's reliable and widely used for both live streaming and recording presentations.

Describe your approach to creating an outline or writing a script for your code review for each of the three categories.

General Approach

- **Preparation**: I start by thoroughly reading through the codebase, referencing the code review checklist and the rubric. I annotate areas of interest or concern and make notes on potential enhancements.
- **Outline Creation**: I create a detailed outline for the screencast, breaking it down into the three required categories: Software Engineering and Design, Algorithms and Data Structures, and Databases.
- **Script Writing**: For each category, I write a script that covers:
- A clear description of the existing code's functionality.
- A critical analysis using the checklist, highlighting strengths, weaknesses, and limitations.
- A practical, organized plan for enhancements, referencing my initial enhancement plan.

Category-Specific Approach

1. Software Engineering and Design

- I describe the overall architecture, modularity, and design patterns in use.
- I analyze code structure, naming conventions, documentation, and error handling.
- I identify areas for refactoring, improved modularity, and documentation.

- I outline enhancements such as breaking up large classes, introducing design patterns, and improving comments and documentation.

2. Algorithms and Data Structures

- I explain how data is currently managed (e.g., arrays, vectors, procedural mesh generation).
- I review the efficiency and appropriateness of data structures and algorithms.
- I look for inefficiencies, lack of spatial partitioning, or missed optimization opportunities.
- I propose enhancements like implementing a scene graph, spatial partitioning (quadtree/octree), and optimizing mesh generation.

3. Databases

- I note the current lack of persistent storage or database integration.
- I discuss the limitations this imposes (e.g., no saved user preferences or scene configurations).
- I propose integrating a lightweight database (like SQLite or JSON), adding serialization/deserialization routines, and validating all data I/O for security.

Using the Checklist

Throughout the review, I reference the code review checklist

to ensure I cover:

- Structure/modularity
- Documentation/comments
- Variable naming/type safety
- Defensive programming/error handling
- Efficiency/maintainability

Recording Strategy

- I use OBS to record my screen as I walk through the code and my outline.
- I keep the tone conversational and professional, focusing on content and analysis rather than perfection.
- I pause to show relevant code sections, explain my reasoning, and discuss trade-offs.
- I conclude each section with a summary of planned enhancements and how they align with course outcomes and professional best practices.

This approach ensures my code review is thorough, organized, and aligned with both academic/industry standards.