CS-499 2-1 Journal

Chris Marrs

11/10/2024

**Journal Entry: What Makes a Productive Code Review?**

**Part 1**

1. **What is code review?**

Code reviews consist of systematic assessments that are designed to find bugs in code, improve code quality and teach developers the source code.

A code review, after a developer completes the coding process, is an essential step to the development of software. It allows a second viewpoint on the implementation and solution before the branch can be merged with an upstream branch such as a feature or main branch. Reviewers can act as second steps in finding bugs, logic issues, edge cases or other issues.

1. **Why is it an important practice for computer science professionals?**

Developing a strong code review process sets a foundation for continuous improvement and prevents unstable code from shipping to customers. Code reviews should become part of a software development team’s process to improve code quality and provide transparency to the codebase. The code review process is also an important part in spreading knowledge throughout an organization.

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

Limit the time spent on code reviews to maintain productivity. Decide what limits work for the team, such as no more than 200 lines or one hour of code. Encourage them to adhere to this limit.

Include everyone, including senior and new members of the team. Code reviews can be a great way for newer team members to get familiar with the codebase. They are a good way to do this by having the senior developers review their code and by also reviewing the code of other team members. Include everyone in the code review process to make it easier for the team to adjust when someone goes on vacation or is no longer part of the team.

Distribute requests for code reviews among the team. It's easy for some developers to receive the majority of code review requests. This is not good for the rest of the team or for the code base in the long run. To avoid this, create a list with domain experts or use a reviewer's roulette.

Ask questions and give context. You should try to learn from the code of others when reviewing it. Not sure why someone did something differently, or have a suggestion on how to improve the code? In the comment, add the reason for the suggestion this.

**Part 2**

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

For my code review, I’ve chosen to use Wondershare Filmora Studio to record the session. The studio is very low coast and feature rich that allows for flexible screen recording, making it easy to capture code demonstrations and walkthroughs. The studio supports multiple recording sources.

1. **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.**

For my code review, I’ve created a structured outline for each category, focusing on specific aspects as guided by the rubric and code review checklist.

Software Engineering and Design:

* Objective: Analyze the overall structure, modularity, documentation, and adherence to coding standards.
* Checklist Focus: I examine the code for redundant logic, opportunities for modularization, and coding style consistency. Highlight any "magic numbers" or blocks of code that could be replaced with reusable functions or constants.
* Script Outline: An overview of the main design, then point out areas where modular functions could improve readability and reusability. Review commenting style and any areas where documentation could be enhanced.

Algorithms and Data Structures:

* Objective: Look at the logic and fit-for-purpose of data structures used within code. Most importantly within its main loop where tasks are executed at regular time intervals.
* Checklist Focus: Assess algorithmic efficiency, opportunities to optimize logic and whether common cases are handled efficiently as well as any unnecessary complexity that needs refactoring.
* Script Outline: Overview of the approach while noting areas for potential improvement within time-based logic. Also discussing the plan to implement a priority queue scheduler, including how it addresses specific checklist items such as loop optimization or minimizing repeated logic.

Databases:

* Objective: Develop a database solution which addresses the absence of persistent data storage within code.
* Checklist Focus: As there's currently no database functionality in my code, with a priority to create a secure database connection using MongoDB with RESTful API access for data secure storage and secure retrieval.
* Script Outline: Describe the shortcomings in current code regarding data persistence, then lay out the plan to incorporate MongoDB as part of it. Furthermore, emphasize error handling techniques as well as defensive programming practices to protect integrity and stability for data.

Personal Progress Update:

So far, I have reviewed my artifacts and identified key areas for improvement using the rubric and checklist. Outline/script preparation for code review has begun with specific code sections marked for improvement. I am also developing enhancement plans focusing on modularization, algorithm optimization and database integration in each of these categories.