2-1 JOURNAL

2-1 Journal: What Makes a Productive Code Review?

CS-499-12473-M01 Computer Science Capstone 2024 C-5 (Sept-Oct)

Michael Zietz

Southern New Hampshire University

Sept 14, 2024

In the Software Development Life Cycle, SDLC for short, code reviews are used at multiple levels throughout a project’s development. They serve as an in-depth analysis of a code base before proposed enhancements are implemented into the existing features, and in larger projects, they are used as a frequent check-in to make sure that deliverables are being adequately and securely met. Here, logic, structure, documentation, security, and variable/function naming conventions are discussed and critiqued for the purpose of giving developers reason to reflect on current work and announce areas that need improvement in addition to the active backlog of features yet created. (Yang et al., 2024) Otherwise, vulnerabilities or errors are likely to be missed and left in the final version of the product. In fact, it has been proven that code reviews improve discovering software issues, improving code quality, and increasing programming knowledge communications. (Yang et al., 2024) A great place to insert a code review is between each sprint of development in the agile methodology.

Once a sprint’s goals and timeline have been achieved, it is an excellent chance for a development team to take the time to review what was made during the sprint, how it improves the original code base, and where areas need improvement in the next sprint or to be added to the backlog. I, personally, do not recommend performing a rigid and thorough code review during a sprint’s lifespan because these processes can slow down progress and possibly conflict with deadlines. This is different from if an issue is identified mid-sprint and needs to be assessed immediately or added to the backlog.

The software I chose to record and partially edit my code review videos was ScreenPal. It’s a free application that can be installed locally on one’s computer. It supports an easy-to-use environment that lets me start new recordings, manage the recording window’s size, quickly retake audios mid recording, and edit silents gaps out of the recording’s final cut. The only downside was this software’s free version only allows fifteen minutes of recording for one file, so I ended up needing to create multiple recordings and edit them together using Microsoft’s built-in video editing software, Microsoft Clipchamp.

My approach for creating an outline and writing a script for each of my code review videos was to first re-familiarize myself with the documents pertinent to the artifact. From there, I created an introduction for each artifact, explaining the premise and solution it aims to provide. I recorded an overview of the code base and introduced the audience to the logic and structure of the artifact. I then followed the provided code review check list and re-read each document with a focus on the major categories, structure, documentation, variables, arithmetic, operations, loops/branches, and defensive programming. At each checklist point, I would assess if it was relevant to the artifact at hand, and if so, write a discussion narrative as to why. This checklist served as my analysis for the code base, and when I finished my explanation, I would state the enhancements I wanted to implement as well as areas that needed to be adjusted. Finally, I restated the skills and outcomes I wanted to emphasize for each artifact.

Resources

Yang, Z., Gao, C., Guo, Z., Li, Z., Liu, K., Xia, X., & Zhou, Y. (2024). A Survey on Modern

Code Review: Progresses, Challenges and Opportunities.