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

Ryan DeBraal

CS-499-X6397 Computer Science Capstone 21EW6

A code review is as important to a software developer as a second draft is to an author. Good software is not written in bursts of inspiration but rather honed and perfected over the course of many iterations.

Code reviews come in a variety of forms including pair programming, over-the-shoulder presentations, and even tool assisted code checks. A typical code review usually consists of one developer reviewing another developer's code and trying to identify mistakes.

This process has been shown to improve overall code quality in the same way that a proofreader can more easily identify another writer's mistakes in spelling or grammar because they do not have a personal association with the code.

When someone reviews code they should be looking for common issues such as missed data validation checks or null exceptions, but they should also be conscience of the overall solution that the code is attempting to attain and be thinking about ways on which the process could be improved.

Code reviews should occur frequently throughout the development process. The Agile process is specifically designed to place code reviews at pivotal junctures throughout the SDLC. After a user story is completed and submitted as a pull request, a developer should be made responsible for reviewing the code. This way there is a system of checks in place to make sure that more than one person has signed off on a piece of code and that responsibility for the correct functioning of a piece of code is shared between multiple developers.

Personally, I have found live demonstrations to be the most effective method of code review. It has been my experience that software developers can become very myopic when looking at the code rather than how the code functions from the perspective of the user. No code is bug free, and all code can always be improved in one way or another, but it is important that the developer understands that what they are building is ultimately to satisfy a customer's needs. So, the first step in developing code is to make it do what the customer wants, the second step should be to retain the quality of the code but still make sure that it does what the customer wants. I have seen numerous occasions when code refactoring has resulted in much cleaner code that no longer performs the basic operations that the "spaghetti code" mysteriously was able to achieve.