1

WHY ARE CODE REVIEWS SO FRUSTRATING?

Software is taking over our world. These days, nearly every company or organization needs some amount of software to run smoothly. From e-commerce to work order systems, banking to health care, software continues to grow and grow.

As such, the interest in becoming a software developer has grown too. The potential for strong salaries and flexible schedules continues to entice increasingly more people to open up a terminal and learn to code.

But with the boom of software comes a cost. Software development has increased in complexity, responsibility, and competition. Companies are competing against each other to have the best product and be the first to bring it to market. This new competition is leading teams to be as nimble (dare I say Agile) as possible. Companies are looking to implement the most effective processes and win by delivering innovative value.

This pattern leaves many developers feeling exhausted, overwhelmed, and oftentimes burnt out. With new technologies to master and intense deadlines, developers often feel perpetually behind. To evade burnout and find a sustainable pace, we begin looking at the industry for best practices and tools. We seek guidance on how to navigate the landscape and hopefully arrive where we want— with a team working effectively, delivering quality software with speed, and without falling apart.

One best practice often used is code reviews. Code reviews are the process of inspection in which one or several people check the quality of software by viewing and reading parts of its source code. Research promises that code reviews eventually speed up your process by keeping quality and maintainability high.[1] Organizationally, code reviews are a powerful tool for not only keeping software quality high but for keeping costs low.[2] Many organizations now require code reviews and have a minimum of one approver before allowing any code to be merged to the mainline.

However, not every engineer agrees that code reviews are good. Developers seem to fall into two fiercely held camps wherein code reviews are either completely useless or extremely useful. You don't have to look hard to find articles about how code reviews are ruining your team.[3] When not done properly, code reviews tend to be a source of frustration and conflict. Additionally, code reviews tend to be the lengthiest part of a typical development life cycle.[4]

Why is there such a disparity between the research and everyday experiences of developers? Why do some teams love code reviews while others hate them?

As a developer who has done hundreds of code reviews and had my own code reviewed to the same degree, I think the reason is simple: we don't know *how* to review code. We've only been told that we should. We don't know how because no one taught us.

1 Bird and Bacchelli (2013)

2 Fagan (1986)

3 Fox (2020)

4 Czerwonka and Greiler (2015)

NO ONE TEACHES CODE REVIEWS

If you were taught the skills of reviewing code at your college or boot camp, you are part of a select few to ever receive such training. Most software development programs focus on the technical aspects of building software—coding, understanding computer science theory, algorithm analysis, etc. This makes sense because these programs are designed to get students jobs as developers. Very few software organizations want to hire a developer who doesn't know the core basics of writing software.

But a largely neglected part of building software is how to build software *as a team*. Some programs have robust curriculums that require students to work on teams for some of their assignments instead of working alone, which is a great step. But few programs discuss the inner workings of how to succeed as a team. There's an expectation that teams will learn to work together, but the concrete skills of *how* to work together—such as when performing code reviews—are missed.

This means that most developers are learning the art and science of code reviews on the fly in their first professional role. Some of us will gain experience during our internships if we're lucky. Even so, few of us are shown what to look for, how to give good feedback, and how to determine what issues are important enough to insist on being changed. We weren't taught how to respond when our own code gets reviewed either.

The purpose of this book is to teach you those skills—to teach you how to be a great code reviewer. More than that, this book will show you how to be a champion of the entire code review process and give you the tools you need to help your team's code review practice become the best it can be.

But before digging into that, let's talk about some of the key obstacles we all experience that prevent us from being great code reviewers. We need to acknowledge and understand these obstacles before we can start learning skills to overcome them.

These obstacles, or blockers, come in the form of four *fears*. Let's dive into each one.

FEAR OF LOOKING DUMB

No one wants to look dumb.[5] We want to appear competent and professional at a minimum while most of us aspire to be considered an expert in at least one domain. We want people to respect our opinions, listen to our advice, and esteem us in their minds. We want to be viewed as credible and intelligent. Case in point: I even found a source to cite, so I didn't look dumb saying we are afraid of looking dumb!

Yet we all know that such fears often lead to the very outcome we want to avoid. Our fear of looking dumb keeps us anxious,

5 "Afraid of Looking Dumb" (2013)

uptight, and unwilling to share ideas. It keeps us in our comfort zone, preventing us from learning new skills and expanding our responsibilities.

In the context of highly technical and complex work such as coding, this fear can be pervasive. Many junior (and senior!) developers feel like impostors when joining a new team; they are unsure of their skills and whether they can do the job. They might ask themselves, "Do I know what I'm doing?" when in fact, many developers quickly realize that they *don't* know what they are doing and need help in their day-to-day work.

A code review can be a spotlight of sorts illuminating our strengths and weaknesses. Whether presenting our own code or reviewing someone else's, we're trying to ensure that we put our best foot forward. Whether a junior, mid-level, or even senior developer, we want to appear knowledgeable and skilled. We become afraid of any misstep or incorrect comment as we fear being exposed as an imposter.

Overcoming this fear is tricky. Gaining more knowledge won't do it. Instead, we must start with sharing what we know, with confidence and a willingness to learn. This book will give you tools and techniques for learning how to give feedback with such humility and willingness to help you overcome this fear.

FEAR OF CONFLICT

On the other side, many developers might hold back comments for fear of creating conflict. No one wants to look like a gatekeeper, and no one wants to be an instigator (at least well-meaning people don't).

For example, suppose a young developer has left a rather blunt but correct suggestion on a code review just the other day on how to prevent a bug. Today they are greeted by a comment from the author debating the merits of their suggestions. Over the course of the day, a long thread builds up between the developer and the code's author. Neither seems willing to collaborate on a solution. There is tension within the team, and the two developers are finding it hard to work together in other meetings throughout the rest of the week.

The next week, a different developer on the same team has a concern about implementation. However, having seen the tension from such a comment last week, this developer is afraid of creating more conflict. As a result, they might withhold their suggestions to remain in the tribe of their team, even if this means that a security concern makes it to production.

The root of this conflict comes from our inability to give and receive feedback. Often, the missing ingredients are humility and patience. Once we learn how to stay humble, objective, and diffuse conflict, we can quell our fear of creating conflict.

FEAR OF SPEAKING UP

Another fear might be that a developer, who is certain they are right about an error in code, is afraid to say anything about the code of a senior engineer. They are afraid of looking dumb and saying something wrong to the leader of the team. Senior engineers don't make mistakes anyway, right?

The fear of speaking up is a mix of the fear of looking dumb and the fear of conflict. The differentiator here is that the context is often due to an authority figure, such as a manager or senior engineer. It can be intimidating to say, "I think you're wrong" to someone with four or five times your experience.

Fear of speaking up is sadly pervasive in many technical domains. When this fear exists on a team, details get overlooked, and warnings are never spoken, leading to devastating consequences. One such example is Korean Airlines, which had several plane crashes in the late 90s and early 2000s. An investigation found that their hierarchical culture was keeping junior officers from pointing out errors or mistakes to superior officers.[6] Respect is one thing, but we can't pull all our feedback, or we very well might head for disaster.

The fear of speaking up also comes from groupthink, where all team members seem to agree except for one. The authority here isn't one person but the combined authority of the group.

6 DeHart (2013)

It can be disorienting to be the only person willing to speak up on an issue. We start to question ourselves again… "Do I know what I'm talking about?"

FEAR OF SLOWING THINGS DOWN

Another fear is that by giving a thorough review, the reviewer will slow down the team. It's a reasonable fear, too, as we've already pointed out that code reviews can be one of the lengthiest parts of the development cycle.

This fear, though, has more to do with the sentiment that we as individuals review code too slowly, not necessarily the process overall. We are afraid that our speed of reviewing code will slow down our teammates. In a way, this fear stems most directly from the initial core issue: we weren't taught the skills to be effective reviewers. Code reviewing is a skill that can be taught, refined, and mastered—just like every other skill in the world.

However, there is more to the story. The reviewer isn't the only part of the equation when it comes to how long a review takes. As you'll see, the time it takes to review code is just as dependent on the author as the reviewer. There is also more than one way to review code and minimize waiting times that teams should aim to leverage when possible (hint: what type of programming happens when two people work together on the same code at the same time?).

This is where teaching a whole *team* great code review practice becomes important. And our whole team will need to learn how to review code to overcome this fear effectively.

To recap this chapter, code reviews are a great tool that we can use to inspect our code and build better software. However, they can be frustrating because we were never taught how to be great code reviewers. We were never taught how to create a code review practice that serves and strengthens our teams, but we were told we needed to do it anyway.

Let's start learning how to review code the right way. We must develop a practice of reviewing code, remembering that practice is the real path toward mastery and growth.

Building such a code review practice begins differently than you might think. It begins by learning how to receive feedback.