97 Things Every Programmer Should Know

– COLLECTIVE WISDOM FROM THE EXPERTS

Chapter 21-30

Chapter 31: Floating-Point Numbers Aren’t Real

In programming, floating-point numbers are a bit like close guesses of real numbers. They're useful for science stuff but can be a bit off. It's like using a ruler that rounds to the nearest inch—it's not always super precise. So, it's important to know their limits and use them carefully.

Chapter 32: Fulfill Your Ambitions with Open Source

If you're not satisfied with your work projects, open source can be considered. It lets you be into projects you're passionate about, learn from others, and make meaningful contributions. Sure, it might mean sacrificing some free time, but the rewards are worth it. Just be mindful of your commitments and dive in. It's a fantastic way to grow as a programmer and pursue your dreams.

Chapter 33: The Golden Rule of API Design

When making software tools for others, like APIs, it's like building a puzzle. If you seal too many pieces, it's hard for it to fit them together. But if you leave them too loose, the puzzle might fall apart. So, it's essential to strike a balance between flexibility and stability. Testing your tools thoroughly helps ensure they fit into others' projects smoothly, like finding the right puzzle pieces.

Chapter 34: The Guru Myth

In software, there's a belief in "gurus" who have all the answers magically. But even the best programmers use logic and learn over time. We should focus on helping each other grow instead of relying on mythical experts.

Chapter 35: Hard work does Pay-off

As someone who's been in the programming field for a while, I've learned that putting in endless hours doesn't always yield the best results. It's tempting to believe that working harder means accomplishing more, but in reality, it's about working smarter. Taking breaks, reflecting on progress, and continuously learning are essential for long-term success. Programming is like navigating through a maze; rushing through it won't get you to the finish line any faster. So, I've embraced a more balanced approach, dedicating time to both work and personal growth. It's not about how many hours you put in; it's about the quality of those hours and the lessons learned along the way.

Chapter 36: How to Use a Bug Tracker

Using a bug tracker effectively is crucial for smooth project management. When reporting a bug, be detailed about how to reproduce it and what you expected to happen. Provide context and avoid blaming others, as bugs are part of a collective conversation. When changing a bug's status or priority, explain your reasoning to save time later. Avoid overloading fields for personal use and ensure everyone knows how to find and prioritize bugs. A bug is not a standard unit of work; it's just one part of the development process.

Chapter 37 : Improve Code By Improving It

Improving code doesn't always mean adding more; sometimes, it's about taking away. Improving code often involves simplifying it by removing unnecessary parts. Recently, I've been doing just that in our codebase, cutting out features that were over implemented and slowing down the product. Often, it's because of the temptation to add extra features for fun or uncertainty about future needs.

Chapter 38: Install Me

When it comes to installing software, I'm all about convenience and clarity. If I can't quickly grasp where the program will be stored and how it will enhance my workflow, I'm less inclined to proceed. Transparency is key—I want to feel in control of my system and confident in my ability to remove any software that doesn't meet my needs. Provide me with a straightforward installation process and clear instructions, and I'll be more likely to engage with your software. But if it feels overly complex or uncertain, I'll likely seek alternatives.

Chapter 39: Interprocess Communication Affects Application Response Time

Communication between different parts of a program affects how quickly it responds. When one part talks to another remotely, it can slow things down, especially if they have to talk one after the other. For example, when an app asks a server for lots of small things one at a time, it can take a while. To make it faster, we can try sending only the important stuff, doing things at the same time when possible, or storing previous answers to avoid asking again. Thinking about how parts of a program talk to each other is important for making it run faster.

Chapter 40 : Keep your Code Clean

Keep your code clean by addressing compiler warnings promptly. Even if a warning seems minor, deal with it to avoid clutter and maintain readability. Fixing warnings, even those seemingly insignificant, helps maintain code quality and makes it easier for others to understand and work with your code. Don't ignore warnings or let them pile up, as they are crucial for maintaining code hygiene and ensuring a smooth development process.