97 Things Every Programmer Should Know

– COLLECTIVE WISDOM FROM THE EXPERTS

Chapter 11-20

Chapter 11: Code in the Language of the Domain

The passage explains the importance of coding with domain-specific terms rather than abstract data structures. By making domain explicit in the code, developers enhance readability, maintainability, and adaptability, fostering a clearer understanding of the system's intent and facilitating future changes.

Chapter 12: Code Is Design

I thought that code is just a structure of functions to make the system work. It's design, making it too complacent about what you do for the sake of complying, it will surely affect the project's performance and status. - Sometimes the design of what we thought is done, it's not. It requires several tests to know its status and capability. When you design you must dedicate yourself upon the mastery of your work, in that sense it will mirror the output. Being passionate about something will greatly affect the output of what you are doing.

Chapter 13: Code Reviews

I never considered a layout when coding. In making a good documentation, one must have to set a layout. A layout helps a document when it comes to organizing. In terms of coding, a layout will help the programmer set their code easy to understand. When the code is readable and easy to understand by the other programmers, it will make the task run smoothly and be done fast. It must be easy to scan, compact format.

Chapter 14: Code with Reason

It proposes a middle ground for ensuring software correctness: dividing code into short, understandable sections and reasoning informally about their correctness. This involves following coding best practices like avoiding go to statements, minimizing global variables, and keeping functions short and focused. Communicating insights gained from this process helps enhance understanding across the team.

Chapter 15: A Comment on Comments

It reflects on the importance of comments in programming. While coding, it's easy to focus on functionality, but comments are crucial for understanding and collaboration. Striking a balance between clarity and conciseness in comments is key, ensuring they enhance code readability without becoming overwhelming. Evans' experience serves as a reminder of the value of clear communication within software development.

Chapter 16: Comment Only What the Code Cannot Say

I'm struck by how tricky it is to balance clear and cluttered code comments. It reflects the real-life struggles of dealing with comments in code. It warns about wrong comments hits home for me, reminding me of the importance of being accurate in my comments. The idea of "noisy" comments, which don't add much value or just repeat what the code says, makes me rethink how I comment my code. Too many unnecessary comments can make programmers ignore them, which isn't good. So, I'm reminded to comment thoughtfully, making sure each comment adds something unique and helpful to the code.

Chapter 17: Continuous Learning

Learning all the time is super important because things are always changing, especially in development. If you don't keep up, you might fall behind. Even though your job might offer training, it's up to you to keep learning on your own. Luckily, there are tons of ways to learn, like online tutorials and getting help from mentors. I like to stay up to date by trying new things, going to conferences, and joining study groups to learn from others.

Chapter 18: Convenience Is Not an -ility

Convenience in API design can sacrifice code readability and clarity. Making an API overly convenient by cramming multiple tasks into one method can make code confusing to understand. While it might seem easier to write, breaking tasks into clear, separate steps makes code more efficient and easier to work with in the long run. APIs are like languages, so having a diverse vocabulary of methods helps developers communicate clearly with the software. By focusing on clarity over convenience, we ensure that our code is easy to understand and flexible for others to use.

Chapter 19: Deploy Early and Often

It’s important to deploy projects early and keep doing it regularly. Waiting until the end to fix deployment issues can cause big problems. The way customers first use the product when they install it affects what they think about how reliable it is. So, start with a strong installation process and keep improving it while working on the project. Testing the deployment on new systems helps make sure everything works well and stops any wrong assumptions that could cause trouble later. Basically, treat deployment testing as seriously as testing my code to make sure everything goes smoothly for the customers.

Chapter 20: Distinguish Business Exceptions from Technical

I've learned it's crucial to tell the difference between two kinds of problems: technical issues and business problems. Even though both are shown using errors, they're different. Technical problems, like coding mistakes or computer troubles, should be handled generally by the program. But business problems, like running out of money in a bank account, need a special kind of attention. Separating these types of problems makes it easier for others to understand the software, and it keeps everything working smoothly.