**97 Things Every Programmer Should Know**

**– COLLECTIVE WISDOM FROM THE EXPERTS**

**Chapter 1: Act with Prudence**

I was warned against the habit of quick and rushing through tasks with the thinking to fix them later because it often leads to a lot of technical debt, it will make the code harder to work and cause problems. “The longer you leave it, the worse it gets.” For a better management, do not leave errors, you might get another unexpected bug and another, so that explains it. I used to have no idea that there is a coding standard, I understood that it makes the documentation clean and easy to work in a project and debug. In short, it's about being careful not to take too many shortcuts in coding, fixing them promptly, and understanding how they affect a project.

**Chapter 2: Apply Functional Programming Principles**

I realized that understanding functional programming goes a long way or beyond just handling multiple core challenges, prioritizing consistency and clarity , basically a game- changer, developers can create code with fewer defects and better organization. Mastery of Functional Programming Paradigm improves quality of code, especially there are so many languages, and, in the industry, you have to adapt modern coding. Embracing functional programming principles enables teams to build more strength and maintainable software, benefiting projects across various domains.

**Chapter 3: Ask, “What Would the User Do?” (You Are Not the User)**

I don’t have to think like a User, and users don’t think like programmers. We should not and assume others will think like us, this is false consensus bias by psychologists. People think differently and that’s okay. Programmers has struggle to understand how users think because users don't spend as much time with computers and don't know the technical stuff. We should seek what the client and user needs. To understand users better, we should observe and watch them with similar software to what you're making. Pay attention to how they do tasks . You'll see patterns in how they work and where they get stuck. Users tend to focus on one thing when they're stuck, so if you need to give them help, put it where they need it most. Keep things simple for users - one clear way to do something is better than a bunch of confusing options. Also, what users say they want might not match what they do, so it's important to watch them in action to really understand their needs.

**Chapter 4:** **Automate Your Coding Standard**

I was not familiar with the coding standard , I’ve come to understand that a coding standard is guidance. It is to keep our code consistent and prevent common bugs. Automated coding standard and uniformity makes it easier to work in a project. It is important to not be in a private way of coding, or your own. Whether we want to prevent developers, a coding standard should be maintaining in development speed, and smoother from start to end. It is also a must to produce code quality reports and to document. Automation is key and enforced.

**Chapter 5: Beauty is in Simplicity**

– “Beauty of style and harmony and grace and good rhythm depends on simplicity.” – Plato, basically a programmer should have simplicity in coding , simplify and make the work done smoothly and on time. Readable , Maintainable and development speed and quality . In the beauty of coding , is in it simplicity. We can know and find bugs and errors and convenience as well in fixing it.

**Chapter 6: Before you Refactor**

It would be like renovating, but its not an always best step to just to break and remove everything. I've learned to consider a few points, Instead of rewriting everything, reusing tested code prevents the loss of valuable fixes and knowledge. Breaking down the process into smaller, incrementing it, makes it more manageable and reduces risk. Maintain test, validate changes and guard against new bugs. It's important to set aside personal biases. Finally, accepting mistakes happen and learning from them, it helps us in decision making.

**Chapter 7: Beware the Share**

Understanding context , will help us produce and for effective development. Reduce bugs in a existing code. It’s not okay to copy and not understanding the whole code, again avoid making new errors, it may be okay to use and searched source codes for the problem without understanding its context, despite everything the error may still exist in the code. It can affect the program and unforeseen complications.

**Chapter 8: Boy Scout Rule**

Robert drawn inspiration from the Boy Scout Rule were leaving camp clean than how you found it, he suggested to apply it similar ruling to code development: make it cleaner than how you checked it. By consistently making small improvements to the code, aside from its original state, teams can stop software from getting worse over time and encourages everyone to take responsibility for upkeeping it.

**Chapter 9: Check Your Code First Before Looking to Blame Others**

It is a struggle to accept a code may be flawed, preferring to blame system software like compilers or operating systems. However, the reality is that such issues are rarely caused by bugs in these tools. Instead, it's more productive to focus on debugging your own code and questioning assumptions, both yours and others'.

**Chapter 10: Choose Your Tools with Care**

Modern applications now rarely start from scratch, developers assemble them using existing tools, frameworks, and libraries to be efficient and its cost-effective. I realized its hard still to mix of tools it takes extra care and consideration , to avoid of tools mismatching, lifecycle complexities, configuration overheads, vendor lock-in, and unexpected costs. It’s advocating, starting with essential tools and gradually to add more as needed, with simplicity and flexibility. Segregating external tools from core business logic, developers can build leaner, more adaptable applications that minimize errors and maximize maintainability.