Nicole P. Satiembre BS Computer Engineering-Intern February 20, 2024

Cebu Technological University-Main Campus

97 Things Every Programmer Should Know
Chapters 26-30

26. Don't ignore that Error- Pete Godliffe

It is common for us programmers to sometimes ignore errors or warnings that could cause potential errors in our code. Some of us delay fixing errors which is a big risk for bugs that may arise in the future. Not handling errors properly could lead to Brittle Code, Insecure Code, and Poor Structure. We report errors in our code in several ways, including Return codes, errno, and Exceptions. Just as we should check all potential errors in our code, we also need to expose all potentially erroneous conditions in our interfaces. We should not hide them, pretending that our services will always work.

27. Don't Just Learn the Language, Understand Its Culture-Anders Norås

I have learned that it takes more than just learning the syntax to learn a language: you need to understand its culture. We can choose to use any language but to truly learn a language, we have to embrace it. As novice programmers, it is common to make an excuse of using only the concepts comfortable to us which sometimes causes the code to grow large because of redundancy. We should challenge ourselves and understand the usage of other concepts in our language so we can make the best use of it. For example, instead of writing long static variables in C#, we must challenge ourselves to understand the concept of classes and why we use them. I learned not to fear away from using new concepts that seem to be difficult, rather we should force ourselves to use them. By doing so, we also get a better understanding of design patterns by moving between different languages.

28. Don’t Nail Your Program into the Upright Position-Verity Stob

The metaphorical phrase "nailing the corpse in the upright position" is used to humorously convey the attempt to keep the application running despite encountering exceptions. The author admits to the impracticality and lack of forethought in the design, drawing parallels to the absurdity of Yossarian's logic in Catch-22. The narrative is used to criticize an argument the author had with someone over Java code and exception handling. The other person advocated for catching and blocking exceptions on the spot, citing a UI design rule to never show exception reports to users. The author ridicules this perspective, questioning the practicality of catching exceptions without a clear plan for handling them. Overall, the passage serves as a cautionary and humorous tale about the pitfalls of overcomplicating exception handling and the importance of sensible and minimalistic error-reporting mechanisms in software development.

29. Don’t Rely on “Magic Happens Here”- Alan Griffiths

We often have this tendency to underestimate the job of others thinking that their job is easy because we don’t know the ins and outs of what they do. When we aren’t actively involved in things, there is an unconscious tendency to assume that they are simple and happen “by magic.” Underestimating the difficulty of a task can be dangerous especially when you find out only near the project deadline that magic does not exist and that it is really hard to fix the issues plus the time is very limited. That is why it is very important to be involved in each part of our project. In every project we do, we must understand how every bit works from the ground up. You don’t have to understand all the magic that makes your project work, but it doesn’t hurt to understand some of it—or to appreciate someone who understands the bits you don’t.

30. Don’t Repeat Yourself - Steve Smith

One of the most fundamental principles in programming is DRY- Don’t Repeat Yourself. As developers, we must learn to recognize duplication and understand how to eliminate it through appropriate practice and proper abstraction. This way we can produce a much cleaner code less prone to bugs. Duplication needlessly bloats the codebase, resulting in more opportunities for bugs and adding accidental complexity to the system. DRY requires that “every piece of knowledge must have a single, unambiguous, authoritative representation within a system.” Many processes in software development are repetitive and easily automated. Manual testing is slow, error-prone, and difficult to repeat, so automated test suites should be used where possible. The formulation of design patterns themselves is an attempt to reduce the duplication of effort required to solve common problems and discuss such solutions. When followed about structure, logic, process, and function, the DRY principle provides fundamental guidance to software developers and aids the creation of simpler, more maintainable, higher-quality applications.