Nicole P. Satiembre **March 2, 2024**

Cebu Technological University-Main Campus- Intern

97 Things Every Programmer Should Know

Chapters 86-90

86. Two Wrongs Can Make a Right (and Are Difficult to Fix) - Allan Kelly

Code never lies, but it can contradict itself. This chapter uses the example of the Apollo 11 Lunar Module Software the software controlling the engines contained a bug that should have made the lander unstable. However, another bug compensated for the first, and the software was used for both Apollo 11 and 12 Moon landings before either bug was found or Fixed. The interplay between two code defects that appear as one visible fault makes it hard to fix the problem and leads developers down blind alleys, only to find they tried the right answers early on. The issue extends beyond code to written requirements and user understanding, with single errors being easier to fix than problems with multiple causes. The passage suggests that addressing such faults requires awareness, a clear mindset, and a willingness to consider various possibilities. Overall, it underscores the intricacies of dealing with intertwined defects in software development.

87. Ubuntu Coding for Your Friends-Aslam Khan

The passage emphasizes the social aspect of software development, highlighting that code is often written in isolation, reflecting personal interpretations and solutions to problems. It stresses the importance of recognizing that code created individually will be used and relied upon by others, emphasizing the shared responsibility within a team. The text explores the idea of "Ubuntu" philosophy, stating that a person is improved through interactions with others. Applied to coding, the quality of one developer's code can impact another developer's work. The author suggests that coding with an "Ubuntu" mindset, considering the collective improvement of the team, results in better code quality. The passage contrasts the individual focus of Zen with the group-oriented nature of Ubuntu, concluding that code is rarely created solely for personal use.

88. The Unix Tools are Your Friends - Diomidis Spinellis

The passage encourages us to become proficient with Unix tools over using an Integrated Development Environment (IDE), especially in a diverse and evolving development environment. It highlights that IDEs are language-specific, while Unix tools can handle any textual form, making them versatile in a world with constantly emerging languages. Unix tools allow users to create custom commands by combining simple tools, providing flexibility and adaptability. The example showcases a text-based implementation of file analysis, revealing the power of Unix tools in handling diverse tasks.

The passage underscores that learning IDE operations is task-specific, whereas mastering Unix tools makes one effective across various tasks. It emphasizes the efficiency of Unix tools in handling large datasets, thanks to their design originating from a time of limited computing resources. Unix tools, working as filters, process data line by line, offering scalability without upper limits. The ubiquity and open-source nature of Unix tools make them widely available, even on resource-constrained platforms. The passage also mentions the ease of extending the world of Unix tools by writing programs that adhere to specific rules. Overall, it promotes the versatility, efficiency, and adaptability of Unix tools in a diverse and resource-conscious development landscape.

89. Use the Right Algorithm and Data Structure -Jan Christiaan “JC” van Winkel

In this chapter, we learn the importance of efficient coding, especially in large systems. The bank faced slow computer issues due to a poorly optimized program that needlessly checked the length of a constant string thousands of times. The programmer could have saved resources by determining the length beforehand. It highlights the significance of choosing the right algorithms and data structures, as the wrong choices can drastically impact system performance. The example emphasizes that programmers should not only avoid reinventing the wheel but also be educated about algorithms, data structures, and when to use them. Reusing existing libraries is essential, but understanding the problem domain ensures appropriate choices. The story concludes by suggesting that a good programmer should know when to use unconventional solutions based on specific constraints. Overall, the message encourages programmers to prioritize efficiency, make informed choices about algorithms and data structures, and continuously educate themselves in the field.

90. Verbose Logging Will Disturb Your Sleep - Johannes Brodwall

When working with established systems, a messy log, flooded with excessive messages from routine actions, often signals trouble. After completing your part, someone else takes over, ensuring the system serves customers. Monitoring becomes crucial, and logs play a key role. If a log entry could wake you at night, it should denote a serious issue. A clean error log during testing is a positive sign, indicating system robustness. Distributed systems add complexity, requiring a strategy for external dependency failures. In essence, a well-behaved system has content INFO-level logs for major events, avoiding clutter that complicates live system management. Keeping the error log uncluttered makes handling unexpected issues more manageable.