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

Cebu Technological University-Main Campus- Intern

97 Things Every Programmer Should Know

Chapters 81-85

81. Test Precisely and Concretely-Kevlin Henney

It is important to test for the desired, essential behavior of a unit of code, rather than for the incidental behavior of its particular implementation. The passage emphasizes that code should be tested based on what it is supposed to achieve and not how it was implemented. It does so by referring to a few things that are related to sorting algorithms to show some of the misconceptions regarding testing. This requirement for an inclusive postcondition must take into account more elements such as sort order and permutation. The text points out that tests have to be both accurate and precise, for they must be clear enough to remove all ambiguity from the code under the test itself. There should be good cases that bring out any questionable factors or unnecessary complexities in a simple manner without obscuring general behavior's nature thereby highlighting the need for legible but exact tests when evaluating code behavior effectively.

82. Test While You Sleep (and over Weekends)-Rajith Attapattu

This chapter encourages us to draw our attention to how much computing power we have at our disposal. Suggestions for making better use of idle computing power include running automated testing during off-hours. Larger test suites can be broken down into smaller profiles that are "fast to run on every code commit," advises Sirosh. New automation, "ideally written to run overnight," provides necessary tests of "stability, performance and platform permutations," that are not sensible for manual testing. "The frequent AGILE cycle means that you will actually run a lot of these tests so that you can fix it before your next morning coffee," he quips. "You maximize compute during down time to run hundreds of hours of tests and catch any issues as quickly as possible."

83. Testing Is the Engineering Rigor of Software Development - Neal Ford

This chapter states that, using physical engineering metaphors for writing software is a no-go, because while a bridge or a truck engine are subject to physics, and therefore conserved principles, software development has no such principles established. The passage advocates for testing, as it is the most cost effective way to ensure correctness at almost any stage during development: even such tools as unit testing can assist in this process, if put into practice from the start. The text urges developers to think of testing as the primary method of verification of their work – while this takes time, it is absolutely necessary in software development, the way that structural analysis is necessary for the unspoken, taken for granted but essential, correctness of a truck engine. The passage seems to imply that the responsibility of testing solely lies with the developers for the sake of the quality of the software.

84. Thinking in States - Niclas Nilsson

The passage highlights how people often have a casual attitude towards "state" in everyday situations, like being "super-duper, mega–out of milk." However, the author argues that programmers can face issues when handling states in their code. Using an example of a webshop, the author points out the redundancy in state checks and the importance of proper state handling in complex scenarios. The passage suggests thinking in terms of states, understanding state machines, and using tools like Design by Contract to ensure code simplicity and robustness. It emphasizes that incorrect states can lead to bugs and data issues, and advocates for techniques like code generation or aspects to manage state checks efficiently.

85. Two Heads Are Often Better Than One - Adrian Wible

Programmers are often stereotyped as “Loners” because of the nature of their job that requires deep thoughts in solitude. However, the opposite is the truth. Being the expert technologist is no longer sufficient. You must become effective at working with others. The author emphasizes that collaboration goes beyond simple question-and-answer interactions or attending meetings. Instead, the author sees collaboration as an active and hands-on process, likening it to rolling up one's sleeves and jointly tackling work with a partner. The author particularly expresses enthusiasm for pair programming, referring to it as "extreme collaboration." In pair programming, the author believes that personal skills and knowledge grow significantly. If one is weaker in a certain domain or technology, they learn from the partner's experience. On the other hand, if one is stronger in a particular aspect, they gain insights by explaining their thought process. The passage emphasizes the mutual learning and skill development that occurs through collaborative efforts where both individuals contribute their strengths and learn from each other's expertise.