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

Cebu Technological University-Main Campus

97 Things Every Programmer Should Know
Chapters 31-35

31. Don't Touch That Code-Cal Evans

In every project, the tasks are to be organized. In most web-based environments, the architecture can be broken down into Local Development, Development Server, Staging Server, and Production Server. Using this model, a developer—even a senior developer—should never have access beyond the development server. The development stage is done on a developer's local machine, once checked, it will be rolled over to the development server for testing. From this point, the developer is just a spectator. The staging manager is assigned to roll the code to the staging server for the QA team. The QA team and users do not need to touch anything from the development server. The release manager is the only person who should have access to both. A developer should never have access to the production server. Some of the biggest programming disasters have taken place because of violating this rule. I have learned that if a program has defects, production is not the place to fix it, and as a developer, I shouldn't touch anything in production.

32. Encapsulate Behavior, Not Just State - Einar Landre

Containment is a very important concept for a software developer. Modules and packages address the larger-scale needs for encapsulation, while classes, and subroutines. When I was still learning OOP in college, it was hard for me to understand the concept of encapsulation but I realized the importance of it. Containment is extremely useful when dealing with large and complex system structures. I would admit that it is one of the trickiest things to master and a lot of novice developers would rather write long lines of code because they failed to understand the beauty of handling objects and how they make lives easier. An object encapsulates both state and behavior where the behavior is defined by the actual state. Allocation and delegation of responsibility to the different objects including the inter-object interaction protocols are made simpler with objects. If we utilize if-then-else statements, these tasks are tedious, complicated, easy to break, and harder to maintain, mainly because encapsulation is broken. As final thoughts, we should learn how to utilize the power of encapsulation using our programming language of choice.

33. Floating-Point Numbers Aren't Real Chuck Allison

When I was a freshman taking up Computer Engineering, our teacher always instilled in us the importance of choosing the right data type, and the range of bits it can handle and that consideration includes the IEEE floating-point numbers. Real numbers have infinite precision and are therefore continuous and non-lossy; floating point numbers have limited precision, so they are finite, and they resemble "badly behaved" integers. Floating-point operations round to the nearest floating-point number. Since floating-point numbers are approximations of real numbers, there is inevitably a little error present. This error, called roundoff, can lead to unpredictable outputs. The problem here is that the roundoff in the large, positive terms is in a digit position of much greater significance than the true answer. As a lesson, shouldn't use floating-point numbers for financial applications—that's what decimal classes in languages like Python and C# are for. Floating-point numbers are intended for efficient scientific computation. But efficiency is worthless without accuracy. Lastly, this lesson does not only apply to floating point numbers, we shouldn't take lightly the difference between data types as that basic concept could bring out the best or the worst if not considered appropriately.

34. Fulfill Your Ambitions with Open Source - Richard Monson-Haefel

Open-source projects are widely available and are very helpful if we want to enhance our skills by contributing to these projects. I haven't tried it but I am interested to know that we can contribute to projects like these. Of course, there is no pay, but for me, it is one of the best ways to give back to the community because I have also benefited from open sources in a lot of situations, and even free tutorials online. I wouldn't be here knowing how to code without the free benefits of open-source software. Aside from that, it is also a good opportunity to enhance our skills. We can learn a lot from reading other people's source code. We also enhance our cognitive thinking by contributing our amazing ideas. And the fun part is we meet a lot of great people with the same passion and these open-source friendships can last a lifetime. Another fast way to learn is by writing test code for other people's software. This is a pretty golden advice that I would highly consider doing.

35. The Golden Rule of API Design-Michael Feathers

When building an API, novice programmers would be tempted to use the keyword “final” for most of the classes and methods in Java, although not inherently bad constructs, there are better approaches for this type of task. Over the past decade, unit testing has become an extremely important practice. If we try to write a unit test for an arbitrary untested class that uses third-party API we’ll find out that API is stuck to it like glue and there’s no way to impersonate the API classes so that you can sense your code’s interactions with them, or supply return values for testing. That’s where the Golden Rule of API Design fits in: It’s not enough to write tests for an API you develop; you have to write unit tests for code that uses your API. When you follow this rule, you learn firsthand the hurdles that your users will have to overcome when they try to test their code independently. Finally, there is no one way to make it easy for developers to test code that uses API. What’s most important is to be aware of the testing issue and, to do that, we must have first hand experience to it.