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

Cebu Technological University-Main Campus

97 Things Every Programmer Should Know
Chapters 11-15

21. Distinguish Business Exceptions from Technical-Dan Bergh Johnsson

I have learned in this chapter to distinguish technical exceptions from business exceptions which are the two basic reasons for things to go wrong at runtime. Mixing these could confuse what the method contract is, what conditions it is required to ensure before calling, and what situations it is supposed to handle. Technical exceptions are thrown when something goes wrong that we cannot fix and usually cannot respond to in any sensible way. A business exception is thrown when a business rule within our application is violated. It's up to the client of the above code to ensure the business rule is satisfied. An example of this is making sure that the input variable water does not go beyond a certain maximum level or else it throws an exception.

It is therefore important to separate the cases to provide clarity and increase the chances that technical exceptions will be handled by some application framework. In contrast, the business domain exceptions are handled by the client code.

22. Do Lots of Deliberate Practice-Jon Jagger

I have learned in this chapter that we do not just practice to complete a task, rather we must deliberately practice–meaning to repeatedly do something. This way we can increase our mastery of one or more aspects of the task. By repeated practice, we slowly build our skills and memory until we achieve our desired level of mastery. The ultimate goal of deliberate practice is to improve our overall performance.

Coding, just like exercise or learning an instrument is a delicate craft. No one can build a skill by performing a task once or twice. It needs repetitive practice for one to hone expertise in such areas. The expertise arrives gradually over time—not all at once in the 10,000th hour! Nevertheless, 10,000 hours is a lot: about 20 hours per week for 10 years. Becoming an expert demands commitment and discipline. Deliberative practice doesn't also mean repeating the same things over and over again. Just like working out, we need to challenge ourselves and gradually increase our challenges. Learning not only improves our skills and performance but most importantly it positively changes our attitude and behavior.

23. Domain-Specific Languages-Michael Hunger

Jargons– a lesson from my English class, these are words that are used by people who work in the same field. For example, a class may mean differently between programmers and teaching professionals. I think this is the simplest way to explain what Domain-Specific Languages are. A specific domain has a specialized vocabulary to describe the things that are particular to that domain. DSLs are about executable expressions in a language specific to a domain, employing a limited vocabulary and grammar that is readable, understandable, and—hopefully—writable by domain experts.

Internal DLS is written in a general-purpose programming language whose syntax has been bent to look much more like natural language. Most internal DSLs wrap existing APIs, libraries, or business code and provide a wrapper for less mind-bending access to the functionality. They are directly executable by just running them.

External DLS are textual or graphical expressions of the language—although textual DSLs tend to be more common than graphical ones. Textual expressions can be processed by a toolchain that includes a lexer, parser, model transformer, generators, and any other type of post-process.

The lesson I have learned is to consider the target audience of our DSL. Are they developers, managers, business customers, or end users? We have to adapt the technical level of the language, the available tools, syntax help (e.g., IntelliSense), early validation, visualization, and representation to the intended audience

24. Don’t Be Afraid to Break Things-Mike Lewis

In the real world, as programmers, we will have to work on already-built systems that are precarious in the state. I remember during my application process to different companies, others showed me their code bases. I was in awe, and in my head, I was afraid to touch it as I might break the code and cause a whole havoc for the project. I learned in this chapter that we are like doctors of the programs. These programs need to be fixed and we could make it worse without being careful. Nonetheless, doctors would still cut a patient for surgery knowing that the temporary pain is worth it for the whole healing process. As programmers, we face the same challenge. We cannot stay in the corner, afraid, forever. We must break things to make things and it is part of our job. I learned that to be confident in fixing things, I must consider the best practices so I know that I am contributing to the betterment of the program as a whole. I should not be afraid to break things but rather face them to fix the problems.

25. Don’t Be Cute with Your Test Data-Rod Begbie

As I was reading this chapter, I couldn’t help but laugh because it is relatable and the author’s sentiments are embarrassing and hilarious. I think most of us fall into the temptation of writing temporary variables or value names that we thought of as just temporary and nobody will see it anyway so sometimes we use funny or cute words as aliases. However, this thing could lead to a very embarrassing experience especially if we forget to review it and we need to present our output. This and other programmer’s familiar stories can very much relate to the author’s experience. I have learned that even our source code isn’t necessarily free of scrutiny. In 2004, when a tarball of the Windows 2000 source code made its way onto file-sharing networks, some folks merrily grepped through it for profanity, insults, and other funny content. Lesson learned, when writing any text in our code—whether comments, logging, dialogs, or test data—always ask yourself how it will look if it becomes public. I am taking more cautious steps now because I have gone through the same funny experience myself.