Nicole P. Satiembre **February 27, 2024**

Cebu Technological University-Main Campus- Intern

97 Things Every Programmer Should Know
Chapters 46-50

46. Know Your Limits-Greg Colvin

In software engineering, knowing yourself, and your team, and managing your time and money wisely is important. This way, we will understand the best of our capacity can make this is because our resources are limited. We must understand the ins and outs of data structures, algorithms, and system architecture to create efficient software that fits real-world machines. Choosing the right algorithm will help lessen the complexity of a program thus if you are time-constrained, you must consider how to finish things fast. One way to do this is to conduct Complexity Analysis. The effectiveness of algorithms and data structures in utilizing caches becomes a critical consideration. Acknowledging limited resources, such as personal effort, time, and the capabilities of tools and target machines, is necessary. On the other hand, we must also be aware of budget limits, both in terms of resources and the capabilities of our tools. As a golden rule, we must keep things simple, and always be ready to learn and adapt to stay on top of our game in this ever-changing field. Striking a balance between continuous learning, adapting to technological advancements, and respecting these resource constraints is key to sustainable success in the dynamic field of software engineering.

47. Know Your Next Commit-Dan Bergh Johnsson

One important thing I have learned in this chapter is to have a clearly defined goal when coding. We shouldn't be too broad about our tasks because we might get lost and we will not know the scope and time of what we are doing. This could lead to a huge wasted time. Instead, the best approach would be to work on a few tasks at once, with feasible goals and timelines, and if it fails to work, we can set it back to the original code and work on the next feature instead. This way we will feel more productive because at least we have accomplished tasks little by little. On the other hand, if we tried to make one massive program all at once, we wouldn't feel as much effective because we get lost on which parts to finish first. The vision is very broad and there is little assurance that we can accomplish it all at once. Know your next commit. If you cannot finish, throw away your changes, and then define a new task you believe in with the insights you have gained. Do speculative experimentation whenever needed, but do not let yourself slip into speculative mode without noticing. Do not commit guess work into your repository.

48. Large, Interconnected Data Belongs to a Database - Diomidis Spinellis

If your program deals with lots of connected data that needs to stick around, you can consider using a relational database or RDBMS. In the past, these databases used to be pricey, hard to find, and kind of tricky to handle. Now, things are different. You can easily get your hands on relational database systems, and some good ones like MySQL and PostgreSQL are free. There are also embedded databases like SQLite and HSQLDB that you can just link to your app, and they work super efficiently without needing much setup or fuss. Another benefit of building your code with an RDBMS is how it manages relationships between your data. You can specify consistency rules for your data in a clear and declarative manner, steering clear of the chance of having leftover connections (dangling pointers) if you happen to forget to update your data in certain situations. You have the flexibility to establish effective connections between the entities stored in the database whenever necessary, just by setting up an index. No need for costly and extensive restructuring of class fields. Moreover, coding with a database enables multiple applications to safely access your data. So, if you have a massive source of data to handle, relational databases are a smart and accessible choice.

49. Learn Foreign Languages - Klaus Marquardt

As programmers, we are responsible for bringing ideas to life. That is why we require a lot of communication. We communicate with the clients, stakeholders, and end users who are not familiar with the tech jargon. It is our responsibility to know the language of the domain so we can communicate with them effectively. The way to do this is to learn their language of domain. We must learn another language, be it machine language or human-specific language. Beyond communication with machines, programmers need to communicate with their peers. Today's large projects are more social endeavors than simply the applied art of programming. It is important to understand and express more than machine-readable abstractions can. We must try to master another language because if we don't, it will be hard for us to express our thoughts and thus our communication will not be as effective. Ideas are wasted because of communication skill issues. Speaking a language well also leads to a clarity of thought that is indispensable when abstracting a problem. And, of course, life is more than software projects. As noted by Charlemagne, to know another language is to have another soul. For your contacts beyond the software industry, you will appreciate knowing foreign languages. To know when to listen rather than talk. To know that most language is without words

50. Learn to Estimate - Giovanni Asproni

As a programmer, it's important to provide estimates to managers, colleagues, and users. This way, they can have a reasonably accurate idea of the time, costs, technology, and other resources needed to achieve their goals. We must learn what estimates, targets, and commitments are. An estimate is an approximate calculation or judgment of the value, number, quantity, or extent of something. This definition implies that an estimate is a factual measure based on hard data and previous experience—hopes and wishes must be ignored when calculating it. The definition also implies that being approximate, an estimate cannot be precise. A target is a statement of a desirable business objective. A commitment is a promise to deliver specified functionality at a certain level of quality by a certain date or event. One example could be "The search functionality will be available in the next release of the product." Estimates, targets, and commitments are independent of one another, but targets and commitments should be based on sound estimates. Thus, the purpose of estimation is to make proper project management and planning possible, allowing the project stakeholders to make commitments based on realistic targets.