Book Report of Pragmatic Thinking and Learning

Xiaoxiao Wang (xwang104)

November 30, 2016

Abstract

This is a book report of Pragmatic Thinking and Learning: Refactor Your Wetware, by Andy Hunt. In this report, I will introduce the background and motivation of this book. Then briefly summarize the content in each chapter. After that, I will take some useful advices from the book and relate them to my own project experience.

1 Introduction

Programming, or software development, is a creative activity that combines rich, flexible human thought with rigid constraints of a digital computer. It is complex and difficult, and the cost of failure can be huge sometimes.

According to a study from Capers Jones via Bob Binder mentioned in the book, despite of the great advance in programming languages, techniques, project methodologies etc these years, the density of bugs in programs has remained almost constant. The fundamental reason, according to the author, is that instead of focusing on new technologies, we should improve the skills of ourselves as program developers. And that's where the phrase "refactor your wetware" comes from. The wetware here is an analogy made by the author that models human thought processes as a computer. "Refactoring your wetware", compared to refactering a program, means redesigning and rewiring your brain to make you more effective at your job.

I personally cannot fully accept this reasoning. The problems that we modern programmers facing today are much more complicated than people did back in the dark ages forty years ago when the world saw its first personal computer. In addition, the scale and complexity of modern softwares as well as development teams has been increasing exponentially. If the density of bugs remains constant, shouldn't we be proud of ourselves? What's more,

in old times, only very few well educated experts did programming, but nowadays thanks to the popularity of programming tools and online tutorials, even a motivated high school student can write pretty fancy apps.

However, I agree that software development highly depends on the quality of the developers even they are equipped with same techniques and methodologies. So let us take a look at the author's suggestions.

The author points out that there are two important modern skills to success:

1. Communication skills. As software can't be built or perform in isolation, the communications between team members and between customers and development teams are important. That's why the agile methods get so emphasized.

2. Learning and thinking skills. Nowadays programmers have to learn constantly, not only new technologies, but also the bigger context. They also have to think creatively and critically.

The purpose of this book is to guide us through accelerated and enhanced learning and more programmatic thinking. Besides, it also includes many useful tips in improving communication skills and social interaction in teams.

Two basic principles are emphasized at the beginning and mentioned now and then throughout this book. First, because every person is unique and our brain is such a complicated and unknown system, we should not simply follow all the advices blindly. Instead, we should do what works for us, and this is actually the essence of "pragmatism". Second, everything in the outside world and our inner thoughts are interconnected, which implies that small things can results in unexpected large effects. In that sense, we should not only pay enough attention to details, but also consider a larger context when solving problems. This ability actually differs an expert from a novice, which is talked about later in the Dreyful model of skill acquisition. Lastly the author suggests that the ideas in his book are not only useful for programmers but also for people in many other diverse areas.

The ideas and advices presented in this book are like a complex graph where many things are interrelated and connected, but the book is somewhat loosely and linearly organized. In this report, I will not follow the flow of the book. Instead, I will extract the important ideas and summarize them in different sections. The rest of the book report is structured as follows: In section 2 we give a brief summary to each chapter of the book. Section 3 gathers author's advices for programming since this is most relevant to our course. Section 4 discusses useful advices in learning and thinking. Section

5 is about getting along with other people in teams. I will also relate to my personal experiences of doing projects in each section.

2 Summary of the Book

This book introduces some theories in cognitive science, neuroscience, leaning and behavior. Based on these theories and models, the author propose many suggestions and advices for learning and thinking and for programming. Chapter 1 is an introduction and acknowledgement. In chapter 2, a Dreyful model of skill acquisition is introduced. Chapter 3 takes a quick look at how our brain works and models of our brain as a dual-CPU computer. Chapter 4 gives advices on how to extract more power from our brain "computer". Chapter 5 discusses common bugs in human thinking and suggest how we should try our best to avoid them. Chapter 6 introduces many learning techniques. Chapter 7 proposes good learning strategies. In Chapter 8, the author suggest ways to managing our focus, which is very important for programmers. And the last Chapter is a short summary.

In this section, we will mainly focus on the background theories in Chapter 2, 3, 5 and 8. Since the advices and techniques appears in many places in each chapter and many of them are mentioned more than once in different level of details, I will summarize the ones that I feel useful based on my personal experience in later sections.

According to the Dreyfus Model in Chapter 2, people can be divided into five stages in skill acquisition. They are Novices, Advanced Beginners, Competent, Proficient, and Experts. The abilities, attitudes, capabilities, and perspectives in different stages are discussed in details. In particular, the author emphasizes that novices needs recipes, which are basically context-free rules that cannot fully specify everything. But experts have the ability to recognize patterns in context. And they work from their intuitions which are often hard to explain to others. According to the author, a large majority of programmers are advanced beginners who can try tasks on their own but don't have the big picture or the ability to troubleshoot. We need to improve our skills at least to the level of Competent to be able to not only follow orders but also doing troubleshooting.

This model does not only guide us to be more ware of our need for learning and personal growth when we are at different skill levels. It also tells us how to listen and react to other people's need in the team.

Chapter 3 takes a quick look at how our brain works and proposes many practices that can better utilize its power.

The author uses a computer that has dual-CPU sharing the same bus

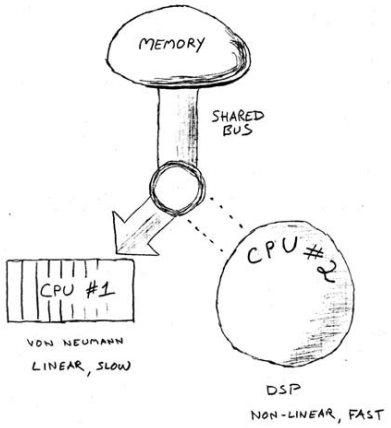

Figure 1: This is your brain.

to the memory core as a metaphor for our brain. The two CPUs are very different in their processing styles. CPU 1 is responsible for linear, logical thought, and language processing. Its relatively slow and can only access small amount of memory. This processing style is called linear mode, or L-mode. L-mode gives us the power to work through the details and make things happen. CPU 2, on the other hand, is more like a magic digital signal processor that is super powerful in searching and pattern matching. But it's not verbal, not under direct conscious control, and usually non-rational and unpredictable. This processing style is called rich mode, or R-mode. R-mode is critical for intuition, problem solving, and creativity, which is the hallmark of experts. Its analogic and holistic thinking styles are very valuable to software architecture and design.

Besides, as both CPUs share the bus to the memory core; only one CPU can access the memory banks at a time. That's why great intuitions or ideas often come to you while you are not sitting in front of your computer. However, since each mode contributes to our mental engine, and for best performance, we need these two modes to work together.

Chapter 4 looks at more techniques to exploit the power of thinking and encourage better creativity and problem solving from our brain. We will discuss this in more details in later sections.

Chapter 5 discusses how to debug our brain. Our intuition can be wrong because of the weakness (or bugs) in human thinking. We have built-in biases in our cognition. One's emotional state, background culture, personality, people around you, and even hardware bugs, can affect the decision making process. Knowing these common bugs can help us mitigate them. Being aware of weakness of yourself helps you test your intuition more systemat-

ically. What's more, it is also important to keeping in mind that different people may have a different set of bugs when collaborating with other people.

Chapter 6, 7, 8 suggest many useful techniques for leaning and practicing. Chapter 6 takes a more deliberate look at how to take advantage of our brain and talks about many specific techniques to help accelerate our learning such as planing techniques, reading techniques and so on. Chapter 7 is about strategies for effective learning. For example, we need to create an efficient learning environment. We need to manage pressures and give our brain positive suggestions in order to succeed. Chapter 8 gives us tips about how to manage our attention and focus on the critical work, how to recover quicker from unexpected interruptions.

Chapter 9 is a little summary and encourages readers to embrace changes and apply the ideas mentioned above in real life.

3 Useful Tips for Programming

3.1 Write Beautiful Code

Our R-mode works better with well-designed and attractive interface. So the layout of our code and comments, the choices of variable names, or the arrangement of the files in the project has great effect on our productivity.

Besides, to facilitate the function of R mode, the author suggest us to create larger patterns in our code for it to be more human-readable. Try to aid visual perception by using consistent typographic cues. Even though the compiler may not care about these, we human beings do, and this is also an important communication tool with team members.

I cannot agree more with this point. And I think that's part of the reason that I like python so much since it use indentation for code blocks instead of brackets, and the latter is so hard to read unless properly formatted and highlighted.

When I wrote my first web application for the database course, since I haven't had much experience in developing big projects before, I didn't have a fixed style in writing code. I chose variable names largely depends on my mood and didn't wrote enough comments as I thought these parts are too trivial to explain. What was worse, my teammate has a different style (or none at all) in indentation and many other places. So later on when our project became bigger and bigger, none of us were willing to read each other's code, and I didn't want to read my own code either. We ended up not being able to finish implementing all functionalities we planned to have at the beginning. And, one year later, none of us want to touch this project

anymore.

With this experience, when I actually took Professor Ranjitha's course in Web Programming, I spent more effort in making my code readable, consistent, and beautiful, and tried to be very verbose at the beginning of programming. What's more, I learned more techniques in responsive design and user interface design. Our webpage looked much prettier. And this made me feel excited every time I opened my project and added more things to it.

3.2 Pair Programming

An interesting way to get L-mode to work with R-mode is to use another person for the other mode, and pair programming is an effective way embracing this idea. In pair programming, one programmer types code while the other offers suggestions and advices in general. Since the person who is typing code is locked in the verbal mode at a particular level of detail, the other one can engage more nonverbal centers.

I experienced this when I was doing this class project in implementing controlling white spaces between tokens. To my understand, the character 'd', which is the 'default spacing', can have different meanings: it should be empty before and after all tokens, but a single white space in between tokens. And since we have to process every token recursively, some of them may have a specified format while others may not, the conditions for when to add white space or not become so complicated. I spent a lot of time trying to figure this out but was still so confused. My pair programming partner Matthew, who didn't actually write any code, pointed out that I should simply put the string representation of each child token in an array. If the format of the parent token is specified, we should follow the format, otherwise just use a single white space to join the array. And, it turns out that this strategy works!

At that time I as so devoted to figuring out the relations between parent token and child token, what kind of additional parameters should I pass to the recursive function to tell children the parent's format, and so on. I finally get lost in so many details and convoluted conditions and cannot catch the big picture, which, was actually easy and clear. Thanks to Matthew, we finally figured this out.

After reading this book, I become more aware of my switching between L-mode and R-mode when doing works. For example, when I actually typing this book report, my brain has to concentrate more on spelling and my poor syntax. But I have to keep going back and read it as a whole to get an idea of the big picture and make sure the flow is reasonable and every section gets its appropriate length and depth. These things are similar in programming.

With a partner, we don't have to stop so often and step away from the keyboard trying to see the larger relationships.

What's more, having a partner allow us to explain our confusions to other people, almost every one of us have such an experience that when you try to describe a difficult bug to others, you suddenly catch a flash of insight in your head, and find the bug by yourself. Pair programming makes this process natural and common.

3.3 Draw Maps

The author suggests us to draw mind maps that shows topics and how they are connected when reading books. He claims that this graphic enhancement does not only helps us visualize the insight and see hidden relationships, but also bring up our R-mode.

Despite that this technique is proposed for effective learning, I learned a similar technique from a classmate recently. When dive in a code base of a large project such as k, it is a good practice to draw a picture that visualize the relationships among different classes and functions. This doesn't only help me understand the flow of logic better, but also let me grab the important parts that is relevant the the task that I want to approach.

3.4 Manage Focus

Unlike computers which are built to swap context easily and naturally, our brain are wired in such a way that doesn't support context switching very well. If something interrupt us, breaks our flow, or causes us to lose focus, it take really great effort to drag everything back in. And this process can take as long as twenty minutes on average.

I agree with this point. I found that my most productive time is after midnight when I have finished reading interesting news happened during the day and have responded to important emails and finally can fully concentrate on my projects.

The author shared many useful tips to manage our focus and improve our concentrations. For example, to avoid distractions, we can configure single-tasks interfaces that hide everything other than the application that has focus. Process each pile of work in order to avoid context switching. Keep to-do lists in a calendaring or dedicated to-do list tool instead of in brain. As dynamic refresh of mental lists is very expensive, and you'll be distracted by all other things you are supposed to do.

Advices are also given on how to recover sooner from unavoidable interruptions. For example, only check important emails at fixed time. Go for

context-friendly breaks when you get tired, such as doodling on a paper or go for a walk alone.

One most useful advice that I read from the book is to save your stack in order to be prepared to be interrupted. When you know you're being interrupted, take several seconds and write important cues that you can pick up once you get back to resuming the task. Or simply leaving little reminders of where you are constantly. I will try to do this in the future.

The author also suggest multiple monitors to keep a big enough context in order to avoid context switching. Despite that many programmers would agree with this, I think larger monitors are only great for reading code. I personally prefer smaller interface when writing since this helps me feel more sharp and concentrated.

3.5 Consider the Context

It is always crucial for programmers to take the context into account in viewing the world. When debugging a program, all the variables, object interrelationships, and so on needs to be kept in mind. When designing a software, the domain of its application, the need of its user community are important factors to be considered. When working in a team, the dynamics of the project team and the skill levels as well as characteristics of teammates are important factors. When investing time learning new technologies, we need to consider the shifting trend of of the industry. One technology might be popular and easy to learn, but with fairly low return. New emerging technologies, however, despite of its possible high reward, may go nowhere in short time.

4 More General advices on thinking and learning

There are many more general advices on effective thinking and learning in this book. Since they are not directly related to our Software Engineering class, I will just mention some of them briefly.

One useful tip is to always write down your ideas. Since our R-mode intuitions are so unpredictable, great idea may come to you at any time. If you keep track of them, it is possible that your brain will produce more. In addition, buy putting similar ideas together, it's more likely for you to observe patterns from them and get more ideas.

Another tip that I learned is that positive emotions are essential to learning and creative thinking. Being "happy" broadens your thought processes

and brings more of the brain's hardware online. If you are under big pressure from deadline, your brain will shut down its R-mode completely, and the performance of you L-mode is lowered. To ease up on the pressure, we should keep an open mind to failures.

Other useful techniques including giving your brain positive suggestions, looking at a problem from the other way around, making good objectives and plans for learning, learn by teaching, learn from playing and so on. Each of these advices are covered in details and many of them are very pratical.

5 Advices for getting along with other people

I think the most important idea that the author tries to convey to us on this topic is to understand and accommodate the need of different people in the team.

As we mentioned before, different people in the team may at different stage of skill acquisition. Novices need specific rules while experts need more freedom for their intuitions. We human beings all have our own weakness in our personality. To understand them and try to avoid them helps us become a better person in a team.

I used to involve in a project that made me feel frustrated for quite a long time. Since I didn't make good progress in that project, I became reluctant in meeting with my advisor and talking about the details of my work. Being afraid of talking about failures is the weakness of my personality. Another reason was that I thought the advices I got from my advisor seems to come from nowhere and I wasn't fully convinced. After reading this book, it looks like my advisor was working in the expert way, that is, working from intuitions. But since I didn't provide him with enough details of my progress, he didn't have enough context to make right decisions. Later on, it turned out that we were working in a completely wrong direction on that project and finally we gave up.

It is easier to analyze reasons and predict directions after things happened and have been finalized. But when you yourself are in the maze, it is always hard to grab the big picture and choose the correct way out. Probably that's why we all like reading novels and watching TV shows about other people's life. As we are provided with enough background information and contexts of different people, we feel like god that is so much cleverer than those people struggling in the stories.

6 Summary

In this book report, we summarize the content of this book. The author provides many advices on how to make our brain more efficient and productive. I took some of these advices and relate them to my past project experience. And hopefully after reading this book, I can think smarter, work better, and learn more in the future.

References

[1] Andy Hunt . Pragmatic Thinking and Learning: Refactor Your Wetware