THE STANDARD APPROACH TO BECOME A COMPETENT AND MATURE SOFTWARE ENGINEER

BRIAN IBRAHIM QAMARDEEN

The Standard Approach to Become a Competent and Mature Software Engineer

by

BRIAN IBRAHIM QAMARDEEN

[Version 1.1.2]

COPYRIGHT NOTICE

TABLE OF CONTENT

FOREWORD

The question “I have learnt how to program, how can I become a good [competent] software engineer?” is highly prevalent in our industry. In an industry with many people claiming to be experts, you will expect good answers to the question, but surprisingly, the vast majority of answers provided are foolish (if you follow them, they will not make you competent, that is the definition of foolish).

A good example of the foolish answers seen is “Learn data structures and algorithms, practise all the questions on HackerRank, learn React, and learn Node.JS.”

I can start to hear a lot of incompetent engineers who put on the mask of competent engineers grumbling, “What do you mean? Who told you data structures are not important? React is one of the best frameworks ever. Bla Bla Bla.”. Well, I am glad I made you angry.

Foolish answers to the question are becoming too much, a dangerous thing for the development of upcoming engineers. It is high time someone stood up to address this issue: the motivation behind this book.

This book will share a proven approach that can be taken to become a good (competent and mature) software engineer. The approach will only be summarized, as I only have quite small free times. Hopefully, if the book turns out a success, more details will be added in the later versions.

This book will be of great help to all upcoming software engineers, regardless of whether you are undergoing formal education or learning by yourself. Engineers who are already in the industry may also find the book useful since the approach shared is not widely found in the public.

INTRODUCTION
How to Become a Competent and Mature Engineer
★★★★

To become a competent and mature software engineer (a CME), we first need to know what a competent and mature software engineer looks like. If we do not know what we want to be like, how can we possibly become that thing?

Who is a competent and mature software engineer? Well, a competent and mature software engineer is: (i) an engineer who understands informatics and computer, (ii) an engineer who is capable of creating good softwares, (iii) an engineer who is capable of creating good softwares efficiently, and (iv) an engineer who is good to their stakeholder.

If you can develop those 4 characteristics, you will become a true CME and one of the leading engineers in the world.

CHAPTER 1
Understanding Informatics and Computer
★★★★

To become a competent software engineer, you need to understand informatics and computer. You also need to acquire other useful knowledge about them. If you do not understand them, you will have a very hard time becoming a competent software engineer.

Note, there is a difference between "knowing" and "understanding". We usually think we understand when in reality we only know. A lot of engineers know the name "computer", they can name the top computer manufacturers in the world, they can tell you if an OS is a Windows or a Mac, they can tell you how much RAM a computer has, etc. Those are examples of knowing a computer. To know means to be aware of some things about the subject, but to understand means to be aware of what the subject is, in its complete, core, and fundamental sense. So when I say, understand informatics and computer, I am saying become aware of what they are in their complete, core, and fundamental senses. Also become aware of their histories, how they relate to us and humans at large, etc.

Note, understanding informatics and computer are not things you should do for the sake of doing. You should do them out of genuine interest to understand them. If you do not approach them this way, then there is probably no point in even trying to understand them.

To make your learning easier and faster, I will briefly explain what informatics and computer are, then you can read further about them.

Informatics

Informatics is the science of information processing (the creation, changing, exchange, and storage of information). Since the genesis of man, information processing has been prevalent in our lives. There is arguable no human who does not process information every minute of their life, including babies. When we calculate, we are creating information, when we edit pictures, we are changing information, when we speak, we are exchanging information, and when we download a book, we are storing information. Information processing is all around us all the time, some of us even earn our living by processing information. Informatics helps us to understand information processing.

Computer

Computers are the machines we built to process information. We built them initially for convenience and accuracy, but over time they became also desirable for other reasons like speed.

CHAPTER 2
Creating Good Softwares
★★★★

It is one thing to create **softwares**, it is another thing to create **good softwares**. As a competent engineer, you must be able to create good softwares.

What is a good software? Well, a good software is a software that can be described with the following words:

1: Maintainable
Maintainability refers to a software being easily changeable in the future. Hardly are softwares created and never came back to. For instance, you may have to fix a bug in the future, you may have to add new features, and so on.

Not being able to create maintainable softwares can be very bad. We have seen companies abandon software codebase because modifying the codebase without creating a bug had become incredibly difficult and expensive. There are so many other ways not being able to create maintainable softwares can be bad.

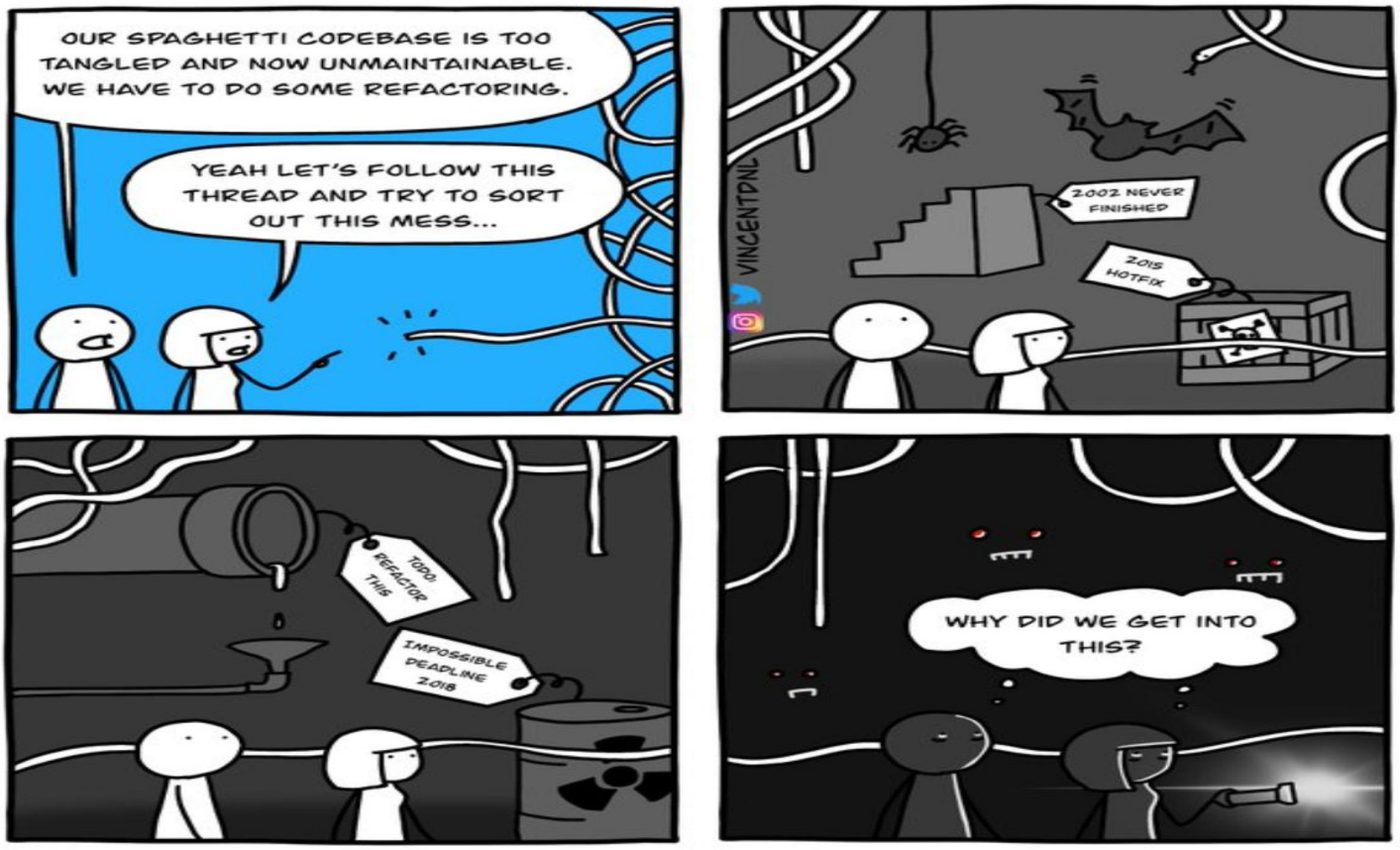


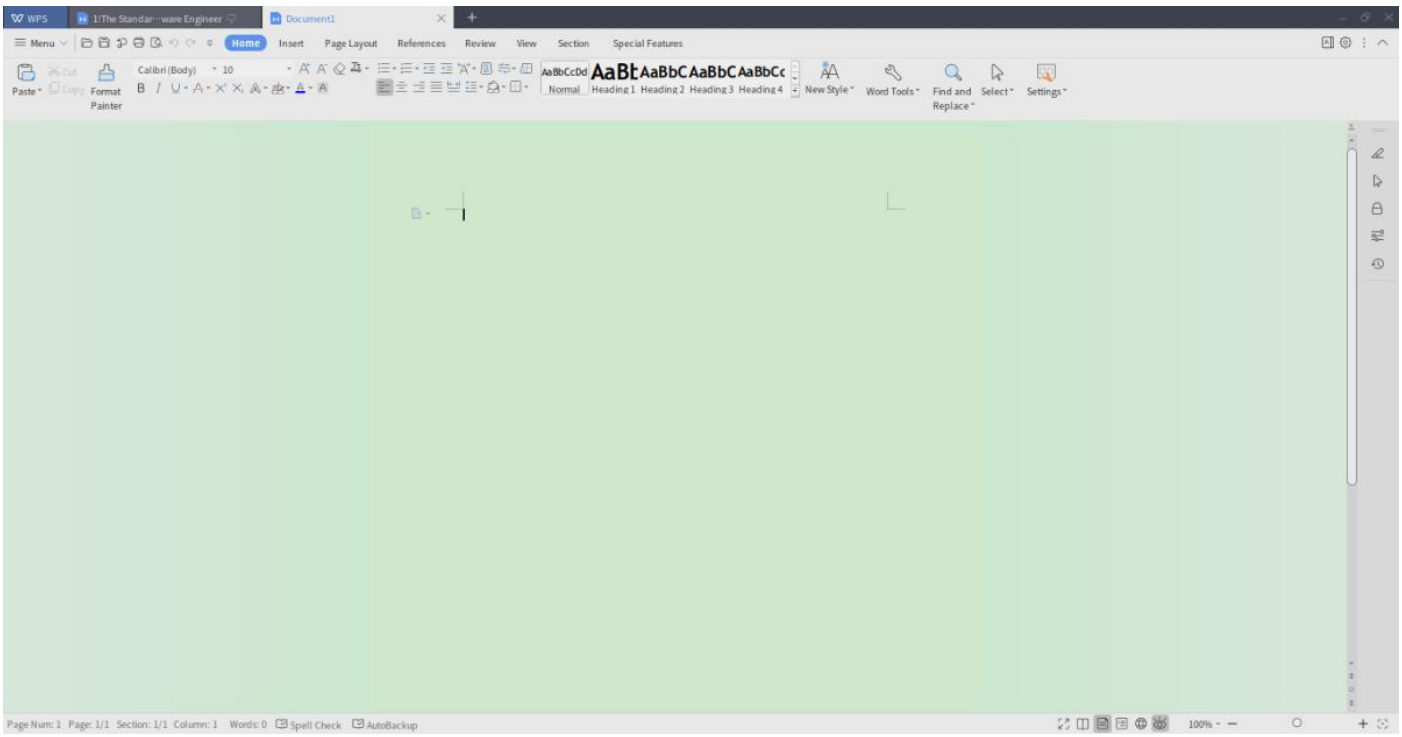
Image source: AlternativeTo on Facebook

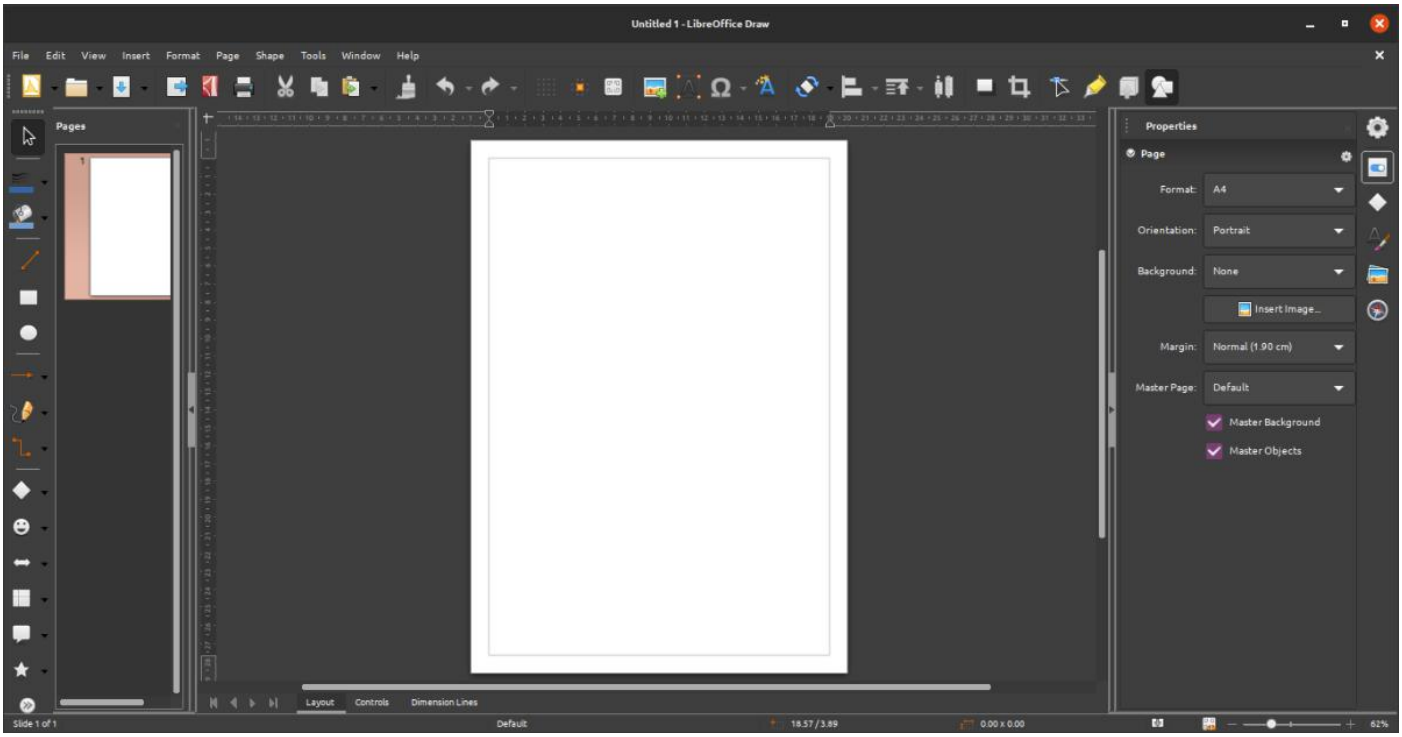
You must be able to create maintainable softwares.

Due to lack of time, I will not get into the details of "how to create maintainable softwares", but typically, learning to create maintainable softwares may involve things like: learning to document your codes, learning to structure your codes coherently, learning to use descriptive identifiers, learning to use full words (not abbreviations) in identifiers, learning to use verbal words and phrases to name functions, learning to avoid cyclic dependencies, etc.

2: Convenient

Convenience refers to a software being positive from the observatory view of a user. Convenience consists of things like: how responsive a software is, how easy it is to accomplish tasks with it, how visually appealing it is, how auditory appealing the sounds produced by it are, etc. A good example of a convenient software and an inconvenient software are WPS Office word document processor (the Linux desktop model) and Libre Office word document processor (the Linux desktop model). WPS document processor is clean, organized, and aesthetically appealing, Libre document processor, on the other hand, is one of the most clueless softwares I've ever seen (this is not meant to condemn the well-meaning and intelligent engineers behind the software. With all honesty, I love your intention and effort, but truth be told, that software needs a lot of work in the aspect of convenience.).





Not being able to create convenient softwares can be very bad. We have seen softwares fail or get replaced by competitors' softwares due to poor convenience. Your ability to create convenient softwares can make all the difference between your software succeeding and failing. Examples of softwares that failed due to convenience are some of the Linux-based GUI OS that existed in the early days of operating systems. These OS were functional but far from being as convenient as the likes of Windows and Mac.

You must be able to create convenient softwares.

Due to lack of time, I will not get into the details of "how to create convenient softwares", but typically, learning to create convenient softwares may involve things like: learning human psychology, learning colour theory, learning algorithms, learning data structures, learning design thinking, etc.

To those engineers who think I do not appreciate or understand the value of learning algorithms, data structures, and the likes, I hope you can now see how wrong you have been all along. Yes, algorithms and the likes matter, but what you do is like someone asking you to give them a place to live in, and you gave them a window instead of a house.

I understand a lot of intelligent software engineers disregard human psychology, art, colours, and so on. The thinking is usually along the line of "They are not that intellectually stimulating.". Yes, I know that feeling, but if we must do what matter, then we must embrace learning them.

3: Available

Availability refers to a software being available for use when it is needed. If you will be creating softwares that will run locally on users' computers, you may have no problem achieving this, however, if you will be creating software that will run remotely, then this becomes quite challenging to achieve.

Not being able to create available softwares can be very bad. We have many cases of companies losing huge amounts of money to poor availability. A good example is Dyn, a French company providing DNS services. In 2016, the company was hit by a 1 terabits per second DDOS attack. The attack affected about 150,000 domains names. Dyn was able to bring back its service online before the end of the day. Even though the company did not go out of business, the attack did some serious damages to its reputation, resulting in some huge financial losses.

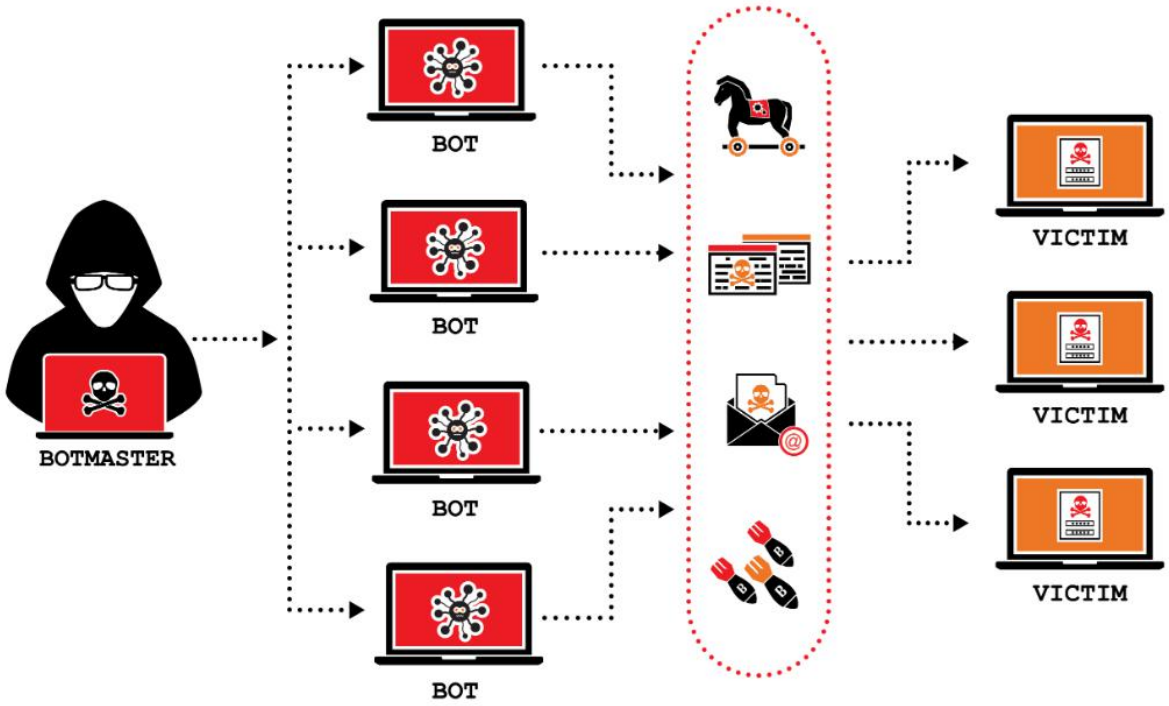


Image source: The SSL Store

You must be able to create available softwares.

Due to lack of time, I will not get into the details of "how to create available softwares", but typically, learning to create convenient softwares may involve things like: learning performance monitoring, leaning failure detection, learning DDOS attack detection and mitigation, learning distributed systems, learning redundancy, learning health checks, etc.

4: Efficient

Efficiency refers to a software being able to take only the minimum amounts of resources needed to carry out its function. User time and memory space are often two very valuable resources in the software world. If you don't learn to create efficient softwares, you will end up creating time-wasting and memory-wasting softwares. A software that should ordinarily take 1.5 seconds and 14 MB to carry out a task may end up taking 11 seconds and 143 MB to carry out the same task.

Not being able to create efficient softwares can be very bad. For instance, being able to create an efficient softwares can determine whether a software-as-a-service startup company you are working for will survive or not. If you are not good at creating efficient software, you may create softwares that will cost the startup more money than it can afford to run its softwares.

You must be able to create efficient softwares.

Due to lack of time, I will not get into the details of "how to create efficient softwares", but typically, learning to create efficient softwares may involve things like: learning algorithms, learning data structures, etc.

5: Accurate

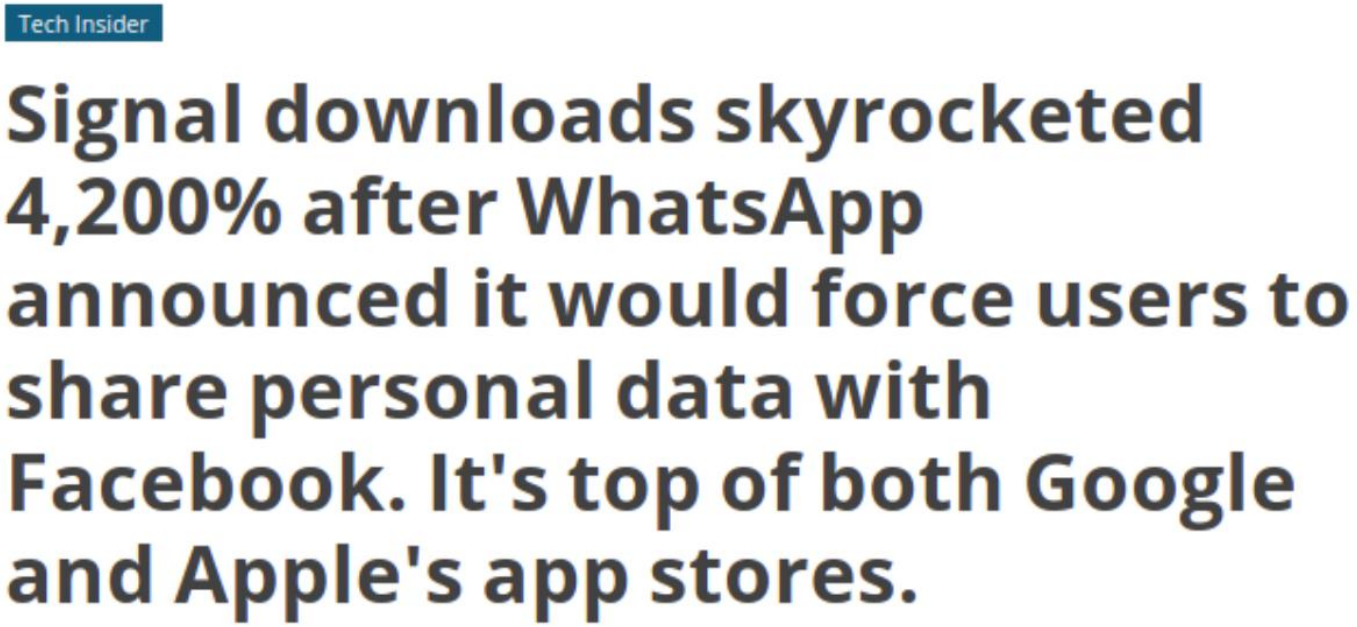
Accuracy refers to a software being able to consistently do exactly what it is designed to do. Accuracy is always an important characteristic. It is even more important in aspects like self-driving cars, planes, stock markets, etc, where inaccuracy can be very bad.

With mindfulness, engineers generally get accuracy right, so I guess, there's no need for further words on this.

6: Scalable

Scalability refers to a software being able to accommodate more loads with no or little help from you. If you are creating a single-user software, this may not be a concern, but when it comes to multi-user softwares, it is usually a serious concern. Imagine creating a platform like Amazon, your software may be able to power such a platform, but will it still be able to serve when there is a massive increase in the number of users (online shoppers) on a black Friday?

Not being able to create scalable softwares can be very bad. Not being able to create scalable softwares can deny your company a once-in-a-lifetime opportunity. In the January of 2021, WhatsApp updated its privacy policy to something a lot of privacy-conscious people were not satisfied with and Elon Musk further tweeted in response, recommending Signal. That day, Signal gained well over 2 million new users. Imagine Signal not having a scalable software that day how terrible do you think that would have been?



You must be able to create scalable softwares.

Due to lack of time, I will not get into the details of "how to create scalable softwares", but typically, learning to create scalable softwares may involve things like: learning distributed systems, learning horizontal scalability, learning caching, learning load balancing, etc.

7: Private

Privacy refers to a software being able to keep its sensitive data private (prevent them from falling into wrong hands). If you are behind a software system like GMail, it will be critical to protect users messages, contacts, passwords, and so on.

Not being able to create private softwares can be very bad. In 2011, a US email communication company, Epsilon, suffered a database hack. Millions of names and emails of credit card companies customers were stolen. Companies that were affected include big names like Capital One, Barclays Bank, U.S. Bancorp, Citigroup, and J.P. Morgan Chase. Although, Epsilon did not go out of business over this hack, it was estimated that the hack cost them about $4 billion (that is enough to pay 33,000 engineers throughout their whole careers).

You must be able to create private softwares.

Due to lack of time, I will not get into the details of "how to create private softwares", but typically, learning to create private softwares may involve things like: learning cryptography, learning hashing, learning to no run softwares with sensitive data on servers with unencrypted storages, learning identity and access management, etc.

8: Secure

Security refers to a software being safe from getting controlled by unauthorized persons.

Not being able to create private softwares can be very bad. In 2016, a total of $2 million was stolen from ATMs across Taiwan. The hackers pick a victim ATM, cripple its digital security, then remotely ask it to spit out money. While the hack did not drive the manufacturers of the victim ATMS out of business, it cost the manufacturers huge amounts of money.

Image source: The News Lens

You must be able to create secure softwares.

Due to lack of time, I will not get into the details of “how to create secure softwares”, but typically, learning to create secure softwares may involve things like: learning the gotchas of your programming language, learning identity and access management, learning to choose cryptography-based authentication over password-based authentication, learning brute-force attacks, etc.

CHAPTER 3
Creating Good Softwares Efficiently
★★★★

Creating a new software is usually very expensive, entities do need to create new softwares within a short period. Your ability to create good softwares at the smallest cost possible and within the shortest time possible can make all the difference between a bad thing happening and a good thing happening (e.g. whether a startup will go out of business or remain in business). You need to be able to create software efficiently, money-wise and time-wise.

Due to lack of time, I will not get into the details of "how to create good softwares efficiently", but typically, learning to create good softwares efficiently may involve things like: learning to create softwares using reusable components, learning how to hunt for and hire competent and mature engineers, learning how to hunt for free codes and softwares, learning frameworks, learning the agile methodology, and so on.

CHAPTER 4
Being Good to Your Stakeholders
★★★★

Typically, the stakeholders of software projects have no or poor understanding of software engineering, consequently, they try to do certain things in ways they should not be done, e.g. they may ask you to come up with an estimate of how much money a project will cost, when such project can not be accurately estimated. If you are not good to them (interested in helping them and understand their world), a struggle may surface, or in a worst-case scenario, a very expensive project may fail.

To become a mature developer, you have to: (i) develop a genuine interest in helping your stakeholders and (ii) understand their world. In the example above, if you are a mature engineer who wants to help and understand the world of their stakeholder, you may refuse to come up with an estimate but may also help them understand why it is impossible and possibly give them some little information they can work with, in the place of an actual cost estimate.

Do not get me wrong, this is not to say that you should please your stakeholders. There is a difference between pleasing and being of genuine help. Pleasing can be bad for both parties in the end, but helping will be beneficial to both parties in the end.

CLOSING
Recapitulation
★★★★

To become a competent and mature software engineer (a CME), start by understanding informatics and computer, and acquire other useful knowledge about them.

When you understand those to a reasonable extent or completely, start creating good softwares. Take on each characteristic of a good software one after the other, or if possible more than one at a time.

When you are quite good at creating good softwares, learn to create them efficiently.

When you are quite good at creating good softwares efficiently, learn to become good to your stakeholders.

When you are done with all the above, for the rest of your career, go over the four aspects over and over, in whichever order you deem appropriate, until you master them all.

WANT MORE?

I have just started to share my knowledge about software engineering. There is a lot more I'll like to share in the future. If you'll like to be aware when I share new things, from time to time, check my blog [Qeetell.vip/2106-3016-55].

If you're a LinkedIn user, you can save yourself the stress of having to check from time to time, by following me [LinkedIn.com/in/qeetell/].

CREDITS

I will like to say a big thank you to my friend (Ellah Abraham Izekor) and my younger brother (Abdul-Mateen Adebowale Qamardeen). Ellah assisted with the design of the cover of this book and offered some helpful suggestions on how the draft of the book could be improved. Abdul-Mateen also offered some helpful suggestions on how the draft of the book could be improved.

APPRECIATION

If the information in this book were to have been packaged as a seminar and marketed ingeniously, $1,500 is a very small price people will gladly pay. But I did not do that. I gave out this information free without holding back. I did that because a few of you may be unable to pay for it, even though you need and desire it. I have been human to you.

Just like I have been human to you, be human to me. Show financial appreciation on my financial appreciation page: [Paystack.com/pay/behuman]. The process is short and easy.