Table of Contents

Why this book?

This book has been written during the COVID-19 crisis we all suffered in 2020, a pandemic induced by the SARS-CoV-2 virus. It was imperative to mitigate contamination to the maximum extent possible by keeping distance and maintaining high levels of hygiene. Besides being warned not to shake hands anymore, people were asked to work from home office as much as possible. Inexorably, working from home highly relied on IT, which, in turn, highly relied on software running the internet.

I myself was working at Bosch heading a software development group, where 90% of the engineers suddenly started to work from home. Luckily enough, things went smoothly, whereas equipment was taken home and based on a reliable networking infrastructure. As a result, engineers could continue to work. Some of them were needed in the office to support our big test systems, containing hundreds of devices, but the majority could seamlessly work from home, remotely accessing the code repository, the build machines and test systems, while simultaneously using *Skype* and *Teams* for communication and collaboration purposes. All of this powered by software. In particular, it was during this period that I once again realized how much we, as a society, depend on software….

Software is becoming increasingly important in our lives. Just cast your glance everywhere where software is available and

you will conclude that software indeed runs (make that rules) the world.

A proper look at your close environment will convince you that software is ubiquitous. Your smartphone is run by software, your computer is run by software, your vacuum cleaner is run by software, your television is run by software, your car is run by software. Can you imagine any device that is not influenced by software in some shape and form?

Then, if you have a look at society, you will conclude that even more is run by software. Financial systems are run by software, the internet is run by software, public transportation is run by software, air traffic control is run by software. Let's face it, software runs the world. Imagine a world that is bereft of software and you will struggle to make sense of the world we live in.

On August 20, 2011, The Wall Street Journal published an article of Marc Andreessen titled *Why software is eating the world*. In this article, Andreessen describes the growing importance of software worldwide, to the point of changing business models and destroying complete industries. The influence of software on our businesses is, to put it mildly, huge!

Due to this increasing influence of software, companies are starting to realize that digitalization is not only a fact, but also a necessity. Companies need to transform into software

companies and upscale their investment into software. For its irrefutably prominent role in all aspects of our lives, the quality of software is paramount

When reading books and articles about quality assurance in software, I often get the impression that quality in software is only about testing. Well, in my humble opinion, it is not; software quality is much more than testing. Testing is a necessity because we are not able to produce error-free software.

This book provides a holistic view on software quality. In addition to addressing testing to achieve quality, it also looks into internal software quality and even what enables quality. This book aims to give an overall picture on what matters in software development and why, without going into the pedantic details, which are described much better in other books, like the ones mentioned in the bibliography.

Who should read this book?

This book should be read by managers who manage software development teams so that they understand why software professionals make certain choices.

Specifically, it should be read by quality managers so that they know which questions should be answered in order to validate the quality of the software.

This book can serve as an aid for software professionals who need to elucidate certain aspects of software development to management with a non-software background and why certain choices need to be made.

It should be read by recruiters, who need to recruit competent software engineers despite lacking knowledge about software per se.

It can be used for education, especially in generic technical education programs, such as Automotive or Health Care programs wherein one is trained to become a generalist in different engineering disciplines rather than becoming a specialist in one specific discipline.

Finally, this book can be read by everybody interested in software quality and wants to be convinced that quality is more than testing alone.

How this book is organized.

The first two chapters of the book provide a basic overview on software and software development lifecycles. If you are familiar with software development, feel free to skip these chapters. However, even if you are familiar with software, it still makes sense to read these chapters. Still, they might make some implicit knowledge explicit. These two chapters are specifically meant for people who are not so familiar with software development, to understand what software is and

how it is developed. This knowledge is needed for the remainder of the book.

In chapter 3, a model for software quality is introduced. A high-level model covering external quality, as experienced by the users of the software, in addition to internal quality as produced by the software developers. And, last but not the least, it covers enabling quality; the capability of an organization to develop software. This model is used in subsequent chapters to explain the different aspects of software quality.

Chapter 4 addresses the balance between investing in quality and rectifying poor quality and associated costs. It is never a black and white situation; it is always a matter of balancing driven by business considerations.

Quality Debt is introduced in Chapter 5 of which, Technical Debt is an important part.

Chapters 6 and 7 comprise practical suggestions for achieving and maintaining quality. Chapter 6 addresses ways of mitigating Technical Debt while chapter 7 addresses testing and defects.

Meanwhile, Chapter 8 talks about measuring aspects of software quality. How to get a notion of something which cannot be measured as such.

With all of this knowledge in mind, one could reconsider how to set priorities in a development team, which is addressed in Chapter 9.

Finally, after being provided with all this information about software quality and what really matters in software development, some food for thought is presented in Chapter 10.

I do understand that it might be difficult to read this book if you are unfamiliar with software development. Unintentionally or even intentionally, I might be using jargon which is technical in nature. At the end of the book, a List of Concepts is added which might help you better understand these jargons when reading.