Table of Contents

Introduction i

Why this book? i

Who should read this book? iii

How this book is organized. iv

Chapter One 1

What is Software? 1

Programming Languages 1

Huge Software Programs 5

Separation of Concerns 6

Abstraction 8

From Requirements toward Software 9

Design 10

The desired design 11

The intended design 11

The implemented design 11

Source Code 12

Execution Paths 12

Configuration Management 15

Version Control of Source Code 16

Configuration as Code 18

Software diversity 19

Build, Installation and Deployment. 20

Build, installation and deployment on a small embedded device 23
Build, installation and deployment on a personal computer 24
Build, installation and deployment in the Cloud 24
Open Source Software 25
OSS licenses 26
Copyleft 27
Permissive 27
Risks of using OSS 28
Snippets and libraries 29
OSS management 31
Software versus Hardware 31
Software is intangible, hardware is tangible 31
Laws of Physics 32
Software instantiations are identical, hardware instantiations deviate. 33
Producing, storing and distributing software is cheap 33
Manufacturing of software is an integral part of development 34
Software development “never ends” 34
Software Types 35
System software 35
Application software 36
The Cloud 36
Summary 38
Chapter Two 41
Software Development Life Cycle 41

From Waterfall to Incremental Development 41
Spiral 44
Agile 46
Scrum 48
Scrum team 49
Product Backlog 50
Definition of Ready 52
Sprint Backlog 53
Definition of Done 53
Scrum meetings 54
Scaling Scrum 55
Scaling Agile 57
SAFe 57
Continuous X 61
DevOps 63
The Stacey matrix 64
Product Lifecycle Management 67
Innovation 68
Analysis 68
Development 68
Deployment 68
Maintenance 69
End of life 70
Summary 71
Chapter Three 73

What is Software Quality? 73

The 1+3 Software Quality Model 74

Product Quality 74

Design Quality 76

Code Quality 76

Organizational Quality 77

The Icons in the 1+3 SQM 78

The Icons of Organizational Quality 78

The Icons of Product Quality 80

The Icons of Design Quality 81

The Icons of Code Quality 81

Transcendent Quality vs. Modeled Quality 82

Transcendent Quality 82

Modeled Quality 83

Both need to be considered 84

ISO-25010 84

Functional Suitability 86

Performance Efficiency 86

Compatibility 86

Usability 87

Reliability 87

Security 87

Maintainability 88

Portability 88

Summary 89

Chapter Four 91
Total Cost of Quality 91
Cost of Implementing Quality 92
Product Quality 92
Design Quality 93
Code Quality 93
Organizational Quality 94
Cost of Rectifying Poor Quality 95
Product Quality 95
Design Quality 95
Code Quality 95
Organizational Quality 96
Total Cost of Quality 96
Product Quality 98
Design Quality & Code Quality 99
Organizational Quality 100
Different sweet spots? 100
Determining your sweet spot 101
Summary 102
Chapter Five 105
Quality Debt 105
Cost of Change 107
Quality Debt Examples 109
Product Quality 109

Design Quality 109
Code Quality 112
Organizational Quality 114
Technical Debt 115
Technical Debt Quadrants 116
Summary 118
Chapter Six 119
Mitigating Technical Debt 119
Creating Awareness of Technical Debt 120
The importance of Clean Design and Clean Code 120
Complexity and the Speed of Software Development 123
Understood Design 126
Technical Debt and Developer Morale 127
Causes of Technical Debt 128
Nature of the Business 128
Change in Context 129
Development Process 129
People and Team 130
Software Craftsmanship 130
Building a Clean Design and Clean Code Mindset 132
Software Engineering Practices 133
Requirements Engineering 134
Model Visually 136
Apply Patterns 138

Architecture Risks Analysis 140
Communicating and documenting your architecture.............142
SOLID..145
Apply Clean Code 148
Peer Reviewing 149
Traceability...151
Refactoring and Restructuring 156
Static Code Analysis 159
Static Design Analysis...163
Strategic Adoption Model for Tracking Technical Debt165
unaware 166
no tracking 166
ad-hoc 166
systematic 166
measured 167
institutionalized 167
fully automated...167
Putting Technical Debt in Context 168
Summary...169
Chapter Seven...171
Testing and Bugs 171
Bugs 171
Heisenbugs...174
Managing Defects 175
Debugging 176

Debugging – scope matters. 176
Debug-activities 180
Testing 184
Alternative Test Scenarios. 185
Happy flows 186
Alternative flows 186
Sad flows 186
Data driven test scenarios 187
Boundary testing 187
Equivalence partitioning 188
Short Feedback Loops 189
Testing Pyramid 190
Agile Testing Quadrants 191
Test Driven Development 194
Behavior Driven Development 195
Graphical User Interface testing 197
Chaos Testing 198
Test Automation is Not For Free 201
Artificial Intelligence in Test Automation 202
Testcase selection 202
Self-healing tests 203
Unit tests 203
Summary 204
Chapter Eight 207
Safety, Security, Legislation 207

Safety 208

Safety engineering 209

ISO-26262 211

Security 212

Vulnerabilities 212

Organizations which Fight Exploitation of Vulnerabilities 216

Security Engineering 218

Legislation 222

New technologies require modified legislation 223

Intellectual Property Rights 224

Imposed legislation 225

Legislation in relation to safety and security 226

Summary 227

Chapter Nine 231

Measuring Software Quality 231

Measuring What You Cannot Measure 231

Lagging and Leading Measurements 234

Business Oriented measurements 235

Measuring Product Quality 235

Customer feedback 237

Found defects after deployment 238

Found defects during development 238

Test coverage 239

Measuring Design Quality 250

Models and design documentation 251
Gap between intended and implemented design 252
Measuring Code Quality 254
Understandability 255
Static code analysis 256
Secure code..258
Measuring Organizational Quality 259
Morale..260
Education 260
Programming skills...261
Infrastructure 262
Processes 262
Velocity 264
Deployment 266
Summary...267
Chapter Ten 271
Process Reference Models 271
CMMI..272
Maturity levels 272
Practice Areas 273
Determining the CMMI Maturity Level....................................275
ASPICE..277
Processes 277
Capability levels 281
Process Assessment..282

CMMI, ASPICE and Agile 283
The Usefulness of Process Reference Models 285
The power and the danger of a process reference model 285
Summary 287
Chapter Eleven 289
Setting the Right Priorities 289
Enabling should have Highest Priority 290
Culture 290
Intrinsic Motivation 295
Professionalism 298
Hiring the best Engineers 299
Reconsidering Operational Priority Setting 300
Ensure a Smooth-Running Development Environment 301
Mitigate Quality Debt 303
Improve Continuously 303
Develop Functionality 304
Summary 305
Chapter Twelve 307
Infamous Software Failures 307
Mars Climate Orbiter crash 308
Boeing 787 Dreamliner Power Failure 309
Millennium Bug 309
A2LL 311
Boeing 737 MAX; a safety related accident 312

Unintended Acceleration 314
Retrospect 316

Chapter Thirteen 319
Food for Thought 319

Trial and Error Programming 319

Punch Cards 320
Press F5 321

The Best of Both Worlds 322
Should Software Engineers be Certified? 323

Low threshold to become a software engineer 324
Software engineering is more than programming alone 324

A closer Look at Test Driven Development 325
Follow the Red- or the Green-line? 330
Speed versus Pace 332

Qualification versus race 333
Software Development 334
Pace versus speed, a balancing act. 335

Process versus Skill 336

Process and Skill 336
Software Engineer 337
ASPICE or CMMI level 338

Software Estimates..., Why is it so Hard? 339

Cone of uncertainty 340
Puzzle analogy 341

Chapter Fourteen 345
Closing 345
Abbreviations 349
List of Concepts 353
Bibliography 359
Acknowledgments 363
About the Author 365
index 367

Introduction

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.

A separate chapter, Chapter 8, is dedicated to safety, security and legislation, simply because lack of quality of these aspects may have severe consequences,

Meanwhile, Chapter 9 talks about measuring aspects of software quality. How to get a notion of something which cannot be measured as such.

Although processes are not the most important aspect of Software Quality, the industry pays a lot of attention to processes to achieve Software Quality. Therefore, Chapter 10 describes two important process reference models used in software development.

With all of this knowledge in mind, one could reconsider how to set priorities in a development team, which is addressed in Chapter 11.

To better understand the importance of the different aspects of Software Quality, Chapter 12 covers some infamous software failures.

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 13.

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.