CHAPTER 1

Introduction

The quality of the people is the single most important factor contributing to success in software development projects, but good people are hard to find.

If manpower were only a matter of money, it would not be that bad. But people are not only expensive. They are also volatile and unpredictable. They are hard to manage. They quit. They die. In software as in many trades, depending on people increases uncertainties.

Taylorism was all about the rationalization of factories, by reducing the individual's latitude to personal initiative, turning people into robots, by a scientifically designed division of labor. Formalizing well-defined tasks by decomposing them in the simplest possible movements was a way of ensuring that people could be trained and replaced easily.

Software recipes are Taylorism applied to software development. It starts with division of labor: gurus design recipes, in the form of methodologies, rulebooks and processes. These recipes are then applied by less acute and less experienced software development teams, in the hope that zealously following the recipe can replace personal involvement of gurus in projects, that their intelligence will apply transitively in their absence, just by being faithful to their gospel.

> *Me calling them recipes is more than a colorful and slightly demeaning twist. Software recipes aim at replicating the success of chefs designing balanced, tasteful and creative dishes to be replicated, reimplemented by their staff or enthusiast amateurs. If only it were that simple!*

This book is about recognizing that the multiple attempts to implement software recipes over the years have failed. It is now time to see this statistical evidence as a damning fact more than a circumstantial trend. It may have made sense in theory, but reality has proven a different matter.

As a consequence of this failure to industrialize software development, this book is about claiming that software development remains intrinsically a people's business.

Musings

This book talks about software recipes as a central theme, but it is not a scholarly textbook where unity of purpose requires one to focus on a single subject exclusively. It follows a colorful tradition of computer scientists musing about their work, their joys and their pains. It is made of a collection of loosely connected essays, which makes it immensely entertaining to write, because of the myriad of subjects, topics and domains one can choose from to make a point. Technical matters, personal anecdotes, facts, opinions, business issues – nothing is off-limits. Everything goes.

> *Just as G.H. Hardy could muse about cricket, with just as much passion as he wrote about math…*

I would have loved to name this “Confessions of a coder”, but this title is already used by one of the early volumes of Jon Bentley’s wonderful series of books [Bentley1988]. Lame variations such as “Confession of a Belgian coder” or “Boudoir confessions of a coder” would have been disrespectful.

“The Art of Computer Programming” would have been a great title as well, but it would be deceiving. This book is not intended as the ultimate reference on a wide range of techniques, as the magnum opus of Donald Knuth [Knuth2011]. More importantly, this is not a book about computer science. It is a book about how a computer scientist views the world starting with his immediate technical surroundings, then expanding on how this very dense occupation shapes his views on many other topics, such as rice on chessboards.

“A software scientist’s apology” would have been a nice compliment to G.H. Hardy’s classic book about mathematicians [Hardy1992], but this text is too tainted by engineering to make any claim to pure science. Its material comes from the trenches more than the classroom or research lab. I admire Hardy’s work, writing and posture (he confessed his humility when confronted with Ramanujan’s talent and achievements, not a popular stand in post-Victorian England), but my feelings would probably not have been reciprocated would our paths have crossed over time and distance. He expressed contempt for applied mathematics, as a mercantile and aesthetically degraded version of the real thing, the pure math designed for elegance and intellectual elation only. How would he have considered computer science, which is at best a by-product of applied math? And even worse, software engineering, which is unglorified plumbing applied to electronic devices?

> *This book is more personal than a purely scientific textbook. It is about experiences, opinions, and perceptions. I’ve recycled a stylistic trick I started using for my master’s thesis, where I distinguish facts printed in normal text, from more personal and colloquial reflections, indented and in italics.*

The early drafts of this book complied to politically correct standards, by referring to anonymous people by “his/her”, “him/her”, etc. It flattered my liberal stance, but the effect on style was dreadful, so dreadful in fact that I considered introducing monikers such as “heezorhur” instead. The cure would have then become more of a curse than the disease, and I reverted to “his”, “him” or “her” more or less at random, in effect implementing gender diversity rather than merely promoting it.

Scientist vs. engineer vs. businessman

At heart, I am a scientist, but I’ve run a software company for over twenty-five years, and against my best instincts, I have developed some acumen in business matters. I have learned how some of the great scientific and technical truths we geeks so firmly believe in translate in the real world, where real people pay us real money to get real things done.

This is the glass half-full narrative. The half-empty one tells a different story altogether. The engineers I work with, or people I meet in academic circles sometimes kindly discard my opinions, as if I had moved too far to the dark side, as if my scientific mind was too much tainted by market forces, sales and money, to remain relevant. Similarly, some people I do business with consider me as a techie with little understanding of the harsh reality of today’s market and economy. Being equally mocked by the two sides of the fence consoles me: I take it as a sign that I am standing at the right distance from both to make educated and balanced judgment calls.

Raincode

My area of expertise is not broad enough to allow me to write competently about everything. I leave such feats to world-class polymaths such as Erdős[1] and Knuth. Many technical examples in this book are related to my area of specialization, namely programming languages and compilers.

> *Everybody in this trade has used a compiler at least once. Anyone with some academic background in computer science has been exposed to the underlying theory. But how compilers are designed in the real world is closer to black magic than to industry-accepted standard processes and techniques.*

Compilers are my daily bread and butter at Raincode, which is a company I founded with my two partners Alain Corchia and Juan Diez Perez in 1999, and where I've been working ever since. Raincode specializes in programming language processing (analysis, conversion, compilation, metrics, etc.) mainly applied to legacy modernization. Our tools and services allow aging mission-critical software systems to have their lives prolonged beyond technical obsolescence, whether it comes from running on antiquated hardware, barely maintained database engines, ancient programming languages, etc.

We are exposed to very different kinds of software projects. We process our customers' application portfolios, where functional adequacy trumps technical excellence, and which are written and maintained by large teams within even larger organizations. At the opposite end of the spectrum, Raincode is organized in small teams of technical experts who develop highly specialized products with little understanding of any functional domain. And developing these products, as complex as they are, is the relatively easy part of the process. The real challenge lies more in keeping them alive in the long run (twenty years and more), controlling their entropy, training people to take over maintenance, adapting them to an evolving technical landscape, etc.

[1]Even if one could arguably refer to Erdös as a monomath, the breadth of his expertise in all branches of mathematic is just amazing.

Platitudes and metaphors

Be warned: this book will also allow me rant at two dreadful trends in IT, namely, platitudes and metaphors.

Platitudes are pre-munched statements, so obvious and so trivial, that the opposite statement simply makes no logical sense, like “Quality is important” or “Software is ubiquitous”. Such platitudes are used mechanically, as substitutes for reflection or true opinions one can evaluate and possibly consider contradicting. So mechanically in fact, that one no longer questions these platitudes, even when they make no sense whatsoever.

Metaphors are intrinsic to what we are. Recognizing patterns is an evolutional trait of human nature. It is thus – literally – natural for us to go for metaphors, reusing names and concepts outside of their original context, at the risk of stretching the reality. And software development is an ideal area for this natural tendency, as it is all about abstractions. When conveying, comparing and exchanging ideas, these abstractions must be given a name. Rather than invent new words all the time, we reuse existing words based on metaphorical similarities of variable accuracy.

“Comparaison n’est pas raison”

Software architects are not architects. Software systems do not require maintenance in the industrial sense as they don’t wear out the way industrial machines do. Bad smells don’t smell bad. A plugin is not actually plugged into anything. Cloud computing does not take place in actual clouds, etc.

Pinpointing places where metaphors are taken way out of proportion has turned into a bad habit for me.

Other than this warning about me ranting, I am not going to tell you how you should read this book. If you are at all like me, it is the kind of advice you ignore anyway. At best, I can tell you how I would, namely by opening it at random and reading a few pages here and there. Or cover to cover, but starting with the end.

Anything goes.