Cover

Contents

Introduction 1
Conventions: 1
Why this book? 2
Who this book is for 2
Role and responsibilities 3
The role of agile technical leaders 3
The need for constraints 3
Technical leadership is a tough job 4
Technical leadership and self-organization 4
One technical leader or the whole team? 5
So the technical leader dictates the rules? 5
Agile technical leader vs. Scrum Master? 6
Constraints 7
Configuration management policy 7
Testing strategy 8
Error management and logging policy 8
Code review policy 8
Design Elements 8
Practices of an Agile Technical Lead 8

Introduction

Conventions:

The following conventions are used throughout this book:

- "Developer" means anyone contributing to the success of the product. They can be programmers, testers, designers etc.
- Singular "they" instead of "he or she", as in: "The product owner attended the meeting. They clarified the goals".
- "Technical leader" stands for any person who influences considerably the technical practices, architecture and design in a company. Typically the roles are technical lead / team leader, architect, CTO and, in certain contexts, Scrum Master, XP Coach and

the developers.
- "Technical leader" will also be used as a collective noun, for situations when the whole team or a group of people plays this role.
- "Agile" means any type of method that helps agility of the business. This includes lean techniques such as kanban or extreme programming technical practices (XP)

Why this book?

I've met in my 15+ years of working in the software development industries hundreds of people complaining about same issues: estimations, legacy code, ... I have come to understand that many of them come from disfunctional technical leadership.

I found two reasons for these situations:

1. Self-organization is misunderstood (either as anarchy or limited to almost non-existence)
2. The technical leaders lack the necessary knowledge, and sometimes skills

It's not the fault of the leaders. They're busy people, trying to keep projects, teams and companies afloat. There's little time to learn and no technical leadership manual. Most technical leaders are promoted based on being very good programmers, but there's little guidance to adjusting to their new role.

I did my best to write the missing manual for agile technical leaders. You will find in it practices, tools and knowledge that helps you become a better technical leader.

There's one caveat. To keep it short, I will present my style of technical leadership, based on my experiences. I leave it to each reader to find their own style, but we all need to start somewhere. If you're a beginner in technical leadership, start by copying. If you're more advanced, adapt. Either way, this book will help you understand and perform better in your very important role.

Who this book is for

This book will discuss general practices for all types of technical leaders, from team level, to product level to organization level. The typical job titles of these people are: technical lead, team leader, architect (on different levels) or CTO.

This book is for you if you're planning to take over such a role, if you're just beginning or if you already exercise it.

Role and responsibilities

The role of agile technical leaders

It's easy to recognize leaders when meeting them, but very difficult to explain what differentiates them from the others. When speaking about leadership, people usually think about charisma, vision, clarity, commitment etc.

We'll be discussing a very specific type of leader. A leader that influences the technical aspects of the development and who works in an agile team. In this context,

The role of technical leaders is to set constraints related to technical practices that create the best chances of success

The need for constraints

Imagine a barren piece of land left to its own rule. Give a few years, and vegetation will take over it in a chaotic way. The result is not pleasant, and can even become detrimental.

Compare it with another piece of land that has a gardener. The gardener will place constraints on what grows in the garden, where and how much. These constraints help create beautiful places where the plants flourish and that are in harmony with the rest of the land.

This is a typical example of a complex adaptive system. It's a system, because it's formed out of things that coexist and influence each other. It's complex because a very small variation in the initial conditions can result into a huge variation after some time. It's adaptive because it will respond to changes in the environment.

In this metaphor, constraints help focus the energy of the system towards a certain goal. The gardener does not control each plant in the garden - that would be impossible. Instead, the correct set of constraints helps getting desired results.

Self-organized teams are also complex adaptive systems. So is the code they produce. But left unchecked, they might evolve into a less ideal, maybe even detrimental direction. Constraints are needed to ensure the best results.

This is a metaphor, and like any metaphor, it tears down quickly. After all, programmers are smart people and we can't compare them with plants. But programmers have their issues: lack of discipline, stubbornness, wishful thinking, propensity towards execution, avoidance of unknown and uncertainty etc. Any constraint that helps deal with these issues will make things better for the whole system, although it might make it temporarily worse for individuals.

It would be ideal if each developer developed ways to deal with these issues. Software craftsmanship is a movement that tries to do just that. Unfortunately, we're a long way from getting there. Truth is, most teams need technical leadership, and most teams need dedicated people to play this role. For a very good reason.