Preface

This is not a philosophical, theoretical or motivational book, but a practical one. Its purpose is to enable readers — software developers, managers involved in IT, and educators — to benefit from the good ideas in agile methods and stay away from the bad ones.

Agile methods are undeniably one of the important recent developments in software engineering. They are also an amazing mix of the best and the worst. This is an extraordinary situation: usually, when a new set of concepts bursts forth, one can quickly assess its overall contribution as beneficial, neutral or detrimental. Agile texts, however, defy such a simple judgment: you may find in one paragraph a brilliant insight, in the next paragraph a harmless platitude, and in the one after some freakish advice guaranteed to damage your software process and products.

No wonder then that practitioners have massively disregarded injunctions to use this or that agile method — such as Scrum, Extreme Programming, Lean Software and Crystal, the most prominent ones today — in its entirety. Industry knows better, and every agile team in the field makes up its own cocktail of agile practices, rejecting the ones that do not fit. Until now, however, each organization and project has had to repeat for itself the process of sorting out the gems from the gravel. What a waste of effort. This book spares you the trouble by presenting a comprehensive **description** and **assessment** of the key agile ideas.

DESCRIPTION AND ASSESSMENT

The first goal is *description*: you can use this book as a primer on agility, presenting the approach concisely, coherently and comprehensively. If agile development is new for you, this presentation will, I hope, teach you what it is about, enable you to apply to your own projects the agile ideas you decide to retain, and prepare you if you wish to read the more specialized literature (such as the texts advocating a particular agile method) in the most effective and profitable way. If you have already read about agile methods, and perhaps practiced them, I hope it will help you put all the concepts in place, understand them in depth, and apply them better.

What makes this descriptive component of the book necessary is that until now, in spite of the already enormous literature on agile methods, there was no place, as far as I know, where you could find a complete yet concise presentation of the essential agile ideas and techniques, not tied to a particular agile method, not drowned under anecdotes, and not interspersed with a constant exhortation to join the cult. Sermons have a role, but for most people, I think, it is more interesting to find out what exactly is meant by "velocity", "continuous integration", "user story", "self-organizing team", "sprint review", "planning game", "mob programming" and so on. That is what I have tried to provide — in 162 pages.

The second goal is *assessment*: we take an even-handed look at agile methods and sort out what helps, what is not worth the attention, and what harms — the good, the hype and the ugly. The assessment is unbiased (I have no horse in this race) but that does not mean it is the only possible one, since empirical software engineering, the objective study of software processes, is still a science in progress. So you will not necessarily agree with all the conclusions, but I think you will agree with most, and where you disagree you will be able to appreciate rational arguments on both sides.

The two aspects — "news" and "editorial"— are separated: you are entitled to know at any stage whether you are reading the factual presentation of an agile technique or a discussion of its merit. Judgmental elements are marked by the icon shown here on the right. The scope of its application will be clear from its position: at the start of a paragraph, it generally applies to the remaining part of the current section; at the start of a section, to the full section; and in the case of the final assessment, to the full chapter.

KEEPING A COOL HEAD

Anyone trying to gain a clear, cool-headed understanding and appreciation of agile methods has, so far, faced three difficulties that I hope this book removes: partisanship, intimidation and extremism.

Most of the existing texts are **partisan**. At issue here is not just the normal phenomenon of inventors arguing for their inventions, but a lack of restraint that sometimes borders on religious fervor and demands from the reader a suspension of disbelief. The first presentations of structured programming, object technology and design patterns — to cite three earlier developments that each imprinted a durable mark on how the world builds software, as agile methods have already started to do — were enthusiastically promoting new ideas, but did not ignore the rules of rational discourse. With agile methods you are asked to kneel down and start praying. This is not the right way to approach solutions to engineering problems involving difficult technical and human aspects.

The agile literature is often **intimidating**. It dismisses previous approaches as passé, scornfully labeling them "waterfall" (even though no company applies a strict waterfall process), and leaving the impression that anyone supporting them is a rigid, pointy-hair-boss type. We will encounter the typical example of an author for whom any objection to agile methods is a mark of "*bureaucracy*", "*incompetence*" and "*mediocrity*". The very name for the approach, "agile", a brilliant marketing decision — no, a stroke of genius! —, is enough to make any would-be skeptic think twice: who wants to

→ *See "Intimidation", 2.2.3, page 23.*

be cast as *not* agile? If you search the dictionary for antonyms to "*agile*", you will find such niceties as "*awkward*", "*lumbering*" and "*ungraceful*". If those are the alternatives, you, I and everyone else want to be agile! This name is just a name, however; we must unemotionally assess, one by one, the concrete principles and practices that it covers.

Clear, no-nonsense assessment is also complicated by **extremism**: the insistence of some method designers that you must apply their prescriptions entirely. There are exceptions; Crystal, for example, is more of a flexible, your-mileage-may-vary approach. But the prevalence of the all-or-nothing view in many of the foundational texts further complicates the task of identifying which techniques will work for your own project, and which will not.

PREVIOUS ATTEMPTS

Among the many books on agile methods, I know of only three that have not taken an adoring tone. The first is McBreen's *Questioning Extreme Programming*, whose "questioning" is plaintive, leaving the reader uncertain about any serious problems with XP. *Extreme Programming Refactored: The Case Against XP* by Stephens and Rosenberg does not suffer from such angst; it is a pamphlet, both funny and enlightening, but like any pamphlet it does better at highlighting absurdity than at performing a fair pro-and-con analysis. The book that made the most serious attempt at such an analysis, Boehm and Turner's *Balancing Agility with Discipline*, contrasts agile approaches with traditional plan-driven software engineering techniques. Its great strength is that it relies on empirical data from studies comparing the effectiveness of agile techniques to their classical counterparts. For my taste it tilts a trifle too much to the side of cautiousness; perhaps because Boehm is such a respected figure in software engineering and feared being branded as a proponent of the old order, the authors avoid sounding too critical.

[McBreen 2002].

[Stephens 2003].

[Boehm 2004].

Do not expect such timidity in the present book (mentioning this just in case you were worried). Respect yes, deference no. It will highlight and praise the good ideas, and when it encounters balderdash it will call it balderdash.

STRUCTURE OF THE BOOK

The book has a simple structure and is intended for sequential reading.

The opening chapter, entitled "overview", presents a summary of agile ideas and a first overall assessment. It sets the stage for the rest of the book and serves as a summary of it.

The second chapter is a short foray into the style of agile descriptions, serving as a form of immunization against the risk of unjustified generalization. Working from examples in the agile literature, it analyzes the intellectual devices that agile authors use to convince the world.

Chapter 3 is a sketch of everything that agile methods do not want to be and agile texts love to lambast: traditional plan-based software engineering methods, including the derided "waterfall".

The next five chapters, the core of the book, review agile ideas: **principles** in chapter 4, **roles** (in the sense of personnel roles, such as managers and users) in chapter 5, **practices** in chapters 6 and 7, and **artifacts**, both material and virtual, in chapter 8. Here we do not focus on any specific method but look instead at the concepts and tools shared by all or most. This approach illuminates the many commonalities between the various methods. It will allow you to examine agile ideas by themselves, in a non-denominational way, so that you can decide which ones are suitable for your context. When some of them apply more specifically to one method, the discussion points this out, and includes in the margin one of the icons shown here on the right. The focus in those chapters remains, however, on individual methodological concepts and techniques.

These symbols were designed for the present book and are not official logos of the methods.

That focus moves to the methods themselves in chapter 9, which studies four of the principal agile methods in existence today, the four already cited: Scrum, Lean, XP and Crystal. Since the constituent ideas have been presented in the preceding chapters, 4 to 8, we can in the presentation of each method concentrate on the particular combination of principles, roles, practices and artifacts that it has chosen, and just as importantly on the characteristic *spirit* of that method. The analysis shows that each of them has "one big idea" that sets it apart, supported by a number of auxiliary concepts.

Chapter 10 is brief; it describes precautions that organizations should take when adopting agile methods, in particular when some are more agile than others. It warns that the laws of software engineering continue to apply, and cautions against the "either-what-or-when" fallacy that works well for consultants but not for their clients.

Chapter 11 is the final assessment: an overall examination of the agile canon, appraising which ideas stand up and which just do not make sense. It shows indeed that, as the book's subtitle indicates, agile ideas can be classified into three categories:

- *The good* (including the "*brilliant*"): principles and practices — some new, some not — that agile authors rightly present as helpful to software quality and productivity.
- *The hype*: widely touted ideas that will make little difference, good or bad, to the resulting software.
- *The ugly*: agile-recommended techniques that are just plain wrong, contradicting proven rules of good software engineering, jeopardizing the success of projects, and harming the quality of the resulting software.

PERSPECTIVE AND SCOPE

Any book is colored by its author's experience. What mostly characterizes mine is the mix of industrial practice (for most of my career) and academic work (for the past decade).

It is also useful to note what this book does *not* include: a comprehensive approach to software development. My previous books describe techniques of quality software development and argue for specific approaches, particularly object technology, formal specification and Design by Contract. This one, in contrast, studies other people's work. Even when I felt that my own work is relevant to the discussion or predates some of the successful agile ideas I have (except for a hint or two) refrained from talking about it.

ANALYSIS: INSTINCTIVE, EXPERIENTIAL, LOGICAL OR EMPIRICAL?

Software methodology is a tricky business because it is difficult to prove anything. Many ideas get adopted on the strength of an author's powers of conviction. It does not mean they are good, or bad.

Authors use four kinds of argument: gut feeling, experience, logical reasoning and empirical analysis.

Do not laugh at **gut feeling** as a means of persuasion; after all, the mother of all software methodology texts, Dijkstra's 1968 *Go To Statement Considered Harmful*, largely relied on it:

> *Recently **I discovered** why the use of the go to statement has such disastrous effects, and **I became convinced** that it should be abolished from all higher-level programming languages.*

[Dijkstra 1968], emphasis added.

But if you are not Dijkstra your gut feeling will not take you very far in a quest to convince the community.

Experience was also part of Dijkstra's rationale:

> ***For a number of years** I **have been familiar** with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce.*

Experiential arguments are among the favorite tools of agile authors. The typical agile book is a succession of alternating general observations and personal anecdotes of project rescues (rescued, remarkably, by the author) and project failures (failed, remarkably, after not following the author's advice). These anecdotes are usually entertaining and sometimes enlightening, but a case study is only a case study, and we never know how much we can generalize. One can, after all, summon an experience in support of almost any recommendation.

> Anecdotes and individual cases, by the way, can have force of proof, but only in one case: *disproving* a general law. If such a law has been proposed, it suffices of a single experiment to negate it (the technical term is "falsify"). For example if someone — say, Aristotle — told you that bodies fall at a rate that depends on their mass, just go up that tower in Pisa, drop a light ball and a heavy ball, and see them reach the ground at the same time.

Logical reasoning is a powerful tool; it played a significant role in Dijkstra's advocacy (and for Galileo too, who according to some authors proved his hypothesis solely by thought experiment). But it is only as convincing as the hypotheses from which it starts, and there is the risk that it will remain academic.

Ideally, we should use **empirical analysis**. Does pair programming lead to better results than code inspections? Is constant customer interaction preferable to a solid requirements process? Credible answers to questions of software methodology require systematic, rigorous, realistic studies of projects. This book relies on such results when available, but there are not enough of them; the burgeoning field of empirical software engineering has not yet provided answers to many fundamental issues. This has been perhaps the biggest obstacle in the preparation of the book. Where not enough empirical evidence was available, the discussion largely relies on analytical reasoning.

I have not completely avoided anecdotes and personal experience, but have tried to confine them to the illustration of points supported by logical argument and to the task, mentioned above, of disproving undue generalizations.

FREE CRITICAL INQUIRY

Given that this work includes critical comments, a word is in order to explain the spirit in which it has been written.

Progress in science and engineering relies on free, critical inquiry of previous work. In reviewing the agile literature, I have found a number of reasons to disagree with its authors, and a few reasons to be shocked; I have not been coy about taking their claims to task. I have also, however, found elements to admire, and learned new insights about software development. This observation is worth remembering whenever you encounter criticism in the following pages.

I would not have spent a good part of my last three years immersing myself in agile methods and the supporting texts if I had not felt that I had something important to learn. The path has been tortuous at times; with this book I hope to spare you the path and share the lessons.

In no case does the criticism mean disrespect; the agile pioneers are experienced professionals, passionate about software. Even when I find them to be wrong, I value their views and share the passion. We are all in the same boat.

Bertrand Meyer
January 2014

ACKNOWLEDGMENTS

Since this book makes a number of judgments, the customary caveat that its content commits only the author is more than perfunctory: by acknowledging sources of influence and help I do not mean to imply that anyone listed endorses the views expressed. This caveat particularly applies to the first group of people to be thanked, some of whom may be expected to disagree: the authors of the best agile books. I have learned a lot from reading about agile methods, and am particularly indebted to the books and articles of Kent Beck, Barry Boehm with Richard Turner, Alistair Cockburn, Mike Cohn, Craig Larman, Mary and Tom Poppendieck, and Ken Schwaber with Mike Beedle. I credit my first encounter with agile ideas to a presentation of Extreme Programming by Pete McBreen at the Le Bréau EDF/CEA summer school in 1999. I am grateful to Mike Cohn for clarification of the origin of two of his citations. I also benefited from a lively Scrum workshop by Jeff Sutherland in Moscow, enabling me to become a proud Certified Scrum Master.

I have given several industry seminars at ETH on the theme of this book and gained from the participants' comments. I am grateful for the advice of Ralf Gerstner at Springer on refining the focus of the book, and am also indebted to his colleague Viktoria Meyer. Patrick Smacchia brought some recent agile practices to my attention. Claude Baudoin, Kent Beck, Judith Bishop, Michael Jackson and Ivar Jacobson were kind enough to encourage me after seeing a draft. Paul Dubois and Mark Howard sent me important comments which helped focus and refine the text. Claudia Günthart and Annie Meyer helped with editing. I have a special debt to Raphaël Meyer's for his thorough reading of the text, which led to essential improvements.

If I ever felt like pontificating abstractly about software engineering, I would quickly be brought back to earth by the development group at Eiffel Software; we are fighting the battle every day. Together we have seen it all: successes as well as those less glorious moments, the development iteration that seemingly will never end, the critical bug that surfaces two days after a release, the amorously crafted feature that turns out to interest nary a user. We are agile, in the best sense of the term, but we are learning all the time.

I have drawn on some material published on my personal blog, at bertrandmeyer.com, and on my blog at *Communications of the ACM* (cacm.acm.org/blogs/blog-cacm). I am grateful for reader comments on blog articles.

I am indebted to members of the Chair of Software Engineering at ETH Zurich for many discussions on software engineering issues. I cannot cite everyone but should mention that a remark by Till Bay was the spark that led to switching the EiffelStudio development to an agile-style time-boxed release process, and that Marco Piccioni first brought Scrum to my attention. He also made a number of important suggestions on the draft of the present book. In the ETH course "Distributed Software Engineering Laboratory", where students from a dozen universities around the world work together on a challenging distributed project, my co-instructors of many years, Peter Kolb and Martin Nordio, contributed numerous insights, as did the assistants (Roman Mitin, Julian Tschannen, Christian Estler) and the students and instructors from the participating universities. The course led to a number of published empirical studies which significantly helped my understanding of the field.

se.ethz.ch/dose.