CHAPTER

1

Introducing the Agile Methods

Contents

This chapter provides a brief and basic introduction to the Agile methods. It is designed to provide a minimal foundation for the contents of this book but also contains pointers to sources of more complete information for those interested in investigating them more deeply.

Historical and background information

Although some of the Agile methods have existed in one form or another for a decade or two (a relative eon in the software business), the term "Agile Method" was coined more recently, in February 2001, by 17[1] of the leading developers and proponents of what were then known as the "light" methodologies. These people met "to see whether there was anything in common among the various light methodologies" [1]. The meeting resulted in four levels of agreement among participants.

1. There is a need for methods designed to respond to change during software projects. Further, they adopted the term "Agile" to identify those methods. They agreed that the term "light" was not appropriate because certain projects (e.g., those with many programmers or those that develop safety-critical software) would not employ a "light" methodology but could still require agility.
2. The second level of agreement was on the four statements of the "Agile Manifesto".[2] These four statements capture the

1. The 17 people were Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Stephen J. Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas.
2. The Agile Manifesto is quoted and discussed in Appendix A.

core values on which all of the Agile methods are built, as well as the spirit in which they should be implemented. The Agile Manifesto states:

> We are uncovering better ways of developing software by doing it and helping others to do it. Through this work we have come to value:
>
> - **Individuals and interactions** over processes and tools.
> - **Working software** over comprehensive documentation.
> - **Customer collaboration** over contract negotiation.
> - **Responding to change** over following a plan.
>
> That is, while there is value in the items on the right, we value the items on the left more.

3. The next level of agreement was on a set of 12 Agile Principles.[3] In these statements, the values are fleshed out in more detail and given more concrete meaning.
4. The final level of agreement was that agreement at a more detailed level (e.g., actual activities or tactics for running projects) was beyond their grasp at the time. They were content to leave that fourth level for each of the Agile methods to define in its own way.

Since that meeting, the Agile methods have become a popular topic in software development circles, giving rise to much confusion and contention. The Agile Alliance[4] has grown to become the voice of the Agile methods. Their Web site is an active forum for practitioners to share their experiences and the practical matters related to using the Agile methods. The Web site is probably the best source of up-to-date information about the Agile methods.

This book is designed to cut through the partisan positions of the various factions in the Agile method debate. It presents a balanced view of the Agile methods to give you the information you need to make an adoption decision based on an analysis of facts and consideration of your organization's needs.

The Agile methods, generally

The suspicion that led to that February 2001 meeting where the Agile Manifesto was developed was confirmed. These methods do, in fact, have much in common. The commonalities revolve mainly around the topics of agility, change, planning, communication, and learning.

3. The 12 principles are quoted and discussed in Appendix B.
4. Refer to http://www.AgileAlliance.org.

Agility

The name that they adopted, "Agile," pinpoints one of the key attributes that these methods share. What image does the word "agile" bring to your mind? Perhaps it makes you think of a gymnast's performance, or a mountain goat navigating steep rocky crags, or a gazelle running in the wild. Someone who is agile is able to move quickly but decisively, to react to changing situations with speed and grace, to change direction while maintaining his or her balance and poise.

This image contrasts sharply with the more traditional software development methods, which more closely resemble a large military encampment. These methods commit to a war plan and then steadfastly march forward toward the stated goal, all the while controlling and mitigating any outside effects, as opposed to accommodating them.

That the Agile methods are designed to be able to move quickly and react to change is clear. What is less clear is the extent to which they can do these things with speed and grace … with balance and poise. The concern many people have about these methods is the cost that may be incurred in order to achieve agility. What must be given up to achieve those ends? And is the achievement worth the cost?

Change

The philosophy of agility shows up most pointedly in the methods' approaches to change. The Agile methods treat change as an equal partner in the project. Change is welcomed to the table and encouraged to shed new light and introduce new information continually throughout the project. These methods are designed not just to *accept* change, but also to welcome it and capitalize on it.

The traditional methods treat change as the enemy. They accept that it cannot be avoided, so they spend significant effort to control it and mitigate its effects. Far from welcoming change, most methods shackle it in Change Request systems and try it in the court of Configuration Control Boards (CCB). Each change is interrogated and examined in an effort to determine what should be done about it.

An appropriate position on the subject of change is a point of contention that we will discuss in some detail in Part V. For most projects, neither of the two extremes described is optimal; rather, some middle ground is more likely to be to the project's advantage.

Planning

Although planning is as central to the Agile methods as to any traditional method, the Agile methods treat deviations from plan very differently. When an Agile project does not progress as planned, the methods treat the deviation as new information about the project, and they generally replan in light of that new information. Their goal is to bring the plan into conformance with reality.

The more traditional methods view deviations from the plan as undesirable events. Therefore, they react to deviations by adopting corrective actions. In extreme situations, the "corrective action" may involve replanning, but these methods much prefer to bring reality back into conformance with the plan.

So, should reality be molded to the likeness of the plan? Or should the plan be reworked to match reality? Clearly, neither position is optimal. Rather, each deviation should be evaluated and an appropriate reaction adopted. This subject will also be discussed in Part V.

Communication

All of the Agile methods are designed to optimize communication among the various stakeholders. They strongly favor face-to-face communications and tend to de-emphasize written documents, except where those documents provide real-time support for the more favored face-to-face communication. Although none of these methods actually intends to eliminate all documentation, they do militate against documentation that is primarily archival in purpose. They all pay primary attention to communication among project team members and between the team and the customer (in whatever way the word "customer" is defined).

The traditional methods do not argue *against* communication; rather, they tend to *assume* that appropriate communication will result from the prescribed activities and documents. Unfortunately, this assumption sometimes remains unmet as each party does what is required of them and reads their own biases into documents, only to find that misunderstandings surface later in the project.

While the Agile methods' emphasis on communication is welcome, their tendency to not document the results of those communications can cause problems. People's memories are often faulty, and two people can have very different memories of the same exchange. So, a more appropriate philosophy might state, "If something is worth talking about, then it is also worth recording what was said." These topics will be addressed in Parts II and III.

Learning

Each of the Agile methods treats a project as a learning experience. They acknowledge that, at the beginning of the project, neither the customers nor the developers have a complete understanding of what must be built. Therefore, the following occurs:

- The methods foster copious communication among stakeholders to accelerate the learning that will take place.
- The new learning results in changes to the requirements for the system, the technical constraints on it, and the ways in which it will be used.

- Those changes become the basis for evolving plans, as the project adapts to the new information.

The traditional methods regiment learning into the project life cycle. It is assumed that all that is needed to plan the project is available during the planning phase, that all of the requirements are understood during the Requirements phase, and so on. When this turns out not to be the case, the resulting deviation is documented, and corrective action is taken.

The reality is that every stakeholder in every project continually learns throughout the life of the project. To expect that this will not be the case is to expect miracles. But at the same time, (as discussed in the "Change" and "Planning" sections) not everything that is learned is necessarily beneficial to the project. Each new piece of information should be evaluated and integrated into the project only if it provides value. These topics will be addressed in Parts IV and V.

The Agile methods, specifically

This book focuses on the following six[5] major Agile methods:

- *Adaptive Software Development (ASD),* as discussed in Appendix C, is based on Complex Adaptive Systems theory and treats software development as a collaborative learning exercise. ASD is based on the "Adaptive Life Cycle" (which continually cycles through three phases named "Speculate," "Collaborate," and "Learn) and the "Adaptive Management Model" (also called "Leadership-Collaboration" management).
- *Dynamic System Development Method (DSDM),* as discussed in Appendix D, is not properly a "method" because it does not provide guidance about how development projects should be run. Rather, it is mainly a philosophy about system development that consists of nine principles. DSDM focuses on *system* development and does not get into the details of writing software, so it can be used in conjunction with any of the more software-intensive Agile methods, like XP.
- *Extreme Programming (XP*), as discussed in Appendix E, is a collection of 12 practices that focus specifically on the mechanics of developing software. These practices include such topics as The Planning Game, Pair Programming, Refactoring, and Testing.
- *Feature-Driven Development (FDD),* as discussed in Appendix F, treats software development as a collection of features that are implemented one at a time. Unlike the other Agile methods, FDD includes upfront

5. These six methods were chosen because they are widely recognized as being Agile methods and there was enough published information about them at the time of this writing to support evaluation. Absence of any particular method from this list should not be interpreted as a judgment of that method. It is merely an artifact of the available information at this time.

architectural analysis, such as the development of a Domain Object Model, which becomes the basis for planning the project iterations. It also includes a unique (among the Agile methods) mechanism for objectively reporting progress against plan.

- *Lean Software Development (LD),* as discussed in Appendix G, is not really a software development method. Based on the principles of lean manufacturing, LD provides a set of seven principles for making software development more efficient, and it amplifies those principles with 22 tools.
- *Scrum,* as discussed in Appendix H, is primarily a product development method. Its seven practices focus on planning and managing a development project but do not address any specifics about software. Therefore, it can be used in conjunction with any software development method.

Each of these six methods embodies the philosophies of the Agile Manifesto. Each is widely recognized as an Agile method. And each provides a good basis for the discussions in this book.

Reference

[1] Cockburn, A., *Agile Software Development*, Reading, MA: Addison-Wesley, 2002, p. 215.