Introduction

We can't solve problems by using the same kind of thinking we used when we created them. — Albert Einstein

One of the goals of this book is to give you a better perspective on Lean and Agile and how to use them in software development. This requires an understanding of the roots of Agility, the software development "pendulum," and the importance of paradigms and practices and of being pragmatic. Lean offers a way forward.

This book takes the reader beyond Agile's standard practices by teaching how to incorporate Lean thinking into software development. Although Agile, as it is usually practiced, is effective at the team level, it gives little guidance on how it fits at the enterprise level. This is somewhat for historical reasons, as you will see. Lean-Agile is an approach to Agile software development using Lean principles and practices for guidance.

You can think of Agile in one of two ways: as a set of values and beliefs that leave it to the practitioners to decide how to apply them or as a set of practices that are suggested to manifest good results. Practitioners typically use a combination of both, believing the mandate of the Agile Manifesto and then using either Scrum1 or eXtreme Programming2 (or some of each) as the basis for their methods. The challenge with this approach is two-fold—one resulting from the roots of Agility and the other from the lack of a theoretical foundation for the Agile practices themselves – as we will discuss later.

How This Book Will Help You

This book aims to change how you look at software development. Doing so will enable you to solve seemingly intransigent problems with much less effort than you might have thought possible. One of our guiding principles is that we need to drive from business value: Deliver the value (software) that will provide the greatest return to the business by providing the greatest value to the business's customers. For an IT development group, this could mean either internal or external customers.

Together, we will explore what software development actually is and how it must be managed. We will investigate ways to help our customers through the process of selecting what work to

[1] Scrum is a popular Agile process created by Jeff Sutherland and Ken Schwaber. It is commonly used at the team level and is characterized by self-organizing, cross-functional teams doing iterative development in what are called Sprints.

[2] eXtreme Programming is an iterative development process for teams centered around several engineering practices. The most common of these is test-driven-development, paired programming and continuous integration.

accomplish through development, deployment, and, ultimately, ongoing support and enhancement.

We will drive from principles throughout the book and provide a good many that you can apply in Lean Software Development. This book will not give you all the answers; instead, it will help direct your thinking so you can create answers that will work for you in your company, in your situation, for your customers, and with your products.

The Roots of Agility

The development of the Agile Manifesto (Beck, 2001) was a breakthrough event for the software industry. The manifesto (See Figure I.1) and its "Twelve Principles" (See Figure I.2) describe the essential ideology that underpins Agile software development.

Manifesto for Agile Software Development[3]

We are uncovering better ways of developing software by doing it and helping others 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.

Figure I.1 Manifesto for Agile Software Development

Principles behind the Agile Manifesto

We follow these principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

[3] Copyright © 2001 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, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas; this declaration may be freely copied in any form, but only in its entirety through this notice.

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Figure I.2 Twelve Principles behind the Agile Manifesto

The Software Development Pendulum

The Manifesto is a strong statement. It is consistent with the intentions of most people in the software development industry. But it says that we must develop in a way that is different from the ways we have often tried in the past. It stands in opposition to the myth that the way to create quality, sustainable software is to conceive large plans and then use command and control management[4] to realize them. When it was written, the Agile Manifesto presented a great opportunity for exploring new, better ways of developing software. Unfortunately, it also left a huge hole. It did not attempt to describe how to achieve the promise.

This lack of instructions is not a shortcoming of the Agile Manifesto. Its purpose was to create a vision for a better way to develop software. It is instructive to look at the Manifesto in its historical context. During the decades preceding the Manifesto, the principles of and approaches to software management swung like a pendulum, between free-form and command-and-control, from little process to too much process. Each was responding to the challenges of the other.

In the 1960s, several large system failures demonstrated the need for both better engineering methods and better processes. Certainly, software development during this time was not an ad hoc affair, but the industry was new and there was little experience with large-scale systems. In the 1970s, the idea of software as "engineering" surfaced. We began to use structured analysis and design, top-down programming, structured programming (`goto` statements were considered bad form) and, notably, the Waterfall methodology emerged. The industry was growing up and standard practices for design, programming, and management arose. By the

[4] Apologies to military experts who properly use this term to mean vision at the top with implementation at the bottom. We're using this term in the way most people interpret it – top-level people telling lower-level people how to get their job done.

1980s, PCs and fourth-generation languages enabled small projects to flourish. Small teams produced much more software than large teams did.

Prototyping was popular. Speed was king. If you were the first, you were the best.

But quality often suffered. Speed to entry was so important that a product's sustainability was often ignored. This led to different kinds of failures. Since it was easy for anyone to enter the market, the competitive edge of getting in first was lost if the product lacked quality. Failures in this era triggered an upsurge in rigorous process. The sense was that if we can't do it ad hoc, then we'd better control it.

Tick tock. Tick tock. The pendulum continued to swing. Maybe even faster.

The 1990s brought us the Capability Maturity Model (CMM). Y2K dominated the last few years of the decade, emphasizing the need for planning ahead. But the '90s also brought us the Internet, which again enabled small teams to have great impact. The dot-com boom brought rapid software development. Again, a proliferation of small teams found initial success but subsequently had difficulty maintaining the software that they had developed.

Now, the twenty-first century has given rise to Agility—small teams working with customers to develop software quickly. There have been many successes and there have been many failures.

And the pendulum continues to swing. What can we do to stop it? Or can we at least find a balance?

The Agile Manifesto was an attempt to find such balance. Let's respect our teams. Let's respect our customers. Let's work with the business. Process can be good, but process that doesn't help a team get its job done is not.

Unfortunately, the world is messy and the promise of the Manifesto has not been entirely realized. The Manifesto itself showed the potential, but it did not provide a means to stop the pendulum. In fact, it has been used to justify letting teams rule. We have mostly lost the enterprise view, because that view seems to lead right back down the path of command-and-control management. If the choice is between that and using teams with Agility, then abandoning command-and-control seems reasonable.

It is not either-or. There is a way to balance command-and-control with the need for effective teams. Lean provides the way. To see why, we must first examine the beliefs, principles, and paradigms on which we build our thinking.

Principles and Paradigms

A principle is a comprehensive and fundamental law, doctrine, or assumption. Principles may exist at the level of the individual, may be held by a community, or may even apply universally. For example, individual principles may relate to one's integrity or one's way of living. A

communal set of principles might include moral or religious beliefs or a set of beliefs that the community accepts as the true way to be living. Universal principles are those that apply everywhere—beyond the effect of the beliefs of any set of individuals. Perhaps we should actually call these laws of the universe. Principles are often stated in the form of guidance since there is often a corresponding principle (law) that should be followed. For example, one of the best known Lean principles is "eliminate waste." That's not a law, as much as it is something you should do, as a rule.

A paradigm is a combination of assumptions, values, beliefs, and practices that define how to view reality, how to look at a situation. It is a worldview that characterizes what is true. Paradigms tend to last a long time (consider how long people believed the earth was the center of the universe). Paradigms are shared by a particular community or group of people. In the software world, Waterfall, Scrum, and Lean-Agile each have their own paradigms, or way of looking at how to best build software.

Since a paradigm defines what is real and true for someone, changing one's paradigm is quite difficult. It requires the individuals and their community to grapple with the underlying assumptions, values, and beliefs and assess whether the paradigm actually squares with what is indeed "real" or whether some shift is required.

The paradigms we hold constrain what we consider possible and shape what we do. Unexamined paradigms can therefore be very limiting.

A Pragmatic Approach

Software development professionals are pragmatists (pragmatism is part of our worldview). We favor what works over what is represented as theoretically "correct." It is not that theory is bad, but theory must be grounded in real work if we are going to embrace it.

With that in mind, we would like to suggest taking a pragmatic approach to evaluating the essential paradigms we, as software developers, hold. That approach is to use the scientific method in whatever we do: Propose a hypothesis and then run an experiment to validate or invalidate it. If the experiment supports the hypothesis, then we have some evidence that the hypothesis is correct. If it doesn't, then the hypothesis is incorrect and must be modified.

We suggest that in the software development world, our processes must be consistent with our hypotheses about the best way to practice software development. If we get good results, we have evidence that our process (that is, our hypothesis) is good. If we get poor results, our process needs updating.

Critique the process; work together

Let's be clear: This is all about critiquing the process, *not* the people involved. How many teams have run into problems because they are following a poor process and yet management, being

overly committed to the process, blames the people? Assuming the process is right, they believe "if only the people had done it right, it wouldn't have been such a disaster."

Or how many projects have failed because teams decide to follow their own approaches regardless of the larger needs of the business? They assume management is just getting in the way—bureaucrats who must be worked around.

It seems that the tendency is for management to over-focus on process while teams underestimate its value. One side sees management as crucial to making the process work; the other wants to be protected from management's command-and-control mentality so that they can just get their work done. And so they go back and forth, not working in concert.

What we need is a new attitude about process and how to manage process. Processes must be designed to assist the team in achieving management's goals. Processes help the team get its job done: They represent accountability among team members about how they will work. The team is the steward of its processes—creating, sustaining, and improving them so that the team can improve constantly. Processes are dynamic: They are the team's baseline for change.

Lean provides the way forward

Is this possible? Yes! Lean provides the principles we need to do this. And we will not follow these principles blindly. Blind faith doesn't work. Instead, we will use Lean as a guide and use our own experience to refine our own process.

If you have been building software for a few years, we invite you to use the hypothesis-and-test approach yourself: Run "backward-looking experiments," [5] that look back over your own past experiences to validate or invalidate the process we are developing. This is a lot more pragmatic and a lot less painful than trying new processes on future projects. You will be able to verify relatively quickly whether the process works.

As we do this, we will be building a pragmatic "theory" about why and how software development works. We recognize the truth in Jan L.A. van de Snepscheut's and/or Yogi Berra's comment "in theory, theory and practice are the same, but in practice, they are different." We also believe Kurt Lewin's notion that sometimes "there is nothing more practical than a good theory." In other words, do not follow theory when it does not match practice. But when you are not sure what to do, an understanding of why your practices work may give you guidance in unfamiliar situations.

[5] A backward looking experiment is a term Alan Shalloway coined to mean looking into your past to validate or invalidate an hypothesis made in the present. For example, if someone says "coding conventions help" he is actually postulating that coding conventions result in better code. You can actually look into your past to see when that was true (adding evidence to the hypothesis) or when it was false (disproving the hypothesis). If you disprove the hypothesis, you can modify it with a condition to see if there is a set of circumstances that would make it true. This enables us to learn about and test our understanding by taking advantage of our past experience.

This pragmatic approach embraces the principles (or laws) that we have discovered work in all situations. For example, the principle that overloading an individual with work, that is, giving her many tasks to do at the same time, will degrade her performance.

Principles lead to many practices. However, practices must change depending upon the context, or situation, in which they are being used. Relying totally on principles may not work unless the principles are proven. Relying totally on practices will work only if you are in situations you've been in before. Effectiveness requires a proper blend of proven principles with practices appropriate for the situation in which they are being used.

Evaluating Paradigms

As we begin this approach, let's look at some of the core beliefs upon which the Waterfall method and the Agile framework are based. These are described in Figures I.3 and I.4. Are these universal principles? Or are they unexamined paradigms—rules that just must be followed?

We believe that the core beliefs of Agile are more helpful than the core beliefs of Waterfall. They are helpful, but they are not enough. To follow them effectively, more is required. That is where Lean comes in.

The core beliefs of Waterfall include:

- You can know everything required to build a software product properly at the start of the project.
- Customers can accurately tell you what they want at the start of the project.
- You don't need to get feedback from the customer until the end of the project.
- Managers, developers, and customers can gauge the status of a project by looking at completed milestones as reflected in documentation. That is, given proper documentation, it is not necessary to deliver complete, tested software until the very end of the project.
- You can effectively have separate groups do analysis, design, code and test. That is, there is little loss of information in the hand-off between these groups.
- Hand-offs between people in different roles can be done efficiently by writing down what was done in each step.
- You can test at the end of a project and achieve the required quality.
- Management can demand that certain work be done at a certain time and should expect to get them.
- Giving people many projects to work on simultaneously is a good approach to achieving 100% productivity because they are always busy.

Figure I.3 Core beliefs of Waterfall

The core beliefs of Agile include:

- You cannot know everything required to build a software product at the start of the project.
- Customers cannot accurately tell you what they want at the start of the project; instead, they will gain clarity as the project proceeds.
- You want feedback from the customer as often as possible and you want to give

developers feedback on how they are doing as soon as possible.
- Working code is the most accurate way of seeing the progress of the development effort.
- Groups working together minimizes delays as well as the loss of information between people.
- Moving test to the front of the development cycle improves the conversation between developers and customers and testers and, thus, improves the quality of the code.
- While management can set expectations for what work is done, management must not demand how that work is done.
- Working on one project at a time improves the productivity of a team.

Figure I.4 Core beliefs of Agile

We Do Not Know It All

Although software development is not exactly like other types of product development, we can still learn a lot from how other industries approach product development. In particular, Lean gives us a lot of information, based on decades of experience, that can be particularly useful to Agile teams. In fact, Scrum, one of the more popular Agile methods, is based on Lean principles. Unfortunately, an understanding of Lean is not widespread in the software community. It is unfortunate because teams lose out on the potential guidance that Lean offers. Moreover, without a grounding in Lean, software developers often lack the basis for explaining to management why certain practices would be useful. Lean provides a new set of beliefs, shown in Figure I.5. The question still remains: Even if these beliefs are true, how do we manifest good practices that are consistent with them?

Of course, merely believing something doesn't make it so. It is worth looking at the beliefs presented here and then deciding which ones represent actual principles. We suggest using backward-looking experiments for this.

The core beliefs of Lean include:
- Most errors are due to the system within which people work rather than to the individuals themselves.
- People doing the work are the best ones to understand how to improve the system.
- Ad hoc is not an acceptable process.
- Looking at when things are done in a process is a more useful guide than trying to make sure every step of the way is as efficient as possible.
- Our measure for success must be related to the amount of time between when ideas come in and when they are manifested as value to our customers.
- Management must work with the team to improve the way it works to improve its own efficiency.
- Teams are most efficient when the amount of work is limited to their capacity.
- Team efficiency improves by minimizing the amount of work in progress at any one time.
- When evaluating actions, we must optimize the whole, not merely improve individual steps in the process.
- There are principles in software development that must be followed in order to reduce

waste.

Figure I.5 Core beliefs of Lean

Lean Provides More than Beliefs

Fortunately, Lean provides more than a paradigm and a belief system. It provides a set of principles in its own right as well as many practices based on them. While these practices cannot usually be taken straight from Lean (since practices must change depending upon the context in which they are used), Lean principles and practices can be readily adapted to software development. By learning these principles and practices, one can manifest the intention of the Agile Manifesto—developing software effectively. And we can do it at both levels—enterprise and team.

We will see that Lean provides a paradigm of management in which managers are not encouraged to command and control teams and developers are not required to insist they are craftsmen who cannot and should not be managed. Rather, Lean provides a paradigm under which managers and developers can work together toward a common goal—providing the best return on software development efforts. Lean provides such a paradigm through its focus on the process by which the team works—but a process that must be the best one for the team to get its job done. Process is no longer something imposed on the team, but rather something owned by the team to make its work more productive as well as more enjoyable.

Lean combines this management paradigm with concepts, tools, and practices that give both sides a way to work together and improve visibility to management, direction from management, and team productivity.

Going Beyond Lean

Of course, Lean is not all there is from which to pull. But our experience is that it is consistent with other useful paradigms, beliefs, and principles that come from other disciplines. For example, we learn from the building-architecture discipline and the software-design-patterns community that we should develop products by starting with the big picture. That is, don't try to create a product by building it from small pieces. Keep the big picture in mind. This, unfortunately, is a lesson many Agile practitioners and consultants have long ignored (probably due to Agile's heritage, which sprang up on smaller projects).

In this book, we incorporate what may be non-Lean practices but they are otherwise consistent with a central principle of Lean: "Optimize the whole." In particular, we'll see this in these areas:

An enterprise focus instead of a team focus both for product-portfolio management and for team coordination, thereby providing a working alternative to Scrum-of-Scrums.

A product focus instead of a project focus (where projects are enhancements to products).

Managing requirement elicitation from the big picture instead of starting with stories and combining them into epics and themes.

Driving release planning from business value instead of trying to manage the effective release of a collection of stories.

Summary

This introduction explored the roots of agility: starting with the Agile manifesto and its principles and its historical context in the swing between management command-and-control and development teams wanting to apply their local knowledge to get work done. What is needed is a proper understanding of process as both/and: both as a tool for management and a responsibility of the team to steward what it knows.

Getting to this better understanding involves examining core paradigms, principles, and practices that everyone in software development holds. Lean-Agile offers a thinking practice to help form a better way of understanding. It is based on the solid foundation of Lean thinking and is entirely consistent with Agile practices.

Try This

These exercises are best done as a conversation with someone in your organization. After each exercise, ask each other if there are any actions either of you can take to improve your situation.

- Look at the beliefs of Waterfall listed in Figure I.3. Which of these are true?
- Look at the beliefs of Agile listed in Figure I.4. Which of these are true?
- Look at the beliefs of Lean thinking listed in Figure I.5. Which of these are true?

Business-Driven Software Development™

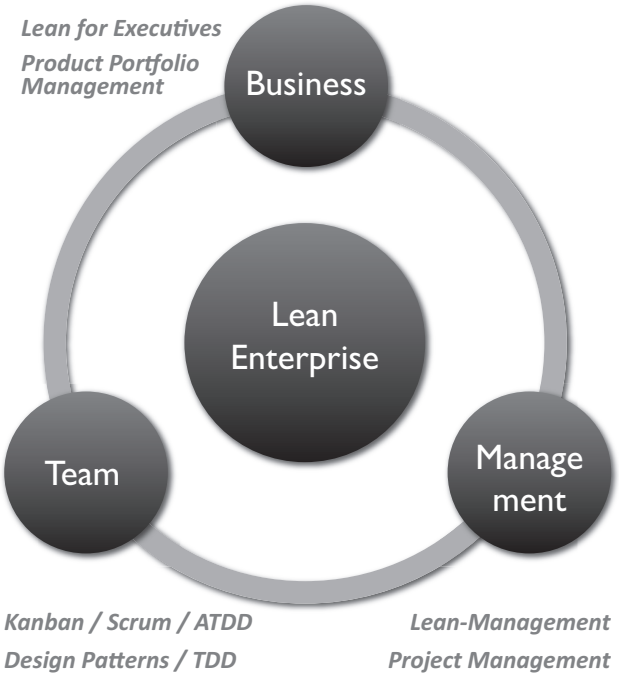
Assessments, Consulting, Training, and Coaching

Business-Driven Software Development (BDSD) is Net Objectives' proprietary integration of Lean-Thinking with Agile methods across the business, management and development teams to maximize the value delivered from a software development organization. BDSD has built a reputation and track record of delivering higher quality products faster and with lower cost than other methods

BDSD goes beyond the first generation of Agile methods such as Scrum and XP by viewing the entire value stream of development. Lean-Thinking enables product portfolio management, release planning and critical metrics to create a top-down vision while still promoting a bottom-up implementation.

BDSD integrates business, management and teams. Popular Agile methods, such as Scrum, tend to isolate teams from the business side and seem to have forgotten management's role altogether. These are critical aspects of all successful organizations. In BDSD:

- **Business** provides the vision and direction; properly selecting, sizing and prioritizing those products and enhancements that will maximize your investment
- **Teams** self-organize and do the work; consistently delivering value quickly while reducing the risk of developing what is not needed
- **Management** bridges the two; providing the right environment for successful development by creating an organizational structure that removes impediments to the production of value. This increases productivity, lowers cost and improves quality



Become a Lean-Agile Enterprise

All levels of your organization will experience impacts and require change management. We help prepare executive, mid-management and the front-line with the competencies required to successfully change the culture to a Lean-Agile enterprise.

Prioritization is only half the problem. Learn how to both prioritize and size your initiatives to enable your teams to implement them quickly.

Learn to come from business need not just system capability. There is a disconnect between the business side and development side in many organizations. Learn how BDSD can bridge this gap by providing the practices for managing the flow of work.

Why Net Objectives

While many organizations are having success with Agile methods, many more are not. Much of this is due to organizations either starting in the wrong place (e.g., the team when that is not the main problem) or using the wrong method (e.g., Scrum, just because it is popular). Net Objectives is experienced in all of the Agile team methods (Scrum, XP, Kanban, Scrumban) and integrates business, management and teams. This lets us help you select the right method for you.

Assessments

See where you are, where you want to go, and how to get there.

Business and Management Training

Lean Software Development

Product Portfolio Management

Enterprise Release Planning

Productive Lean-Agile Team Training

Team training in Kanban, Scrum

Technical Training in ATDD, TDD, Design Patterns

Roles Training

Lean-Agile Project Manager

Product Owner

For more information about Business Driven Software Development Services
Contact Mike Shalloway at Net Objectives: mike.shalloway@netobjectives.com / 404-593-8375
NetObjectives.com / 1-888-LEAN-244 (888-532-6244)