

# CHAPTER 2

---

## The Business Case for Agility

*“The battle is not always to the strongest, nor the race to the swiftest, but that’s the way to bet ‘em!”* —C. Morgan Cofer

---

---

### IN THIS CHAPTER

This chapter discusses the business case for Agility, presenting six benefits for teams and the enterprise. It also describes a financial model that shows why incremental development works.

#### Takeaways

Agility is not just about the team. There are product-management, project-management, and technical issues beyond the team's control. Lean-Agile provides benefits for all, such as:

- Getting customers to tell you everything they are ever going to need from you with any sort of accuracy; they tend toward speculation, much of which is inevitably wrong.
- Emerging designs that are change-tolerant, which addresses the need for software development to get a return on investment as quickly as possible in order to respond to changing market conditions as competitors advance and adjust their own products.
- Delivering value incrementally provides much more business value to your customers.
- Lean-Agile methods provide management greater visibility into software-development projects than Waterfall methods do.

Lean-Agile addresses all of these points. However, it requires the team to pay attention to a few things that *are* in its control:

- Delays and thrashing, which are impediments
- Resource- and product planning
- Code quality, coding standards, and planning for future modifications

## The Benefits of Agile

Despite the buzz about Agility, it is important to keep in mind that the primary driver is, and must always be, the benefit to the enterprise via adding value to its customers. Helping teams become more Agile is good. It is a necessary part of the journey toward helping the enterprise become more Agile. Enterprise Agility enables an organization to react to market needs and to competitive changes and helps it realize the most value from the resources at hand. In the end, using Agile methods must improve the enterprise's bottom line. If it does not, then we need to question why we are using Agile or if we are using it correctly.

Agile can benefit an enterprise and its teams in the following ways:

- Add value to the business quickly
- Help clarify customers' needs
- Promote knowledge-based product development and better project management
- Motivate teams and allow failing (that is, learning) early
- Focus on product-centered development
- Improve team efficiency

Each of these is important and is discussed below individually. However, there is a synergistic effect as well. Growth in one area spurs growth in other areas, resulting in greater capacity to create value.

### ***Add Value to the Business Quickly***

Whether you are in an IT shop that produces software for internal customers or in a company that produces software products for external customers, there is tremendous value when products can be released to the customer quickly<sup>1</sup>. The benefits to the business include the following:

- **Revenue/Return on Investment** The sooner customers can begin using a product, the sooner the business begins to receive a return on its investment—either in direct revenue or as a way to satisfy business needs.
- **Increased Customer Satisfaction** All things being equal, customers prefer to get new or improved features sooner rather than later so that they can begin to use them.
- **Market Position** The best way to maintain or increase market position relative to the competition is to deliver valuable features sooner than they can. Products that look fresher and more capable satisfy customers more deeply, which creates credibility for the business and builds customer loyalty. In today's world, competition often comes from smaller, more nimble (as in Agile) companies, and so releasing quickly is more important than ever.
- **Lower Risk** Releasing quickly shortens the feedback loop with the customer, enabling you to discover mistakes in your project earlier. It gives you an opportunity to fix the product—or abandon it—in order to minimize your losses.
- **Greater Profit Margins** Investment costs are recouped sooner on products that are released more quickly and are further mitigated by the incremental delivery of smaller releases. Additionally, it is often possible to charge more for your products if you are considered a market leader.

### **A Financial Model for Software**

Suppose you are responding to a Request for Proposal (RFP) which spells out the following:

- Scope
- Due date

---

1. Merely releasing a product or software supporting a service adds no value if the customers of the organization can't actually use the product or service of the company.

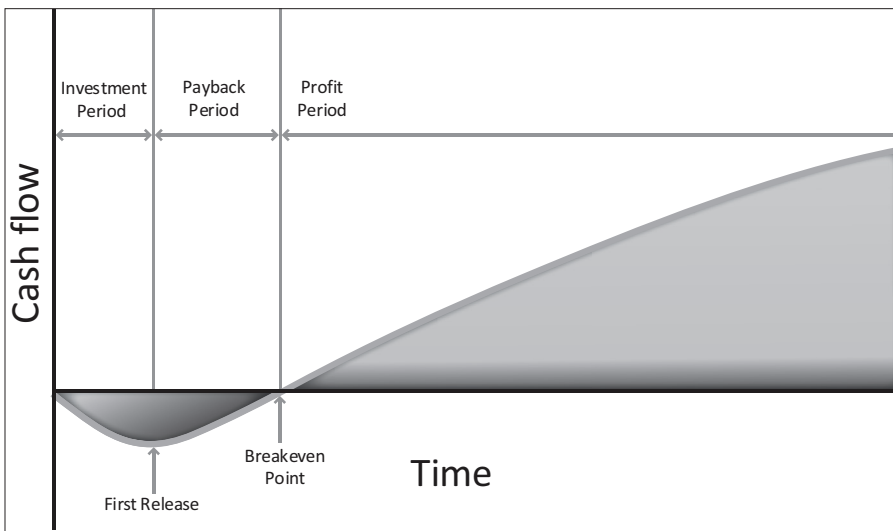
- Quality
- Cost

The RFP says that none of these four can be modified. That is, you cannot charge less for the product or give better quality without being disqualified. Where can you compete? Credibility may come to mind, but that might be difficult to establish. What else could there be?

Mark Denne and Jane Cleland-Huang offer valuable guidance here in their brilliant book, *Software by Numbers* (Denne & Cleland-Huang 2003). The following is based on their analysis.

Figure 2.1 shows the cost and return for a typical, successful software project. In this graph, the “Investment Period” represents the time spent creating the software. This is when work is being done before money can be made. The Payback Period begins when the customers begin to realize value from—and start paying for—the product.

What would this curve look like if you could release the same product in two phases? Each phase of the system would contain half the features required. Suppose the team can build and release the first phase in half the time period and the second phase during the remainder of the time period. Each of these releases would have a financial curve similar to the one shown in Figure 2.1, except that they would occur at different times. For example, on a 10-month project with 100 features, you might release

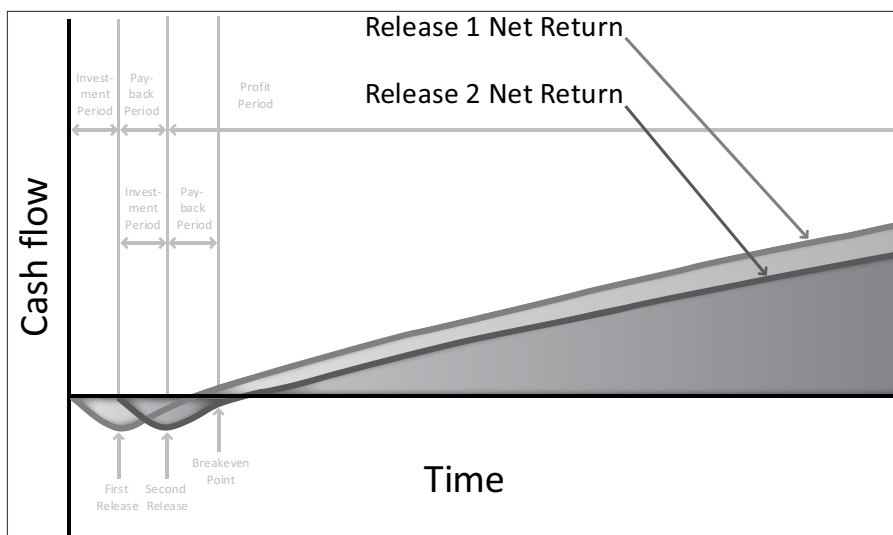


**Figure 2.1** A successful application development project

features 1–50 after five months and features 51–100 after another 5 months. Figure 2.2 shows these two curves overlaid on each other.

Each half contributes some value and some revenue. If you are a product company, you are getting revenue for half the system; if you are an IT organization, you are providing new value to your internal customer. It is often the case that the first half of the system provides more than half the value (remember the Pareto rule, “80 percent of the value comes from 20 percent of the work”). However, it is also possible that no value will be returned until all of the system is in place (for example, you cannot release an airplane guidance-control system that only partially works). Let’s suppose we have the former case, in which half the system provides half the value. Figure 2.3 shows the total profit generated from these two phases.

Figure 2.4 compares two release strategies: a single-release strategy, in which all of the features are delivered in one release package, and a strategy of staged (iterative) releases, in which features are delivered as they are ready. With the single-release strategy, the business must wait significantly longer to realize revenue than with the staged-release strategy. The amount of money that the business must fund on its own, before money begins to come in, is significantly more, the breakeven point is significantly extended, and the ultimate profit is significantly less. The business realizes less overall value and so does the customer. Too much delay may actually prevent the business from entering or extending the market altogether.



**Figure 2.2** Developing a system in two halves

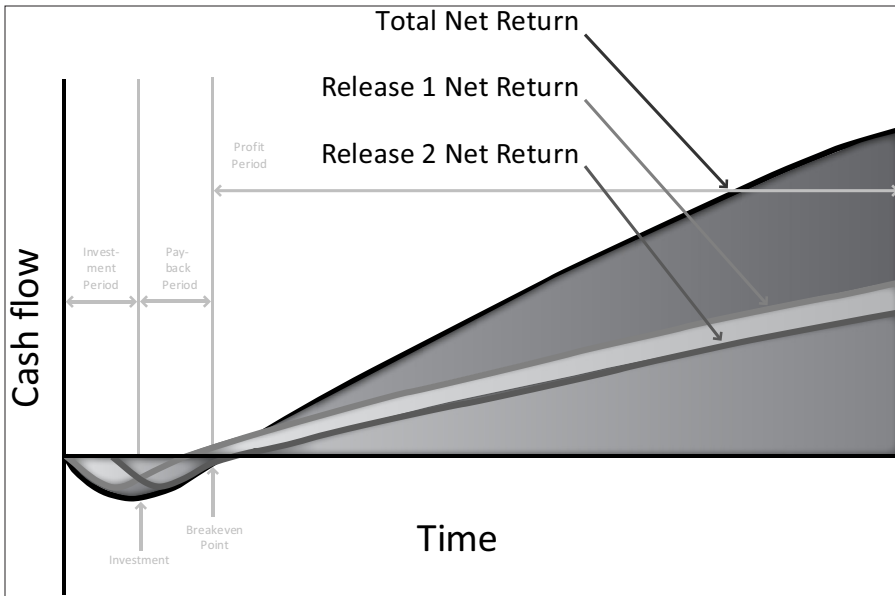


Figure 2.3 The net return on a successful software project built in stages

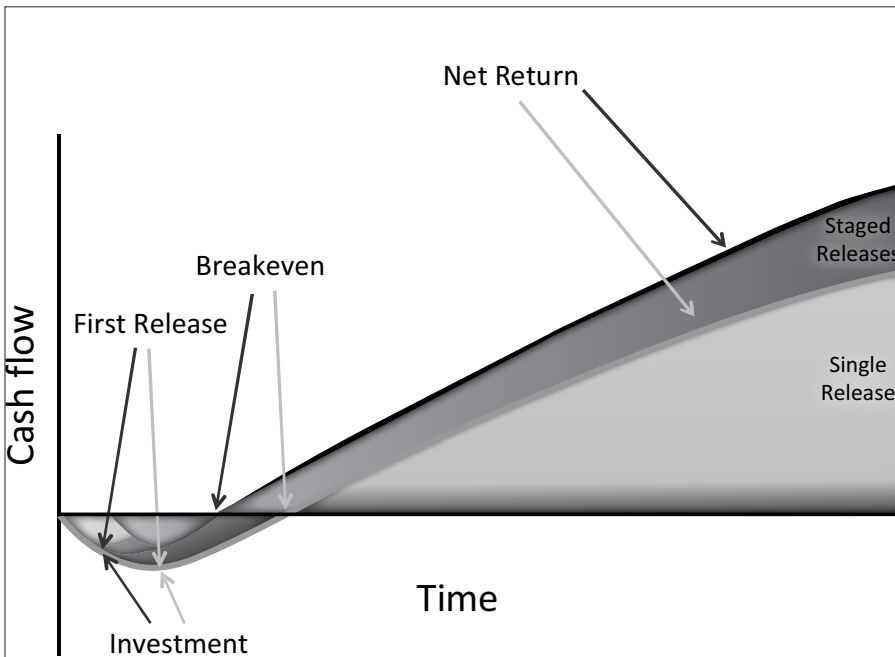


Figure 2.4 Comparing profit and breakeven points for release strategies

Releasing features sooner means that everyone gets value sooner: profit, market penetration, and utility. Additionally, it helps clarify what the ultimate product feature set *should* be: As customers understand more about the product, they can speculate less about what they think they need.

Of course, there are other considerations such as delivery and installation costs; but even so, we have understated the case. For example, in many large projects half of the requested features often represent 80 percent of the entire project's value. In that case, the first curve would be sloped higher than we have shown.

Although not all software can be delivered in stages, very often it can be—particularly in IT organizations. How do we determine which aspects of a product can be delivered quickly? Denne and Cleland-Huang (2003) suggest identifying “minimum marketable features” (MMFs). A minimum marketable feature is the smallest amount of functionality that makes sense to market.

This can be taken further: Plan which features to deliver first so that the business and the customer get the most value as soon as possible. A relentless focus on adding value is at the very core of Lean thinking.

### ***Help Clarify Customers' Needs***

Frequently, developers complain that customers do not really know what they want. They change their minds; they cannot specify what the software should do; their requirements are imprecise and incomplete.

It is easy to think that if this is true, it must be a customer issue. But dig deeper. The developer team is also involved. Consider the following case study.

---

#### **Case Study: Building Components**

Alan was working on a system that involved four components: scheduling, accounting, resource management, and content delivery. The intended development approach, as shown in Figure 2.5, was to build and release each component in turn until the product was completed. From the perspective of contributing value, delivering with four iterations would be good because customers would get the components sooner.

Unfortunately, it was clear that the customers did not really understand the fairly complex scheduling aspect of the product. At first, Alan thought that if each component could be delivered in stages, perhaps the customers would be required to know less up front and would be able to build on what they did know. So he asked the team, “If we build *half* of the scheduling component, would that provide value?” This would require the customer to know only

### Initial development sequence



**Figure 2.5** Time frame for initial development sequence

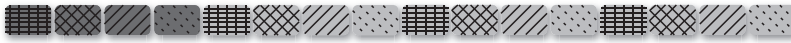
### Iterative development sequence



Build and deliver the first half of all features. Then go back and build the rest of system.

**Figure 2.6** Building the components in stages

### Suggested development sequence



**Figure 2.7** Building components 25 percent at a time and cycling through them until the whole project is done

half as much. We would deliver half of each component, release it, go to the next one, and release that until the first half of each component was done. We would then repeat the process to add the features we hadn't built yet. The thinking was that after the customers had used each component for a while they would know how to describe the rest of the features. This is shown in Figure 2.6.

The team's response was disappointing. "Yes, we could deliver half and find value, but our customers don't even know *half* of what is needed."<sup>2</sup>

This was when he asked a simple question that resulted in an epiphany for him. "How much do customers *really* know?" How often do we developers have the attitude that customers really aren't sure about *anything*? The truth is that they do have certainty about *some* things, even if it isn't everything. It seems reasonable that they usually understand about 20 to 25 percent of their problem domain.

This leads to an even more useful question, "What *part* of the system do they know?" Do they know the core stuff or do they know the fancy, extra stuff? In fact, customers typically know more about the core stuff, the things that led to the project and product enhancements in the first place. In other words, the customers know what we can start with.

The key insight is this: Start with what customers know.

Alan asked the team if building and delivering the components in pieces representing only 25 percent of the system would be valuable. They responded, "Yes." This evolved into the plan shown in Figure 2.7.

---

2. We are not suggesting that you can always deliver in stages; that is why we had to ask the team. But you should always look for the opportunity to deliver value in stages when you can.

This has become a fundamental approach for us:

Start building the system based on what the customer is clear about, even if it is only 20 percent of what is required. Build a little, show them, gain more clarity, and repeat. That is the best way to address constantly changing requirements.

When customers are changing their minds frequently, it is probably because developers are asking them to speculate and try to be clear about things that they do not yet know. What happens? Anything built under those circumstances will, of course, be inadequate. Note that the word “speculate” is never used—but the pressure for more information causes the customers to do just that.

The reality is that customers usually do know some of what they want, at least the features that are most important to them right at the moment. What many cannot do well is speculate about what is unknown. (“This is a six-month project, so tell me what you will need six months from now”). Speculation is risky.

To mitigate the risk of uncertainty, Agile prescribes short iterations with lots of feedback. As customers learn more about what the system looks like and what it can do, they get clearer ideas about what they really need and can ask for it more precisely. The design emerges with experience over time.

Lean product development takes this further with its focus on delivering value. It guides teams to concentrate on what their customers have the greatest clarity on: the core functionality of the product. Core functionality is closely tied to the business process, to what they are already doing, even if it is not automated or is poorly done. Customers can be very specific about these features. What they are not clear about—and so what they have to speculate about—involves user interface, esoteric features, and other cool bells and whistles. Often, these offer the least real value to the customer but are the most seductive—both to the customer and to the developer—and thus are the most likely to get them off track.

The Lean principle here is to create knowledge all the time. Focus on delivering value! Start with what the customer is clear on, on core processes. Let them gain experience and knowledge and then specify the next bit.

This is a better approach to product planning than trying to specify every requirement at the beginning. Even if it is not possible to release in stages, it can still be useful to build in stages because doing so

concentrates first on that part of the system that the customer is clear about; it develops customer clarity. We do customers a favor by not requiring them to speculate, by focusing on what they are comfortable with. And it does developers a favor by reducing risk, making it less likely that they will build unnecessary features. And not building unneeded features means less wasted time and, even more importantly, a less complex system.

The point is to involve the business and the customer in planning for quicker delivery in multiple iterations, adding value in stages. This is Lean thinking.

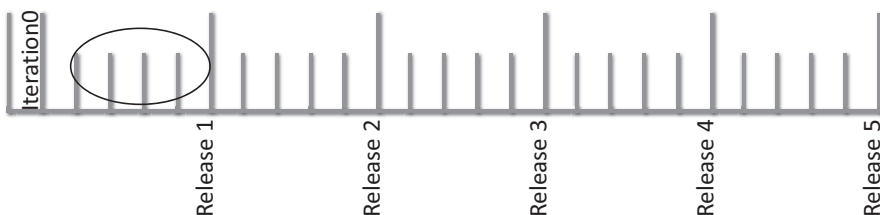
### ***Promote Knowledge-Based Product Development and Better Project Management***

The most obvious benefit of Lean-Agile product development is that you discover how you are doing very early in the project. In non-Agile projects, you don't really know about the quality of your code until you enter the test phase of the project, which is usually late in the schedule if not at the end itself. You also don't know how customers will react to your product until the first beta testers see it. This is a bit late in the cycle. In many ways, the feedback you get on a project at this point, which is illustrated in Figure 2.8, is really only useful for setting up the next release.

#### Waterfall Project



#### Iterative Development



**Figure 2.8** Waterfall versus Agile: When you get a real indication of where you are

This feedback covers both the pace at which you are working and whether or not you are building what the customer wants. In a Waterfall project, although customers may see the system while it is being built, it is often difficult to incorporate their feedback into the system. Agile processes are geared toward enabling the developer to utilize the feedback rapidly, as soon as it is given.

Most of us have been on projects that were doing fine after six months but then were two months behind after seven months without being given any new requirements. Unless you are not using source control, this actually isn't possible (you could back up to where you were one month ago and be just one month behind schedule). The reality is that you were not on schedule at the six-month mark but did not know it. Now you do, but it's a bit late in the project to do anything about it.

Imagine a ten-month Agile project in which you discover after a couple of months that you are behind schedule. You have many choices at this early point—add more resources, remove scope, reset the end date, even cancel the project. When we learn late in the game that a project is in trouble, it affords little opportunity for management to adjust. We talk about “death marches” at the end of many projects. At that point, management is often angry and upset. And where there is anger, there is usually fear. Fear runs rampant in many non-Agile projects when management senses that the team has lost control of the job. That can happen when a team relies on too many predictions and assumptions that do not bear themselves out.

What is different about Lean-Agile projects is the way they are planned, the role of testing, and the sense of urgency about them. These present new challenges for the project manager, who must facilitate communication between teams and customers, manage sustainability and a constant sense of urgency, maintain visibility, and communicate the true pace of progress to all concerned. But these are relatively positive challenges to have.

### **Short Planning Horizons**

One of the myths of Agile is that you do not plan. In fact, you are constantly planning—just in smaller increments and with more people than you are probably used to. You plan for releases. You plan before each iteration. You plan in the daily standup meetings. Maybe 15 percent of the team's time involves some sort of planning. The difference is that, at the project-level, you plan for what is known: two to four weeks out, the

next iteration, and the work we will do today. Planning for what is nearly certain is a whole lot easier than planning for what you don't know. This is what makes Lean-Agile projects easier to plan.

### **Testing to Improve the Process as Well as the Quality**

A key Lean principle is to aim for perfection, to improve constantly. In Lean-Agile projects, this strongly drives the way testing is done. Testing occurs very early in the development process in order to shorten the cycle-time between finding a defect and fixing it. This makes it likely that fairly well perfected code is delivered at the end of an iteration. It also makes it more likely that any defects that are found will not have serious consequences.

Perfection means that testing is not just for finding bugs. Even more, the testing activity should discover the *causes* of errors and eliminate them. Root-cause analysis is part of testing's portfolio of work. This approach to testing is different from the usual approach, in which bug elimination is done by developing rigorous specifications and then testing in a separate phase, usually after development is completed. Removing the delay between development and testing is what makes risk management easier in Lean-Agile projects. Many errors stem simply from misunderstanding customer requirements. Creating acceptance tests up front (at least defining them, if not implementing them) changes the conversations between customers, developers, and testers. Discussing things at the implementation level as well as the requirements level improves the conversation, which assists developers in understanding what customers really mean and need.

### **Planning without Fear**

Lean-Agile projects always have a sense of urgency about them. Short deadlines and close contact with the customer tend to ensure this. However, there is rarely a sense of panic or fear. Even if there is a setback, the project never feels out of control, and that helps prevent management from getting angry and afraid about the project. They get a sense early on whether the project is on track or at risk and they can adjust to it; they don't get surprised by a lack of progress at the end of a long project (and surprising management is always a bad thing). The stress level among Agile development teams seems moderate and healthy. Contrast this with more traditional approaches, where the workload at the front end is rel-

atively low and then ramps up to crises and all-consuming 80+-hour weeks at the end; and then when there are setbacks, the consequences feel dire and the pressure builds. Such a pace and environment cannot be sustained or be healthy for long. Lean-Agile projects tend to feel more in control and to be more sustainable over time.

### ***Motivate Teams and Allow Failing (Learning) Early***

The feedback that customers get with Lean-Agile helps them clarify their needs and instills confidence that their needs will be met. This feedback helps developers, too.

By and large, developers enjoy working with the business and with customers when there is a sense of common purpose and teamwork. They like doing work that positively impacts users of their product; they have a passion to help. Developing short iterations and working closely with customers results in lots of good feedback and lots of quick wins. This is a great motivator for development teams.

In this process, both customers and developers discover very quickly what is not working<sup>3</sup>, what assumptions are invalid, and what they do not know. That information, too, is a good motivator. It may not make them happy but at least it presents them with challenges to overcome while they are still in a position to react positively. Nothing feels worse than discovering a major flaw late in the cycle.

By getting early feedback on how the team's methods are working, developers can make changes quickly and avoid waste. Sometimes this feedback has more to do with the progress of the project than with its process. If it is slower than desired, management can choose to add more resources, de-scope the project, push out the end date, or even cancel the project early in the cycle. With wise project management and Lean thinking, this should be a motivator: Stopping early on the wrong project or on a project that is more costly than feasible is in everyone's interest. It prevents the agony of working on a project for six months or a year only to find out that there is no business support for it and that it is going to be cancelled. That feels just terrible, especially if the team was doing what it was asked to do and was doing it well. Ultimately, it is better to work on projects that have a solid business tie-in.

---

3. In the Agile community, they say that they want to "fail fast" but we prefer to say that we want to "learn fast." That may involve failing, but it may not. It is better not to fail if you don't have to! But if you are failing, learn fast . . . and correct quickly.

### ***Focus on Product-Centered Development***

Suppose you are working on a project for which the following are true:

- There is no benefit from a partial release of the product; the entire product must be finished before it can be released. (This is not uncommon.)
- The customer has perfect clarity about what is needed. (This almost never happens; even with rewrites, the customer usually has better ideas the second time around.)
- Management trusts the team to produce the result that is needed; therefore, the project does not have to be controlled through iterations. (This happens on occasion.)
- The team perfectly understands Agile. There is no learning curve.

Would you still want to use Agile and an iterative approach? Yes!

Why? Because building software in stages enables you to take advantage of what you have learned as you build it. All developers know much more at the end of a project than they do at the beginning. When you design everything up front, there is a tendency to overdo the architecture, to include more than you need. This adds complexity that the team has to deal with for the life of the project.

Lean gives insight into how to manage unfolding requirements and design, as described in detail in *Emergent Design: The Evolutionary Nature of Professional Software Development* (Bain 2008).

### ***Improve Team Efficiency***

A problem many teams face is that key people are busy with several projects. This degrades their efficiency not only because they are pulled in multiple directions and must continually switch tasks, but because it creates delays for everyone else who needs their assistance. Large projects tend to have more people working on several projects at one time, which lowers the teams' efficiencies.

Lean helps us minimize the time it takes to complete prioritized business and customer solutions. For most legacy organizations, this means they should organize around sequencing and completing smaller, high-value requests before starting new ones. In these organizations, the drive to keep people busy (to maximize efficiency) seems to motivate professionals to start more and more projects without concern for the conse-

quences. The Lean approach is to finish work and deliver it to the end user before starting something new. In simple terms: Stop allowing queues of unfinished work to build up. This tends to reduce contention for critical resources because projects go through the development pipeline faster; this alone can generate huge efficiency improvements.

## Summary

This chapter describes six benefits of using Agile for software development:

- It adds value to the business quickly.
- It helps clarify customers' needs.
- It focuses on knowledge-based product development, which leads to better project management.
- It motivates teams and allows "failing early."
- It promotes product-centered development.
- It improves team efficiency.

It also offers a financial model that describes the benefits of realizing the value of products incrementally rather than in final releases. In addition to financial benefits, everyone on the team—customers and developers—can learn more quickly and incorporate that learning back into the product and process while there is still time to do it.

## 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.

- How involved are customers and stakeholders in day-to-day product development?
- What are impediments to frequent feedback from users of products in development?

- How would smaller, more frequent deliveries of verifiable components help improve the ability of customers and stakeholders to provide feedback?

## Recommended Reading

The following works offer helpful insights into the topics of this chapter.

Bain. 2008. *Emergent Design: The Evolutionary Nature of Professional Software Development*. Boston: Addison-Wesley.

Denne and Cleland-Huang. 2003. *Software by Numbers: Low-Risk, High-Return Development*. Upper Saddle River, NJ: Prentice Hall.

**Business-Driven Software Development (BDS)** 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. BDS has built a reputation and track record of delivering higher quality products faster and with lower cost than other methods

BDS 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.

BDS 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 BDS:

- **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

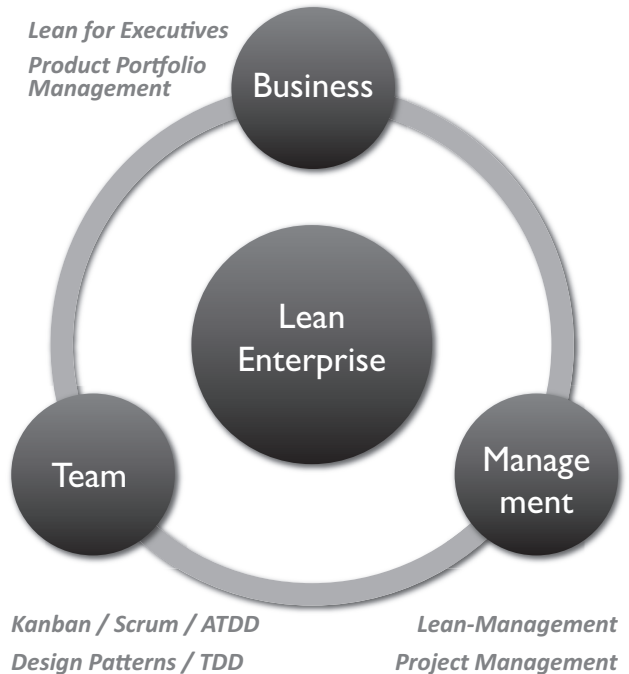
All levels of your organization will experience impacts and require change management. We help prepare executive, mid-management and the front-line with the competencies required to successfully change the culture to a Lean-Agile enterprise.

**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 BDS 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