The following is an excerpt from Lean-Agile Software Development: Achieving Enterprise Agility *by Shalloway, Beaver, and Trott. No portions may be reproduced without the express permission of Net Objectives, Inc.*

Chapter 5. Going beyond Scrum

It isn't what we don't know that gives us trouble; it's what we know that ain't so. —Will Rogers

In This Chapter

This chapter discusses how to approach learning a new methodology and how to extend your knowledge of the methodology. We start with Scrum as a base because it is widely used in Agile projects. Although Scrum is quite effective at the team level, it needs to be expanded to work well throughout an Agile enterprise. We explore several of the mistaken beliefs that Scrum practitioners (new and experienced) hold and the unfortunate consequences of these beliefs as well as some of Scrum's actual limitations. After considering these issues, we offer two approaches to help teams go beyond Scrum: Scrum#™ (which embeds Scrum within Lean thinking) and Kanban software engineering (which focuses directly on the flow of work). The chapter concludes with a brief case study to illustrate these ideas.

Takeaways

Key insights to take away from this chapter include

- Scrum is a powerful, purposely lightweight framework that naturally limits work-in-process at a team level and empowers teams to work effectively with business partners.
- Scrum, executed correctly, exposes impediments so that an organization can react.
- While Scrum works well at the team level, using it as the primary method to guide agility at the enterprise level has severe challenges.
- Lean brings proactive guidance to Scrum, and offers clear principles that explain why Scrum works (and why it fails when it does).
- Both teams and management have a common responsibility for good process.
- Always use the best practices and approaches for your particular situation, informed by good principles and the experience of others. And then improve continuously through Plan-Do-Check-Act.
- Scrum# is an enhancement of Scrum that results from embedding Scrum with Lean thinking.
- Kanban software engineering is a Lean approach that manages WIP directly to improve the flow of work through the value stream.

Learning a New Way

There are many models that describe how people learn new skills. It is widely acknowledged that people go through different stages: beginner, basic competency, competent, advanced,

© 2009 Net Objectives. All Rights Reserved

expert, virtuoso, master. Transitioning from stage to stage involves starting with a basic set of principles and practices and then expanding on them as the learner becomes more adept. The model evolves by building precept upon precept.

For example, when learning how to drive a car in the United States, a new driver may be told a few principles, such as "first be concerned about safety" and "use your own judgment . . . don't let others hurry you." They will also be given a few practices, such as "drive on the right side of the road," "don't follow anyone too closely," and "put on your blinker before you make a turn." This is the basic set, the "100"[1] level for a beginner.

Before long, the beginning driver has to transcend these rules and understand the principles upon which they are based. "Driving on the right side of the road" actually is a manifestation of a principle: "Driving in the same direction as the other cars in your lane is safer than driving in the opposite direction." But, even in the US, if you want to make a left turn on a 2-lane, one-way road, you should do it from the left lane. So the rule, "drive on the right side of the road," is insufficient.

The point is that to become adept at any new process, we need to keep adding principles and practices to what we already know. For some methods, this can be burdensome. If we have no guiding force, it can be overwhelming. Fortunately, Lean-Agile has a mindset that can give guidance without being burdensome. Lean-Agile presents a way of thinking to solve new problems as they come up as well as a way of organizing practices that have been established to work in particular situations (contexts).

Defining a Method while Not Being Restricted by It

In the introduction, we discussed how our industry seems to cycle between too much process and not enough process. Processes can be used to micro-manage teams or can be so loose as to be useless to the organization. Teams react against or are cowed by rigid processes. But they also go astray when process is missing.

Product development requires discovery and feedback so that what is learned can be incorporated; it also requires systems so that what is learned can be incorporated efficiently. It requires both fire and a fireplace: bright people who know how they will work together for the benefit of the product and for the benefit of the organization.

Lean-Agile balances the extremes of too much or not enough process by looking for a process that supports the team. Lean thinking assumes that most errors are systemic in nature. To resolve them, it is critical to understand the way we do our work. Team members must

[1] We use this term to emphasize that these concepts are like those taught in a beginning (freshman) class.

© 2009 Net Objectives. All Rights Reserved

understand the process they are following. Lean thinking assumes that the purpose of the process is to support the team. The team understands their work and conditions better than anyone else; therefore, the team is responsible for their process. When the process dictates actions that don't best fit the situation, the team must modify (improve) it.

Defining a Process

How do you define a process that supports teams? By balancing the following:

- Principles apply in all contexts; practices apply in only certain contexts.
- When learning something new, people are in transition and can learn at only a certain rate.
- Process definitions need to be updated as the team learns and they should help people transition from beginner to expert.

We want a model that can be picked up fairly easily by beginning teams and that can then expand as the team learns and becomes capable of incorporating more knowledge. This balance enables us to provide a definition of a methodology that is not too overburdening to new practitioners yet is rich enough for people once they gain experience.

This concept represents a significant break in thinking between Lean-Agile and many other Agile methods. eXtreme Programming (XP) started with a dozen practices and a few values. As it became more popular and people became more skilled with it, however, little was done to explain the principles on which XP was based.

Several practitioners went beyond the specified practices and used their intuition and experience to take XP beyond situations where the practices as specified would work. This was a good thing. Unfortunately, little effort was made to codify these new practices or the thinking behind them, which meant it was hard for others to transition quickly from beginner to expert. A lot needed to be relearned. This learning was expedited, of course, with coaching from others who had already undergone the transition. But coaches of this nature are often either unavailable (if one looks internally) or expensive (if one has to learn externally).

We see the same thing happening with Scrum. Scrum is propagating through the industry because it is easy for individual or small groups of teams to adopt it. Scrum's practices readily work inside organizations that have well-defined teams, good communication channels with customers or their representatives, few interruptions in terms of supporting existing products (where the teams adopting Scrum are the only ones to support them), and there are not many projects in process at any one time (about one at a time per team). This is the context in which the basic Scrum practices work well.

However, most organizations comprising several teams do not work under these conditions. Very often, an organization decides to adopt Scrum and it special cases the context just

© 2009 Net Objectives. All Rights Reserved

described for one select team; resulting in success with Scrum for that team. Unfortunately, when they try to expand Scrum to other situations in the organization, it does not work nearly as well because the core organizational issues that are necessary are never addressed.

Scrum, like XP before it, has responded to these challenges by relying almost entirely upon developers learning as they go. Developers are expected to "inspect and adapt." This is good, but it's not nearly enough. Both XP and Scrum have good belief systems and values but speak little of the principles underlying their practices.

Lean thinking uses a richer approach, the "Plan-Do-Check-Act" (PDCA) cycle. PDCA requires the team to do its work according to an explicit plan of execution—a "model" if you will, that is guided by experience, good principles, and lessons that others have learned. This plan becomes their standard process. They do their work according to the plan then stop and check their experience against the plan. Based on their observations, they decide how to adjust the *plan*—what to change and what to keep doing. Then they begin to plan again.

The difference between "inspect and adapt" or "sprint plan, execute, retrospect" and PDCA is that PDCAincludes an explicit statement of the workflow being used by the team. We plan what and how we will do our work. We do that. We check to see the results this "model" achieved. We then act accordingly – replanning as necessary. The former approaches leave the team to figure out for itself what to do based on their own intuitions and with little guidance. PDCA, built upon a model of Lean principles, provides more specific guidance and enables us to check the validity of our understanding of the work involved.

PDCA also applies more widely; it is not limited to the development cycle but becomes part of the mindset in all aspects of the team's work. A testing team can use PDCA in its work that is founded upon Test-Driven Development (TDD) principles. A UI team adjusts its interview process after each set of user interviews based upon what they got versus what they expected.

To become a true profession, we must use work processes guided by good principles and by the experience of others while constantly being open to critique and learning.

Principles and Practices Open the Door for Professionalism

There is an analogy here with design patterns. In 1994, Gamma, Helms, Johnson, and Vlissides published their seminal book, *Design Patterns: Elements of Reusable Object-Oriented Software*. Most people have understood patterns as "solutions to recurring problems in a context." That is good for the beginners' level. But patterns are much more than that. Christopher Alexander, author of *Timeless Way of Building* (1979) inspired the patterns community when he said that patterns are really about resolving the forces (or issues) that need to be resolved in recurring situations. Learning how to resolve these forces leads to a discovery of much deeper principles

© 2009 Net Objectives. All Rights Reserved

rather than mere solutions. This deeper understanding of patterns[2] can become a foundation for establishing a Lean-based thought process to use for designing practices that solve problems teams face.

The Lean-Agile approach to creating a model for undertaking software development is a combination of foundational principles, beginning practices, and a thought process that teams can use to expand on their knowledge and to incorporate lessons learned from others. This creates the basis for a level of professionalism in software process that heretofore has not been achieved.[3]

Knowing Where You Are

Any skill that is really useful in life takes time to master. Sometimes you make great progress when you are first learning a skill. Danger lies in thinking that your surface understanding is deeper than it is. Wise people keep pressing on to learn and improve so that they can handle the inevitable challenges. You need to be prepared when the crisis comes—that's not the time to begin preparing.

As educators in several different areas (Lean, Agile, Kanban, Scrum, product management, design patterns, Test-Driven Development), we have seen the importance of clearing away misunderstandings before proceeding on to new concepts. While we never want to forget what we know, misimpressions about what we are learning can stand in our way.

We have chosen to discuss the misunderstandings of Scrum to illustrate the difference between Lean-Agile and other Agile methods because Scrum is widely used and reasonably well known. It represents much of the current attitude in the industry about Agile and is therefore representative of much of the industry's thinking, particularly with new teams attempting to adopt Agile methods.

The following sections cover several beliefs we have encountered that impede learning Scrum effectively. The first involves misunderstandings about Scrum itself—things that people believe but that are not true about Scrum. The second takes on concepts that Scrum does seem to advocate but that we feel are not effective. Errors in understanding must be cast aside before the true intentions can be grasped. Whenever we discover limitations in our thinking, or in this case, the thought process we are learning (Scrum), that becomes a place where we can expand our thinking.

---

[2] See *Design Patterns Explained: A New Perspective on Object-Oriented Design* (Shalloway, Alan, and James R. Trott. Boston, MA: Addison-Wesley, 2004).

[3] This is analogous to Scott Bain's exhortations of creating a basis for the technical aspects of software development in *Emergent Design: The Evolutionary Nature of the Software Profession* (Boston, MA: Addison-Wesley, 2008).

© 2009 Net Objectives. All Rights Reserved

Scrum is a really useful approach. It seems simple and yet it, too, requires skill and determination. To be prepared, you have to understand its principles and practices so that you can adapt to and address the challenges you will face.

After we explore some misunderstandings we have witnessed in the industry and how to get beyond them, we will conclude with a few concepts that many Scrum trainers believe but that we do not agree with.[4]

Scrum Is a Framework

Scrum is a framework for creating an effective Agile development process. It is based on the belief that software development must be controlled by responding to feedback received during the course of development. That is, although software development is inherently empirical (you can't predict it) you can *control* it with feedback. The more frequent the feedback, the more effective developers can be. Scrum suggests building software in stages—say, every two to four weeks. Assess where you are, reprioritize, and develop the next step. Doing this helps expose problems and impediments. And problems are not to be avoided, but rather to be solved. For example, a team may find it does not have enough contact with someone who can speak for their customers. This is a problem that needs to be solved (increase the level of contact) or the team's efforts will be hampered.

Following Scrum means bringing problems to the surface, solving them, then moving forward until more problems surface, and then solving them. There is not a one-size-fits-all approach to this because each team—and the problem domain they work in—is different. They must learn to learn. They must also throw away any limiting beliefs.

Misunderstandings, Inaccurate Beliefs, and Limitations of Scrum

This section discusses a variety of beliefs about Scrum that we have heard from many Scrum practitioners, both new and experienced. These can be grouped into three categories as follows:

- Misunderstandings commonly held by new Scrum practitioners
    - There is no planning before starting your first Sprint.
    - There is no documentation in Scrum.

[4] We are not saying that experienced Scrum practitioners cannot succeed at the enterprise with Scrum. After all, Scrum is merely a framework for building software and the team members need to fill in the framework with their own knowledge. What we are saying is that many of the things Lean provides fit nicely into that framework and should be used. Also, experienced Scrum practitioners know when to break the rules, so to speak—that is, use their intuition. We believe many of the things they intuit on their own can be explained consciously with Lean thinking. Our experience is that intuiting solutions is good, but being able to explain *why* you did what you did is better.

© 2009 Net Objectives. All Rights Reserved

  - There is no architecture in Scrum.
- Scrum beliefs we think are incorrect
  - Scrum succeeds largely because the people doing the work define how to do the work.
  - Teams need to be protected from management.
  - The product owner is the "one wring-able neck" for what the product should be.
  - When deciding what to build, start with stories: Release planning is a process of selecting stories to include in your release.
  - Teams should be comprised of generalists.
  - Inspect-and-adapt is sufficient.
- Limitations of Scrum that must be transcended
  - Self-organizing teams, alone, will improve their processes beyond the team.
  - Every sprint needs to deliver value to the customer.
  - Never plan beyond the current sprint.
  - You can use Scrum-of-Scrums[5] to coordinate interrelated teams working on different products.
  - You can use Scrum without automated acceptance testing or up-front unit testing.

Misunderstandings Commonly Held by New Scrum Practitioners

These are only a few of the misunderstandings we have run across. We mention them because they are some of the most frequent and some of the most damaging.

There is no planning before starting your first Sprint

Many people think that Scrum says just to jump in on day one and build the first part of the system. Actually, Scrum acknowledges that some pre-planning is necessary. We will talk about this in great detail in chapter 6, Iteration 0: Preparing for the First Iteration.

There is no documentation in Scrum

Actually, Scrum doesn't address this directly, but it suggests that there be no documentation unless there is business value for it. This does eliminate many types of documentation. As in all Agile methods, document things only when that documentation will actually be useful. Don't write documentation simply because the process says to do so (and if your process *does* say this, you should change your process).

---

[5] Scrum-of-Scrums is a method of several teams. Each each team sends a member to a meeting where all of the teams working together are represented. These meetings occur to coordinate the teams as often as necessary – very often weekly or even twice a week.

© 2009 Net Objectives. All Rights Reserved

There is no architecture in Scrum

Again, Scrum doesn't address this directly. Scrum is more about managing the team than defining the work the team should do. Chapter 13, The Role of Architecture in Lean-Agile Projects, discusses the proper use of architecture.

Scrum Beliefs We Think are Incorrect

We have found that these beliefs either lower a team's effectiveness or make it less likely that improvements will be made. This is not a complete listing, but these are what we see as the more common and harmful beliefs.

Scrum succeeds largely because the people doing the work define how to do the work

It is true that Scrum follows Lean's mandate that the people doing the work define how the work is done. However, the biggest improvement that many teams achieve when they initially practice Scrum has little to do with this. Consider "teams" that

- Have to pull people from other parts of the organization to get all of their needed skills
- Work on many projects at a time
- Are not co-located
- Have to follow what amounts to bureaucratic policies that are counterproductive

Now, imagine that you are to pilot a Scrum project and are told:

- You will have a cross-functional team with all of the skills you need.
- You will work on only one project at a time
- You team will be co-located.
- You will not have to follow bureaucratic polices that are counterproductive.

You probably would expect a great productivity increase. We have seen teams like this be three times more productive than other teams in the same company even when they work on the same types of projects and have the same level of personnel. We believe that Scrum's first times-three improvement level is often because thrashing stops and delays are cut out. There may be cases when we're unable to do iterative development to the extent we would like, but we still see a huge productivity increase by taking advantage of this.

We want to understand this because we can often make productivity improvements by implementing these items even if Scrum can't be started. Misidentifying the cause of improvement can result in missed opportunities for further improvement.

Teams need to be protected from management

Many Scrum practitioners say that the team should be insulated from management. They misunderstand the Scrum mandate to protect the team from *interruptions*.

This misunderstanding is based on the experience in some organizations that management causes most of the interruptions and therefore the team must be protected from management. This creates an "us (developers) versus them (management)" conflict that has caused so many problems in the software industry. Some generally understood Scrum principles are closer to folklore than actually a part of Scrum. Many of these tend to undermine management.

The famous "chickens and pigs"[6] story used by many Scrum practitioners encourages this attitude. The intent is to illustrate how some people on a project are committed to it while others are merely interested in it. Unfortunately, it is usually used as a way to keep management from being involved. Management is not an impediment to be removed—it's an asset that keeps the entire enterprise moving in the right direction. Chapter 11, Management's Role in Lean-Agile Development, describes attributes of good Lean-Agile management. Again, Lean provides a way for management and workers to work together. Lean's directive that management supports the team while the team creates their process provides guidance here.

It is true that in some organizations teams won't self-organize unless management steps back and allows them to. In this case, the Scrum Master must encourage the team to make decisions in the new vacuum that management's disappearance has created. But this views managers as capable only of managing and not leading. Lean-Agile considers management's role as one of leadership, which is a distinct difference from the way many Scrum practitioners view it.

Leaving this attitude unchecked can result in dysfunctional teams that never quite become effective due to thrashing and poor integration with other groups. At that point, only management's involvement can bring it under control.

Beyond this, there are times when management is essential because teams are constrained by their own local concerns. For example, if reorganization is needed, management must be involved; if impediments are introduced by factors outside the team's control, then management can help resolve them.

Management is a partner in improvement.

The Product Owner is the "one wring-able neck" for what the product should be

Actually, no one's neck should be "wring-able"! The Product Owner is the keeper of priorities but the entire team is responsible for building a quality product. Lean provides a way for developers and managers to work together. It starts with the name of this role: To indicate that

[6] "Chickens and Pigs" is based on an old joke. A chicken and a pig are in a bar having a drink when the chicken says to the pig, "We should open up a restaurant."The pig says, "Oh? And what will we serve?" The chicken responds with "ham and eggs." The pig considers for a moment and then answers, "I don't think so. While you would be interested, I would have to be committed!"

© 2009 Net Objectives. All Rights Reserved

it is leadership—not ownership—that we need, Lean uses "Product Champion" instead of Product Owner.

The Product Champion leads the developer team in discovering what the customer truly needs and she or he assists and guides the rest of the team in this discovery. The Product Champion and the team are responsible for the quality of the product. The Product Champion may be responsible for prioritizing stories, but the development team is no less responsible for the product as a whole.

Practical experience from the field suggests that the Product Champion role comprises a team of product managers, business stakeholders, business analysts, and client-facing personnel who are committed to providing the required service levels of feedback and validation so that the development organization can move quickly. Chapter 10, Becoming an Agile Enterprise, covers this in detail.

When deciding what to build, start with stories: Release planning is a process of selecting stories to include in your release

It is almost always better to start with the big picture. In particular, Agile analysis should be a progression from business capability to sets of features to stories to tasks. The concept of minimum marketable features, described in chapter 4, Lean Portfolio Management, is essential. If you are losing the big picture while working on little pieces, this misunderstanding may be why—you shouldn't be starting with the little pieces. Lean's principle to "optimize the whole" helps provide guidance here. Chapter 7, Lean-Agile Release Planning, describes this in greater detail.

Teams should be comprised of generalists

This is an overly simplistic view. If everyone on the Scrum team can do every task then it is definitely easy to manage how stories are built. In reality, many applications have complexities that require specialists, such as database analysts and developers of stress-test algorithms in aviation.[7] What is really needed is a team that is organized so that it has all the skills it needs to complete the work in a short time. The more knowledge is shared, the better. But the guiding rule is that the team have the necessary blend of skills.

Inspect-and-adapt is sufficient

We discussed this earlier, but it bears repeating. Inspect-and-adapt is good and necessary but it is not sufficient unless it includes explicitly improving the process on which the team works. It is also necessary to incorporate learning and guidance from others and from past experience. The better model is Plan-Do-Check-Act. Having to relearn should be considered a type of lost knowledge.

---

[7] Some of these skills require a Ph.D. and years of experience—not an easy thing to replicate.

© 2009 Net Objectives. All Rights Reserved

Limitations of Scrum That Must be Transcended

Scrum works extremely well for teams within functional organizations. Unfortunately, many people are trying to adopt Scrum in less than fully functional organizations. Although Scrum may help teams isolate themselves from dysfunction in the organization (which can lead to limited improvement), it is better for them to help the organization become more functional. While this is not necessary for an individual team to accomplish its work, it is necessary for multiple teams to work together effectively. Lean-Agile's broader perspective can help here and it is essential if we are to achieve enterprise Agility.

Self-organizing teams, alone, will improve their processes beyond the team

This clearly relates to the prior misconceptions concerning management. Purely self-directed teams have a history of not succeeding well. Scrum teams should be self-organizing, not self-directing. Continuous process improvement can be accelerated by a partnership between teams and management. Although the Scrum Master can provide some of the leadership necessary to prod teams into self-assessment, it is not enough. In fact, the Lean Enterprise Institute (Womack and Shook 2006) described the crucial role that middle management plays, such as asking intelligent questions and establishing an environment in which there is both leadership and collaboration while avoiding being either an autocrat or a hands-off manager. Lean's paradigm of management providing leadership to teams that continuously improve their process provides guidance here.

Every sprint needs to deliver value to the customer

Until the software is released, there is no value delivered, no matter what you do. While each iteration (which Scrum calls a "sprint") should include something that can provide feedback, it is not necessarily true that the iteration is always for the customer's benefit. There are cases when you need to learn something about the system. These situations don't occur as often as some developers think—usually the biggest risk is in building what the customer doesn't need. But there are times when not discovering something about the system now will cause you great problems later (such as redesign or higher integration costs). In these cases, building what can most mitigate your risk may be more appropriate. Lean's principles of "optimize the whole" and "eliminate waste" provide guidance here. We don't want to overbuild, thus adding waste, but at the same time, we must keep the big picture in mind.

That said, you risk losing interest and feedback from key business stakeholders if iterations fail to show verifiable progress against a roadmap. Being able to deliver end-to-end slices of capabilities is a higher-level technical skill that cannot be achieved in a hand-off, legacy organization. It requires cross-functional teams working together. In making the tradeoff between delivery of infrastructure and verifiable business value, always lean toward the latter to ensure that business and stakeholders stay attentive and engaged. They should serve as a natural constraint to "building what is not needed" or looking too far ahead.

© 2009 Net Objectives. All Rights Reserved

Never plan beyond the current sprint

Oh, if only this were possible! On small teams and small projects, it might be; however, as the effort gets larger, it becomes increasingly difficult. An iteration backlog (visual control) sophisticated enough to handle multi-iteration and multi-team stories can manage this and it isn't really that complicated. We talk about this in chapter 8, Visual Controls and Information Radiators for Enterprise Teams. Lean's larger view contrasted with its mandate on eliminating waste helps here again—look as far as you need, but no farther. Dependencies between teams that work at different rates will require looking ahead to make sure their efforts are well coordinated.

You can use Scrum-of-Scrums to coordinate interrelated teams working on different products

Scrum-of-Scrums is a great practice for coordination. However, when the different teams involved in the Scrum-of-Scrums have different purposes, motivations, or driving metrics, Scrum-of-Scrums simply does not work well. When the pressure is on and when teams have different motivations, they tend to home in on solving their problems. It is human nature to focus on those closest to us—so when we try to coordinate teams, naturally we will do what is in our own team's best interest.

Lean can provide a bigger view. We all know the challenge of creating teams from individuals—we must provide a common goal. Creating an organization from individual teams has the same problem. Having a product-coordination team that reaches across teams can solve this. Chapter 12, The Product-Coordination Team—Going beyond Scrum of Scrums, provides a better alternative.

Another characteristic of Scrum-of-Scrums is that often they become reactive in nature, and are simply a place to discuss impediments. Lean guidance would suggest that if impediments exist, then the process must be improved. The product-coordination team is proactive and creates a structure for cross-team planning and visibility, which is critical for scaling and sustainability.

You can do Scrum without automated acceptance testing or up-front unit testing

Most people are not aware that Scrum, as Jeff Sutherland originally created it, included automated testing practices and other quality engineering practices. Unfortunately, these were removed so that Scrum would catch on more readily. We say unfortunately because without them the quality of your code will degrade and you will find it hard to change and dangerous to change as well. Not to mention that the process of getting good acceptance tests helps clarify the customer's needs and therefore lowers the risks of both building the wrong and thing building things wrong.

It is interesting to note that the Lean principles "optimize the whole" and "build quality in" are almost always violated when automated testing is not included. Optimize the whole, in this situation, means that when you are building a product you need to consider the entire time

© 2009 Net Objectives. All Rights Reserved

span of both the coding and testing as well as its whole lifespan, not merely the coding phase. Getting out a quick release that will cost a lot to maintain is not a good practice. As Scrum teams mature, most are starting to realize that automated testing should be a part of Scrum.

Here are three essential references if you want to learn more about this:

- *Emergent Design: The Evolutionary Nature of the Software Profession* (Bain 2008)
- *eXtreme Programming Explained, Second Edition* (Beck and Andres 2004)
- *Working Effectively with Legacy Code* (Feathers 2004)

Consequences of these Beliefs

These beliefs combine to form additional challenges.

Management should not prod teams for information if the teams decide they don't need to give it

This sometimes arises from a combination of the fallacy that self-organized teams don't need management's help (they do!) and that teams need to be protected from management. The visual controls used by most teams should provide information to both the team and to management. Management has a need and a right to understand what is happening in the team. If it is difficult for a team to show this, there is something wrong with how the team is tracking their work. Lean's use of visual controls provides guidance here.

No part of a team's process can be dictated from the outside

It is unfortunate that many people still have the attitude that a central process group can find the right process for all teams. This is a holdover from a legacy mentality that does not work. Teams need to be responsible for their own process. However, Lean's principle of optimizing the whole does mean that certain standards need to be established for how teams work together. Typically this is better accomplished with a "here is what needs to be achieved" mandate and not a "here is how to do it" one. Having consistent methods teams must use to work together is a good practice – as long as each team can determine how best to meet this requirement.

Lean Thinking Provides the Necessary Foundation

Leaving teams to figure out for themselves the practices, workflows, and approaches almost guarantees misunderstandings, errors, and limitations; at best it is inefficient. Teams that guess well will succeed because Scrum is a useful approach. Teams that do not guess well may fail. Certainly, that is inefficient and results in a lot of unlearning, relearning, and adjusting.

The most effective method is to approach Scrum within the system of Lean thinking. Lean offers a well-established model to guide a team in its practices and in its workflow. Table 5.1 illustrates what help Lean thinking provides.

© 2009 Net Objectives. All Rights Reserved

The following is an excerpt from Lean-Agile Software Development: Achieving Enterprise Agility *by Shalloway, Beaver, and Trott. No portions may be reproduced without the express permission of Net Objectives, Inc.*

Table 5.1. Scrum and Lean perspectives

Belief	Scrum Perspective	Lean Perspective
Iteration structure	Use time-boxed iterations; discover and build in relatively small iterations.	Use the type of iteration structure that best addresses your needs (e.g., time-boxed as in Scrum or flow based as in Kanban).
Product direction	The Product Owner is the "one wring-able neck."	The team is responsible for the product. The Product Champion sets priorities and leads the team to discover and build what is needed.
Management	Scrum tends to insulate teams from management.	Managers lead and coach teams. Management and teams work together.
How to organize	Get teams working and then scale them by coordinating teams with Scrum of Scrums	Create a context for all work, such as the value stream. Have teams figure out how to work in this context.
How to learn	Inspect the results of your work and then adapt to improve your situation. Focus on the team's practices and try to improve them.	Work from known good practices. Understand the fundamentals of flow, WIP and do everything with the big picture in mind. As such, plan your work, do it, check it against your understanding and then act accordingly. Don't just "inspect and adapt"; create a model of how things are working and refine it.
Story prioritization	Focus on value to the customer.	Focus on value to the customer and to the Business but also pay attention to cost of delays, not necessarily what the most valuable feature is.
Where to start	Let teams figure things out for themselves.	Begin with a solid understanding of Lean thinking. While teams may be able to solve problems on their own, it is essential that we know as much as possible about what we are doing. Pay particular attention to batch sizes, queues, WIP, and flow.

Introducing Scrum# - Scrum Embedded in Lean Thinking

Looking at Table 5.1, it is reasonable to expect that a team operating from the perspective on the left would get significantly different results—even with the same practices—than one using the explicitly stated Lean perspective on the right. Since both are following Scrum, we have

© 2009 Net Objectives. All Rights Reserved

chosen to label this second type—practicing Scrum within Lean's context and belief system—as Scrum#. Scrum# is Scrum infused with Lean thinking.

It is useful to take this a little deeper than simply the beliefs. Using Lean thinking, what approaches should Scrum# practitioners follow?

At the beginning of any Agile transition, it's best to avoid making too many changes at once; but starting with the four in Table 5.2 virtually guarantees better results. The quality of the team's work will improve and cycle time will decrease.

Table 5-2. Four Essential Practices for Scrum#

Practice	Description
Do timely builds and use swarming.	Many new teams are plagued by difficulties creating builds in a timely manner. Everything may compile and link but still the code fails because of unknown dependencies—such as some other team using an old API, not realizing it has changed. This happens when teams are working on the same stories but are not in sync. A cure is to use team swarming on stories. Difficulties in getting quality builds often result from not enough swarming. Swarming is the practice of bringing all of the people required to work on a story together at the most appropriate time when this will decrease the overall time required to complete the story. This is the Lean approach of focusing on cycle-time instead of individual productivity.
Define acceptance tests prior to writing code.	This practice enhances conversations among customers, analysts, testers, and developers. It also helps testers stay in sync with developers. If developers cannot write code before the testers specify the tests, then the developers need to help prevent testers from getting behind.
By the end of the iteration, complete all stories that have been started.	Avoid opening new stories just because someone is slightly impeded. Many new Agile teams do not realize that having a lot of WIP is an impediment itself. It is better to have fewer completed stories than to have many that are 90 percent done—you cannot demonstrate a 90-percent story to the customer at the end of the iteration.
Ask good, reliable questions.	This provokes the team to think about what they are doing and helps them learn to recognize the gaps between what they are doing and what is expected.

Anti-Patterns: Practices to Avoid

Teams just starting out with Scrum seem to always make the same mistakes. They try an approach that seems good only to discover it causes problems. While some coaches contend teams must learn for themselves we believe teams should learn from others' mistakes?8 There are already plenty of other, new things to learn. We have seen these common missteps so often that we can describe them as "anti-patterns"—approaches that are known to work against you. Some common anti-patterns for Scrum teams are

- Stories are not completed in an iteration.
- Stories are too big.
- Stories are not really prioritized.
- Teams work on too many things at once.
- Acceptance tests are not written before coding starts.
- Quality Assurance/Testing is far behind developers.

Here are questions we always try to use:

- Does the team's workload exceed its capacity?
- When was the last time you checked your actual work process against the standard process?
- When was the last time you changed the standard process?
- Where are the delays in your process?
- Is all of that WIP necessary?
- How are you managing your WIP?
- Are developers and testers in sync?
- Does the storyboard really help the team keep to its workflow?
- Are resources properly associated with the open stories?
- How much will limited resources affect the team's work?
- What resource constraints are you experiencing?
- Can these constraints be resolved with cross-training or are they something to live with?
- Does the storyboard reflect constraints and help the team manage them?
- What needs to be more visible to management?
- How will you manage your dependencies?

[8] We think the advice of learning from your mistakes is often poor advice. You want to learn from *others'* mistakes.

© 2009 Net Objectives. All Rights Reserved

Introducing Kanban Software Engineering

This section introduces Kanban software engineering,[9] a relatively new approach to developing software rooted in Lean thinking. Based on long experience and good principles, many development teams see it as a healthier, Leaner alternative.

As Table 5.1 shows, most Agile methods use time-boxing, that is, managing software development by discovering and building in relatively small iterations. This indirectly improves workflow because the team works on small things and gets quick feedback to ensure it is working on the right things. Kanban software engineering focuses more directly on workflow.

Kanban software engineering (referred to as Kanban from now on) is based on the following beliefs: [10]

- Software development is about creating and managing knowledge.
- Software development processes can be described in terms of queues and control loops, and managed accordingly.[11]
- As information flows through the system, we must have some representation of it.[12]

The Kanban model, illustrated in Figure 5.1, is based on the notion that the team works on the appropriate number of features through completion. When the team is ready to begin on the next feature, they pull a feature from a small queue of potential work. This allows for proper management of both selecting what to work on and how to do the work.

- It focuses the team on building features that are as small as possible and that add value to the customer.
- The development pipeline has small queues and batches and so is more efficient.
- The team still gets quick feedback to keep them on track.

---

[9] "Kanban software engineering" is perhaps an unfortunate name because it conjures images of Toyota's *kanban* cards that Toyota uses to manage their pull manufacturing systems. Kanban software engineering is much more than merely using cards to manage WIP.

[10] Most of the ideas regarding Kanban in this chapter come from (Ladas 2009) and (Anderson 2009).

[11] (Ladas 2009, page 10)

[12] ibid, page 26

© 2009 Net Objectives. All Rights Reserved

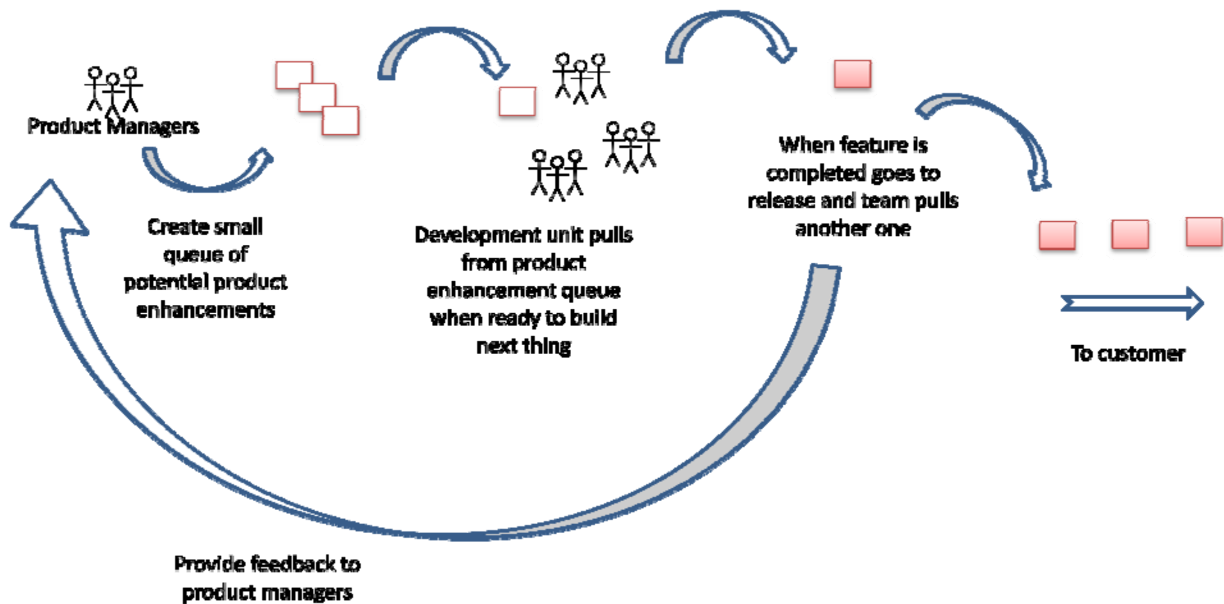


Figure 5-1. Flow of Kanban Software Engineering

The differences between Kanban and common Agile approaches include

- The queues in front of software development teams stay small.
- The software development teams focus on completing features as quickly as possible but are not constrained by a time-boxed system.
- Kanban is explicit about including the entire value stream, from concept to consumption. Ideas from the customer start the value stream and the product managers are directly tied to the teams because of the work-in-process limits that Kanban puts on this flow.
- No estimation is required in Kanban

Managing the Work in the Kanban Team

Kanban does not specify a technique for managing how work is done: It can be done individually or by a team swarm. Instead, Kanban seeks to control the amount of WIP that is allowed. Kanban accomplishes this by specifying slots for each available type of activity. Simply by limiting the number of slots available, we can limit the amount of WIP the team has at any step. By defining WIP limits for each activity, we can minimize the average cycle time for any activity.

The Kanban board (illustrated in Figure 5.2) helps the team manage its work. As team members complete a task, they move the card representing that work to the next step in the workflow. At any point in time, the board represents the current state of work. It also shows the process that the team is using and its WIP limits. The Kanban board could be considered a perfect "visual control" (discussed in chapter 8, Visual Controls for Enterprise Teams) because it accurately shows both process and status with minimal effort.

© 2009 Net Objectives. All Rights Reserved

Kanban's approach—based on Lean thinking—is inclusive of management. This means that management is included in the conversations about how the work is being performed and tracked. This is important because it also means that management cannot just say "do more!" Instead, they agree to abide by the methods the team has selected to do their work. Chapter 11, Management's Role in Lean-Agile discusses how managers lead and coach teams. By creating visibility into the team's process (transparency), management can work with the team on improving it.

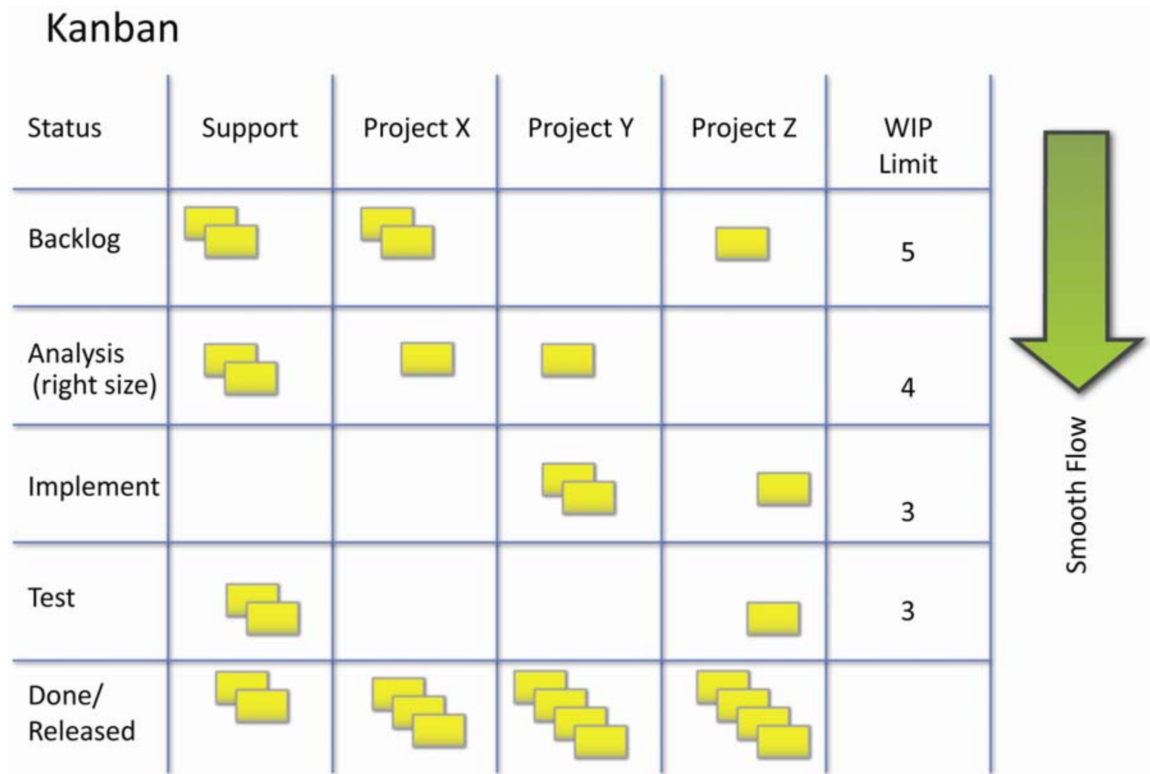


Figure 5-2. A Kanban board limiting WIP

Another diagram used by Kanban is the cumulative flow diagram (CFD), which describes the overall flow through the Kanban system; it provides a measurement for every significant step in the workflow. Figure 5.3 shows an idealized case with four steps: backlog (to be done), analysis, implement, and done. For each step, it shows the count of features at the given time interval. Wide lines indicate an impediment or blockage of flow while thin lines indicate that WIP is too small (sometimes called an "air bubble").

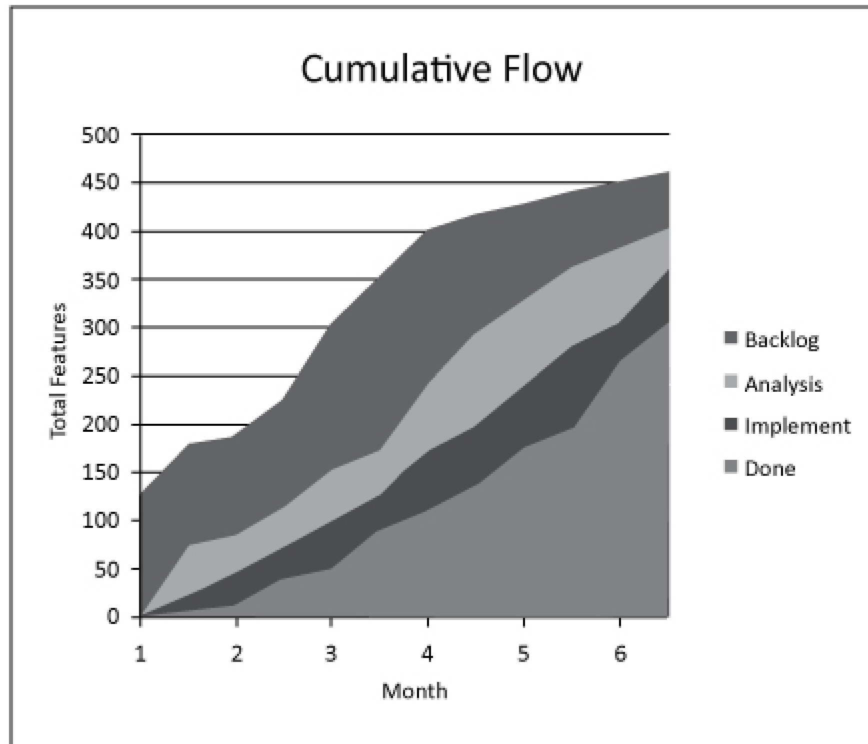


Figure 5-3. Cumulative Flow Diagram

Advantages of Kanban

Many Agile teams spend ten to twenty percent of their time breaking down features into stories and estimating them. Sometimes this can be valuable for improved understanding of the stories; however, when the story breakdown is required simply to make stories fit into an artificial deadline set by your time-boxed iteration scheme, it is wasted work. Compound this with the cost of estimating these smaller stories and it is a whole lot of expense and work for no extra value. Kanban eliminates this type of waste by managing for flow rather than for time boxes.

Kanban doesn't assume all estimation is unnecessary, but it suggests looking at value received for time invested. Kanban pulls well from the features identified by the portfolio team. If you are fairly certain of your feature estimates, you may discover that detailed story estimates are not necessary.

The true value of Kanban lies in its requirement that the team create an explicit workflow with explicitly defined rules and limits. This enables team members to discuss objectively what is working and what is not. That is, it helps the team focus on the *process* rather than on blaming a *person*. Yes, a person might have made a mistake, but what is it about the process that allowed the mistake to happen or to go undetected? Fix the process.

Think of Kanban this way; it combines

© 2009 Net Objectives. All Rights Reserved

- Defining a workflow based on queues and control loops.

  and

- Managing this workflow by limiting the amount of WIP at any step in the workflow.

Evidence suggests that teams learn continuous process improvement faster with Kanban. Some of the reasons are as follows: [13]

- Kanban reduces the fear of committing to a per-story estimate, which is a significant risk for some teams. Fear always impedes learning.
- Kanban is explicitly a team process rather than one for individuals. It highlights the team's performance rather than individuals' and can reduce the fear of embarrassment.
- Kanban focuses on how the workflow process can be improved rather than blaming an individual.
- Kanban allows reflection about concrete measures such as, "Should WIP be 4 or 5?" Reflection about concrete issues is often easier in the beginning than reflecting about more abstract or personal issues.
- Transparency of the process allows management involvement in its improvement.

Case Study: Contrasting Scrum and Kanban

This example contrasts how Scrum and Kanban play out in the real world. Imagine two Agile teams at two different companies. One company uses Scrum, the other, Kanban.

At the first company, the Scrum team and management agree that Product Owner prioritizes the features on the backlog and, once the team selects features, the team is free to do whatever it wants in whatever sequence so as to meet its commitment. There is not much collaborative work between the team and the Product Owner. Management prioritizes (through the PO) and the team implements. If management tries to get heavy handed and demand something, the team can decline to work (that is, they can abort the sprint). If something urgent comes up, managers must wait until the end of the sprint in order to add it to the list. The team thinks this isn't so bad; on average, management will have to wait only a week or, at most, two. Management is none too pleased; they can no longer get the team to work on something immediately as they used to do.

At the second company, the Kanban team spells out to management its workflow and its rationale for the WIP limits. They use language that managers understand: Here is where we figure out what the customer wants (analysis), here is where we decide how to verify that we have done what they want (test specification), here is where we design it, here is where we build it, here is where we validate that we built it right (acceptance test), and so forth. They

[13] Thanks to John Heintz for several of these insights.

© 2009 Net Objectives. All Rights Reserved

make it clear that they are managing their work and limiting that work to the team's capacity. This keeps their efficiency high, yet enables them to respond quickly to requests.

The team reaches an agreement with management that they will always pick the top item in each work queue and do it as quickly as possible, using the best development methods they know, to meet quality objectives. If management must expedite something that is more important than existing WIP, they agree to a "silver card" convention, which always moves the item to the top of each queue. The team can also establish different service level agreements (SLAs) with management if necessary so that certain tasks are generally pt into the queue before others (e.g., critical bugs).

At the Scrum Team

Imagine the following conversation at the company that is using Scrum. A VP is putting some pressure on the development manager.

Product Manager (PM): Joe (one of their VPs) just told me we need to get Feature X done immediately.

Scrum Master (SM): Great. We'll move that to the top of the backlog and do it next sprint.

PM: What part of "immediately" do you not understand? He wants it done now.

SM: Yes, I understand what he wants, but that'll be disruptive and have an overall negative effect. When we started Scrum we agreed that we could work without interruption during the sprint.

PM: Well, that's true, but that was intended for when things are going normally. Now I'm getting heat to get this done. I'm sorry, but just have the team work a little harder this next week to get it in. I don't ask very often.

SM: Well, I think if we put this thing in, we have to take something out.

PM: You know we can't do that. If you don't keep your sprint commitment then that'll impact the other teams that are depending upon you.

SM: Can't you just go back to Joe and tell him it'll be disruptive?

PM: Would you want to do that?

SM: Well, no.

PM: Good. Then we're agreed. You'll all just buckle down a little and get this small thing done. I really appreciate it.

© 2009 Net Objectives. All Rights Reserved

SM (to himself): Great. I guess I could have told him about our agreement to abort the sprint but I know that'll be a CLM [career limiting move]. Well, having the team work an extra weekend is still better than the way it used to be.

At the Kanban Team

Pressure from VPs is nothing new. Here is the conversation that might take place at the company using Kanban when the VP goes to the development manager with something urgent.

Product Manager 1 (PM1): Joe (one of their VPs) just told me we need to get Feature X done immediately.

Kanban Team Leader (KTL): Great. Just move it to the top of the queue and we'll pull it next.

PM1: What part of "immediately" do you not understand? He wants it done now.

KTL: So you want us to drop everything we are doing and get to this?

PM1: Yes, that's what immediately means.

KTL: Do you think it'd be OK for us to let people finish the current task they are working on so they can at least get closure on that? Most everyone would only need a day or so to complete that. Then we'd get on this by using the "silver card" and everyone appropriate would give it their full attention.

PM1: Yes, that's OK. Joe'll be pleased to know that he's going to get his work done without being too disruptive.

KTL: Oh, it'll disrupt us, but if that's the right business decision, no problem.

KTL puts silver card with Joe's request on the board and notifies the other product managers and VPs of its presence.

Product Manager 2 (PM2) calling PM1: I see you've silver-carded feature X. Are you aware that this will slow down the three features we're currently working on? I need those features to be able to hit our release schedule.

PM1: Well, Joe said we needed to get this done. We just got a big account that needs feature X. If we get that done quickly we can make thousands of dollars.

PM2: OK, but we've made a lot of other promises as well. I don't think we can knee-jerk react here.

. . .

Now, we could take this conversation either way. Perhaps the PMs resolve it, perhaps they go to Joe. Maybe Joe realizes his mistake or decides the impact is too great or stays stubborn and can

© 2009 Net Objectives. All Rights Reserved

outrank everyone else. The point is the conversation is elevated to where it should be—at the product-management level.

Compare this with the typical situation in Scrum, which deals with conflict primarily at the level of a single product manager and the team. The problem is not between the product manager and the team, it is that the product managers aren't prioritizing among themselves.

By creating transparency into the process, the demand's impact is seen across the organization. The Kanban team doesn't have to take a strong and uncomfortable stand, they merely let the business side see the impact their decisions will have on the productivity of the organization. Because the software team is not a black box to management, management can work with the team more effectively.

Selecting an Approach

Many choices, which one to use? Use the one that that fits the needs of your own particular context. There is not necessarily a right or wrong answer. Learn as much as you can and then consider the various tradeoffs. Table 5.3 compares how XP, Scrum, Kanban, and general Lean thinking address different factors. [14] Use this to help you select an approach for your team.

Table 5-3. Selecting an approach

Factor	XP	Scrum	Scrum#	Kanban	Lean Thinking
Keep teams intact	Prescribed	Prescribed	-	-	-
Use time-boxed intervals	Yes	Yes	Yes	No	-
Prioritize stories across a team	Yes	Yes	Yes	No	-
When to release completed work	At end of selected iteration	At end of selected iteration	At end of selected iteration	Whenever, at discretion of team	-
Works in a support environment	No	No	No	Yes	Yes

[14] We believe Waterfall is virtually never appropriate if you have Kanban and Lean alternatives. Whereas XP and Scrum may not work in certain situations, with Lean and Kanban, you can always use the underlying principles to figure out what to do.

© 2009 Net Objectives. All Rights Reserved

Co-located teams[15]	No guidance	No guidance	Use fast-flexible flow to create optimal workflow	Manage with appropriate WIP limits	Use fast-flexible-flow to create optimal workflow
Support for the product management organization	No	No	Yes	Partial	Yes
Code quality	Yes	Not discussed	Use a workflow that increases quality	Use a workflow that increases quality	Use a workflow that increases quality

A Case Study: Process control

Recently, we worked with a medium-sized group in a process-control company that was just starting its transition to Agile. [16] This group was comprised of 70 product managers, leads, developers, and integration/build support organized across eight teams. There were three Product Champions for all eight teams.

As is common in development teams building on specialized hardware, the teams were organized around the hardware, each team working on a different component. This is less than ideal: It is like having teams organized around the tiers in an n-tier architecture: one team for the UI, one team for the mid-tier, and one team for the database. It is the exact opposite of swarming and can cause a lot of problems (for example, it is hard to integrate code).

Our first inclination was to reorganize the teams so each team had expertise on each type of hardware component. Upon reflection, the pain of reorganization seemed greater than the benefit, especially given they were just learning Agile. Instead, remembering that principles—not prescribed practices—should drive any approach, we looked for the principle that should guide us. To find this, we asked what problems they currently were having. Their main problem was integrating the work of several teams when a feature cut across several components. It was

---

[15] All methods will find this incredibly valuable. The question is how do they help you overcome the challenge of when teams are not co-located.

[16] This section is written in the first person because it is about a team that was coached by Alan. The description of the product is changed to protect confidentiality.

© 2009 Net Objectives. All Rights Reserved

No portions may be reproduced without the express permission of Net Objectives, Inc.

clear we needed to swarm to avoid the problems (waste) they were having integrating features that cut across the hardware components on which the software resided.

It turned out that 80 percent of the features they were working on were isolated to one hardware component. For the majority of cases, their structure was fine. What they needed was a way to handle the features that required more than one hardware component.

Another question was how could we know that we were building features as efficiently as possible or, alternatively, how could we know when we were *not* building efficiently? This question arose because the team was facing such long build times: Doing a build took less than an hour; fixing the build's problems often took a whole day. This occurred because features across components were being built over a long time span. Each developer (on a different component) would have a branch and then check it in after many changes. This practice was essentially creating errors and then delaying when they would be discovered.

How to create efficient builds? The answer turned out to be fairly simple, if unusual. We decided to create special teams to work on a cross-component feature. This way, 80 percent of the time they could work with the component-oriented structure in place, but when the team needed to swarm, it could. Our approach was to let the cross-component stories drive the work: When the appropriate members of the cross-component team were available, they would pull the cross-component story from the backlog; otherwise, people would continue working on the single-component stories.

This approach required an unusual form of iteration planning. The three Product Champions agreed to meet on iteration planning day with the team leads to determine the amount of work to be done for the iteration. This worked because the team leads were the most knowledgeable about the effort required to build the features. For each team, they created a backlog of work for that team's hardware component. Then, they created a special backlog made up of cross-component work.

During the iteration, when it was time to pull work, they would look first to the special backlog to see if a "teamlet" was available to work on it; otherwise, they would pull from the team's own backlog.

Summary

While Scrum is an effective Agile project-management framework, its history, and common misinterpretations about it, tend to limit its effectiveness when you want to extend it beyond a few teams. It never pays to be dogmatic in following Scrum. Scrum is meant to be a framework for creating an effective process for Agile development. One of its great mandates is to find impediments and remove them. While Scrum tells you not to follow an ineffective practice, it often doesn't tell you what to do when its practices don't seem to apply.

© 2009 Net Objectives. All Rights Reserved

It is much more effective for teams to begin with approaches that are known to be good than to have to start from scratch. There is always more to learn; but you start out ahead when you can learn from the mistakes and successes of others. One effective approach is to let Lean thinking guide the practices of Scrum, to "embed Scrum in Lean thinking." We call this approach Scrum#, and offer some specific practices that this approach prescribes.

Kanban software engineering is an emerging approach to software development that is also based on Lean thinking. Kanban seeks to improve the flow of products through the value stream by managing WIP directly. This is often a better approach than trying to manage flow through short iterations.

Wise development teams will use the approach—Scrum# or Kanban—that best fits their context.

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.

- If you have examples of failed Scrum projects in your organization:
  - What was the reason(s) for failure?
  - How did the organization react?

Did anyone observe any of the misunderstandings presented in this chapter?

- If you are planning to use Scrum for the first time, are any of the misunderstandings we've talked about cited as justification for resistance?
- Review Table 5.3. What approach is best for your situation?
- How much WIP do you currently tolerate?
- How should you organize your storyboard so that it helps
  - Control your WIP
  - Management quickly grasp what they need to know
  - Make visible and address resource constraints
  - Manage dependencies in a visible way

Recommended Reading

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

Alexander, Christopher. *The Timeless Way of Building.* New York: 1979, Oxford University Press.

Anderson, David J. *Agile Management Blog: Thoughts on Software, Management, Constraints and Agility.* June 8, 2009. http://www.agilemanagement.net (accessed June 8, 2009).

© 2009 Net Objectives. All Rights Reserved

Bain, Scott L. *Emergent Design: The Evolutionary Nature of Professional Software Development.* Upper Saddle River, NJ : Addison-Wesley Professional, 2008.

Beck, Kent, and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition).* Boston, MA: Addison-Wesley Professional, 2004.

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

Feathers, Michael. *Working Effectivcely with Legacy Code.* Upper Saddle River, NJ: Prentice Hall PTR, 2004.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series, 1994.

Kennedy, Michael. *Product Development for the Lean Enterprise: Why Toyota's System Is Four Times More Productive and How You Can Implement It.* Richmond, VA: Oaklea Press, 2003.

Ladas, Corey. *Scrumban - Essays on Kanban Systems for Lean Software Development.* Seattle, WA: Modus Cooperandi Press, 2009.

Poppendieck, Mary, and Tom Poppendieck. *Implementing Lean Software Development: From Concept to Cash.* Upper Saddle River, NJ: Addison-Wesley, 2006.

Reinertsen, Donald G. *Managing the Design Factory.* New York: Free Press, 1997.

Shalloway, Alan, and James R Trott. *Design Patterns Explained: A New Perspective on Object-Oriented Design.* Boston, MA: Addison-Wesley Professional, 2004.

Womack, James P, and Daniel T Jones. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation.* New York: Simon & Schuster, 1996.

Womack, James P, and John Shook. "Lean Management and the Role of Lean Leadership Webinar." *Lean Enterprise Institute.* October 19, 2006. www.lean.org/Events/LeanManagementWebinar.cfm (accessed October 23, 2007).

© 2009 Net Objectives. All Rights Reserved

Net Objectives

Business-Driven Software Development™

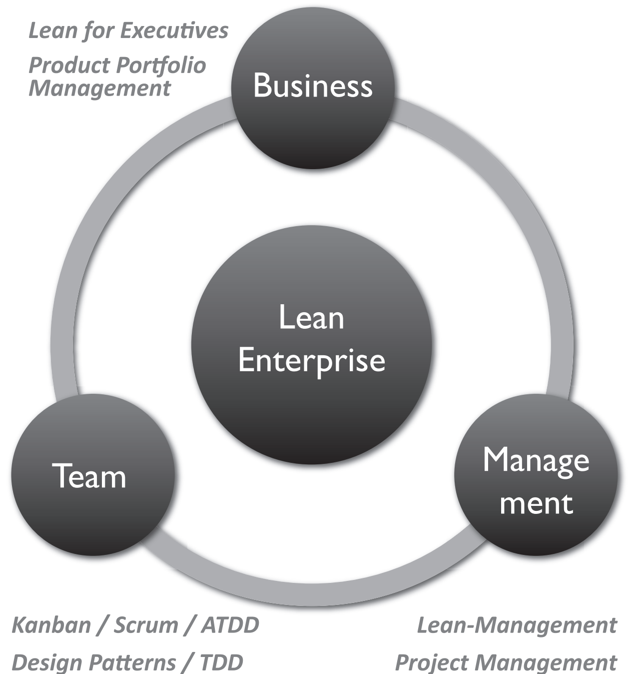
Assessments, Consulting, Training, and Coaching

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)