

# Scrum#: Extending Scrum to the Enterprise

---

## Scaling Scrum

Thousands of teams have adopted Scrum and have met with great success. Dozens of large organizations have adopted Scrum as a standard approach for their enterprise. For that, our industry can thank Jeff Sutherland and Ken Schwaber and the Scrum Alliance.

Yet, for all its success, perhaps *only a third* of Scrum teams succeed. In our work assisting Agile organizations to adopt more effective software development practices, Net Objectives has seen the challenges and frustrations that teams have in taking Scrum to the enterprise. It's not that it can't be done; it is just that Scrum itself provides little guidance in how to do so.

Einstein once remarked "We can't solve problems by using the same kind of thinking we used when we created them." Without guidance in new ways of thinking, people and organizations fall back on what they have always done. It is natural, but will not get them where they want to go.

Scrum# is an extension to Scrum to provide that guidance.

Scrum# was developed by Net Objectives to solve specific challenges that were being encountered by many teams adopting Scrum. These issues include:

- Managing how product enhancements are initiated
- Giving guidance for product managers to work together to perform product portfolio management
- Managing dependencies between the software development organization and the other groups with which it works
- Managing dependencies across the software development organization
- Integrating development and QA roles to increase quality of software and eliminate waste
- Deciding on engineering practice standards across software development teams

The first few of these issues are outside of the scope of Scrum and Scrum has little to say about the others. Scrum does an excellent job at helping individual software development teams improve their productivity; however, it provides little or no guidance beyond the individual team or the project they are working on.

Scrum# extends Scrum by adding the Enterprise view of Lean Thinking and the technical knowledge of Emergent Design. It gives Scrum practitioners the tools they need to extend Scrum from the team to the Enterprise and the guidance about technical skills they need to accomplish iterative development.

## Scrum#: Extending the Scope of Scrum

There are at least two significant dimensions when looking at the scope of a process (Figure 1). The first is the timeline: when does the process start and when does it end? The second is the scope in which work is being done: Is it at the individual level, the team level, the software development organization level, or the Enterprise level?

Scrum's power is in its team and project focus. It should not be surprising, then, that it does not address the many issues that face the software development organization, let alone the Enterprise. Nor that it offers little guidance on detecting how upstream and downstream work will affect the Scrum team's product development efforts or how to address these impacts when they occur. This goes beyond what traditional Scrum is concerned about.

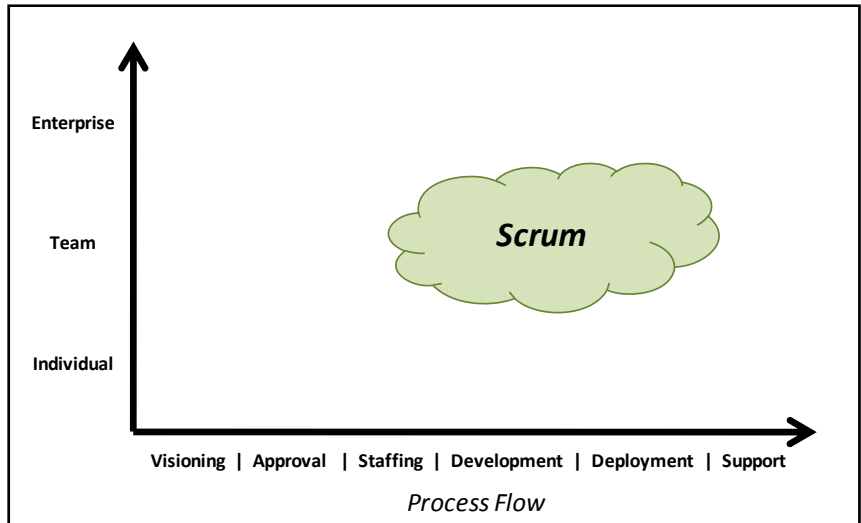


Figure 1. Scrum in the Process Flow

To cover the full range of concerns of the software development organization and the Enterprise, an overarching thought process and technical skill set must be added to Scrum (Figure 2). The thought process is called "Lean Thinking" and the technical skill set is embodied in "Emergent Design", a combination of design patterns and test-driven development which together provide a basis for writing sustainable code.

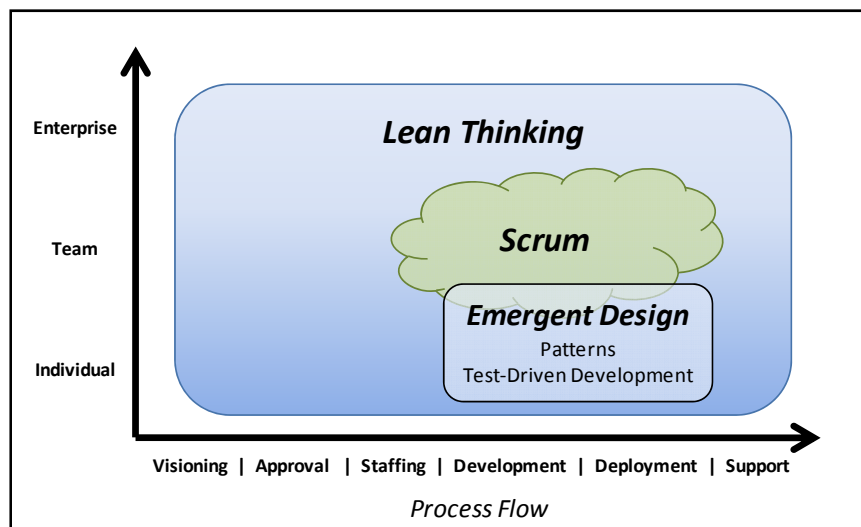


Figure 2. Scrum# addresses the whole concerns of the Enterprise

## Scrum# Extends Scrum with Lean Thinking

Scrum is based on Lean principles. According to Jeff Sutherland,

*Scrum was the first concrete implementation of lean thinking to software development that allowed organizations of all types and sizes to start up lean teams in a couple of days using a standard pattern that was easily understood. What was hard was explaining why and how to implement the pattern to generate continuous quality and productivity improvement. ...Now we can explain how to use Scrum to lean out software development. ...Lean software development views all agile methods as*

*valid, proven applications of lean thinking to software. It also goes beyond agile, providing a broader perspective that enables agile methods to thrive. First, it looks along the whole value chain, from concept to cash and tries to address all the waste and delays that happen before and after the coders contribute their part. Second, it establishes a management context to extend, nourish, and leverage agile software practices. Third, it provides a proven set of principles that each organization can use to adapt tools, techniques, and methods to their own specific unique contexts and capabilities. (Poppendieck, 2006)*

Scrum# applies Lean Thinking principles much more explicitly and broadly. This is vital when one wants to extend Scrum practices beyond a handful of teams to benefit the Enterprise.

The essential elements of Lean Thinking include:

- Focus on optimizing the whole (from the initial concept through delivery of the product)
- Take a systemic view (most errors are due to the system within which one works)
- Focus on eliminating waste to improve quality
- Deliver value with quality (which leads to sustainable high-speed development and lower costs)
- People doing the work must come up with their own process to do the work
- Management plays a key role in coordinating teams and facilitating team process

## Scrum# Extends Scrum with Emergent Design

When teams begin to use Scrum, they often experience great improvements in productivity. Not only will they turn out more functionality more quickly, but, if they pay attention to completing functionality at the end of each sprint, they will see improvements in certain aspects of their code quality. This is good, but is not sufficient. Code quality does not come automatically simply from short iterations.

*Without a disciplined, intentional focus on code quality, quality will tend to degrade over time.*

*Without such focus, teams using Scrum often discover they have new issues of quality to deal with.*

Emergent Design provides the necessary discipline, combining up-front testing (both acceptance- and unit testing) and the design philosophy of patterns. Emergent Design perfectly complements the Scrum practice of building code in stages. Scrum teams

using Emergent Design have a strong technical foundation for evolving software application architectures efficiently over time and for achieving high quality while doing so.

## Helping the Enterprise

### Going for Fast-Flexible-Flow

Think of software product development as a pipeline: Ideas go in and products come out (Figure 3). To have faster flow through the pipeline, we need to eliminate wasted effort, anything that slows down the work being done. And this must be done everywhere in the pipeline: it is never enough to optimize one part of the process if a downstream process is still slow. To become more productive, Scrum# teams focus on optimizing the entire development process.

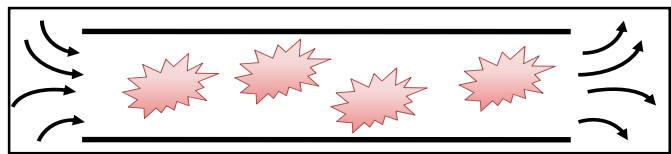


Figure 3. A typical product development pipeline

But increasing throughput is not enough. The pipeline almost must be flexible, able to respond to the needs of the business as quickly as possible. It should not take six months to get to important priorities.

Scrum# aims to focus teams always to be producing and delivering the right, highest value products as quickly as possible in a sustainable way. This is called “fast-flexible-flow” (Womack & Jones, 2003).

To achieve fast-flexible-flow through the pipeline, focus on these three questions:

1. What are the right projects to go *into* the pipeline?
2. What is the *best size* for these projects?
3. How should work that is *in* the pipeline be managed?

The efficiency of the pipeline is governed by what goes into it. If it contains many projects that require people working on multiple projects at once, there will be excessive task-switching and that leads automatically to thrashing. Moreover, there will be delays caused by waiting on key people to complete their work, people who are also spread across several projects.

The answer is to use two different pipelines: one to select the product features and the other to build them (Figure 4). The first pipeline can be called “product portfolio management” – the process of selecting and prioritizing the products that teams are going to build. The second pipeline can be called “development”, where teams build product. The feedback loop between the pipelines provides a better understanding of what is being built as well as allow for quick changes if external factors demand it. Scrum tends to focus on the righthand side, the Development pipeline. Scrum# also encompasses product portfolio management.

## Formal Portfolio Management and the Minimum Marketable Feature

Product portfolio management allows the business to drive development based on current business needs and what has been learned needs while avoiding churn. Scrum# achieves through a technique called the “Minimum Marketable Feature.” According to (Denne & Cleland-Huang, 2003), a Minimum Marketable Feature (MMF) is the

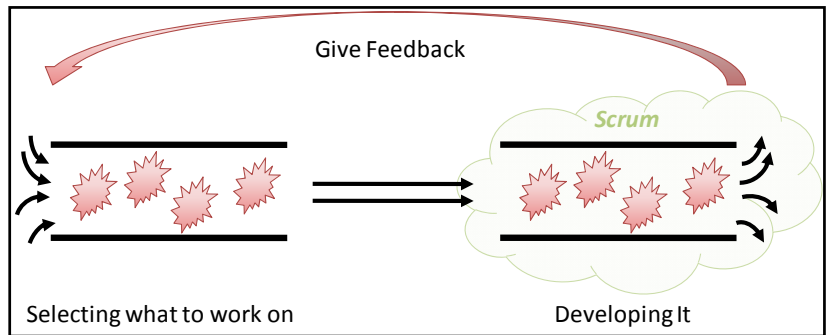


Figure 4. Software Development as Two Pipelines: Selection and Work. Scrum# emphasizes both pipelines

smallest set of functionality that can be released to a customer while providing both value to them and being consistent with the needs of the business. The MMF expresses a defined set of value that can be delivered. Table 1 offers some benefits of this approach

Table 1. Benefits of Portfolio Management with MMF

Benefit	Description
<b>Prioritization by value</b>	The Business can look across <i>all</i> products and projects to prioritize and select the work that most needs to be done now to bring business value now without committing to wasteful additional work  The Business can compare work requests by the value it brings, which gives a better foundation for selecting between new functionality, enhancements, and infrastructure support.
<b>Meaningful metrics</b>	Metrics are showing value delivered rather than hours or dollars spent. Metrics are easily calculated: whenever a MMF is done, but business value burn-up is updated.
<b>Teams always see the larger context</b>	As the team plans its work, it decomposes MMFs into stories that can be completed in a sprint. Since stories trace back to an MMF, the team can tell what is remaining to complete an MMF. All things being equal, they will choose to work on stories that complete the MMF and thus, to deliver value to the customer quickly.
<b>Teams know when to stop</b>	Because teams are focused on completing MMFs, they have a natural way of knowing when to stop work on one feature and turn to the next. More so because Scrum# requires each MMF to have defined criteria for when the MMF is “done.” This is a direct benefit of formal and explicit management of the portfolio by the Business, which is not common with traditional Scrum.

## Collaboration with the Enterprise Team

Whenever software development involves more than a few teams, issues of coordination and communication become at least as important as technical issues. Traditional Scrum attempts to address this with a “Scrum-of-Scrums” structure, a bottom-up approach that tries to tie together various Scrum teams. Many practitioners have not found this to be very effective, particularly when teams are relatively independent: there is little common basis for them to coordinate.

Scrum# starts with an Enterprise view, looking at

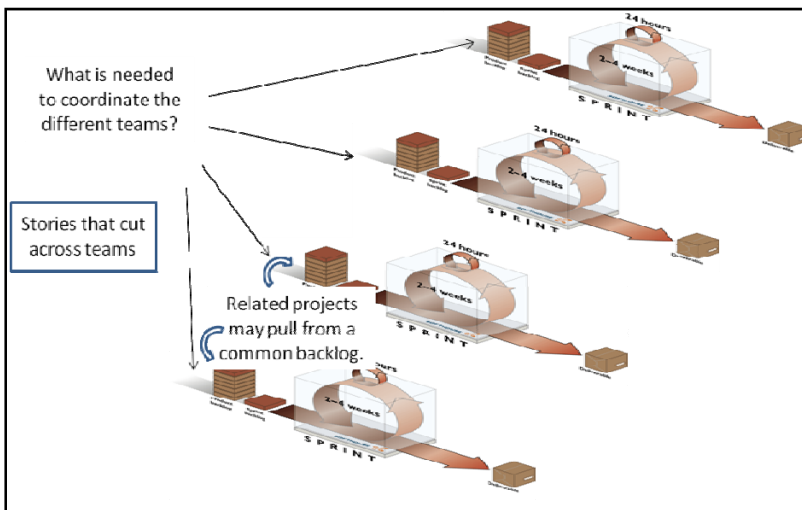


Figure 5. Using a common backlog for multiple Scrum teams

the needs of the whole process and then builds coordination structures that fit those needs. Briefly, an Enterprise Team is responsible for creating structures and opportunities for helping teams share what they know, developing team capacity. For example, when Scrum teams are working on related features, they often create software that may be useful to other teams. A successful approach is for the Enterprise Team to manage a common backlog (Figure 5). Other structures might include facilitated communities of practice, kaizen-type improvement workshops, and even “centers for lessons learned”.

## Helping the Team

Self-organizing teams are at the heart of Scrum. How to build software within the sprint is left up to the local team. Some of the extensions that Scrum# offers are described below.

### The Extended Sprint Backlog

Most Scrum teams work off of two backlogs – the product backlog which contains those stories that will be selected from before each Sprint and the Sprint backlog which represents the work of the current Sprint. Being restricted to these two backlogs works fine when few teams are involved and user stories are small. In other words, if the features being worked on in a Scrum project are small enough to be analyzed and estimated in the planning day and there are few time dependencies between teams, then these two backlogs are sufficient. But how are teams to handle stories that may take days to break down, analyze and estimate. Or stories that require work from other teams where it will take more than a sprint to do the dependent work?

Scrum# uses an extended backlog for visual tracking of the tasks required to discover business value, define how to build it and then build it. Standard Scrum story boards only deal with the last set of actions.

### Organizing Beyond Co-location

Scrum suggests that co-located teams are best. Co-located teams have all of the resources required for the project and should not be interrupted by other tasks. Experience has shown that this is the best organizational structure for a team. But what happens if this is not possible? Lean Thinking principles provide guidance.

## The Role of QA

One of the most useful extensions of Scrum# involves the role of QA. Scrum requires working code at the end of each sprint. What does this mean? Scrum# specifies that working code is both does what the developer thinks it should *and* does what the customer thinks it should. This clearly involves the code going through an entire test cycle. Testing is done up front. Testers are focused more on process (which allowed a defect to occur) than on finding bugs. And the Business has a substantial part to play.

Now, accomplishing this often requires a significant cultural change for most organizations. Culture change is both difficult and not something that can be done directly. As David Mann states in *Creating a Lean Culture*:

*“Should a company target its culture in its efforts to transform its production processes and all the positions – high and low – associated with it? It is tempting to answer: Yes! But that would be a mistake.*

*Culture is no more likely a target than the air we breathe. It is not something to target for change. Culture is an idea arising from experience. That is, our idea of the culture of a place or organization is a result of what we experience there. In this way, a company’s culture is a result of its management system. The premise of this book is that culture is critical, and to change it, you have to change your management system.*

*So, focus on your management system, on targets you can see, such as leaders’ behavior, specific expectations, tools, and routine practices. Lean production systems make this easier, because they emphasize explicitly defined processes and use visual controls.” (Mann, 2005)*

In other words, changing behaviors involves following guidelines whose adoption results in cultural change.

## Helping Individual Developers

While the software itself is not the complete part of the product under development, it is an important part of it. At the end of the day, we are still writing code. The programs produced must be maintainable and extensible. This is even more important in today’s Agile world. The overwhelming amount of code written will be as an extension to existing functionality. Individual developers must always address three issues:

- How do I make my code safe to change?
- How do I make my code easy to change?
- How do I make my code robust so it isn’t likely to break when I change it?

The software development industry has picked up on the need for automated testing. But making code easy to change and robust takes more than testing. It requires knowing what good code is. It requires asking and answering the question, “How do I minimize complexity and rework?”

Scrum# uses the ideas of Emergent Design, including the thought processes of design patterns and test-driven development, to help developers build code in an evolutionary (iterative) manner. Embracing these ideas, developers code faster and with higher quality than they can by building code up front.

The ideas used by Scrum# are well described in books found in the Net Objectives’ Product Development series:

- *Emergent Design: The Evolutionary Nature of Software Development* by Scott Bain
- *An Agile Developer’s Essential Toolkit* by Alan Shalloway et.al.

- *Design Patterns Explained: A New Perspective on Object-Oriented Design* by Alan Shalloway and Jim Trott.

Mann, D. (2005). *Creating a Lean Culture: Tools to Sustain Lean Conversions*. Productivity Press.

Poppendieck, M. a. (2006). *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional.

Womack, J. P., & Jones, D. T. (2003). *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. Free Press.

## Summary

Scrum# extends Scrum with Lean Thinking and best technical practices, as summarized in Table 1. This enables teams to modify standard Scrum practices to their particular situation.

## Works Cited

Denne, M., & Cleland-Huang, J. (2003). *Software by Numbers: Low-Risk, High-Return Development*. Prentice Hall PTR.

Table 2. Scrum# extends Scrum

Area	Description of Extension
<b>Principles and Practices</b>	Scrum is a framework for a team to eliminate the impediments to building software effectively. It offers a few practices – iterative development, daily Scrum meetings, release planning and sprint planning – but little in the way of explicit principles. Scrum# extends Scrum with Lean principles, specific Lean practices, and guidance in what to do when practices do not quite apply.
<b>Perspective</b>	Scrum# extends Scrum to look at interactions between teams and the family of products that the Enterprise may work on. This wider perspective allows the software development organization to work from a larger context.
<b>View of Management</b>	Scrum# believes managers play a key role in software development. Self-directed teams still need capable leadership to guide them, to help resolve impediments, to ask the right questions.
<b>Project Timeline</b>	Scrum# extends the project timeline beyond Scrum, starting at the beginning of the idea until it is delivered.
<b>Organizational Scope</b>	Scrum# focuses on the entire organization, not just local software development teams. The point is to help the entire organization deliver value. Scrum# encourages a perspective of teams working together that are focused on Enterprise wide issues.
<b>Product Scope</b>	Scrum deals with the features/stories of the product backlog being built. Scrum# is concerned both with the features and stories for a particular product and with the entire product portfolio that the product is part of.
<b>Business / Customer Value</b>	Scrum# focuses on delivering value both to the customer and to the Business. For example, eliminating redundancy and maximizing reuse across Scrum teams is also legitimate work.
<b>Technical Practices</b>	Scrum# does not mandate particular technical practices but does require that industry best practices such as patterns and test-driven development.
<b>Product Portfolio Management</b>	Using Minimal Marketable Features assists the team in delivering the smallest useful value as quickly as possible. By providing scope, MMFs make it easier for developers and product owners to be focused.
<b>Tools</b>	Scrum# extends the range of tools from the normal Scrum tools (visual controls, product burn-down and product backlog), to include the many other Lean tools, such as root cause, kaizen, AAR.

Net Objectives provides both fully integrated Lean-Agile training, consulting and coaching solutions for business, management, teams and individuals as well as a complete set of technical training to support these services. Many of our engagements start with an assessment of the effectiveness of your development organization that also gives you a roadmap on transitioning to Lean-Agile methods.

Net Objectives is the training choice of over 20,000 Agile professionals, small and mid-sized organizations and Fortune 100 companies. We offer the most comprehensive Lean-Agile and technical curriculum in the industry; Professional Scrum Certifications, Assessments, on-site Training, custom curricula, Coaching, and Enterprise Agility Consulting Services.

Please call us for a free consultation at **1-888-LEAN-244** (1-888-532-6244) or e-mail us at [sales@netobjectives.com](mailto:sales@netobjectives.com).

For more information, visit our website at [www.netobjectives.com](http://www.netobjectives.com)