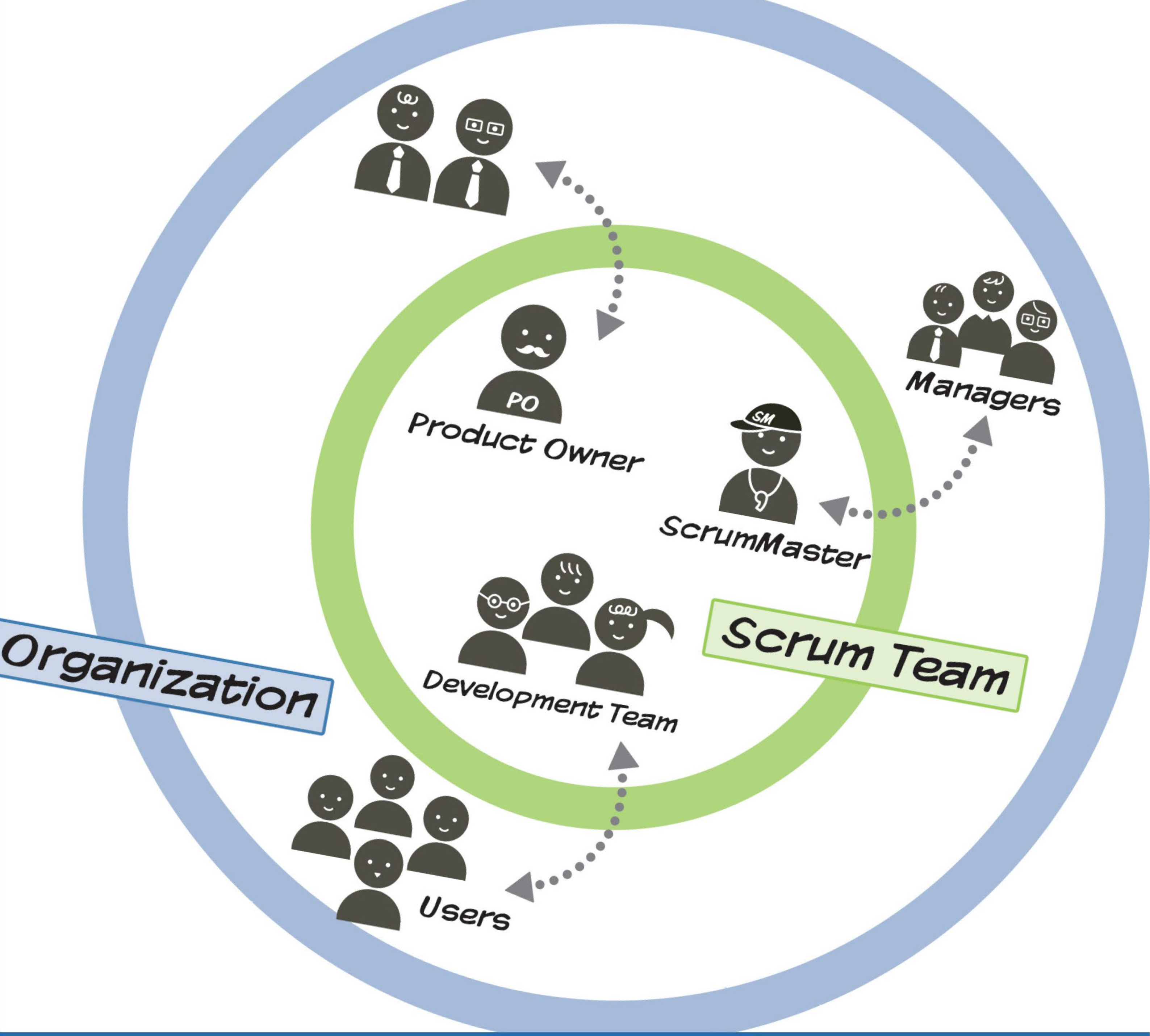


DO BETTER SCRUM

An unofficial set of tips and insights into how to implement Scrum well

VERSION 3 - COMPLETELY REVISED & UPDATED

Peter Hundermark

InfoQueue
ENTERPRISE SOFTWARE
DEVELOPMENT SERIES

Contents

v3.3

Why this guide?

Certified Scrum Trainer and Coach Jim York says: **Scrum is Simple. Doing Scrum is Hard.**

Many people I meet in organizations say that they find it hard to know how to get started with Scrum. Others have teams that are following some Agile practices, yet are far from becoming what Jeff Sutherland terms hyper-productive.

In the four years since I wrote the first edition of this little guide many new and excellent books on Scrum have been published. A few examples that I have read and recommend to my students are Succeeding with Agile [Cohn 2009], Agile Product Management with Scrum [Pichler 2010], Coaching Agile Teams [Adkins 2010], and Essential Scrum [Rubin 2012]. Yet the fact is that most simply don't read them and so there still seems to be demand for brief guides such as this one.

My first aim with this new edition has been to provide a more rounded and complete *reference* to the world of Agile and Scrum, while still sticking to the essence. A second aim has been to update the *practices* described to match what I teach and coach today. Agile is an emergent set of patterns and practices that will, by design, change over time. The third and final aim has been to keep it as *to-the-point* as possible with no 'fluff'. For those who do like to read further, the *reference* section has been expanded significantly.

In the first version I wrote: "I hope this little booklet may be a source of inspiration to help you do Scrum and Agile a little better each day. More importantly I hope it might encourage you to drag yourself, your team and your whole organization away from the old ways of working that simply don't, well, *work* and find new ways that lead to greater quality, faster delivery and above all, more fun." This is still its primary purpose.

One more thing remains certain: you will never become better at doing Scrum (or any Agile method) without practicing. So what are you waiting for? Just go and do it!

Peter Hundermark

Third edition, Cape Town, February 2014

PART ONE

What is Scrum?

Derivation

Scrum is a management framework within which complex products can be developed. Scrum is derived from work in knowledge management, complex adaptive systems and empirical process control theory. The acknowledged genesis is a research paper *The New, New Product Development Game* by Nonaka and Takeuchi [Nonaka 1986]. Indirectly Scrum draws much from what we know as Lean.

Scrum is by far the most popular of the Agile methods. In 2013 72% of teams who would claim to be agile are using Scrum alone or in combination with other methods [VersionOne 2013]. For more information about Agile and other methods, see Chapter 9 – *More on Agile and Lean*.

What is the problem?

- Releases take too long
- Stabilisation takes too long
- Changes are hard to make
- Quality is falling
- Death marches are hurting morale

For decades software developers have been trying to employ defined methods of working and managing projects. Defined methods are appropriate when the inputs are well-defined and the method of converting these into outputs is predictable. Software development and other forms of complex work are not suited to such methods. And the high rate of project failures and customer dissatisfaction illustrates this amply.

How does Scrum help to solve it?

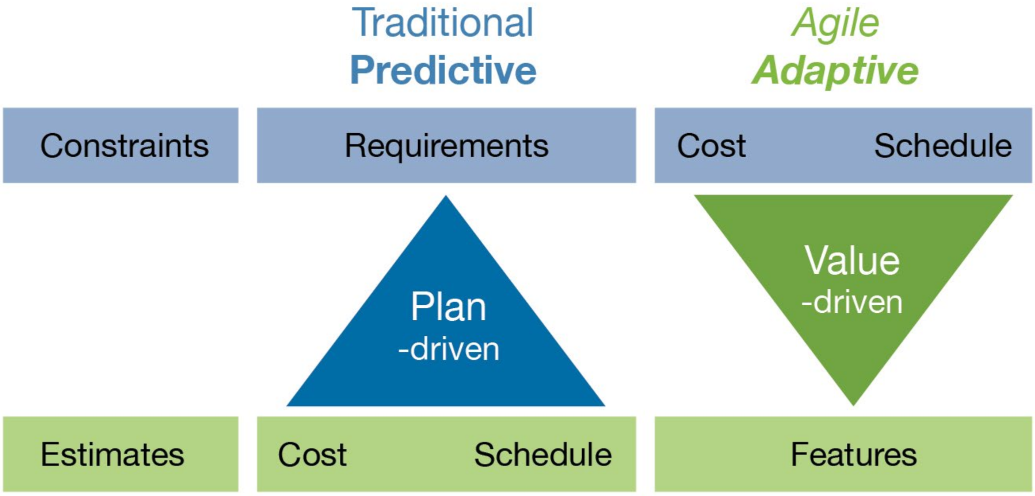
Alistair Cockburn [Cockburn 2008] describes software development as "a cooperative game of invention and communication".

Traditional development methodologies rely on documents to record and pass on knowledge from one specialist to the next. Feedback cycles are too long or even non-existent. Decades of project under-performance has shown these ways of working to be an outright failure.

In August 2012 Gartner published a research paper entitled *The End of the Waterfall as We Know It* in which they claim that long-duration Waterfall projects are the highest risk way to deliver software [Gartner 2012].

Scrum provides a platform for people to work together effectively and relentlessly makes visible every problem that gets in its way.

I have found that C-level managers are able to understand and identify with the following simple model that contrasts the traditional project management model with the Agile model.



In the traditional predictive model, the project manager attempts to constrain scope in order to provide reliable estimates for time and cost. In practice we experience a "triple constraint" or "iron triangle" situation and quality may become an undesired variable. In Agile we accept time and cost as real constraints. The product manager then works with the delivery team to maximize value by delivering iteratively and incrementally. The plan is regularly adapted to match reality. (How could we ever believe the other way around could work?)

I find that senior level stakeholders are delighted to discover that time and cost can be fixed or at least managed in increments. Naturally there is initial anxiety around scope being variable. I overcome this by seeking their permission to conduct an experiment with clearly defined constraints.

The Essence of Scrum

The essence of Scrum is:

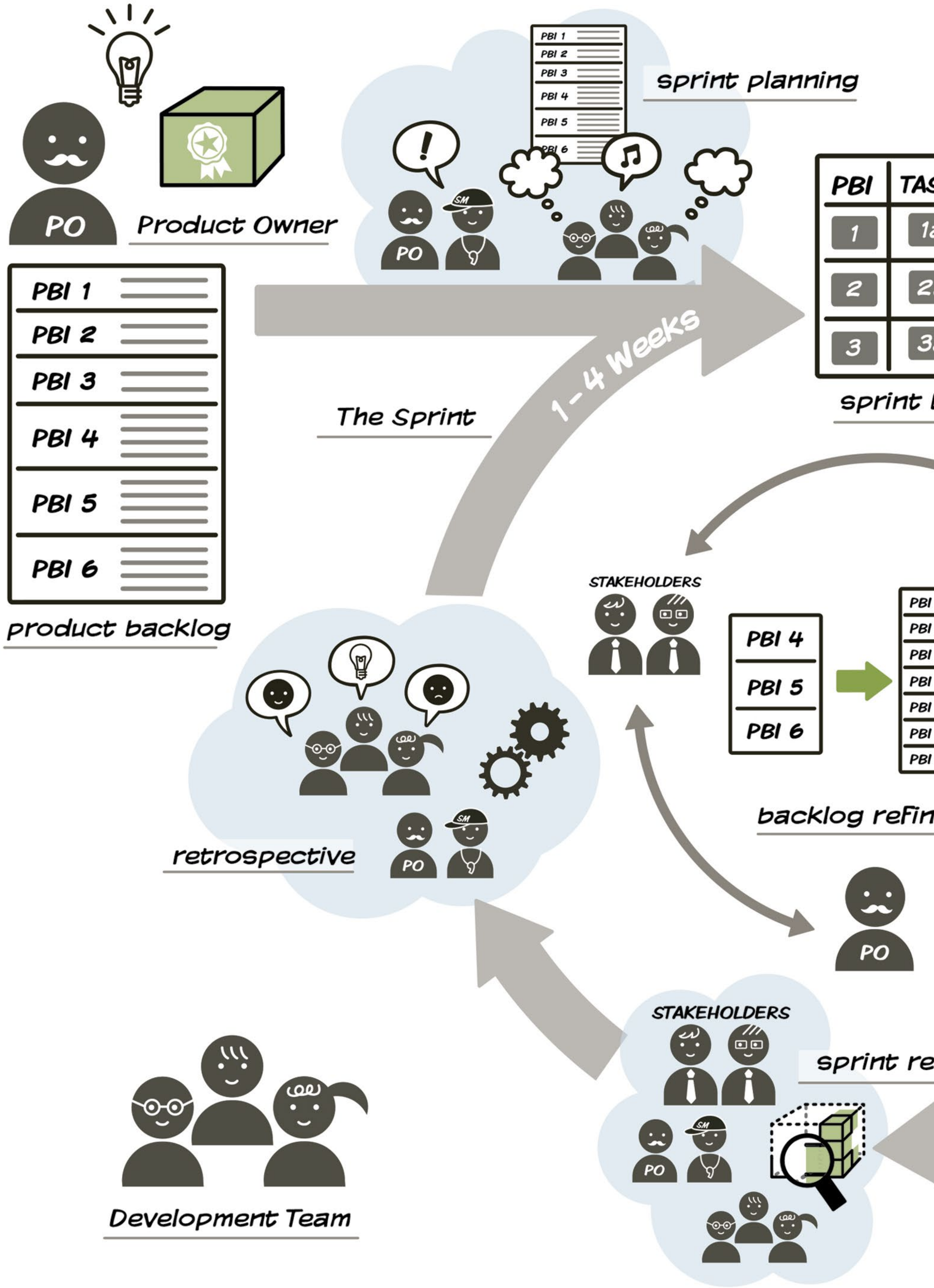
- The team is given clear goals
- The team organizes itself around the work
- The team regularly delivers the most valuable features
- The team receives feedback from people outside itself
- The team reflects on its way of working in order to improve
- The entire organization has visibility into the team's progress
- The team and management honestly communicate about progress and risks

This way of working is based upon the values of Openness, Focus, Commitment, Respect and Courage described in the first book about Scrum [Schwaber 2001].

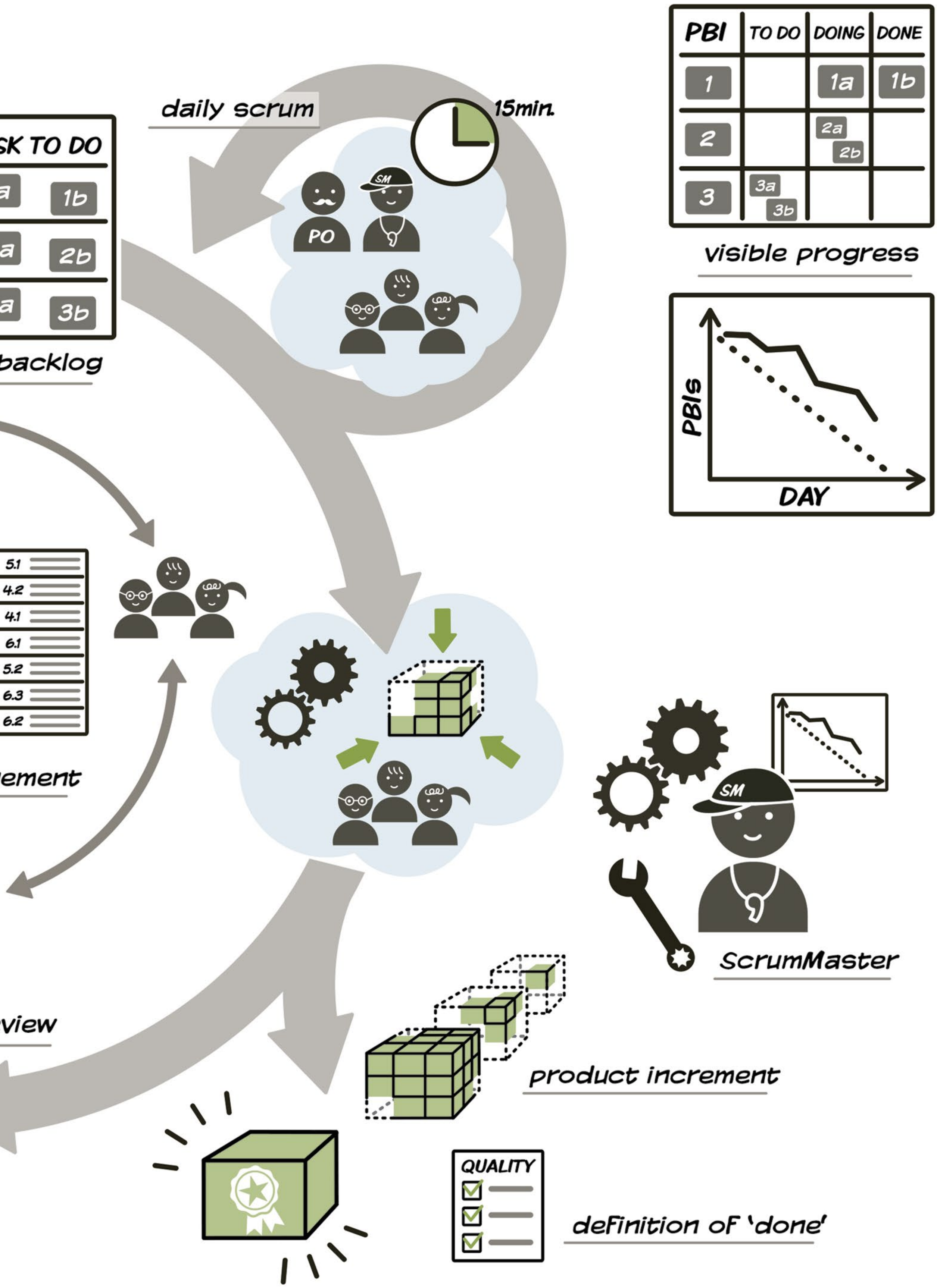
Formal, up-to-date descriptions of core Scrum are available in numerous languages in the form of the Agile Atlas [Jeffries 2013] and the Scrum Guide [Schwaber 2013].

On the next page is a model showing the cycle of events, flow of work artifacts and roles of participants in the Scrum cycle.

Scrum


PO
Product Owner
PBI 1
PBI 2
PBI 3
PBI 4
PBI 5
PBI 6
product backlog
sprint planning
PO
SM
The Sprint
1 - 4 weeks
PBI
TAS
1
2
3
sprint
STAKEHOLDERS
PBI 4
PBI 5
PBI 6
backlog refin
retrospective
PO
SM
PO
STAKEHOLDERS
sprint re
Development Team

Flow


daily scrum
15min.
PO
SM
SK TO DO
a
1b
a
2b
a
3b
backlog
PBI
TO DO
DOING
DONE
1
1a
1b
2
2a
2b
3
3a
3b
visible progress
PBIs
DAY
5.1
4.2
4.1
6.1
5.2
6.3
6.2
ement
SM
ScrumMaster
view
product increment
QUALITY
definition of 'done'

Applicability of Scrum

While Scrum was first applied to development of software products, it is suited to all types of complex work. It is used today to manage software and hardware development, support, advertising and marketing, churches and entire organizations.

Why is Scrum silent on design practices?

Scrum does not attempt to instruct teams how to do their work. Scrum expects teams to do whatever necessary to deliver the desired product. It empowers them to do so. Design practices and tools change and improve all the time and good teams will constantly work to take advantage of these.

If you are developing software you will need to identify and apply a set of software design practices to complement and enable the Scrum management framework. Happily Kent Beck and his collaborators have already packaged a perfect set of practices for you in Extreme Programming (XP) [Beck 2005].

If you are working in a field other than software design, you will need to seek out your own set of design practices.

How does Scrum map to traditional methods?

The short answer is that it does not. Agile and Scrum are based on a different paradigm. Founders Jeff Sutherland and Ken Schwaber have frequently stated that attempts to map defined (prescriptive) to empirical (adaptive) methods are futile [Sutherland 2007].

Some better questions might be: "How can I transition my team and my organization to Scrum? What are the strengths and successes I can build on? What are the likely challenges and how can I prepare myself and my people to overcome them?"

For an erudite discussion on Agile versus traditional methods, one of the definitive articles to read is Martin Fowler's *The New Methodology* [Fowler 2005].

Will Scrum succeed in my organization?

I am often asked for evidence of success of Agile methods. Usually what is sought is proof of it working in situations that are closely analogous to the industry, organization, culture, etc., of the requestor. Sometimes I can point them to examples that I know personally. I also refer them to published surveys and case reports that show that thousands of teams on all continents and in every industry are succeeding in making their world of work better today than yesterday.

More importantly, I think, I can refer them to many people I have witnessed over the past eight years make a personal transition from huge frustration to renewed joy in their work.

Nevertheless success with Agile depends on you! The implementation of Agile methods in your organization will succeed or fail based on the recognition and acceptance that introducing Agile initiates huge and unavoidable changes within the organization.

Learning the rules of one or other Agile method is not the difficult part. It is the effective management of the resulting waves of change at all levels in the organization, including its impact on the organization's culture, that will ultimately determine success or failure.

Scrum offers a way for the organization to unlock more of the unlimited potential of its people. Scrum (and any Agile method) also comes with associated costs, which are measured in multiple currencies, only one of which is money. You need to invest in understanding the economic value proposition that this presents before you proceed much beyond some initial experiments.

Lastly, an organizational transition to Agile will take years. No matter what size your organization, how aligned you think the culture is, or any other factor you imagine may smooth your path, it may take five to ten years to get really good and to institutionalize these new ways of working. So get ready for a long journey. And enjoy the ride!

Takeaways from this chapter

- Scrum is an empirical process framework
- Scrum is based on a cycle of events, a flow of work and a clear definition of roles
- Scrum is broadly applicable
- Success with Scrum depends on your organization's willingness and ability to change

PART TWO

Understanding Scrum

The Scrum Team

The Scrum Team comprises three and only three roles: Product Owner, Development Team and Scrum Master. This composition and interworking between these three roles is foundational and fundamental to the effectiveness of the Scrum pattern. The Scrum Team organizes itself to accomplish the work.

Self-organization

Self-organization does not at all imply a *laissez-faire* approach; on the contrary, self-organized teams are highly disciplined. Self-organization occurs within agreed upon constraints. Within these they are given full autonomy and voluntarily make commitments to one another. They accept responsibility and accountability for delivering to these commitments within the limits of their control. They are encouraged to take reasonable risks and to learn through failure and self-reflection. Self-organizing teams exhibit high trust and possess high intrinsic motivation.

Teams new to Scrum will require some encouragement to explore their new, broader boundaries and to take ownership. They frequently need to overcome strong 'muscle memory' of the poor ways in which they have worked and been managed, perhaps for many years.

Self-organization is not an option in Scrum; it is a core principle. Without this, high-performing teams will not happen. *Caveat emptor!*

Product Owner

The Scrum Product Owner's primary responsibility is to optimize return on investment (ROI) by ensuring that the Scrum Team Members are engaged in delivering the currently most valuable product features.

The Product Owner's primary job is to focus on *effectiveness*, that is building the *right product* for her customers.

The responsibilities of the Product Owner role are:

- Ensuring the existence of a shared vision for the Product
- Managing and prioritizing the Product Backlog
- Helping the Team Members to understand what to build and why
- Accepting the delivered product increment at the end of each iteration
- Managing the release plan
- Informing stakeholders regarding progress and managing their expectations
- Maximizing the economic return of the product.

There is one and only one Product Owner in a Scrum Team. Tobias Mayer has labeled this as the **'What Voice'** [Mayer 2009]. Other people who have a stake in the product are...well...stakeholders!

Metaphor: The Product Owner is a CEO.

» To plagiarize the well-known saying from "Highlander": there can be only one Product Owner in a Scrum Team! All others are stakeholders.

» One common problem relating to this role is the "lame duck" Product Owner. She is not empowered by stakeholders (particularly the Customer or Sponsor) to make decisions. Consequently the Development Team is unable to obtain reliable answers from her. Respect and trust suffer. Motivation drops.

» A second common challenge is the "missing in action" Product Owner. She is never to be found and the Development Team languishes without direction. They will either slow down or make risky assumptions.

Development Team

The Development Team is the collection of people responsible for delivering potentially shippable increments of product functionality at the end of each Sprint.

The Development Team's primary job is to focus on *efficiency*, that is building the *product right* for their Product Owner and users.

The responsibilities of the Development Team are:

- Working with the Product Owner and other stakeholders (often users) to progressively refine up-coming Product Backlog items so they are each well-understood and small enough to fully complete in one Sprint
- Contracting with the Product Owner to deliver a completed increment of the product at the end of the Sprint
- Self-organizing to figure out how to deliver the contracted product increment—and doing so!
- Tracking its own progress daily towards the Sprint Goal
- Ensuring that the product's design remains sound and the code (or other output) is maintainable through ongoing refactoring.

Tobias Mayer has labeled the Development Team as the **"How Tribe"** [Mayer 2009].

» Development Team members are programmers, testers, analysts, architects, designers, writers, and even users—anyone who contributes to doing the work.

» The Development Team is cross-functional, which means that its members between them possess sufficient skills to do the work required to deliver the contracted product increment. Cross-functional does NOT mean that everyone can perform every task. Nevertheless, the best Team Members may be described as "T-shaped" people as opposed to "I-shaped" people [Reinertsen 2009]. I-shaped people have a single specialist skill to offer the team, while T-shaped people complement their specialist skill with additional generalist skills. Therefore they are able to assist their colleagues in multiple ways to deliver the contracted product increment.

- There is no dictated leadership hierarchy within the Development Team. Leadership will emerge to suit the situation at hand.
- The Scrum Team comprises all three roles: one Product Owner, one Scrum Master and three to nine Development Team members. Having fewer than three Team Members has drawbacks such as lack of resilience, difficulty in being cross-functional growing skills. More than nine Team Members will struggle to work as a cohesive team and the communication overhead is too high. Think of the team as a family or tribe who can all keep warm around a camp fire!
- If you have multiple very small teams each dedicated to a product or project, consider rather creating well-formed Scrum Teams and consolidate their work into a single Product Backlog (list of work).
- I strongly advise you to avoid the temptation to fill multiple roles in a Scrum Team or across multiple teams. You will severely compromise the effectiveness of the Scrum pattern. Sorry to be rude, but if you are a novice, what makes you think you can improve on the pattern?
- When the work requires more Development Team members to be involved than can effectively work together in a single team, we call this "scaling" the Scrum implementation. The Scrum community has evolved a number of helpful scaling patterns. You will be well-advised to consult the excellent literature on this topic or call in an expert to assist you if you wish to avoid certain heartache! See Chapter 6 "Scaling Scrum to the Organization" for some more on this topic.

Scrum Master

The Scrum Master manages all aspects of the Scrum Team's process. Since the team is self-organizing around the work, the Scrum Master manages via influence rather than authority. This is a complete paradigm shift for the leadership of most organizations. It is at once the hardest thing to grasp in Scrum and yet is its most powerful enabler of improvement in human performance.

The Scrum Master's primary job is to focus on the Scrum Team's *continuous improvement*, by shortening the *feedback loops* by which they learn. This helps them to get stronger as a team and thereby work better and faster.

The responsibilities of the Scrum Master role are:

- Shepherding the Development Team towards self-organization and continuous improvement
- Working to remove organizational impediments to the Development Team
- Helping the Product Owner to understand and perform his role
- Facilitating Scrum events
- Socializing Scrum to the greater organization.

Metaphor: The Scrum Master is a facilitator, coach, mentor and bulldozer! In fact, she is often called the Team Coach, since the metaphor of a sports coach is often quite apt.

The Scrum Master helps unlock for the Scrum Team the powerful triple intrinsic motivators of *autonomy*, *mastery* and *purpose* so aptly described by Daniel Pink in his seminal TED talk *The Puzzle of Motivation* [Pink 2009] and book *Drive: The Surprising Truth About What Motivates Us* [Pink 2011].

The Scrum Master is a leadership role, yet avoids telling the Team Members what work to do or how to perform it. This is aptly described as Servant Leadership [Greenleaf 2012].

- Many traditional managers imagine the Scrum Master role as some kind of "administrator", perhaps like a junior project manager. I have seen this mistaken understanding result in many difficulties with Scrum adoptions. Agile change agents need to be on their guard for symptoms of this and immediately work to help managers to grasp the value of this new role.
- A related disease is the persistent behavior of allocating a single Scrum Master to multiple teams. Novice teams generally need a full-time Scrum Master. This is especially true when the Scrum Master is also inexperienced and the organization is early in its Scrum adoption.
- Michael James offers us a helpful checklist for all the activities a diligent Scrum Master might do [James 2006].

Other Roles

There is no Project Manager role in Scrum. The responsibilities of the traditional project manager are divided over the three roles in the Scrum Team:

- The Product Owner manages the *product* (and return on investment)
- The Scrum Master manages the *process*
- The Development Team manages *itself*.

This is challenging to individuals who currently fulfill this role and to managers in organizations in which they work. Michele Sliger and Stacia Broderick have written a helpful guide to the transition from Project Manager to Agile Coach [Sliger 2008]. A project manager will usually adopt one of the designated Scrum roles: perhaps Product Owner if she has domain knowledge; perhaps Scrum Master if she has good 'soft' skills; perhaps Development Team member if he enjoys problem-solving.

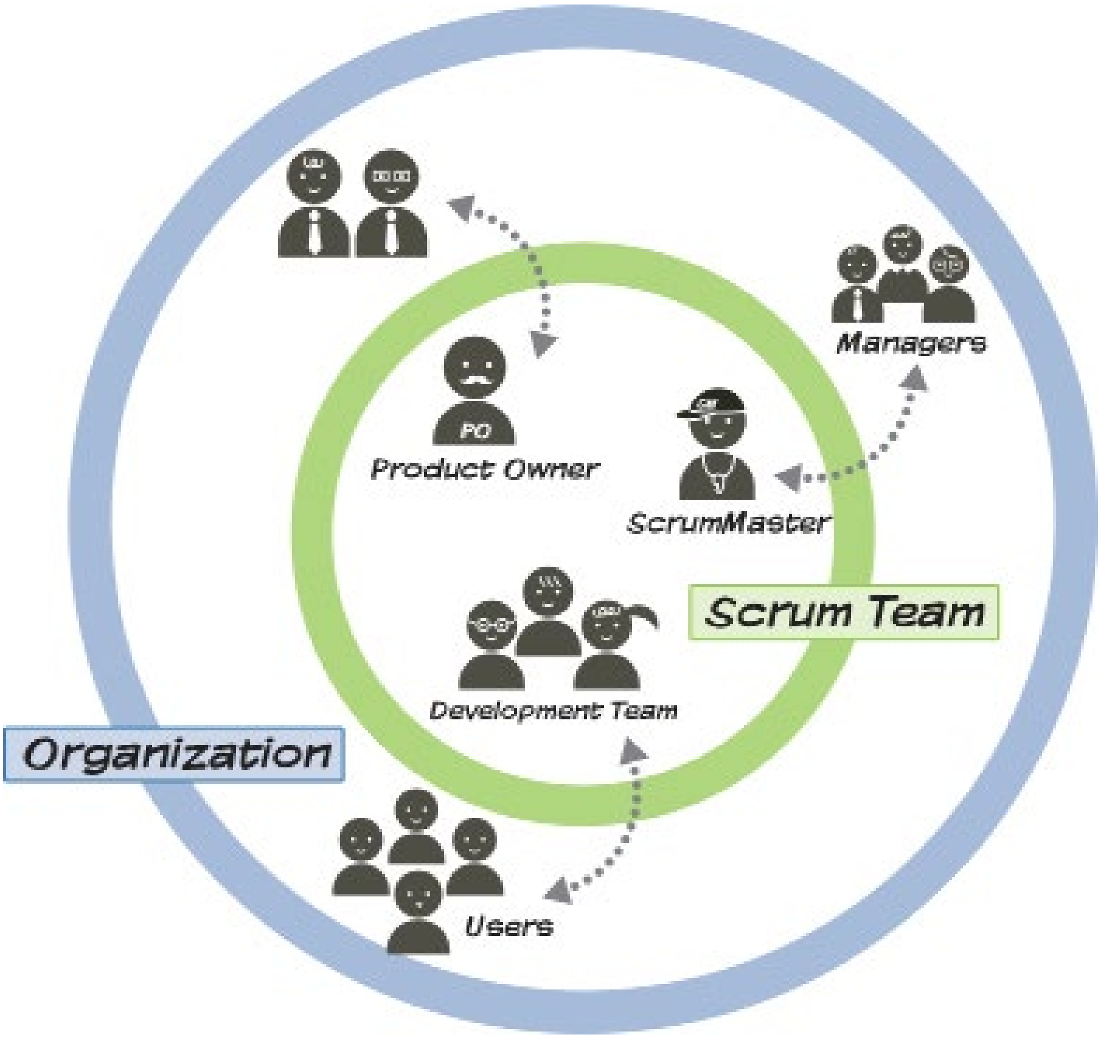
There are no appointed leaders of the Scrum Team beyond the Product Owner and Scrum Master; none is required. The need for line managers is reduced, as teams manage themselves to a great extent. I have seen 40 team members to report directly to a single line manager in an organization that has made the transition to Agile.

Outside of the Scrum Team there are naturally additional roles within the broader organization. I have found it helpful to distinguish three such roles and to describe their interaction with the Scrum Team roles in terms of a simplified communication model:

- The *Customer*, who is financing the work, sometimes known as the *Sponsor*. She talks mostly with the Product Owner. Hopefully she also participates in the Sprint Review events.
- *Users*, who will use the Scrum Team's output. Users will mostly interact directly with the Development Team to help them understand their needs and give them direct feedback about what they have already delivered. Users will often be engaged directly in writing user stories, refining the Product Backlog, giving input during Sprint Planning and feedback during the Sprint Review events.

- *Managers*, who enable the organizational framework within which the Scrum Team operates. Managers and other stakeholders are mostly taken care of by the Scrum Master, who helps them to understand how they can best support the Scrum Team to be effective.

The interactions between these roles is shown in the following diagram. Note that this is just a model that illustrates the major communication flows between the three Scrum Team roles and other major groupings in the organization. In practice communication networks in organizations are far more complex.



Simplified Scrum Communication Flow

Scrum Events

The Sprint

The Sprint is the heartbeat of the Scrum cycle and the container for all other events. It is bookmarked by Sprint Planning at the start and by the Sprint Review and Sprint Retrospective at the end.

The purpose of the Sprint is to meet the Sprint Goal. This is achieved by planning and delivering Product Backlog items to the Product Owner. The completed Product Backlog items are added to the Increment which the Product Owner may choose to release at any time. Quality of the Increment is assured by agreeing on a *Definition of Done* against which every completed item is measured. The Sprint functions to limit the risk that the organization is wasting scarce skills and resources on work that is not the most valuable at the time.

- The Definition of Done (DoD) is an agreement made between the Development Team and the Product Owner. It is made public so that everyone is clear what is included and what is not.
- The DoD is a general statement by the teams and applies to all backlog items.
- Individual backlog items will have specific conditions of satisfaction or acceptance criteria.

Scrum teams choose one, two, three or four weeks as their Sprint duration. The length of the Sprint is fixed and is never extended once the Sprint has begun. This becomes a cadence or rhythm for the team's work. Every event in Scrum is strictly *time-boxed*. The time-box is an upper limit for the duration; it need not use this full time. The time box for each of the Planning (parts 1 & 2), Review and Retrospective events are generally accepted to be one hour per week of Sprint length. For example, for a two-week Sprint, each of these four events will have a maximum duration of two hours. Each day during the Sprint the Development Team holds a Daily Scrum.

Should the Scrum Team or its stakeholders discover at any point during the Sprint that the Sprint Goal is no longer achievable or will no longer serve the organization's needs, the Product Owner is authorized to terminate the Sprint immediately. On termination, any completed (Done) items will be reviewed. I suggest the Scrum Team also holds a Retrospective, since terminating a Sprint is likely a traumatic event for them. Sprint terminations are costly and unusual. Some key attributes of the event are described in the following sections. First, though, I have collected a few experiences I think are worth sharing.

- I often find two-week Sprints a good length to start with. It's a happy medium between one week, which can feel frighteningly short for novice Scrum Teams, and three- or four-week Sprints, which can result in "mini-waterfalls". After three sprints, let the team re-assess the Sprint length and try a different one, if they wish.
- Notwithstanding, I also often start Scrum Teams with one-week Sprints when they are facing short-term delivery deadlines. Short Sprints provide more feedback cycles, which are crucial to helping teams home in fast on the outcomes that are most important to their stakeholders.
- Teams need a minimum of three sprints to grasp the new concepts, break down old habits and start to gel as a team. This three-Sprint rule applies again any time a change is made, such as adding or removing a member, changing the Sprint length, etc.
- As Jean Tabaka advises [Tabaka 2006], never do Sprint planning on a Monday morning. The team is not yet at its best and it is the most common day for holidays and sickness. Never hold reviews or retrospectives on a Friday afternoon.

The team is tired and thinking about the weekend. Therefore choose Sprint boundaries on Mondays to Thursdays.

- Scrum Teams running two-week sprints might be tempted to hold all Sprint boundary meetings in one day. In other words, start the day with the review, then the retrospective; after lunch do Sprint planning parts 1 and 2. The thinking is to get all the meetings out of the way and have 9 full days to do the work. In my experience there are two problems with this approach:
- The team does not get that these meetings are part of the work—in fact the most important part to get right!
- During the last part of the day—Sprint planning 2—the team is brain-dead.
- In such cases I advise conducting the Review and Retrospective in the afternoon and then do Planning the following morning. Yet, as always, let the team try it out if they so wish!

Sprint Planning

The Sprint Planning event marks the beginning of each Sprint. Its purpose is for the Scrum Team to plan the work they will do during this Sprint. It is generally divided into two parts, each with a distinct purpose.

The first part Sprint Planning (often referred to as SP1) answers the question: "What can we deliver by the end of this Sprint?". So we often call SP1 the **"What?"** event. It is really a detailed *requirements workshop*. Starting from the top of his Product Backlog, the Product Owner presents the set of features she would like. The Development Team asks questions to understand the requirements in sufficient detail to enable them to forecast what they can deliver during the Sprint. The Development Team alone decides what is achievable, taking into account the Sprint duration, the size and current capabilities of its members, its Definition of Done, any known holidays or leave days, likely interruptions to their work, and most importantly any improvements it committed to during the Sprint Retrospective held just prior to this event.

The Product Owner is present throughout SP1 to lead the Development Team in the right direction and to answer questions—and they will have many. The Scrum Master ensures that any other stakeholder needed to help the Development Team understand the requirements is present or on call.

Any new backlog items for inclusion in the current Sprint and not previously estimated will be sized immediately during this event. This not, however, an excuse to avoid refining the backlog—see below!

At the end of the first part the Development Team contracts with the Product Owner what they believe they can deliver in the form of *running tested features*. A team with a reasonably consistent record of delivery may use its historic velocity as a predictor (known as "*yesterday's weather*"). This is *velocity-based planning*. Teams starting out can do *commitment-based planning*; that is, the team uses it intuition or "gut" to know what it can do. This may sound unscientific, however in my experience it forces the members to engage their brains and yields rather good results. The backlog items the team forecasts it will complete during the Sprint becomes the basis of the *Sprint backlog*.

If part one is a requirements workshop, the second part of Sprint Planning (SP2) is a *design workshop* and answers the question: "How will we do the work"? So we often call Sprint Planning part two (SP2) the **"How?"** event.

In this session the Development Team collaborates to create a high-level design of the features it has just contracted to deliver. During part two the team may have additional questions regarding the requirements. The Scrum Master must ensure that the Product Owner and, if necessary, other stakeholders are on call to answer them.

Design, as everything else in Agile, is emergent. Also, the event is time-boxed. So it is normal that the team won't get the design perfectly completed in this session and will discover more tasks during the Sprint. This is not a sign that something is wrong. They will simply grab a post-it note and pen and create more tasks whenever necessary during the Sprint.

An outcome of Sprint Planning is the Sprint Backlog, comprising the functional work items, the list of tasks and the plan that the team collectively needs to execute in order to turn the chosen Product Backlog items into running tested features. The Sprint Backlog is most commonly represented on a physical task board. Scrum Teams that are not collocated will likely use a software tool to visualize their task board.

» You will know that SP2 is working when the Development Team is gathered together around the white board discussing noisily or even arguing about the "best" or "right" way to implement a feature.

SUMMARY

Purpose
Plan the work for the current Sprint

Timing
At beginning of sprint; one hour per week of sprint length for each part

Audience
The Scrum Team plus any relevant stakeholders

Inputs
Refined Product Backlog; Improvements from previous Sprint Retrospective; Sprint Goal (if already known).

Outputs
Sprint Backlog

Daily Scrum

The Daily Scrum is one of the primary *inspect and adapt* points in Scrum. The Development Team meets to communicate and synchronize its work and create a plan for the next 24 hours. Since the Development Team is constantly self-organizing towards achieving the Sprint Goal, this collaboration is essential to ensuring continued progress and avoiding work blockages.

The daily Scrum event is not for reporting progress to the Scrum Master or anyone else. In fact the Scrum Guide states that the Scrum Master should ensure no-one other than the Development Team participates [Schwaber 2013]!

The Scrum Master's role is to make sure the Daily Scrum takes place. She will also strive to ensure that any impediments to the Development Team doing its work are bulldozed out of the way as fast as possible. The Scrum Master also ensures the event's 15-minute time-box is honored.

The Daily Scrum is sometimes called the *stand-up*, since it is invariably held standing up. This helps to ensure a short meeting! Jason Yip [Yip 2006] provides a useful guide to help Scrum Masters run this event well.

» As from the July 2013 edition of the Scrum Guide, the Product Owner and other stakeholders are no longer admitted to the Daily Scrum. Given the purpose of this event, I cannot see why they would wish to attend anyway!

SUMMARY

Purpose
Synchronize the Development Team; plan the work for the current day

Timing
Daily at the same time; 15 minutes maximum duration

Audience
The Development Team; Scrum Master may facilitate

Inputs
Sprint Backlog

Outputs
Updated Sprint Backlog; (optionally) updated Sprint Burndown Chart.

Sprint Review

The focus of the Sprint Review is the *product* the Development Team is building.

The Sprint Review is sometimes, and incorrectly, called the "Sprint demo". While it does include a demonstration of the new features completed during the Sprint, its primary purpose is to *inspect* what the Development Team has delivered and gather feedback from the attendees to *adapt* the plan for the succeeding Sprint. Thus it is much more than a demonstration; it is the key point in the feedback cycle that ensures that the product development path is adjusted at minimum monthly.

When asked who should be invited to the Sprint Review, I answer "the whole world". My intent here is to help the Scrum Master and the entire organization understand that the direct attention and feedback of a broad constituency of the organization is crucial to maximizing the value the Development Team will deliver in succeeding sprints. Sprint reviews have many possible outcomes including cessation of the project. In most instances, the Scrum Team is authorized to continue for another Sprint. I find mature teams may already establish the goal for their next Sprint during the review. They find this helps maintain the rhythm or flow from Sprint to Sprint.

- The Sprint Review is a particularly expensive event, since it usually includes many stakeholders and senior people. The Development Team must respect this by setting up and checking their demonstration environment before the meeting.
- I teach Development Teams not to surprise their Product Owner during the Sprint Review: she should be aware beforehand what backlog items they have completed and will be demonstrating.

SUMMARY

Purpose
Stakeholders and Scrum Team inspect the Increment and adapt their plan

Timing
At end of sprint; maximum one hour per week of sprint length

Audience
The Scrum Team plus all interested stakeholders

Inputs
Increment; refined Product Backlog

Outputs
Updated Product Backlog; updated product/release burnup chart; Goal for next Sprint (ideally).

Sprint Retrospective

The Sprint Retrospective is the final event of the Sprint. It follows immediately after the Sprint Review. It is never omitted, no matter what may have happened during the Sprint!

Whereas the Sprint Review is focussed on the product, the retrospective is focussed on the *process*—the way in which the Scrum Team is working together, including their technical skills and the software development practices and tools they are using.

And whereas the Sprint Review is open to the world, the Sprint Retrospective is restricted to the members of the Scrum Team—that is the Product Owner, Development Team members and Scrum Master. Outsiders, including managers at every level in the organization are strictly excluded unless specifically invited by consensus of the Scrum Team.

This rule is to be understood in the context of the goal of the event, which is to inspect at a deep level how the Scrum Team is collaborating and performing and to take action to improve. This often requires deep introspection and sharing, which in turn requires a safe and secure environment. Norman Kerth's Retrospective Prime Directive undergirds this: "Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand." [Kerth 2001].

Boris Gloger [Gloger 2008] offers a simple pattern called *Heartbeat Retrospectives* for new teams to learn to hold valuable retrospectives. Esther Derby and Diana Larsen [Derby and Larsen 2006] provide an excellent model and a host of helpful activities for facilitators of Scrum retrospective meetings. Jeff Sutherland [Sutherland 2011] has produced a very short (2:30) video that does a great job of capturing the essence of what the Sprint Retrospective is about.

» The Product Owner's presence during the Sprint Retrospective is required. He is a full-fledged member of the Scrum Team. As such, his participation in evaluating and improving the Scrum Team's process is as necessary as that of other members.

» The Scrum Master has a key responsibility to assure psychological safety during the Sprint Retrospective. Novice Development Teams with over-bearing Product Owners (it does happen!) may need to hold their first retrospectives without her until sufficient safety can be guaranteed. Such actions must always be seen as temporary and are steps on the journey towards agility.

» Norm Kerth recommends that teams invest 2% of their working time in running retrospectives. I have found this to be good advice. And surely this is not such a high price to pay for learning how to work more effectively and have more fun together? And yet I find teams cutting the time or skipping them altogether. Certified Scrum Trainer Alan Cyment goes as far as saying that all you need to get going with Scrum is an excellent Scrum Master and to start running retrospectives!

SUMMARY

Purpose
The Scrum Team continuously improves its way of work

Timing
At end of sprint (after the Sprint Review); maximum one hour per week of sprint length

Audience
The Scrum Team (others only by specific invitation of the Scrum Team)

Inputs
Anything useful from the current Sprint

Outputs
Improvement items for implementation during the next Sprint.

Scrum Artifacts

Scrum mandates only these artifacts:

- Product Backlog
- Sprint Backlog
- Increment

While not considered artifacts in Scrum, I consider the Scrum Team's *Product Vision* and their *Definition of Done* important assets. I teach all Scrum Teams to create and make both of these highly visible in their work space.

Scrum is purposely silent about all other documentation and artifacts. This sometimes leads to the misunderstanding that agile teams don't need to do any documentation. I coach teams to produce only those artifacts that are *really* valuable to themselves and to others in the future.

My test questions are: "Who is this for?" and "What will they do with it?" This cuts out worthless work and unnecessary destruction of forests!

Product Backlog

The Product Backlog is simply a list of work items, described at a functional level, that need to be done over time. Items may be added to the backlog by anyone, but only the Product Owner has the right (and duty) to determine the order in which they will be executed by the Development Team. Of course a good Product Owner will negotiate this with stakeholders and the Development Team.

Requirements are *emergent*, meaning we do not and cannot know up front every detail about what we want in a product. Therefore the Product Backlog is a living document and requires constant *refinement* to keep it current and useful. Many new items will be added over time; existing items are disaggregated to multiple, smaller items; some items may be removed on realizing that a desired feature is no longer needed. Moreover, items may need to be sized in order to determine the likely relationship between value, time and cost. And, of course, the order of items in the backlog will change as the relative value between them is seen differently today from yesterday.

Therefore backlog refinement is a continuous activity that runs in parallel with every Sprint. The Product Owner convenes regular meetings where the Scrum Team and, if required, other stakeholders refine the Product Backlog in preparation for future Sprints.

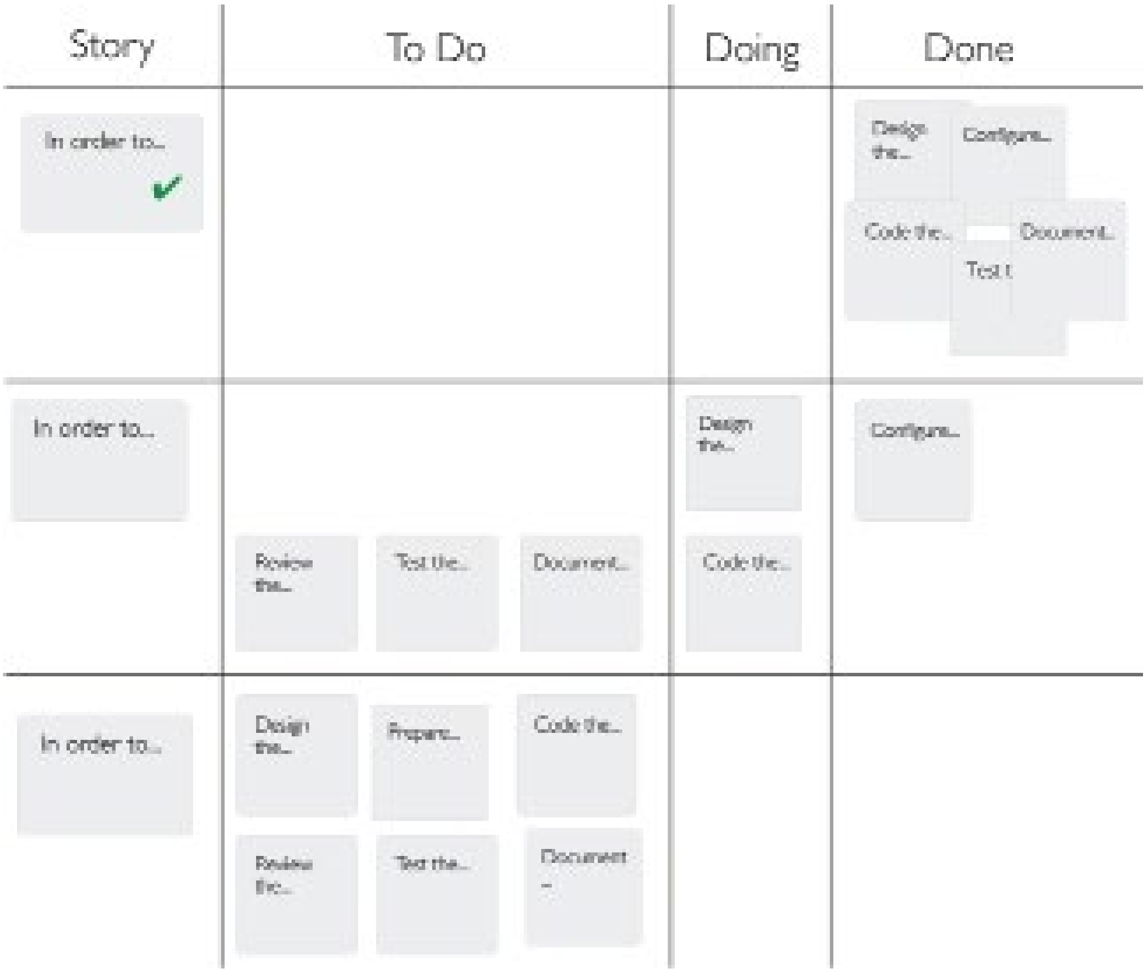
In many cases it is sufficient, and often preferable, to create and maintain the Product Backlog as a set of stories written on physical 125 x 75 mm (5" x 3") index cards. Ron Jeffries [Jeffries 2005] coined the alliterative triplet Card, Confirmation, Conversation (the 3C's) to describe how to work with stories. The stories are written from the perspective of the protagonist. Mike Cohn's book on user stories [Cohn 2004] tells you how.

- An outcome of the refinement activity that surprises some people is the shared learning that occurs by the Development Team members of the business domain in which they are operating. The positive impact of this is products that are a better fit to the users' needs and less rework (a form of waste). It turns out that everyone is a winner when the Product Backlog is kept in good shape!
- A good Product Owner will be adept at saying 'no' to the addition of inessential items to the Product Backlog. This is a crucial activity in support of simplicity, the tenth principle of the Agile Manifesto.
- In my experience the members of the Development Team need to devote 5% to 10% of their time during the Sprint to preparation for forthcoming sprints. This is important to avoid a stop-start effect at the boundary of every Sprint. The implication, of course, is that teams are able to spend 90% to 95% of their time doing the work of the current Sprint!
- I coach teams to hold backlog refinement meetings weekly, no matter what the length of their Sprint. Typically these meetings run for one to two hours and take place at the same time each week. My experience is that regular, short meetings are more effective and less draining on the energy levels of the participants.
- Between these meetings the Product Owner together with business analysts, if the organization includes such roles, will work continuously refining the Product

Backlog. This includes defining the conditions of satisfaction (also known as scenarios) for upcoming backlog items.

Sprint Backlog

Most Development Teams will visualize their Sprint Backlog on a *task board*, which is the physical representation of the list of work they have contracted to do during the current Sprint. The task board is an example of a *Kanban*, a Japanese word meaning token or visible signal. It tells the Scrum Team and anyone else what work they have planned or the Sprint and their current status.



Sprint Burndown Chart (optional)

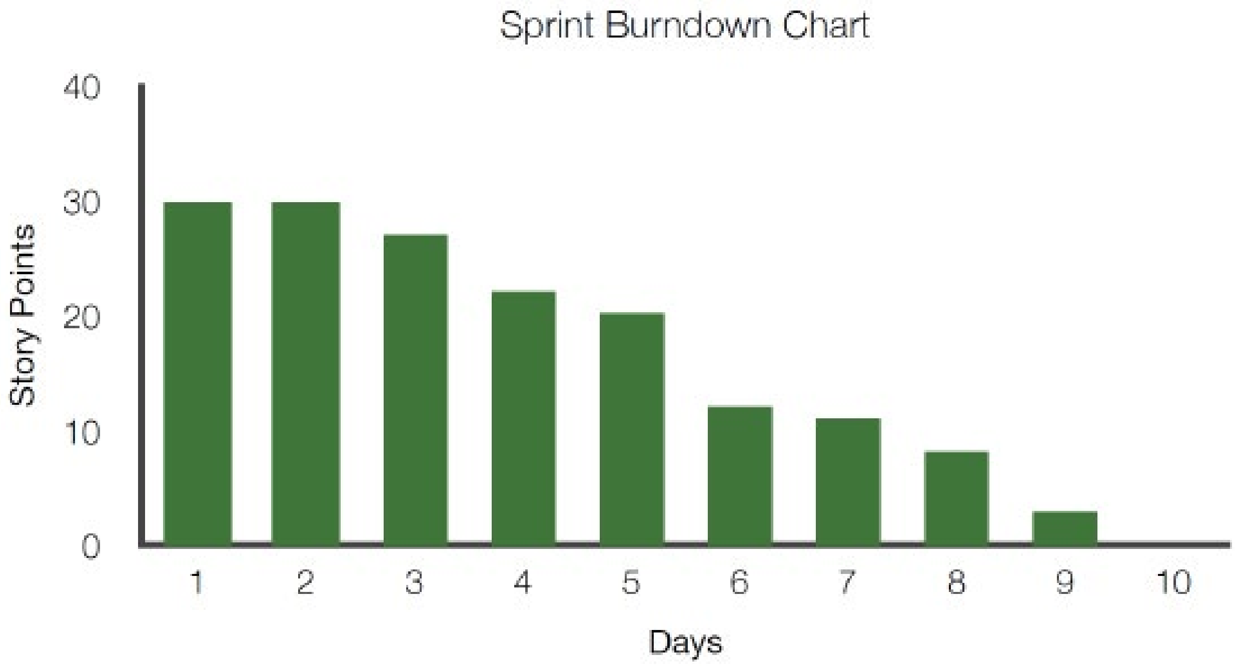
The Sprint burndown chart is not mandated in Scrum, but the Development Team is accountable for managing their journey towards the Sprint Goal. The Sprint burndown chart is designed to be a leading indicator of whether the Development Team will meet the Sprint Goal.

I coach teams to burn down their Sprint in story points. The rationale behind this is:

- Estimating new tasks and re-estimating in-progress tasks requires effort
- Estimates for individual tasks are highly inaccurate
- Estimating in units of time harks back to the bad old ways of working
- Completion of tasks delivers no value; only completed stories (features) deliver value

- Therefore all such effort is not value-adding and simply increases waste.

So in my coaching world the Sprint burndown is just like the product or release burndown (or burnup), except the X-axis scale is days rather than sprints. Jeff Sutherland has supported this as a good practice since 2009 [Sutherland 2009].



Detractors point out that a story-point burndown chart will look jagged rather than smooth, and may tend to flatline until near the end of the Sprint. Exactly! So now there is something obvious to improve on.

Increment

The Increment is the sum of all Product Backlog items that meet the definition of Done by the end of the Sprint. The Development Team will present this at the Sprint Review. The Product Owner will determine when to release it.

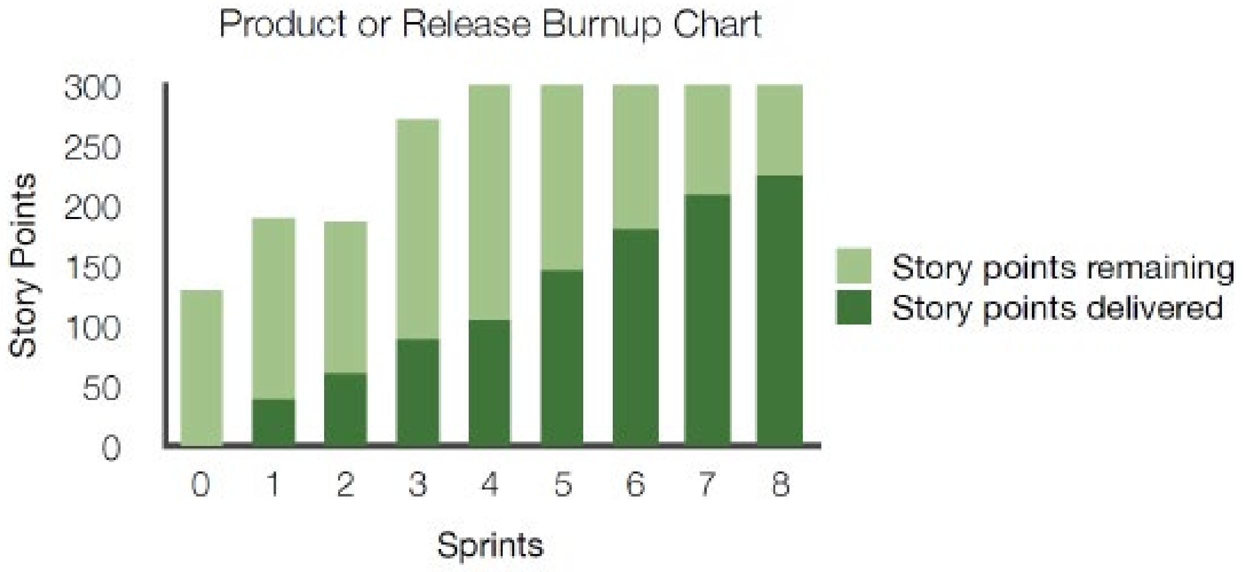
Product or release burnup chart (optional)

The product burnup chart, sometimes known as the release burnup chart, measures the rate of delivery of a stream of running, tested, features over time. This rate is know as the team's velocity. Because features vary in effort, we use a scale to compare their size. The most common method is a unit less relative measure known as story points. Once a team has worked together for a few sprints, it generally establishes its velocity within a definable range. Product Owners then use this velocity to predict the rate at which the team will deliver features in the future, leading to credible release plans.

I coach the Product Owner to use a burnup rather than a burndown chart so that she can simultaneously track changes to the Product Backlog. This is essential, given the dynamic nature of this artifact.

Using this chart Product Owners are able to report progress, determine release dates and predict release scope.

Again, burnup charts are not mandated in Scrum. However I know of no other better way of harnessing the Scrum Team's historical performance data in service of accurate and transparent planning and decision-making. This supports the key principle of empiricism in Scrum.



Takeaways from this chapter

- Scrum is predicated on a self-organizing team
- The structure of the Scrum Team and its constituent roles
- The cycle of events in the Scrum flow
- The artifacts essential to Scrum.

PART THREE

Adopting Scrum

Starting Scrum

Ken Schwaber [Schwaber 2007] says there is nothing that needs to be done before starting Scrum. I interpret this to say that there is no tailoring of the process needed in order to start, as is required with heavy methodologies. Nevertheless the literature is thin on how to get going and probably all of us struggled getting our first Scrum team going without outside help.

The best thing you can do is hire an experienced coach. Failing that, you can try using a pattern that has worked for me with dozens of teams.

Obviously you need a Scrum team. This means a Product Owner, a Scrum Master and three to nine Development Team members. If this is not yet in place, you may want to start with some introductory workshops to check the organizational readiness for Scrum, identify stakeholders for the project or product and form the initial Scrum Team who will do the work. Refer to Chapter 8– *Getting Help* for more information on these activities.

Then follow this sequence of steps, which will be detailed in the next pages.

Day 1: Training	#1 Train the Scrum Team in the basics of Scrum
Days 2 & 3: Product Workshop	#2 Establish the vision
	#3 Form the initial Product Backlog
	#4 Order the backlog items by value
	#5 Size the backlog items
	#6 Re-order the backlog, as necessary, by additional factors
	#7 Create a rough-cut release plan
Day 4: Sprinting	#8 Plan and start the first Sprint

» I start new teams with a full day's training in the essentials of Agile and Scrum. This is sufficient to start a Scrum Team that has an experienced Scrum Master / agile coach to support them during their early sprints.

» Steps two (visioning) to seven (initial release planning) can be completed comfortably in a two-day product workshop attended by the entire Scrum Team. The best workshops are attended by additional stakeholders like enterprise architects, managers and business people.

» On day four, without fail, the Scrum Team is able to commence sprinting!

» If the organization is able to provide the Scrum Team with an experienced Scrum Master, she will facilitate days 2–4 and I will coach her. If not—which is more often the case—I will do so with the novice Scrum Master observing. I try to follow the "see one, do one, teach one" learning principle.

#1 Train the Scrum Team

In my team training I use a lot of group exercises and games to illustrate the principles.

Core topics that I cover on day 1

- Agile values & principles (Agile Manifesto)
- Empirical versus defined processes
- The Scrum flow (cycle of meetings plus artefacts)
- Roles and responsibilities (3 Scrum roles plus more)
- The power of self-organization
- The value of collocation
- The importance of trust
- Cost of multi-tasking; limiting work-in-process

Additional topics that I teach just-in-time during days 2–4

- Using user stories for understanding requirements
- Using scenarios for acceptance testing
- Concepts of size and velocity
- Agile estimating
- Done!
- Using a task board
- Monitoring progress

#2 Establish the Vision

Katzenbach and Smith [2002] have confirmed that having clear goals is essential in the creation of high-performing teams.

The Product Owner will often share her vision for the product. If not, we co-create it with the whole Scrum Team. One technique I use for this is to get each member to write her personal version of the vision. Then members pair up and work to create a single shared vision statement from their initial efforts. The process of pairing up continues until the entire Scrum Team collaborates to formulate a single statement.

This exercise utilizes the power of the group and results in greater commitment to the resulting vision. The team will display the vision prominently in their work space. Visioning takes one to three hours.

#3 Form the initial Product Backlog

The next stage is to hold a user story-writing workshop. It is advantageous to involve business stakeholders here. Certainly the whole Scrum Team is involved.

Writing good user stories is simply a matter of practice. The Pareto principle (80/20 rule) applies here, as always.

Mike Cohn [Cohn 2004] has provided an excellent starter guide to writing good user stories. I encourage every Product Owner to have his personal copy always to hand!

I like to use the User Story form in which the value to the user protagonist is stated first, as shown below. This forces focus on the business reason as the primary purpose of this way of understanding users' needs and planning the work.

> Template:
> In order to «**reason**», as a «**role**», I want «**functionality**».
>
> Example:
> In order to **obtain cash conveniently**, as a **bank customer**, I want **to withdraw cash via an auto-teller machine (ATM)**.

Moreover, I favour using the Scenario construct for specifying acceptance tests from Dan North's Behaviour Driven Development (BDD) model [North 2006], as shown below.

> Template:
> Given **«initial condition(s)»**
> When «**event**» occurs
> Then «**expected outcome(s)**».
>
> Example:
> Given **I have $20 in my bank account** AND **I authenticate myself successfully**
> When **I request a $10 withdrawal**
> Then **I get two $5 notes AND a printed receipt**.

Bill Wake suggested the helpful INVEST acronym for the attributes of a good user story [Wake 2003].

- **Independent**—ideally can be implemented in any order
- **Negotiable**—and negotiated
- **Valuable**—to the customer
- **Estimatable**—enough to rank and schedule it
- **Small**—and with short descriptions
- **Testable**—I could write a test for it

At a minimum the Product Backlog needs to contain enough items for the team to plan the first Sprint. More commonly, the backlog contains all the items that make up the next planned (or hoped-for) product release.

As a guide, the initial backlog should be 80% complete (but not ordered or sized) by the end of day one. The first part of day two can be used to add the final 20% and to resolve any open questions. The overnight break is a very useful hiatus that may spark fresh thinking about the work.

An advanced Product Backlog technique named Story Mapping has been developed by Jeff Patton. You will find more detail on this topic in Chapter 7– *Getting Better*.

#4 Order the backlog items by value

The backlog is now ordered by business value. This appears easier to say than to do well. Many Product Owners I have observed use some combination of HiPPO (Highest Paid Person's Opinion), who is shouting the loudest, and guessing. These techniques will not reliably help to maximize value delivery to your customers.

At the minimum, the Product Owner's subjective judgement of the value of one feature against another might be a starting point. Somewhat better would be a quasi-quantitative relative weighting using affinity groups or planning poker.

However it is done, it is a core responsibility of the Product Owner to order the backlog. Better still would be to measure the actual economic value achieved against that predicted.

#5 Size the backlog items

Project planning has always had estimation as the lynch-pin. Project managers like myself have spent weeks calibrating complex models with historical data.

The simple reality is that the best of these techniques yield no better results than much simpler and faster techniques such as planning poker and affinity estimating. Planning poker works partly because it has a solid basis in theory, but mostly because the people who estimate are those who will do the work. Who would have thought that?

Planning poker is rather fast. A practiced team will estimate at an average rate of 3 minutes per work item. Planning poker is accurate. Estimates using planning poker are as good as the best traditional methods. And planning poker is fun. It takes away the pain usually associated with this topic. Commercial planning poker cards are obtainable from several sources at a cost of about $10 per set of four packs. You can also print your own on plain card for next to nothing.

Affinity estimating is even faster than planning poker. It is great for getting started when we have an entire backlog to estimate and time is more important than great accuracy and information sharing. I employ this technique with novice and advanced teams alike whenever estimation is regarded as necessary and valuable.

We need to understand a few key concepts about agile estimating:

- We size items relative to one another. Why? Because as humans we find this more natural and the results are more reliable. So it's easy to agree that "this story is twice the effort of that one" even though we don't know how long each will take to implement.

- We size items using units of effort rather than time. Why? Because it allows us to separate the rate at which a team works from the size of the work. This shields us from having to change our estimates according to who does the work, or as the teams skills and capacity change over time. We use "story points" as units.
- We use a non-linear sizing scale because the difference between a "1" and a "2" is obviously more meaningful, relatively, than that between "20" and "21". My personal preference is to use the pseudo-Fibonacci numbers : 1, 2, 3, 5, 8, 13, 20, 40 and 100. I define 1 to 8 as the size range of features a team can deliver in one Sprint. Why? Because it turns out that humans are rather accurate within about a single order of magnitude of size difference. The higher numbers are reserved for large stories ("epics") that will need to be split into smaller stories before they can be taken into a Sprint.

#6 Re-order the backlog by additional factors

After sizing the backlog items we may find that some items should be re-ordered. Factors to take into account in addition to the business value include:

- Size: we will naturally choose to implement a simple (small, less effort) story ahead of a complex (large, more effort) story if they have similar business value.
- Learning: we may choose to implement a story earlier if it will help the team learn about the business domain or a new technology.
- Risk: we may choose to implement a story early because doing so will mitigate an identified risk. Obvious examples are items that will help establish performance and scalability limits.

We must always remember that the Product Owner has final say on ordering the backlog.

#7 Create a rough-cut release plan

Once we have ordered and sized the backlog the next step is creating the initial release plan. To do this we need an estimate of our team's velocity. However we haven't done any work yet, so we use a simple technique known as *commitment based planning*.

First, of course, we must be sure to know the team size during the Sprint—whether any member will be away on leave, training, etc. And we must choose the Sprint length—I usually recommend two weeks for a new team. And we must create a definition of *Done* for the team—what does it mean when the team says it has completed a story.

The Scrum Master now picks the first item from the top of the backlog and asks the team "can you complete this item during the Sprint?". She continues to do this until the team is no longer confident to add items. Counting up the story point values of all the committed items, the team has its initial estimate of *velocity*.

This velocity value is used to apportion the remaining backlog items (at least those that have been estimated) over succeeding sprints. This list of items attached to sprints is our initial release plan. It is accurate? Certainly not, but it is probably more credible than anything a project manager could produce before at such an early stage of a project. And as we start work and complete one Sprint after another, we will begin to chart our actual velocity and use this as a predictor of future output. So the release plan is a living thing that becomes more and more reliable as we progress. See Chapter 7 – Getting Better for more on release planning.

- Don't let the product workshop drag on. It is possible within two days for any team and any product to prepare enough backlog items for the first few sprints at least. There is far more value in getting going, inspecting the results and adapting your plan than by striving to formulate the perfect Product Backlog.
- With a new team, new domain or new technology, try to avoid having to create a release plan until you have completed the first few sprints. A release plan based on wild guesses is not helpful to anyone. Rather get three to five sprints under the team's belt and then offer your first plan based on the observed velocity range.
- Scrum emphasizes empiricism as the best way to manage the complexity present in product development. This promotes transparency and acceptance of reality. Traditional release planning and monitoring techniques measure conformance with a plan. Be careful not to allow your release planning and monitoring techniques to subvert the very benefit you seek from Scrum.

#8 Plan and start the first Sprint

Having concluded your product workshop, it's time to cut the cackle and start doing the work. The elegance of Scrum is that we can always start sprinting with a brand new team by day four. We run a standard Sprint Planning meeting, usually first thing in the morning. Assuming a two-week Sprint length, this will be time-boxed to two hours for each part; four in total. So by lunch time or early afternoon the team can start working on the first tasks for their first story. Quite an achievement compared with most other approaches.

Takeaways from this chapter

- Scrum requires minimal preparation to start
- A pattern for starting up a new Scrum Team within three days
- Construction and ordering of the Product Backlog
- Creating an initial release plan.

PART FOUR

Collocation and Team Spaces

Alistair Cockburn [Cockburn 2007] defines software development as a *cooperative game of invention and communication*. The Agile Manifesto says developers and business people must work together daily and that face-to-face conversation is the best way to transfer information.

In order to collaborate effectively, there is no single factor that delivers more value than collocation. A study of collocated software development teams [Teasley, Covi, Krishnan, & Olson, 2000] has shown that they are twice as productive as non-collocated teams.

The definition I use for collocation is that team members sit no further than 6 m (20') from their farthest member. This has to do with the ability to maintain normal visual and aural contact with other team members. It follows that there is a limit to the number of people that can fit into such a space. In practice this number is 8 to 10, which happily matches the maximum size of a Scrum team. Furthermore, every team member must be able to see every other member without doing more than turn her head or swivel in her chair. This means no dividers between desks—good-bye Dilbertville!

For no good reason beyond habit, organizations—note not teams—resist changing their environment to facilitate effective collocation. This is despite evidence that collocated team spaces require no more floor area, on average, than present layouts. Management will challenge you about building dry walls, believing them to inhibit flexibility. This is simply not true—flexibility is required in the organization structure, not team spaces.

My experience is that 90% of Scrum teams in organizations new to Agile struggle on with their miserable work spaces for a year before their management is persuaded to make the necessary changes. And once they have been made, everyone agrees that the improvement was immediate and measurable. Why is this?

In the hope that this absurd resistance will diminish, I offer a simple layout of a collocated team room on the next page. It goes beyond the scope of this little guide to provide all the reasoning behind this design—please contact me if you want additional information.

Some team space layout parameters:

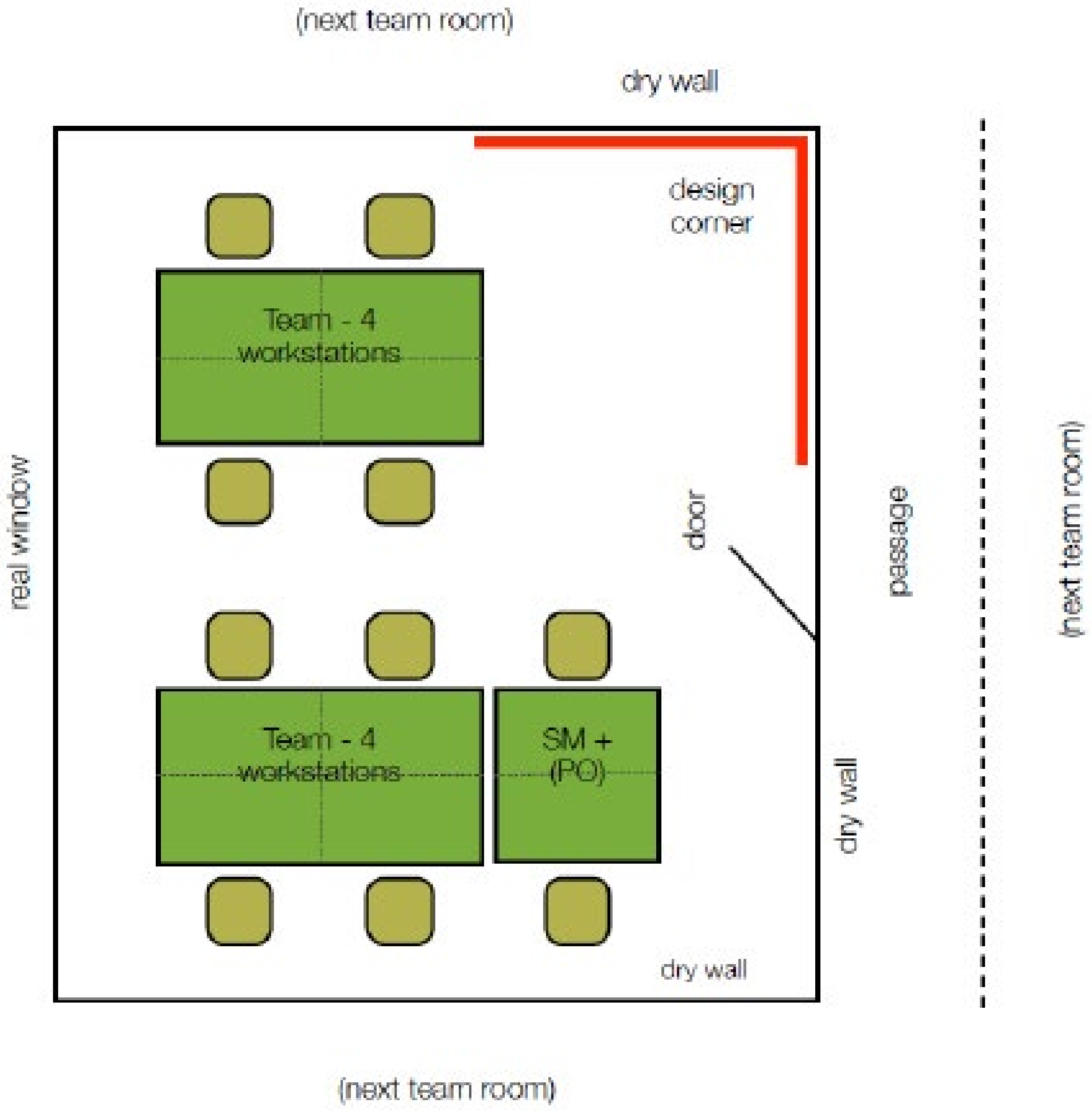
- Rectangular desks to facilitate pairing
- 1.5-1.8 m x 0.8 m (5-6' x 2'8") is a good desk size
- 1.2 m (4') space from desk to wall
- Design corner with white boards 3 x 3 m (10' x 10')
- Typical room size 7 x 8 m = 50 to 60 m^2 (23' x 27' = 550 to 650 sq. ft.)
- Scrum Master can see the whole team
- Product Owner has a floating (not permanent) seat in the room
- Dry walls with internal windows create noise barriers and enough space for information radiators without reducing light.

» Team rooms significantly reduce the need for meeting rooms—teams can do Sprint planning, daily meetings and reviews in their room.

» Be warned that your architect or office designer may not be much help to you. She probably has not been trained in the design of collaborative team spaces.

» The cost of the changes (drywalls and furniture) will be recovered in one week to one month! Sounds unbelievable, doesn't it?

- Given that Scrum is predicated on a self-organizing team, you might try challenging your management to "put its money where its mouth is" and let the Scrum Team determine the layout of its own workspace. After all it's just a decorating decision!



A Typical Collocated Team Space Design

Takeaways from this chapter

- Collocated teams are twice as productive an their non-collocated counterparts
- A simple layout for a collocated team space
- The cost of erecting collocated team spaces can be recovered within a few weeks.

PART FIVE

Measuring

Purpose and perils

As soon as the team understands the basics of the Scrum framework and has some idea of its velocity—usually by the end of their third Sprint—it is probably time to start measuring. Managers in any organization that has even a hint of a controlling culture will want before long to be able to measure the results of a team or organization's transition to Agile methods. How else will you know whether your agile journey is bearing fruit, and what kind of course corrections are needed? And the earlier you begin, the sooner you will have data to support your further decisions.

However the act of measuring brings about a change in the behavior being measured. Eli Goldratt famously said: "Tell me how you measure me, and I will tell you how I will behave" [Goldratt 2006]. And herein lies the challenge in choosing metrics that will help promote the outcomes we want and avoid those we don't. This will differ from one o*rganization* to another, from one *team* to another and from one *time* to another. As the saying goes: "context is king". It follows that metrics, like everything else, must be regularly inspected and adapted to our changing needs.

Moreover, management guru Peter Drucker tells us: "Knowledge workers have to manage themselves. They have to have autonomy." [Drucker 2001]. In keeping with this edict, we must engage the team members themselves in deriving and collecting their own metrics. A good place to start would be to figure out why we have metrics at all.

I would argue that the true value of metrics is as a learning tool. And if we take a systemic view, we should arguably be engaged in what Chris Argyris refers to as double-loop learning. Double-loop learning occurs when error is detected and corrected in ways that involve the modification of an organization's underlying norms, policies and objectives [Argyris 1977].

Starter metrics

With all this in mind I hesitate to offer any concrete metrics here. Nevertheless I will offer a few "starter" metrics for consideration. Most are straightforward to implement; some will require more effort. These are based on my own research and that of other agile practitioners, notably Pete Behrens [Hundermark 2009]. I would encourage you to view the video and slides to get both some background on metrics, an explanation of why I have chosen these particular metrics and some guidance on how to set them up.

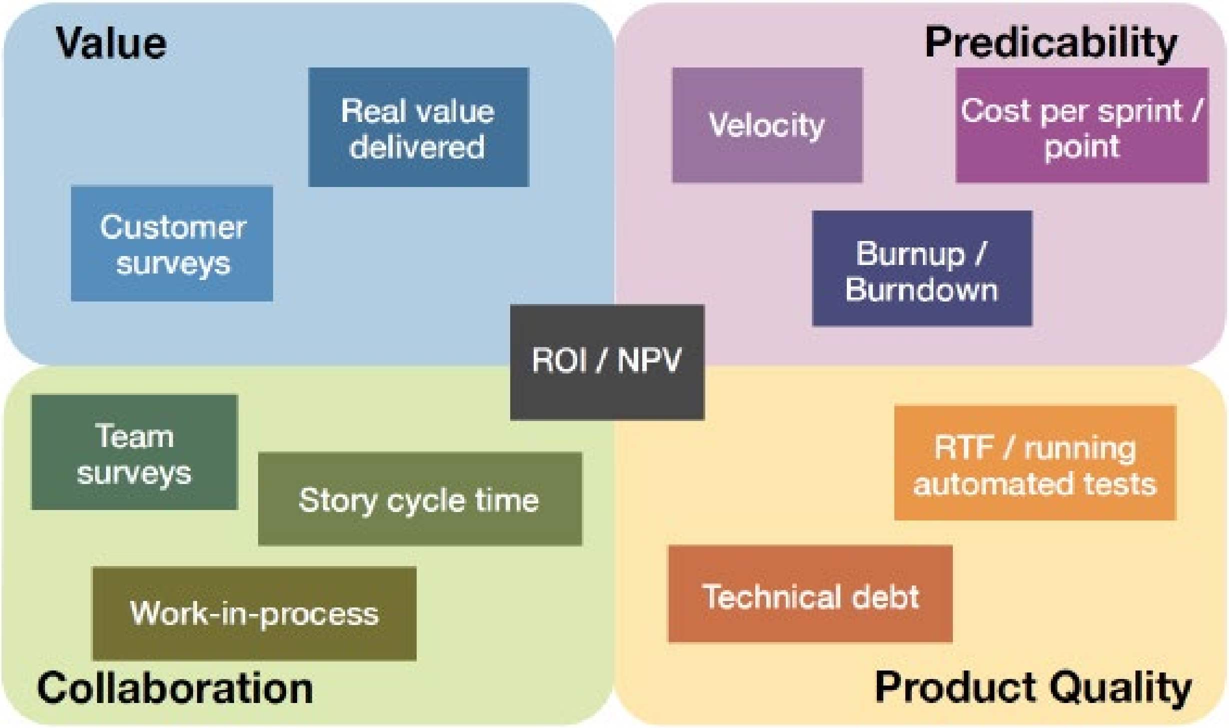
Simple metrics:

- Customer and team surveys
- Velocity chart
- Burnup chart
- Running automated tests
- Technical debt
- Work-in-process / story cycle time
- Cost per Sprint / point / feature

Metrics that require an ability to measure value:

- Real value delivered
- ROI or NPV

These metrics are arranged in the diagram in four dimensions of value, predictability, quality and collaboration, which I also explain in the video.



Lastly, be vigilant about how the metrics are used. For example, a velocity chart, whose purpose is to enhance predictability, may be abused by ascribing the productivity to it, a purpose for which it is not designed and therefore should be used. For every metric you need to carefully and regularly compare its intent with the observed outcome. This is a non-trivial activity.

Key Points

- The purpose of metrics is learning. So start early. The sooner you establish a baseline, the earlier you will be able to begin to use data to understand your results and feed this back into your systems to make better decisions.
- Don't let some group of managers in their "ivory tower" determine metrics. Work with your team(s) and stakeholders to determine what needs measuring and how to measure it.
- What you measure today will not be the right thing to measure tomorrow. Get comfortable with the idea that your team's metrics will be in a more-or-less constant state of flux. CIO and Agile coach Marius de Beer reports that the median lifetime of metrics in his multi-national organization is three months [De Beer 2013].

Takeaways from this chapter

» Scrum Teams must measure themselves
» Metrics are a tool for learning
» Metrics need to be adapted over time based on context
» Some metrics your teams might consider trying.

PART SIX

Scaling Scrum to the organization

Here be dragons!

The first rule of scaling Scrum to multiple teams is "don't". In other words, be sure that you need more than a single, effective Scrum team to do the job before you consider using multiple teams. One study of a single team with four to eight members showed it to be 35 times as productive as the standard [Coplien 1994]. Scrum founder Jeff Sutherland has documented many cases of "hyper-productive" Scrum teams [Sutherland 2003-2012].

Before you attempt to introduce Scrum into multiple teams, do it with one team. Do it "by the book". Learn from your mistakes. Walk before you run. Do it well before you try to change the rules. Follow the *Shu Ha Ri* learning model (see Chapter 7 for an explanation of *Shu Ha Ri*).

My own experience with scaling Scrum is that starting small and adding only what is needed as I go is less painful and more successful than starting with a "heavy" framework and trying to find bits I can remove. Any attempt to scale Scrum will amplify the need for good design and development practices. Be sure to focus attention on getting these right. Every scaling situation is different and requires its own solution. As with everything in Agile, this has to be arrived at via trial, inspection and adaptation.

The greatest challenges to adopting Agile at scale are organizational. Therefore any scaled implementation needs careful attention to communication, culture and change management. I suggest that you accept that there is no "silver bullet" solution to doing Scrum (or any form of Agile) at scale. With this mindset I suggest you read broadly what others have done and are doing. I especially recommend the Larman and Vodde books (above) as well as Henrik Kniberg's papers and presentations. Still worth reading is Ken Schwaber's *The Enterprise and Scrum* [Schwaber 2007]. Mike Cohn's *Succeeding with Agile* [Cohn 2009] also has some useful guidance for applying Scrum at scale.

If you are a novice and are serious about helping your organization get better, find and hire an Agile coach who has a verifiable pedigree in applying Agile at the scale you need. This will shorten your learning curve and lessen the pain!

Lastly, be prepared for a multi-year journey towards organizational agility. Whilst I have seen tremendous improvements within just a few months, serious "sticky" change, especially at scale, will take a long time: think five to ten years. The good news is that it can be a lot of fun with rewards all along the way.

Scaling patterns and frameworks

As Agile has entered the mainstream it is self-evident that it be applied at larger scale and in larger organizations. This has created both challenges and opportunities for agile practitioners.

A number of outstanding contributions have been forthcoming "from the trenches" of Agile practice. Ken Schwaber's book *The Enterprise and Scrum* provided the opening salvo [Schwaber 2007]. Notable contributors in this space are Henrik Kniberg [Kniberg 2008] [Kniberg & Ivarsson 2012], Mike Cohn [Cohn 2010], and Mike Beedle [Beedle 2014]. Pride of place in my personal opinion goes to the comprehensive work of Craig Larman & Bas Vodde, captured in two books: *"Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum"*

[Larman and Vodde 2008] and *"Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum"* [Larman and Vodde 2010].

At the same time older frameworks such as the RUP (Rational Unified Process) have had lipstick applied and been presented with an "agile" flavour. Such offerings need to be approached with care and circumspection.

This topic of Agile in the enterprise is an interesting and important space with much new practice emerging in recent years. I have written about the scaling topic in some detail in my blog [Hundermark 2014].

My own scaling approach

A helpful aphorism to bear in mind is "scale principles not practices".

I offer below some principles and related practices I have drawn from Don Reinertsen's *Principles of Product Development Flow* [Reinertsen 2009], from Larman and Vodde's books, from conversations with other agile coaches, and from my own experiences with applying Scrum at scale with my own consulting clients during the past eight years. Understand that this is only scratching the surface of what it means to apply Agile at scale.

Scrum Scaling Principles & Practices

Minimise cross-dependency between teams, so

- Form feature teams rather than component teams, and
- Ensure each team is cross-functional.

Stick to the "right" team size, no matter what, so

- Merge multiple products into one backlog feeding a single team, or
- Let a single backlog feed multiple teams.

Reduce skill silos and dependencies within teams, so

- Grow T-shaped people

Employ small batches to reduce cycle time, reduce variability and increase efficiency, so

- Avoid large work items in Sprints, and
- Use a regular cadence for all Sprint meetings.

Shorten queuing times for the waiting work, so

- Feed multiple, synchronised teams from a single backlog.

Exploit scale economies of multiple teams, so

- Synchronise sprints for multiple teams.

Retain slack to achieve flow, so

- Allow teams to *pull* from the backlog, based on their observed capacity, and
- Challenge teams to finish early as least as often as they finish late.

Keep feedback loops short, so

- Ensure all teams' outputs are tested and integrated into the Increment every Sprint, and
- Work to eliminate constructs like "integration" or "hardening" sprints.

Optimise the whole, so

- Measure outcomes at the highest possible level, and
- Let teams seek on their own local solutions.

Pay attention to quality, so

- Ensure "technical debt" is reducing, not increasing, and
- Fix errors as soon as they are found.

Pay attention to communication, so

- Institute formal meetings to synchronise teams.

Pay attention to learning, so

- Form communities of practice for different disciplines to share learning, and
- Hold large group retrospectives on a longer cadence (e.g. for releases).

Takeaways from this chapter

» Scaling Scrum to the organization is non-trivial

» Some scaling patterns and principles

» Some guidance and warnings regarding specific approaches

» Some advice on how to proceed.

PART SEVEN

Getting Better

Shu Ha Ri

The Japanese martial art Aikido defines a model of learning named *Shu Ha Ri*. The following brief description is based on an article by Ron Fox [Fox 1995].

The first, *Shu*, state means to keep, protect or maintain. During this stage the student builds her technical foundation in the new art being learned. She will carefully copy the techniques demonstrated by the *sensei* (teacher) without modification and without necessarily understanding the underlying rationale. The idea is that the student will learn fastest by following a single path as defined by the *sensei*. The *sensei*, rather than the student, will determine when she is ready to move to the next stage, *Ha*. It follows that the *Shu* level student requires regular access to a *sensei*. It follows moreover that the *sensei* should be operating at the *Ri* level in order to be an effective teacher and mentor to his students.

Ha, the second stage, means to detach. The student reflects on the meaning and purpose of what she has learned. She develops a deeper appreciation of the art beyond simple repetitive practice. The techniques have been absorbed into her "muscle memory". She breaks free from the traditions to some extent.

The third stage, *Ri*, means to transcend or go beyond. The student is now a practitioner who thinks originally and tests her ideas against reality. At the *Ri* level that student may overtake her *sensei*, thereby advancing the art.

Why is this important? Well I see many teams and managers in organizations too impatient to obtain a solid grounding in the *Shu* stage. Some are hell-bent on going straight to *Ri*! As Ron Jeffries, long-time agilist, says: "My advice is to do it by the book, get good at the practices, then do as you will. Many people want to skip to step three. How do they know?"

If Chapter 3 – Adopting Scrum was mostly about operating at the *Shu* level, this chapter is aimed at the *Ha* level. As your Agile adoption journey progresses, you will find that Scrum still provides a minimal yet sufficient framework within which complex problems can be solved. Continuous improvement can still be achieved simply by doing the essentials well. And when you are ready you will also discover that there is a world of useful tools to complement what you know. At the Ha level the basics will be in "muscle memory" and you can start experimenting with new things without forgetting the fundamentals of Agile.

A pattern language for hyper-productivity

Jeff Sutherland describes a Scrum Team as *hyper-productive* once it achieves a 400% increase in velocity over the team's initial velocity. We might argue successfully that increase in velocity alone is not a sufficient metric, or even a dangerous one when used in isolation. We invariably want more: quality improvements to go hand-in-hand with faster output; better predictability and not only speed; building products and features that customers need rather than simply churning out more features. Yet for the rest of this section let us suspend our disbelief and allow ourselves to focus on a set of process improvement patterns that Jeff and his collaborators have evolved that they believe will help all Scrum Teams to get significantly faster [Sutherland 2013].

My own belief, based on experience working with many teams, is that these patterns are fundamentally sound and will lead not only to increases in output, but also many of the other desired improvements mentioned above. Try them. What have you got to lose?

For the sake of brevity I have simply stated each of the nine patterns that make up the *Pattern Language for Hyper-Productive Teams*. I have added the briefest of comments where this seemed necessary. If you want to implement the patterns, I urge you to download the free PDF with full descriptions of the patterns via the link on Jeff's web site [Sutherland 2013].

Note that the patterns described here are *generative patterns*. This means that the patterns work indirectly by attacking the underlying structure of the problem which may not even be manifest. Generative patterns lead to *emergent* behavior, which is entirely in keeping with the *complex* category of problems that the Scrum framework seeks to address.

Stable Teams

Keep teams stable and avoid shuffling people between teams. Stable teams get to know their capacity, which makes it possible for the business to have some predictability.

Yesterday's Weather

In most cases, the number of Estimation Points completed in the last Sprint is the most reliable predictor of how many Estimation Points will be completed in the next Sprint.

Estimation Points as described here are often termed Story Points. In many cases simply counting Stories (Product Backlog items) will suffice.

Swarming

Focus maximum team effort on one item in the Sprint Backlog to get it done as soon as possible. Whoever takes this item is Captain of the team. Everyone must help the Captain if they can and no-one can interrupt the Captain. As soon as the Captain is Done, whoever takes responsibility for the next priority backlog item is the new Captain.

Development team members and managers may insist that this is inefficient. This may well be true. Remember that our goal is no longer *efficiency*, but *effectiveness*.

Interrupt Pattern

Allot time for interruptions and do not allow the time to be exceeded. Set up three simple rules that will cause the company to self-organize to avoid disrupting production:

1. *The team creates a buffer based on historical data.*
2. *All requests must go through the Product Owner for triage.*
3. *If the buffer starts to overflow, the team Sprint must automatically abort the Sprint.*

Aborting a Sprint has repercussions on delivery dates. Management quickly realizes the cost and will work to help solve the underlying cause.

Daily Clean Code

Fix all bugs in less than a day. Aim to have a completely clean base of code at the end of every day.

Although the Scrum framework is silent on development practices, discipline around code and design quality is absolutely key to agility.

Emergency Procedure

When high on the burndown try a technique used routinely by pilots. When bad things happen, execute the emergency procedure designed specifically for the problem. Do not delay execution while trying to figure out what is wrong or what to do. In a fighter aircraft you could be dead in less time than it takes to figure out what is going on. It is the responsibility of the Scrum Master to make sure the team immediately executes the Scrum Emergency Procedure, preferably by mid-sprint, when things are going off-track. Emergency Procedure steps (do only as much as necessary):

1. *Change the way the work is done. Do something different.*
2. *Get help, usually by offloading backlog to someone else.*
3. *Reduce scope.*
4. *Abort the Sprint and re-plan.*
5. *Inform management how release dates will be affected.*

Scrum is not *laissez-faire* about achieving the Sprint goal!

Scrumming the Scrum

Identify the single most important impediment in the Sprint Retrospective and remove it before the end of the next Sprint. To remove the top impediment, put it in the Sprint Backlog with acceptance tests that will determine when it is Done. Then evaluate the state of the story in the Sprint Review like any other item.

Too many Development Teams struggle with “permission” to make ongoing improvements to their way of work. This is simply ludicrous.

Happiness Metric

Happiness is one of the best metrics because it is a predictive indicator. When people think about how happy they are, they are really projecting out into the future about how they feel. If they feel the company is in trouble or doing the wrong thing, they will be unhappy. Or if there is a major roadblock or frustrating systems they have to deal with, they will be unhappy.

Before you reject this as “fluffy” and idealistic nonsense, I challenge you to try it! First read Ben Linders’ InfoQ article *How to Measure and Analyse Happiness* [Linders 2013].

Teams That Finish Early, Accelerate Faster

Teams often take too much work into a Sprint and cannot finish it. Failure prevents the team from improving. Therefore take less work into the Sprint. Maximize your probability of success by using the pattern Yesterday's Weather. Then implement the four patterns that reduce Impediments within the Sprint, which will systematically deal with any interruptions and help you to finish early. On early completion pull the next Product Backlog item, which will increase Yesterday's Weather for future Sprints. To increase the probability of acceleration, apply Scrumming the Scrum to identify your kaizen in the Sprint Retrospective. Put the kaizen in the Sprint Backlog with acceptance tests for the next Sprint as the top priority.

More about Impediments

The Scrum Master has the duty, amongst many others, of helping to bulldoze out of the way anything and everything that stands in the way of the Development Team being the best they can. This is a never-ending quest. Impediments range widely from "my computer compiles slowly" to "our CEO does not get Agile".

As a Scrum Master, if you hear words like: *still, expected, waiting, guess, thought, hopefully, not available, assumed, wish, try* or *busy* you have probably discovered an impediment. Many Development Teams won't even realize that they are being held up; they often just assume "this is how things work around here". Your job in this situation is firstly to help them become aware of these blind spots and secondly to realize that they are empowered to take action more often than they think. I teach them the old adage: "it is easier to beg forgiveness than request permission".

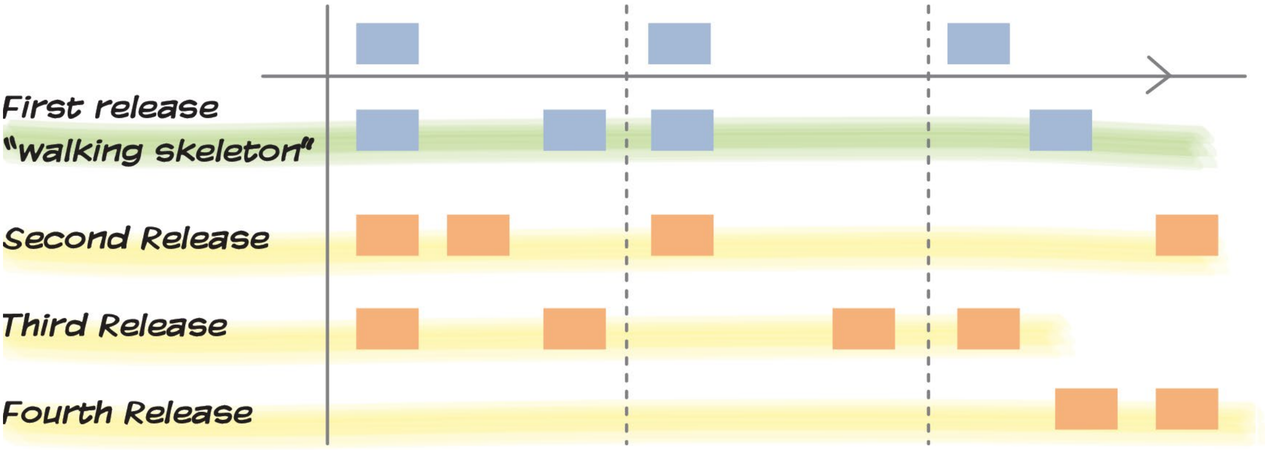
Novice (*Shu*-level) teams will tend to believe that they are powerless to change the status quo. This is often frustrating to middle managers in the organization who have pressed the magic Scrum button and are waiting for the magic of self-organization to occur. This reflects a lack of understanding of organizational dynamics. Such managers and their teams will benefit from engaging an experienced agile coach. And the blunt truth is that self-organization requires time (in combination with appropriate leadership behavior).

Even more, teams are blind to their own impediments. The saying goes: "you can't observe what's going in the box if you're inside the box". The Scrum Master must be the team's coach, mentor and servant leader, helping them identify their own blind spots and then empowering and enabling them to overcome these one by one.

Since our brains respond better to positive impulses than negative ones, I like to focus more on improvements the team wants to make than on impediments. The Sprint Retrospective is the Scrum Team's tool for emerging the next items in its continuous improvement journey. I like to couple this with the *Scrumming the Scrum* pattern [Sutherland 2013] both to bring the appropriate focus to continuous improvement and to get the fast feedback that fuels empirical processes.

Story Maps

Jeff Patton [Patton 2008] developed his Story Mapping concept from the earlier work on Usage-Centred Design by Larry Constantine and Lucy Lockwood, on Use Cases by Alistair Cockburn and on User Stories as described by Kent Beck in Extreme Programming. A Story Map is a two-dimensional depiction of the Product Backlog with a "backbone" of users' activities and the narrative flow running horizontally from left to right and successive incremental product releases running vertically from top to bottom. The main building blocks of a Story Map are user tasks. The figure below shows an example of a Story Map. I teach Story Mapping in my Product Owner training and use it when coaching teams who are beyond novice level.



Update: Jeff Patton's long-awaited book "User Story Mapping" has been released [Patton 2014]. It's outstanding and I urge you to buy and read it!

Release Planning using a burnup chart

The Concept of Velocity

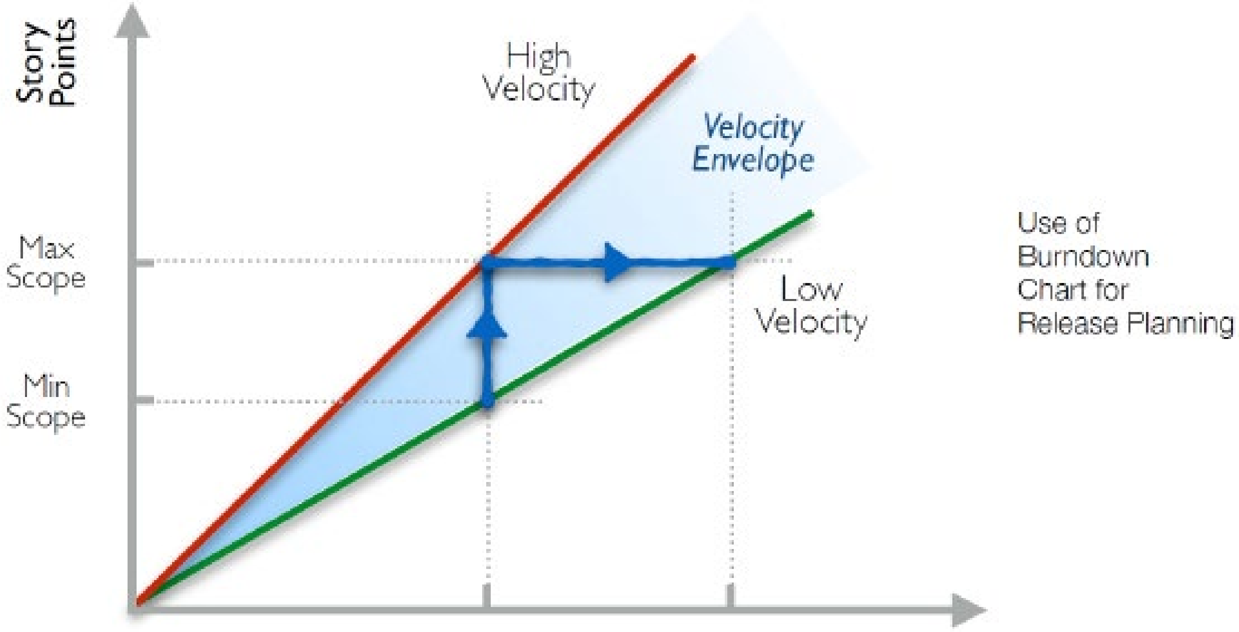
We observe the rate at which a team delivers running, tested features to their Product Owner at the end of each Sprint. We call this their velocity. We say that a team's velocity is "15 to 20 points" when they are able to deliver at the end of every Sprint Done stories whose estimated sizes usually add up to at least 15 points and rarely exceed 20 points.

This velocity range can then be used (with due care) by the Product Owner to provide his stakeholders with a likely lower and upper bound for future feature deliveries.

Using the Velocity Envelope for Release Planning

The product or release burnup chart below shows how we can use the range of velocity measurements from earlier sprints to provide a credible estimation envelope for some time into the future. The green line shows the lower bound of the team's historic velocity to date. The red line shows the upper bound. The area between these lines is the range of probable velocities, or velocity envelope. From the historic data we can estimate an early date and a late date for delivering a specified scope—see for example the horizontal blue line. Or we can estimate how much scope we can deliver by a specified date—see for example the vertical blue line. In practice I always prefer to go for a fixed date with uncertain scope. In this case I deliver some value to my customer at a time she can rely on. And I can deliver more later if she wishes. Compare this with taking the time I need to deliver a given scope: It removes the choice I could have offered her.

We must always help stakeholders to understand that the purpose is to provide transparency so that they know reality and can make appropriate choices. And we must never forget the principle of adaptive planning that underpins Scrum and Agile.



- *In theory the lower the variability in the team's velocities measured over successive sprints, the more precisely the Product Owner can predict the team's future deliveries. However it is a dangerous practice to encourage the team to aim for perfection here. A likely outcome of this is either gaming of sizes, leading to a loss of transparency, or excessive time spent in estimating, resulting in waste. As with all things, balance is desirable. I have found a range of ±20% around the mean to be a good figure.*
- *One team's sizes are generally not comparable to those of another. This is immaterial unless we have multiple teams working off the same Product Backlog. This is an issue of scaling Scrum and is quite easily solved, but is not a topic for this little guide.*
- *With many teams I have coached it turns out after the first few sprints that their velocity did not increase over time. I found this rather surprising until I realized what was happening. As their practices matured and improved, their work got easier and they naturally gradually decreased their story point estimates. A natural consequence was an increase in the value delivered per story point. It matters little if the effort a story point represents changes over time, provided this change is gradual and we are using some sort of moving average to for planning. I mention this to help you to be mindful that story points and velocity are neither a measure of input nor output over the long term; they are simply a planning aid.*
- *Moreover, I hope that the mature Product Owner will herself understand that neither input nor output are valuable measures; agile teams should be focussed*

on maximizing outcomes with the least possible output. And she must educate stakeholders in this.

Takeaways from this chapter

- The *Shu Ha Ri* model of learning
- Patterns for improving Scrum practice
- The nature of impediments
- An introduction to Story Maps
- Release planning using a burnup chart.

PART EIGHT

Getting Help

The guide began with the words **Scrum is Simple. Doing Scrum is Hard.** Agile trainers and coaches the world over use metaphors like "Scrum is a flashlight to discover what is hidden in the dark corners of your organization". They will hold that Scrum will not solve a single problem for you.

So how do you go about using Scrum (or any other method)? Well you can read books like this one and try to apply what you find. And I certainly hope that you do. However you soon find things are hard to figure out on your own. Your specific context is different from what you've read about. Your colleagues and managers, and you yourself are prone to continue doing what you've always done, even when you know this is dysfunctional. Change is hard.

If I have learned one thing in my Agile journey it is to seek help. This has been the single most empowering act for me. And I want to encourage you to do the same. To quote Karl E. Weick and Kathleen M. Sutcliffe from *Managing the Unexpected*: "It is a sign of strength to know when you've reached the limits of your knowledge and know enough to ask for help" [Weick and Sutcliffe 2007].

In this chapter my aim is to help you recognize when you need help, how to get help that is appropriate to your need. I'm mindful that this is a broad topic and I can but scratch the surface. As usual I provide a set of references I have found useful.

The culture elephant

Agile is not an act of *doing* something; rather it is a state of *being*. An implication of this is that people need to change their attitudes and behavior in smaller or greater ways. In my experience this is by far the hardest part of an journey towards agility.

The "elephant" that stands in the way of these changes is the organization's *culture*.

Organizational culture and leadership expert Edgar H. Schein formally defines the culture of a group as "a pattern of shared basic assumptions learned by a group as it solved its problems of external adaptation and internal integration, which has worked well enough to be considered valid and, therefore, to be taught to new members as the correct way to perceive, think and feel in relation to these problems" [Schein 2010].

While this is quite a mouthful, you might more readily resonate with these words also from Schein: "The key to understanding 'resistance to change' is to recognize that some behavior that has become dysfunctional for us may, nevertheless, be difficult to give up because it serves other positive functions".

So what to do? My own journey as a coach has been enriched by engaging with other consultants with far more experience and expertise than myself in understanding and assessing culture as a means of bringing about managed organizational change. And, if you feel up to doing this on your own, Schein offers a recipe for conducting a short culture assessment workshop in your organization [Schein 2010, page 315].

What is coaching?

It is challenging to find a common definition of coaching. One that I like is "The art of facilitating the performance, learning and development of another" [Downey 2003].

Common to most approaches are these characteristics:

- It is a structured dialogue between *coach* and *coachee**, with clear expectations, feedback and accountability;
- The coach facilitates self-directed learning and development of the coachee;
- Coaching aims to enhance well-being and performance of the coachee.

**Some people may have an aversion to the term coachee. However I have not found a better generic word. Note also that the coachee may be an individual or a group.*

Another dimension to coaching in our world is that the *Agile coach* frequently acts in the additional roles of *teacher*, *expert advisor* and as *mentor*. The first two are generally based on expertise in Agile process frameworks (like Scrum). The third is based on personal experience the coach has in the role of, for example, Scrum Master or Product Owner. A good Agile coach will make it transparent to the coachee when she is acting in which role.

How do I know when I need help?

We all need help. Ask any successful leader and she will tell you about those people who have helped her, both formally and informally on her journey.

So perhaps a better question is what kind of help do I need in different contexts?

The Scrum Master role must provide process leadership to the rest of the Scrum Team and to the rest of the organization. Classroom training, such as the Certified Scrum Master course, is an insufficient means to gain the required tacit knowledge to master agile practices and the Scrum framework. The team and the organization need the skills of an experienced practitioner to guide them through the maze of challenges they will inevitably face in transitioning to Agile. An Agile coach will likely be able to help here.

When it comes to organizational transformation, we need help from someone with more formal skills as a coach. Given that Agile (and its close cousin Lean) focus on the whole system, perhaps it follows that a systemic coach will be of most help. For a better understanding I commend the article *Agile and Systemic Coaching* by Sigi Kaltenecker and Bent Myllerup [Kaltenecker and Myllerup 2011].

Another truism is that we all have areas of weakness of which we are unaware, or *blind spots*. This is not because we are stupid or unskilled, but because we are human. A trusted party external to our immediate environment—for example, a systemic coach—is well-placed to observe and inform us about our own blind spots.

During his research into successful Agile adoptions Agile coach and CIO Marius de Beer observed that the best organizations combined the use of internal and external coaches [De Beer 2012].

What are the economics of coaching?

Engaging a coach is but one component of making an Agile adoption. There are costs associated with some or all of training teams in Agile methods, allocating teams time to practice, restructuring the organization, helping managers to migrate to new roles and much more.

My observations based on doing this with scores of teams in diverse organizations is that the benefits of an Agile transition have outweighed the costs in every case. In no case have the people involved sought to return to how they worked before; in fact the resounding cry is: "I would rather leave than go back to how we worked before".

Nevertheless there remains a conundrum: why is it that some organizations show dramatic and ongoing improvement, while others tend to stall after a modest gain? I have seen some teams treble their ROI in a few sprints and I have seen others satisfied with a small improvement. I believe the answer lies in the cultural aspects I discussed at the start of this chapter.

So I don't believe it is the right question to ask what coaching will cost. I would rather ask: "How can we improve X?", where X might be customer satisfaction, employee happiness, ROI, etc. And then "what investment will be required by us and how can we measure value?" Your coach should be ready and able to partner with you to understand your needs and answer these questions.

When asked what they would change if they did their transition to Agile and Scrum over, the most common comment heard from managers in organizations worldwide is "more coaching"!

Takeaways from this chapter

» The importance of organizational culture

» What coaching is

» Knowing when to ask for help

» The economics of coaching

PART NINE

More on Agile and Lean

The Agile Manifesto

In February, 2001, seventeen independent "lightweight" software methodologists and thinkers met to talk and find common ground. Amongst them were Scrum co-founders Jeff Sutherland and Ken Schwaber, together with Mike Beedle, who worked on the initial Scrum patterns and co-authored the first book on Scrum. The group named themselves "The Agile Alliance" and agreed on a *Manifesto for Agile Software Development*. They further defined a set of twelve principles behind the manifesto, reproduced in full on the facing page [Highsmith 2001].

Commentary

The Agile Manifesto and its twelve principles have stood up (thus far) for more than a decade. They remain the litmus test for all Agile methods and practitioners by which to evaluate their way of working. The most vocal criticism has been the bias towards software development, when Agile methods are actually more broadly applicable.

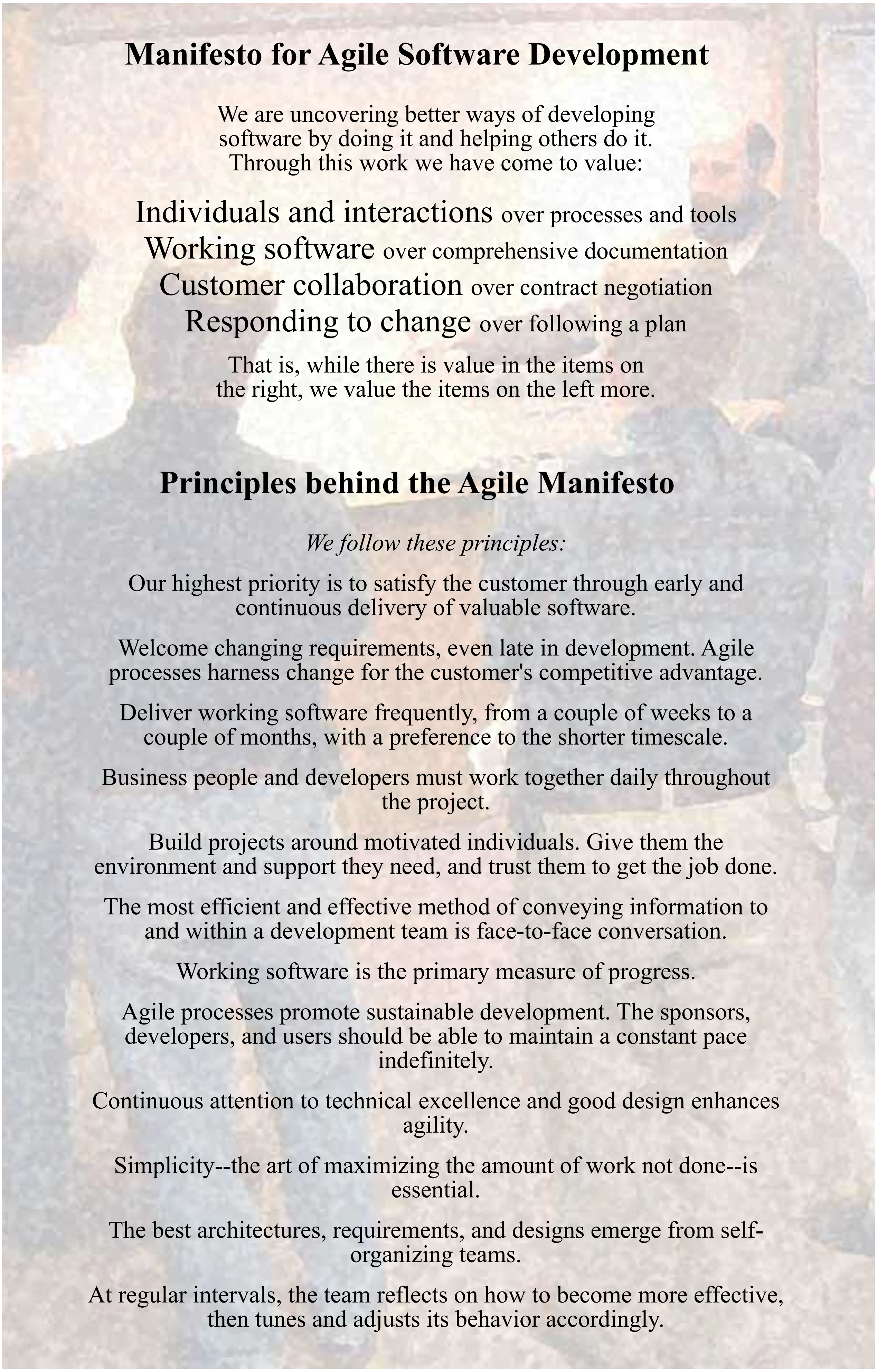
A common misconception is that Agile seeks to eliminate all process, documentation, contracting and planning. Simply reading the value statements should help you spot the fallacy in this.

For more on the history of the manifesto as well as an analysis of the four values and twelve principles, I suggest you read Martin Fowler and Jim Highsmith's original article in Dr. Dobbs Journal [Fowler and Highsmith 2001].

Agile and Lean Product Development

Scrum is one instance of Agile. Adherence to the Agile Manifesto and all its principles does not necessarily mean a team or organization is following Scrum. Ken Schwaber and Jeff Sutherland make this clear: "Scrum's roles, artifacts, events, and rules are immutable and although implementing only parts of Scrum is possible, the result is not Scrum. Scrum exists only in its entirety and functions well as a container for other techniques, methodologies, and practices." [Schwaber 2013]. This may seem harsh to the novice. To the Scrum journeyman who is able to make a thoughtful analysis of the Scrum pattern the reasons are more obvious. Scrum provides a minimal pattern for establishing order.

Agile and Lean Product Development (as distinct from Lean Manufacturing) are close cousins. Both focus on delivering a flow of value to customers and both have respect for people as a foundational value. Agile relies on adaptive planning with fast feedback loops, while Lean focuses on achieving flow. Both encompass rapid learning and continuous improvement.



Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Other Agile and Lean Methods

The most popular "lightweight" method at the time of the Agile Manifesto was Extreme Programming (XP). A fundamental distinction between Scrum and XP is the former's exclusive focus as a management pattern and the latter's strong emphasis on technical software development practices. Actually they have much in common and every good Scrum team I have seen embraces many of the XP principles and practices. For a full description of XP I recommend Kent Beck's book *Extreme Programming Explained* [Beck 2005].

Another important method has emerged more recently called the Kanban Method. This movement has been led by David Anderson and others. It is arguably a Lean rather than an Agile method, but this distinction is perhaps esoteric for most people.

My own experience is that both methods have strong merits and are likely to lead to better ways of organizing knowledge work. Yet both have suffered blame for poor implementation. A case of shoot the messenger?

Kanban and Scrum are not mutually exclusive and experienced teams will embrace ideas from both patterns. Since this is really a book about Scrum, it is not within its scope to go into great detail, but the following table highlights a few similarities and differences.

Kanban	**Scrum**
Statistical process control method	Empirical process control method
Manages a given flow of work by visualizing the workflow, limiting the quantity of work-in-progress, and removing bottlenecks	Manages the iterative design and incremental delivery of value from a dynamic list of work that is fed by a continuous assessment of users' needs
No roles, four principles, six practices	Defined roles, events & artefacts
Perceived as simple; actually highly sophisticated	Simple to learn; hard to do well
Evolutionary change	(More) revolutionary change
Sweet spot is for addressing Complicated* problems	Sweet spot is for addressing Complex* problems

- The Kanban Method manages a given flow of work by visualizing the workflow, limiting the quantity of work-in-progress, and removing bottlenecks.
- Kanban imposes few rules. This may create an impression of simplicity. In practice it is a sophisticated method and requires stewardship.
- Change with the Kanban Method is evolutionary. This can make Kanban a more palatable choice for organizations whose culture makes it hard to introduce rapid change.
- Scrum, by contrast, attacks complexity head-on and requires a more radical shift in attitude. Scrum provokes fundamental change. This is hard for organizations and people, but the results can be astonishing.
- It will be wise to familiarize yourself with both methods, or to seek expert advice, so you make an appropriate choice. You may well profit from using both in different areas of application or in tandem.

- A simple, yet excellent guide to Kanban written by Arne Roock is now available [Roock 2012]. The more serious-minded should read David Anderson's definitive book on the subject [Anderson 2010].

* I view the Kanban Method as a good—probably the best—tool for solving problems in the Complicated domain. Scrum, by contrast, is optimally suited to solving Complex problems. Some readers will find this a controversial statement. You may find Snowden and Boone's HBR article [Snowden 2007] a helpful reference to understanding complexity.

Takeaways from this chapter

» The Agile Manifesto

» Relationship between Agile and Lean Product Development

» An introduction to other methods: XP and Kanban

Acknowledgements

The genesis of the first version of this little book was the need, as a sponsor, to provide an insert to the "goodie bag" for the first South African Scrum Day in 2009. Since then a number of people have been kind enough to recommend it and also to suggest improvements. The agile practitioners I have been fortunate to meet and interact with have all in some measure influenced what you find written here.

At the risk of omission I want to mention some who have had a significant influence on my agile journey: Boris Gloger, my Scrum trainer and my first Scrum coach and mentor; Jim Cundiff, who as MD of the Scrum Alliance encouraged and helped me to found the South African Scrum User Group (SUGSA); the many people who selflessly gave their time to serve SUGSA during my tenure: Neil, Sue, Steve, Mike, Carlo, Charl, Evan, Karen G, Karen V, Kevin, Neill and Alwyn; Tobias Mayer, who encouraged me and promoted this booklet; Pete Behrens and Roger Brown, who created the CSC programme; Andreas Schliep and Peter Beck, who have collaborated frequently in training and coaching; Mike Freislich, my first partner at Scrum Sense; Carlo Kruger and Marius de Beer, my business partners for two years from whom I learned much; David Anderson, who introduced me to Kanban; Karen Greaves, who co-trained with me more times than I can remember; and Sigi Kaltenecker, who has mentored me in organizational development over the past four years.

In addition I have learned so much from other CSTs with whom I have co-trained, Lean/Agile conference delegates with whom I've had intensive conversations, participants at my training classes and from clients I have consulted, coached and mentored during the past eight years. You are too many to mention by name.

And I want to acknowledge Ken Schwaber, Jeff Sutherland and the other Scrum pioneers who paved the way for all of us.

My sincere appreciation goes to the volunteers who selflessly took it on themselves to provide translations of the second version into several popular languages:

- Alan Cyment (Spanish—Un Mejor Scrum)
- Peter Beck, Sebastian Lauber, Andreas Schliep Henning Wolf and Marion Eickmann (German—Scrum besser machen)
- Samuel Gonsales (Brazilian Portuguese—Melhor Scrum)
- Stéphane Wojewoda, Matthieu Nguyen and Yann Le Moal (French—Du Meilleur Scrum).

You will find links to the soft-copy of this booklet and all available translations at www.agile42.com/DoBetterScrum. Please send translation requests and offers to info@agile42.com.

Thanks to Michi Ukawa for the smart new agile42 layout, diagrams and cover. And thanks to Ana-Maria Ciobotaru and Lawrence Nyveen for editing and publishing this booklet on InfoQ.

I have tried to give proper credit where I have used the work of others. If I have quoted your work and failed to acknowledge this, please let me know so that I can remedy the error.

References

Adkins, Lyssa (2010). *Coaching Agile Teams: A Companion for ScrumMasters, Agile Coaches and Project Managers in Transition. Addison-Wesley Professional.*

Ambler, Scott M. and Lines, Mark (2012). *Disciplined Agile Delivery: A Practitioners' Guide to Agile Software Delivery in the Enterprise. IBM Press.*

Anderson, David J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press.*

Argyris, Chris (1977). *Double Loop Learning in Organizations. http://hbr.org/1977/09/double-loop-learning-in-organizations/ar/1*

Beaumont, Serge (2009). *The Definition of Ready. http://blog.xebia.com/?s=definition+of+ready*

Beck, Kent with Cynthia Andres (2005). *Extreme Programming Explained (second edition). Addison-Wesley Professional.*

Beedle, Mike (2014). *Enterprise Scrum. http://www.enterprisescrum.com/*

Cockburn, Alistair (2007). *Agile Software Development: The Cooperative Game (2nd edition). Addison-Wesley Professional.*

Cohn, Mike (2004). *User Stories Applied. Addison-Wesley Professional.*

Cohn, Mike (2006). *Agile Estimating and Planning. Prentice Hall.*

Cohn, Mike (2009). Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional.

Conway, Melvin (1968). *Conway's Law. http://www.melconway.com/Home/Conways_Law.html*

Coplien, James O. (1994). *Borland Software Craftsmanship: A New Look at Process, Quality and Productivity. ftp://ftp.gwdg.de/pub/languages/smalltalk/st.cs.uiuc.edu/pub/papers/cope/borland-process.ps.*

De Beer, Marius (2012). *Data-driven Agility: An analysis of Agile adoption in North American organisations. http://www.scrumsense.com/consulting/data-driven-agility/.*

De Beer, Marius (2013). *Personal correspondence and conversations during 2012 & 2013.*

Drucker Peter F. (2001). *Management Challenges for the 21st Century. HarperBusiness.*

Fowler, Martin and Highsmith, Jim (2001). *The Agile Manifesto. http://www.drdobbs.com/open-source/the-agile-manifesto/184414755.*

Fowler, Martin (2005). *The New Methodology. http://www.martinfowler.com/articles/newMethodology.html.*

Fox, Ron (1995). Shu Ha Ri. *In THE IAIDO NEWSLETTER Volume 7 number 2 #54 FEB 1995. http://www.aikidofaq.com/essays/tin/shuhari.html.*

Gartner (2012). *The End of the Waterfall as We Know It. http://www.gartner.com/id=2127715. (Not available for individual purchase.)*

Gloger, Boris (2008). *Heartbeat Retrospectives. http://www.infoq.com/presentations/Heartbeat-Retrospectives-Boris-Gloger.*

Goldratt, Eliyahu M. (2006). *The Haystack Syndrome: Sifting Information Out of the Data Ocean. North Riven Press.*

Greenleaf, Robert K. (2012). *The Servant as Leader. The Greenleaf Center for Servant Leadership.*

Highsmith, Jim, et al (2001). *Manifesto for Agile Software Development. http://www.agilemanifesto.org/.*

Hundermark, Peter (2009). *Measuring for Results. http://www.scrumsense.com/coaching/measuring-for-results.*

Hundermark, Peter (2014). *Scaling Scrum to the Organisation. http://www.scrumsense.com/blog/scaling-scrum-Organisation.*

James, Michael (2006). *An Example Checklist for ScrumMasters. http://www.scrummasterchecklist.org/.*

Jeffries, Ron (2001). *Card, Conversation, Confirmation. http://ronjeffries.com/xprog/articles/expcardconversationconfirmation/.*

Jeffries, Ron, et al (2013). *Agile Atlas. http://agileatlas.org.*

Kaltenecker, Siegfried and Myllerup, Bent (2011). *Agile and Systemic Coaching. http://www.scrumalliance.org/community/articles/2011/may/agile-systemic-coaching.*

Katzenbach, Jon R. and Smith, Douglas K. (2002). *The Wisdom of Teams. Collins.*

Kniberg, Henrik (2007). *Scrum and XP from the Trenches. http://www.infoq.com/minibooks/scrum-xp-from-the-trenches-2.*

Kniberg, Henrik (2008). *Multi-Team Sprint Planning. http://www.scrumalliance.org/system/resource_files/0000/0871/Multi-Team-Sprint-Planning.pdf.*

Kniberg, Henrik and Ivarsson, Anders (2012). *Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds. http://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify.*

Larman, Craig and Vodde, Bas (2008). *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum. Addison-Wesley Professional.*

Larman, Craig and Vodde, Bas (2010). *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum. Addison-Wesley Professional.*

Larman, Craig (2013). *Larman's Laws of Organizational Behavior. http://www.craiglarman.com/wiki/index.php?title=Larman%27s_Laws_of_Organizational_Behavior*

Leffingwell, Dean (2013). *Scaled Agile Framework. http://scaledagileframework.com/.*

Linders, Ben (2013). *How to Measure and Analyse Happiness. http://www.infoq.com/news/2013/01/measure-analyze-happiness*

Mayer, Tobias (2009). *Scrum Roles—an abstraction. http://www.dymaxicon.com/2013/the-peoples-scrum/.*

Nonaka, Ikujiro and Takeuchi, Hirotaka (1986). *The New, New Product Development Game. Harvard Business Review, Jan-Feb 1986.*

Patton, Jeff (2008). User Story Mapping. http://jpattonassociates.com/user-story-mapping/

Patton, Jeff (2014). *User Story Mapping*. O'Reilly Media.

Pichler, Roman (2010). *Agile Product Management with Scrum: Creating Products that Customers Love. Addison-Wesley Professional.*

Pink, Daniel (2009). *The Puzzle of Motivation. http://www.ted.com/talks/dan_pink_on_motivation.html.*

Pink, Daniel (2011). *Drive: The Surprising Truth About What Motivates Us. Riverhead Books.*

Reinertsen, Donald G. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development. Celeritas Publishing.*

Roock, Arne (2012). *Stop Starting, Start Finishing!* Lean-Kanban University.

Rubin, Kenneth S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley Professional.*

Sliger, Michele and Broderick, Stacia (2008). *The Software Project Manager's Bridge to Agility. Addison-Wesley Professional.*

Schein, Edgar H. (2010). *Organizational Culture and Leadership, Fourth Edition. Jossey-Bass.*

Schwaber, Ken and Beedle, Mike (2001). *Agile Software Development with Scrum. Prentice Hall.*

Schwaber, Ken (2007). *The Enterprise and Scrum. Microsoft Press.*

Schwaber, Ken and Sutherland, Jeff (2013). *Scrum Guide. http://www.scrumguides.org/.*

Snowden, David J. and Boone, Mary E. (2007). *A Leader's Framework for Decision Making. Harvard Business Review, Nov 2007.*

Sutherland, Jeff (2003–2012). *Scrum Log Jeff Sutherland > Productivity. http://www.scruminc.com/?s=productivity.*

Sutherland, Jeff (2007). *Origins of Scrum. http://www.scruminc.com/origins-of-scrum/.*

Sutherland, Jeff (2009). *Sprint Burndown: by hours or by story points? http://www.scruminc.com/sprint-burndown-by-hours-or-by-story/.*

Sutherland, Jeff (2011). *Scrum: Inside the Sprint Retrospective. http://youtu.be/MFLvQXMNrO8.*

Sutherland, Jeff (2013). *Finish Early, Accelerate Faster. http://www.scruminc.com/pattern-language/.*

Tabaka, Jean (2006). *Collaboration Explained | Facilitation Skills for Software Project Leaders. Addison-Wesley Professional.*

Teasley, Covi, Krishnan, & Olson (2000). *How Does Radical Collocation Help a Team Succeed? Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (pp. 339 - 346). New York: ACM.*

VersionOne (2013)*: The State of Agile Development | 7th Annual Survey. www.stateofagile.com.*

Wake, William (2003). *INVEST in Good Stories, and SMART Tasks. http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/.*

Weick, Karl E. and Sutcliffe, Kathleen M. (2007). *Managing the Unexpected. Jossey-Bass.*

Yip, Jason (2006). *It's Not Just Standing Up: Patterns of Daily Stand-up Meetings http://martinfowler.com/articles/itsNotJustStandingUp.html*