THE SCRUM PRIMER

By Pete Deemer
Gabrielle Benefield
Craig Larman
Bas Vodde

Version 1.1

Certified Scrum Training Worldwide | www.ScrumTI.com

A note to readers: There are many concise descriptions of Scrum available online, and this primer aims to provide the next level of detail on the practices. It is not intended as the final step in a Scrum education; teams that are considering adopting Scrum are advised to equip themselves with Ken Schwaber's *Agile Project Management with Scrum* or *Agile Software Development with Scrum*, and take advantage of the many excellent Scrum training and coaching options that are available; full details are at scrumalliance.org. Our thanks go to Ken Schwaber, Dr. Jeff Sutherland, and Mike Cohn for their generous input.

Traditional Software Development

The traditional way to build software, used by companies big and small, was a sequential life cycle commonly known as "the waterfall." There are many variants (such as the V-Model), but it typically begins with a detailed planning phase, where the end product is carefully thought through, designed, and documented in great detail. The tasks necessary to execute the design are determined, and the work is organized using tools such as Gantt charts and applications such as Microsoft Project. The team arrives at an estimate of how long the development will take by adding up detailed estimates of the individual steps involved. Once stakeholders have thoroughly reviewed the plan and provided their approvals, the team starts to work. Team members complete their specialized portion of the work, and then hand it off to others in production-line fashion. Once the work is complete, it is delivered to a testing organization (some call this Quality Assurance), which completes testing prior to the product reaching the customer. Throughout the process, strict controls are placed on deviations from the plan to ensure that what is produced is actually what was designed.

This approach has strengths and weaknesses. Its great strength is that it is supremely logical – think before you build, write it all down, follow a plan, and keep everything as organized as possible. It has just one great weakness: humans are involved.

For example, this approach requires that the good ideas all come at the beginning of the release cycle, where they can be incorporated into the plan. But as we all know, good ideas appear throughout the process – in the beginning, the middle, and sometimes even the day before launch, and a process that does not permit change will stifle this innovation. With the waterfall, a great idea late in the release cycle is not a gift, it's a threat.

The waterfall approach also places a great emphasis on writing things down as a primary method for communicating critical information. The very reasonable assumption is that if I can write down on paper as much as possible of what's in my head, it will more reliably make it into the head of everyone else on the team; plus, if it's on paper, there is tangible proof that I've done my job. The reality, though, is that most of the time these highly detailed 50-page requirements documents just do not get read. When they *do* get read, the misunderstandings are often compounded. A written document is an incomplete picture of my ideas; when you read it, you create another abstraction, which is now two steps away from what I *think* I meant to say at that time. It is no surprise that serious misunderstandings occur.

Something else that happens when you have humans involved is the hands-on "aha" moment – the first time that you actually use the working product. You immediately think of 20 ways you could have made it better. Unfortunately, these very valuable insights often come at the end of the release cycle, when changes are most difficult and disruptive – in other words, when doing the right thing is most expensive, at least when using a traditional method.

Humans are not able to predict the future. For example, your competition makes an announcement that was not expected. Unanticipated technical problems crop up that force a change in direction. Furthermore, people are particularly bad at planning uncertain things far into the future – guessing today how you will be spending your week eight months from now is something of a fantasy. It has been the downfall of many a carefully constructed Gantt chart.

In addition, a sequential life cycle tends to foster an adversarial relationship between the people that are handing work off from one to the next. "He's asking me to build something that's not in the specification." "She's changing her mind." "I can't be held responsible for something I don't control." And this gets us to another observation about sequential

development – it is not much fun. The waterfall model is a cause of great misery for the people who build products. The resulting products fall well short of expressing the creativity, skill, and passion of their creators. People are not robots, and a process that requires them to act like robots results in unhappiness.

A rigid, change-resistant process produces mediocre products. Customers may get what they first ask for (at least two translation steps removed), but is it what they really want once they see the product? By gathering all the requirements up front and having them set in stone, the product is condemned to be only as good as the initial idea, instead of being the best once people have learned or discovered new things.

Many practitioners of a sequential life cycle experience these shortcomings again and again. But, it seems so supremely logical that the common reaction is to turn inward: "If only we did it better, it would work" – if we just planned more, documented more, resisted change more, everything would work smoothly. Unfortunately, many teams find just the opposite: the harder they try, the worse it gets! There are also management teams that have invested their reputation – and many resources – in a waterfall model; changing to a fundamentally different model is an apparent admission of having made a mistake. And Scrum *is* fundamentally different...

Agile Development and Scrum

The agile family of development methods evolved from the old and well-known iterative and incremental life cycle approaches. They were born out of a belief that an approach more grounded in human reality – and the product development reality of learning, innovation, and change – would yield better results. Agile principles emphasize building working software that people can get hands on quickly, versus spending a lot of time writing specifications up front. Agile development focuses on cross-functional teams empowered to make decisions, versus big hierarchies and compartmentalization by function. And it focuses on rapid iteration, with continuous customer input along the way. Often when people learn about agile development or Scrum, there's a glimmer of recognition – it sounds a lot like back in the start-up days, when we "just did it."

By far the most popular agile method is Scrum. It was strongly influenced by a 1986 *Harvard Business Review* article on the practices associated with successful product development groups; in this paper the term "Scrum" was introduced, relating successful development to the game of Rugby in which a self-organizing (self-managing) team moves *together* down the field of product development. The first Scrum team was created at Easel Corporation in 1993 by Dr. Jeff Sutherland and the Scrum framework was formalized in 1995 by Ken Schwaber. Scrum is now used by companies large and small, including Yahoo!, Microsoft, Google, Lockheed Martin, Motorola, SAP, Cisco, GE, CapitalOne and the US Federal Reserve. Many teams using Scrum report significant improvements, and in some cases complete transformations, in both productivity and morale. For product developers – many of whom have been burned by the "management fad of the month club" – this is significant. Scrum is simple and powerful.

Scrum Summary

Scrum is an iterative, incremental framework for projects and product or application development. It structures development in cycles of work called **Sprints**. These iterations are 1-4 weeks in length, and take place one after the other. The Sprints are of fixed duration – they end on a specific date whether the work has been completed or not, and are *never extended.*

They are *timeboxed*. At the beginning of each Sprint, a cross-functional team selects **items** (customer requirements) from a prioritized list. They commit to complete the items by the end of the Sprint. During the Sprint, the chosen items do not change. Every day the team gathers briefly to report to each other on progress, and update simple charts that orient them to the work remaining. At the end of the Sprint, the team reviews the Sprint with stakeholders, and demonstrates what they have built. People obtain feedback that can be incorporated in the next Sprint. Scrum emphasizes working product at the end of the Sprint that is really "done"; in the case of software, this means code that is integrated, fully tested and potentially shippable. Key roles, artifacts, and events are summarized in Figure 1.

A major theme in Scrum is "inspect and adapt." Since development inevitably involves learning, innovation, and surprises, Scrum emphasizes taking a short step of development, inspecting both the resulting product and the efficacy of current practices, and then adapting the product goals and process practices. Repeat forever.

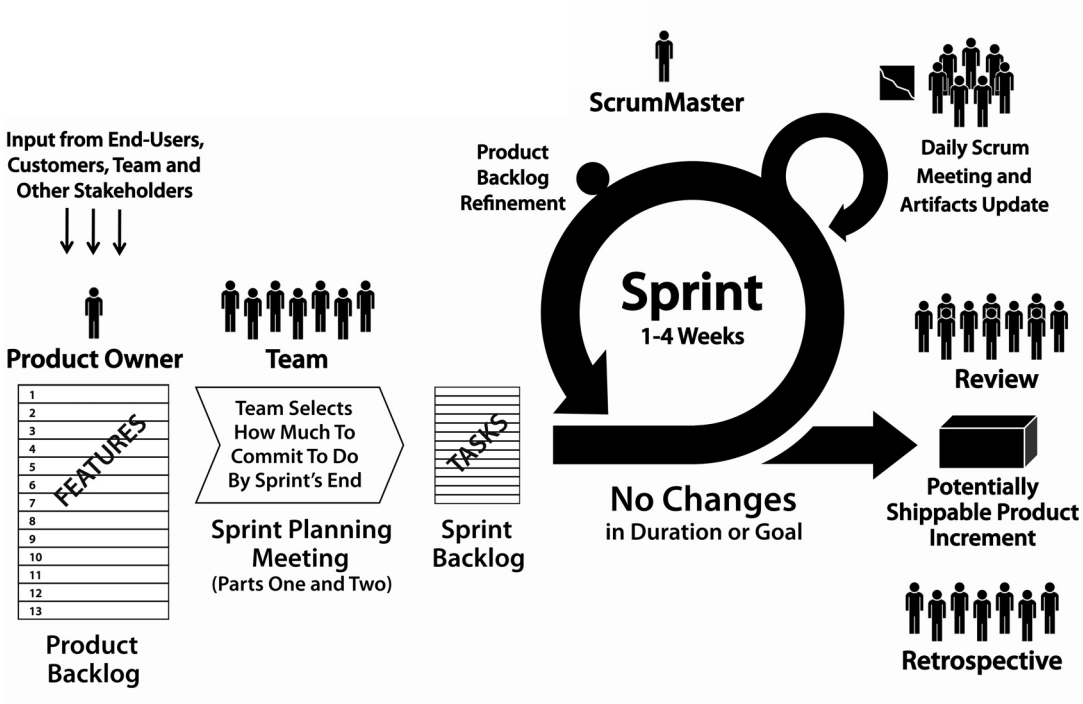

Figure 1. Scrum

Scrum Roles

In Scrum, there are three primary roles: The Product Owner, The Team, and The ScrumMaster. The **Product Owner** is responsible for maximizing return on investment (ROI) by identifying product features, translating these into a prioritized feature list, deciding which should be at the top of the list for the next Sprint, and continually re-prioritizing and refining the list. The Product Owner has profit and loss responsibility for the product, assuming it is a commercial product. In the case of an internal application, the Product Owner is not responsible for ROI in the sense of a commercial product (that will generate revenue), but they are still responsible for maximizing ROI in the sense of choosing – each Sprint – the highest-business-value lowest-cost items. In some cases, the Product Owner and the customer are the

same person; this is common for internal applications. In others, the customer might be millions of people with a variety of needs, in which case the Product Owner role is similar to the Product Manager or Product Marketing Manager position in many product organizations. However, the Product Owner is somewhat different than a traditional Product Manager because they actively and frequently interact with the team, personally offering the priorities and reviewing the results each two- or four-week iteration, rather than delegating development decisions to a project manager. It is important to note that in Scrum there is one and only one person who serves as – and has the final authority of – Product Owner.

The Team builds the product that the customer is going to use: the application or website, for example. The team in Scrum is "cross-functional" – it includes all the expertise necessary to deliver the potentially shippable product each Sprint – and it is "self-organizing" (self-managing), with a very high degree of autonomy and accountability. In Scrum, teams are self-organizing rather than being led by a team manager or project manager. The team decides what to commit to, and how best to accomplish that commitment; in Scrum lore, the team are known as "Pigs" and everyone else in the organization are "Chickens" (which comes from a joke about a pig and a chicken deciding to open a restaurant called "Ham and Eggs," and the pig having second thoughts because "he would be truly committed, but the chicken would only be involved").

The team in Scrum is seven plus or minus two people, and for a software product the team might include analysts, developers, interface designers, and testers. The team develops the product and provides ideas to the Product Owner about how to make the product great. In Scrum, the team should be 100 percent dedicated to the work for one product during the Sprint; avoid multitasking across multiple products or projects. Stable teams are associated with higher productivity, so avoid changing team members. Application groups with many people are organized into multiple Scrum teams, each focused on different features for the product, with close coordination of their efforts. Since one team does all the work (planning, analysis, programming, and test) for a complete customer-centric feature, Scrum teams are also known as *feature teams*.

The **ScrumMaster** helps the product group learn and apply Scrum to achieve business value. The ScrumMaster does whatever is in their power to help the team be successful. The ScrumMaster is *not* the manager of the team or a project manager; instead, the ScrumMaster serves the team, protects them from outside interference, and educates and guides the Product Owner and the team in the skillful use of Scrum. The ScrumMaster makes sure everyone on the team (including the Product Owner, and those in management) understands and follows the practices of Scrum, and they help lead the organization through the often difficult change required to achieve success with agile development. Since Scrum makes visible many impediments and threats to the team's and Product Owner's effectiveness, it is important to have an engaged ScrumMaster working energetically to help resolve those issues, or the team or Product Owner will find it difficult to succeed. Scrum teams should have a dedicated full-time ScrumMaster, although a smaller team might have a team member play this role (carrying a lighter load of regular work when they do so). Great ScrumMasters can come from any background or discipline: Engineering, Design, Testing, Product Management, Project Management, or Quality Management.

The ScrumMaster and the Product Owner cannot be the same individual; at times, the ScrumMaster may be called upon to push back on the Product Owner (for example, if they try to introduce new deliverables in the middle of a Sprint). And unlike a project manager, the ScrumMaster does not tell people what to do or assign tasks – they facilitate the process,

supporting the team as it organizes and manages itself. If the ScrumMaster was previously in a position managing the team, they will need to significantly change their mindset and style of interaction for the team to be successful with Scrum. In the case that an ex-manager transitions to the role of ScrumMaster, it is best to serve a team other than the one that previously reported to the manager, otherwise the social or power dynamics are in potential conflict.

Note there is no role of project manager in Scrum. Sometimes an (ex-)project manager can step into the role of ScrumMaster, but this has a mixed record of success – there is a fundamental difference between the two roles, both in day-to-day responsibilities and in the mindset required to be successful. A good way to understand thoroughly the role of the ScrumMaster, and start to develop the core skills needed for success, is the Scrum Alliance's Certified ScrumMaster training.

In addition to these three roles, there are other contributors to the success of the product, including **managers**. While their role changes in Scrum, they remain valuable. For example:

- they support the team by respecting the rules and spirit of Scrum
- they help remove impediments that the team identifies
- they make their expertise and experience available to the team

In Scrum, these individuals replace the time they previously spent playing the role of "nanny" (assigning tasks, getting status reports, and other forms of micromanagement) with time as "guru" and "servant" of the team (mentoring, coaching, helping remove obstacles, helping problem-solve, providing creative input, and guiding the skills development of team members). In this shift, managers may need to change their management style; for example, using Socratic questioning to help the team discover the solution to a problem, rather than simply deciding a solution and assigning it to the team.

Starting Scrum

The first step in Scrum is for the Product Owner to articulate the product vision. Eventually, this evolves into a refined and prioritized list of features called the **Product Backlog**. This backlog exists (and evolves) over the lifetime of the product; it is the product road map (Figure 2). At any point, the Product Backlog is the single, definitive view of "everything that could be done by the team ever, in order of priority". Only a single Product Backlog exists; this means the Product Owner is required to make prioritization decisions across the entire spectrum.

					New Estimates of Effort Remaining as of Sprint...					
Item	Details (wiki URL)	Priority	Estimate of Value	Initial Estimate of Effort	1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5						
As a buyer, I want to remove a book in a shopping cart	...	2	6	2						
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13						
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20						
Upgrade all servers to Apache 2.2.3	...	5	5	13						
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3						
As a shopper, I want to create and save a wish list	...	7	7	40						
As a shopper, I want to to add or delete items on my wish list	...	8	4	20						

Figure 2. The Product Backlog

The Product Backlog includes a variety of **items**, primarily new customer features ("enable all users to place book in shopping cart"), but also engineering improvement goals ("rework the transaction processing module to make it scalable"), exploratory or research work ("investigate solutions for speeding up credit card validation"), and, possibly, known defects ("diagnose and fix the order processing script errors"), if there are only a few problems. (A system with many defects usually has a separate defect tracking system.) Many people like to articulate the requirements in terms of "user stories" concise, clear descriptions of the functionality in terms of its value to the end user of the product.

The subset of the Product Backlog that is intended for the current release is known as the **Release Backlog**, and in general, this portion is the primary focus of the Product Owner.

The Product Backlog is continuously updated by the Product Owner to reflect changes in the needs of the customer, new ideas or insights, moves by the competition, technical hurdles that appear, and so forth. The team provides the Product Owner with estimates of the effort required for each item on the Product Backlog. In addition, the Product Owner is responsible for assigning a *business value estimate* to each individual item. This is usually an unfamiliar practice for a Product Owner. As such, it is something a ScrumMaster may help the Product Owner learn to do. With these two estimates (effort and value) and perhaps with additional risk estimates, the Product Owner prioritizes the backlog (actually, usually just the Release Backlog subset) to maximize ROI (choosing items of high value with low effort) or secondarily, to reduce some major risk. As will be seen, these effort and value estimates may be refreshed each Sprint as people learn; consequently, this is a continuous re-prioritization activity the Product Backlog is ever-evolving.

Scrum does not mandate the form of estimates in the Product Backlog, but it is common to use *relative estimates* expressed as "points" rather than absolute units of effort such as person-weeks.

Over time, a team tracks how many relative points they implement each Sprint; for example, averaging 26 points per Sprint. With this information they can project a release date to complete all features, or how many features can be completed by a fixed date.

The items in the Product Backlog can vary significantly in size or effort. Larger ones are broken into smaller items during the Product Backlog Refinement workshop or the Sprint Planning Meeting, and smaller ones may be consolidated.

One of the myths about Scrum is that it prevents you from writing detailed specifications; in reality, it is up to the Product Owner and Team to decide how much detail is required, and this will vary from one backlog item to the next, depending on the insight of the team, and other factors. State what is important in the least amount of space necessary – in other words, do not describe every possible detail of an item, just make clear what is necessary for it to be understood. Low priority items, far from being implemented and usually "coarse grained" or large, have less requirements details. High priority and fine-grained items that will soon be implemented tend to have more detail.

Sprint Planning

At the beginning of each Sprint, the **Sprint Planning Meeting** takes place. It is divided into two distinct sub-meetings, the first of which is called **Sprint Planning Part One**.

In Sprint Planning Part One, the Product Owner and Team (with facilitation from the ScrumMaster) review the high-priority items in the Product Backlog that the Product Owner is

interested in implementing this Sprint. They discuss the goals and context for these high-priority items on the Product Backlog, providing the Team with insight into the Product Owner's thinking. The Product Owner and Team also review the "Definition of Done" that all items must meet, such as, "Done means coded to standards, reviewed, implemented with unit test-driven development (TDD), tested with 100 percent test automation, integrated, and documented." Part One focuses on understanding *what* the Product Owner wants. According to the rules of Scrum, at the end of Part One the (always busy) Product Owner may leave although they *must* be available (for example, by phone) during the next meeting. However, they are welcome to attend Part Two...

Sprint Planning Part Two focuses on detailed task planning for *how* to implement the items that the team decides to take on. The Team selects the items from the Product Backlog they commit to complete by the end of the Sprint, starting at the top of the Product Backlog (in others words, starting with the items that are the highest priority for the Product Owner) and working down the list in order. This is a key practice in Scrum: The team decides how much work they will commit to complete, rather than having it assigned to them by the Product Owner. This makes for a more reliable commitment because the team is making it based on their own analysis and planning, rather than having it "made" for them by someone else. While the Product Owner does not have control over how much the team commits to, he or she knows that the items the team is committing to are drawn from the top of the Product Backlog – in other words, the items that he or she has rated as most important. The team has the authority to also select items from further down the list; this usually happens when the team and Product Owner realize that something of lower priority fits easily and appropriately with the high priority items.

The Sprint Planning Meeting will often last a number of hours – the team is making a serious commitment to complete the work, and this commitment requires careful thought to be successful. The team will probably begin the Sprint Planning Part Two by estimating how much time each member has for Sprint-related work – in other words, their average workday minus the time they spend attending meetings, doing email, taking lunch breaks, and so on. For most people this works out to 4-6 hours of time per day available for Sprint-related work. See Figure 3.

Sprint Length	2 weeks
Workdays during Sprint	8 days

Team Member	Available Days During Sprint*	Available Hours per Day	Total Available Hours
Tracy	8	4	32
Sanjay	7	5	35
Phillip	8	4	32
Jing	6	5	30

* Net of vacation and other days out of office

Figure 3. Estimating Available Hours

Once the time available is determined, the team starts with the first item on the Product Backlog – in other words, the Product Owner's highest priority item – and working together, breaks it down into individual tasks, which are recorded in a document called the **Sprint Backlog** (Figure 4). As mentioned, the Product Owner must be available during Part Two (such as via the phone) so that clarification is possible. The team will move sequentially down the Product Backlog in this way, until it's used up all its available hours. At the end of the

meeting, the team will have produced a list of all the tasks with estimates (typically in hours or fractions of a day).

Scrum encourages multi-skilled workers, rather than only "working to job title" such as a "tester" only doing testing. In other words, team members "go to where the work is" and help out as possible. If there are many testing tasks, then *all* team members may help. This does not imply that everyone is a generalist; no doubt some people are especially skilled in testing (and so on) but team members work together and learn new skills from each other. Consequently, during task generation and estimation in Sprint Planning, it is not necessary – nor appropriate – for people to volunteer for all the tasks "they can do best." Rather, it is better to only volunteer for one task at a time, when it is time to pick up a new task, and to consider choosing tasks that will on purpose involve learning (perhaps by pair work with a specialist).

All that said, there are *rare* times when *John* may do a particular task because it would take far too long or be impossible for others to learn – perhaps John is the only person with any artistic skill to draw pictures. Other team members could not draw a "stick man" if their life depended on it. In this rare case – and if it is not rare and not getting rarer as the team learns, there is something wrong – it may be necessary to ask if the total planned drawing tasks that *must* be done by John are feasible within the short Sprint.

Many teams also make use of a visual task-tracking tool, in the form of a wall-sized task board where tasks (written on Post-It Notes) migrate during the Sprint across columns labeled "Not Yet Started," "In Progress," and "Completed." See Figure 5.

				New Estimates of Effort Remaining as of Day...					
Product Backlog Item	**Sprint Task**	**Volunteer**	**Initial Estimate of Effort**	**1**	**2**	**3**	**4**	**5**	**6**
As a buyer, I want to place a book in a shopping cart	modify database		5						
	create webpage (UI)		8						
	create webpage (Javascript logic)		13						
	write automated acceptance tests		13						
	update buyer help webpage		3						
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5						
	complete machine order for pRank		8						
	change DCP and reader to use pRank http API		13						

Figure 4. Sprint Backlog

Figure 5. Visual Management - Sprint Backlog tasks on the wall

One of the pillars of Scrum is that once the Team makes its commitment, any additions or changes must be deferred until the next Sprint. This means that if halfway through the Sprint the Product Owner decides there is a new item he would like the team to work on, he cannot make the change until the start of the next Sprint. If an external circumstance appears that significantly changes priorities, and means the team would be wasting its time if it continued working, the Product Owner or the team can terminate the Sprint. The team stops, and a new Sprint Planning meeting initiates a new Sprint. The disruption of doing this is usually great; this serves as a disincentive for the Product Owner or team to resort to this dramatic decision.

There is a powerful, positive influence that comes from the team being protected from changing goals during the Sprint. First, the team gets to work knowing with absolute certainty that its commitments will not change, that reinforces the team's focus on ensuring completion. Second, it disciplines the Product Owner into really thinking through the items he or she prioritizes on the Product Backlog and offers to the team for the Sprint.

By following these Scrum rules the Product Owner gains two things. First, he or she has the confidence of knowing the team has made a commitment to complete a realistic and clear set of work they have chosen. Over time a team can become quite skilled at choosing and delivering on a realistic commitment. Second, the Product Owner gets to make whatever changes he or she likes to the Product Backlog before the start of the *next* Sprint. At that point, additions, deletions, modifications, and re-prioritizations are all possible and acceptable. While the Product Owner is not able to make changes to the selected items under development during the current Sprint, he or she is only one Sprint's duration or less away from making any changes they wish. Gone is the stigma around change – change of direction, change of requirements, or just plain changing your mind – and it may be for this reason that Product Owners are usually as enthusiastic about Scrum as anyone.

Daily Scrum

Once the Sprint has started, the Team engages in another of the key Scrum practices: The **Daily Scrum**. This is a short (15 minutes or less) meeting that happens every workday at an appointed time. Everyone on the Team attends. To keep it brief, it is recommended that everyone remain standing. It is the team's opportunity to report to each other on progress and obstacles. In the Daily Scrum, one by one, each member of the team reports three (and only three) things *to the other members of the team*: (1) What they were able to get done since the last

meeting; (2) what they are planning to finish by the next meeting; and (3) any blocks or impediments that are in their way. Note that the Daily Scrum is not a status meeting to report to a manager; it is a time for a self-organizing team to share with each other what is going on, to help them coordinate. Someone makes note of the blocks, and the ScrumMaster is responsible to help team members resolve them. There is no discussion during the Daily Scrum, only reporting answers to the three questions; if discussion is required it takes place immediately after the Daily Scrum in a follow-up meeting, although in Scrum no one is required to attend this. This follow-up meeting is a common event where the team adapts to the information they heard in the Daily Scrum: in other words, another inspect and adapt cycle. It is generally recommended *not* to have managers or others in positions of perceived authority attend the Daily Scrum. This risks making the team feel "monitored" – under pressure to report major progress every day (an unrealistic expectation), and inhibited about reporting problems – and it tends to undermine the team's self-management, and invite micromanagement. It would be more useful for a stakeholder to instead reach out to the team following the meeting, and offer to help with any blocks that are slowing the team's progress.

Updating Sprint Backlog & Sprint Burndown Chart

Every day, the team members update their estimate of the amount of time remaining to complete their current task in the **Sprint Backlog** (Figure 6). Following this update, someone adds up the hours remaining for the team as a whole, and plots it on the **Sprint Burndown Chart** (Figure 7). This graph shows, each day, a new estimate of how much work (measured in person hours) remains until the team's tasks are finished. Ideally, this is a *downward* sloping graph that is on a trajectory to reach "zero effort remaining" by the last day of the Sprint. Hence it is called a *burndown* chart. And while sometimes it looks good, often it does not; this is the reality of product development. The important thing is that it shows the team their progress towards their goal, not in terms of how much time was *spent* in the past (an irrelevant fact in terms of *progress*), but in terms of how much work *remains in the future* – what separates the team from their goal. If the burndown line is not tracking downwards towards completion near the end of the Sprint, then the team needs to adjust, such as to reduce the scope of the work or to find a way to work more efficiently while still maintaining a sustainable pace.

While the Sprint Burndown chart can be created and displayed using a spreadsheet, many teams find it is more effective to show it on paper on a wall in their workspace, with updates in pen; this "low-tech/high-touch" solution is fast, simple, and often more visible than a computer chart.

				New Estimates of Effort Remaining at end of Day...					
Product Backlog Item	**Sprint Task**	**Volunteer**	**Initial Estimate of Effort**	**1**	**2**	**3**	**4**	**5**	**6**
As a buyer, I want to place a book in a shopping cart	modify database	Sanjay	5	4	3	0	0	0	
	create webpage (UI)	Jing	3	3	3	2	0	0	
	create webpage (Javascript logic)	Tracy & Sam	2	2	2	2	1	0	
	write automated acceptance tests	Sarah	5	5	5	5	5	0	
	update buyer help webpage	Sanjay & Jing	3	3	3	3	3	0	
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5	5	5	5	5	5	
	complete machine order for pRank		3	3	8	8	8	8	
	change DCP and reader to use pRank http API		5	5	5	5	5	5	
	. . .	. . .	. . .						
		Total (person hours)	50	49	48	44	43	34	

Figure 6. Daily Updates of Work Remaining on the Sprint Backlog

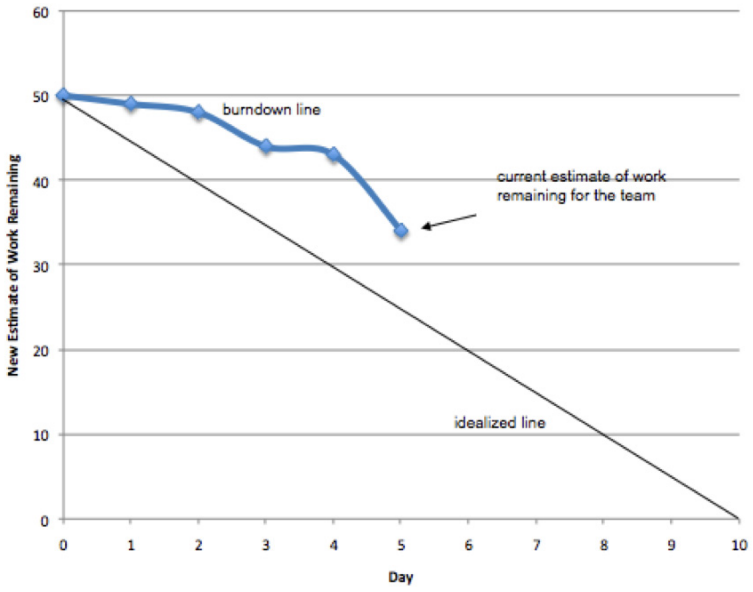

Figure 7. Sprint Burndown Chart

Product Backlog Refinement

One of the lesser known, but valuable, guidelines in Scrum is that five or ten percent of each Sprint must be dedicated by the team to refining (or "grooming") the Product Backlog. This includes detailed requirements analysis, splitting large items into smaller ones, estimation of new items, and re-estimation of existing items. Scrum is silent on how this work is done, but we suggest a focused workshop near the end of the Sprint, so that the team and Product Owner can dedicate themselves to this work without interruption. For a two-week Sprint, five percent of the duration implies that each Sprint there is a half-day Product Backlog Refinement workshop. This refinement activity is not for items selected for the current Sprint; it is for items for the future, most likely in the next one or two Sprints. With this practice, Sprint Planning becomes relatively simple because the Product Owner and Scrum Team start the planning with a clear, well-analyzed and carefully estimated set of items. A sign that this refinement workshop is not being done (or not being done well) is that Sprint Planning involves significant questions, discovery, or confusion.

Ending the Sprint

One of the core tenets of Scrum is that the duration of the Sprint is never extended – it ends on the assigned date regardless of whether the team has completed the work it committed to. Teams typically over-commit in their first few Sprints and fail to meet their objectives. They might then overcompensate and under-commit, and finish early. But by the third or fourth Sprint, teams typically have figured out what they are capable of delivering (most of the time), and they will meet their Sprint goals more reliably after that. Teams are encouraged to pick one duration for their Sprints (say, two weeks) and not change it. A consistent duration helps the team learn how much it can accomplish, which helps in both estimation and longer-term release planning. It also helps the team achieve a rhythm for their work; this is often referred to as the "heartbeat" of the team in Scrum.

Sprint Review

After the Sprint ends, there is the **Sprint Review**, where the team reviews the Sprint with the Product Owner. This is often mislabeled the "demo" but that does not capture the real intent of this meeting. A key idea in Scrum is *inspect and adapt*. To see and learn what is going on and then evolve based on feedback, in repeating cycles. The Sprint Review is an inspect and adapt activity for the *product*. It is a time for the Product Owner to learn what is going on with the product and with the team (that is, a review of the Sprint); and for the team to learn what is going on with the Product Owner and the market. Consequently, the most important element of the Review is an in-depth *conversation* between the team and Product Owner to learn the situation, to get advice, and so forth. The review includes a demo of what the team built during the Sprint, but if the focus of the review is a demo rather than conversation, there is an imbalance.

A useful – but often overlooked – Scrum guideline is that the ScrumMaster is responsible for knowing the "Definition of Done" that was defined during Sprint Planning, and then during this meeting is responsible for telling the Product Owner if any of the items implemented by the team did not meet the definition. In this way, there is increased visibility regarding the quality of the work; teams cannot fake the quality by presenting software that appears to work well, but may be implemented with a messy pile of poor quality and untested code.

Present at this meeting are the Product Owner, Team members, and ScrumMaster, plus customers, stakeholders, experts, executives, and anyone else interested. The demo portion of the Sprint Review is not a "presentation" the team gives – there is no slideware. A guideline in Scrum is that no more than 30 minutes should be spent preparing for the demo, otherwise it suggests something is wrong with the work of the team. It is simply a demo of what has been built. Anyone present is free to ask questions and give input.

Sprint Retrospective

The Sprint Review involves inspect and adapt regarding the *product*. The **Sprint Retrospective**, which follows the Review, involves inspect and adapt regarding the *process*. This is a practice that some teams skip, and that's unfortunate, because it's the main mechanism for taking the visibility that Scrum provides into areas of potential improvement, and turning it into results. It's an opportunity for the team to discuss what's working and what's not working, and agree on changes to try. The Team and ScrumMaster will attend, and the Product Owner is welcome but not required to attend. Sometimes the ScrumMaster can act as an effective facilitator for the retrospective, but it may be better to find a neutral outsider to facilitate the meeting; a good approach is for ScrumMasters to facilitate each others' retrospectives, which enables cross-pollination among teams.

A simple way to structure the Sprint Retrospective is to draw two columns on a whiteboard, labeled "What's Working Well" and "What Could Work Better" – and then go around the room, with each person adding one or more items to either list. As items are repeated, check marks are added next to them, so the common items become clear. Then the team looks for underlying causes, and agrees on a small number of changes to try in the upcoming Sprint, along with a commitment to review the results at the next Sprint Retrospective.

A useful practice at the end of the Retrospective is for the team to label each of the items in each column with either a "C" if it is *caused* by Scrum (in other words, without Scrum it would

not be happening), or an "E" if it is *exposed* by Scrum (in other words, it would be happening with or without Scrum, but Scrum makes it known to the team), or a "U" if it's unrelated to Scrum (like the weather). The team may find a lot of C's on the "What's Working Well" side of the board, and a lot of E's on the "What Could Work Better "; this is good news, even if the "What Could Work Better" list is a long one, because the first step to solving underlying issues is making them visible, and Scrum is a powerful catalyst for that.

Updating Release Backlog & Burndown Chart

At this point, some items have been finished, some have been added, some have new estimates, and some have been dropped from the release goal. The Product Owner is responsible for ensuring that these changes are reflecting in the Release Backlog (and more broadly, the Product Backlog). In addition, Scrum includes a **Release Burndown** chart that shows progress towards the release date. It is analogous to the Sprint Burndown chart, but is at the higher level of items (requirements) rather than fine-grained tasks. Since a new Product Owner is unlikely to know why or how to create this chart, this is another opportunity for a ScrumMaster to help the Product Owner. See Figure 8 and Figure 9 for an example of the Release Backlog and Release Burndown chart.

					New Estimates of Effort Remaining at end of Sprint...					
Item	**Details (wiki URL)**	**Priority**	**Estimate of Value**	**Initial Estimate of Effort**	**1**	**2**	**3**	**4**	**5**	**6**
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5	0	0	0			
As a buyer, I want to remove a book in a shopping cart	...	2	6	2	0	0	0			
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13	13	0	0			
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20	20	20	0			
Upgrade all servers to Apache 2.2.3	...	5	5	13	13	13	13			
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3	3	3	3			
As a shopper, I want to create and save a wish list	...	7	7	40	40	40	40			
As a shopper, I want to to add or delete items on my wish list	...	8	4	20	20	20	20			
. . .				. . .	. . .					
			Total	**537**	**580**	**570**	**500**			

Figure 8. Release Backlog (a subset of the Product Backlog)

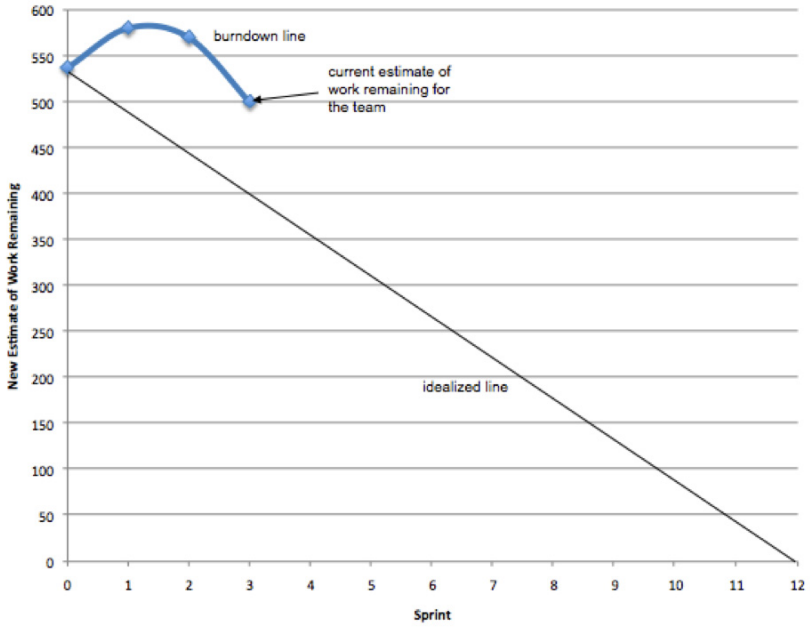

Figure 9. Release Burndown Chart

Starting the Next Sprint

Following the Sprint Review, the Product Owner may update the Product Backlog with any new insight. At this point, the Product Owner and team are ready to begin another Sprint cycle. There is no down time between Sprints – teams normally go from a Sprint Retrospective one afternoon into the next Sprint Planning the following morning (or after the weekend).

One of the principles of agile development is "sustainable pace", and only by working regular hours at a reasonable level can teams continue this cycle indefinitely.

Release Sprint

The perfection vision of Scrum is that the product is potentially shippable at the end of each Sprint, which implies there is no wrap up work required, such as testing or documentation. Rather, the implication is that *everything* is completely *finished* every Sprint; that you could actually ship it or deploy it immediately after the Sprint Review.

However, many organizations have weak development practices and cannot achieve this perfection vision, or there are other extenuating circumstances (such as, "the machine broke"). In this case, there will be some remaining work, such as final production environment integration testing, and so there will be the need for a "Release Sprint" to handle this remaining work.

Note that the need for a Release Sprint is a sign of some weakness; the ideal is that it is not required. When necessary, Sprints continue until the Product Owner decides the product is almost ready for release, at which point there will be a Release Sprint to prepare for launch. If the team has followed good development practices, with continuous refactoring and integration, and effective testing during each Sprint, there should be little pre-release stabilization or other wrap-up work required.

Release Planning & Initial Product Backlog Refinement

A question that is sometimes asked is how, in an iterative model, can long-term release planning be done. There are two cases to consider: (1) a new product in its first release, and (2) an existing product in a later release.

In the case of a new product, or *an existing product just adopting Scrum*, there is the need to do initial Product Backlog refinement before the first Sprint, where the Product Owner and team shape a proper Scrum Product Backlog. This could take a few days or a week, and involves a vision workshop, some detailed requirements analysis, and estimation of all the items identified for the first release.

Surprisingly in Scrum, in the case of an established product with an established Product Backlog, there should not be the need for any special or extensive release planning for the next release. Why? Because the Product Owner and team should be doing Product Backlog refinement every Sprint (five or ten percent of each Sprint), continuously preparing for the future. This *continuous product development* mode obviates the need for the dramatic punctuated prepare-execute-conclude stages one sees in traditional sequential life cycle development.

During an initial Product Backlog refinement workshop and during the continuous backlog refinement each Sprint, the Team and Product Owner will do release planning, refining the estimates, priorities, and content as they learn.

Some releases are date-driven; for example: "We will release version 2.0 of our project at a trade-show on November 10." In this situation, the team will complete as many Sprints (and build as many features) as is possible in the time available. Other products require certain features to be built before they can be called complete and the product will not launch until these requirements are satisfied, however long that takes. Since Scrum emphasizes producing potentially shippable code each Sprint, the Product Owner may choose to start doing interim releases, to allow the customer to reap the benefits of completed work sooner.

Since they cannot possibly know everything up front, the focus is on creating and refining a plan to give the release broad direction, and clarify how tradeoff decisions will be made (scope versus schedule, for example). Think of this as the roadmap guiding you towards your final destination; which exact roads you take and the decisions you make during the journey may be determined en route.

Most Product Owners choose one release approach. For example, they will decide a release date, and will work with the team to estimate the Release Backlog items that can be completed by that date. In situations where a "fixed price / fixed date / fixed deliverable" commitment is required – for example, contract development – one or more of those parameters must have a built-in buffer to allow for uncertainty and change; in this respect, Scrum is no different from other approaches.

Application or Product Focus

For applications or products – either for the market or for internal use within an organization – Scrum moves groups away from the older *project*-centric model toward a *continuous application/product development* model. There is no longer a project with a beginning, middle, and end. And hence no traditional project manager. Rather, there is simply a stable Product Owner and a long-lived self-managing team that collaborate in an "endless" series of two- or four-

week Sprints, until the product or application is retired. All necessary "project" management work is handled by the team and the business owner—who is an internal business customer or from Product Management. It is not managed by an IT manager or someone from a Project Management Office.

Scrum can also be used for true *projects* that are one-time initiatives (rather than work to create or evolve long-lived applications); still, in this case the team and Product Owner do the project management.

What if there is insufficient new work from one or more existing applications to warrant a dedicated long-lived team for each application? In this case, a stable long-lived team may take on items from one application in one Sprint, and then items from another in the next Sprint; in this situation the Sprints are often quite short, such as one week.

Occasionally, there is insufficient new work even for the prior solution, and the team may take on items from *several* applications during the same Sprint; however, beware this solution as it may devolve into unproductive multitasking across multiple applications. A basic productivity theme in Scrum is for the team to be *focused* on one product or application for one Sprint.

Common Challenges

Scrum is not only a concrete set of practices – rather, and more importantly, it is a framework that provides visibility to the team, and a mechanism that allows them to "inspect and adapt" accordingly. Scrum works by making visible the dysfunction and impediments that are impacting the Product Owner and the team's effectiveness, so that they can be addressed. For example, the Product Owner may not really know the market, the features, or how to estimate their relative business value. Or the team may be unskillful in effort estimation or development work.

The Scrum framework will quickly reveal these weaknesses. Scrum does not solve the problems of development; it makes them painfully visible, and provides a framework for people to explore ways to resolve problems in short cycles and with small improvement experiments.

Suppose the team fails to deliver what they committed to in the first Sprint due to poor task analysis and estimation skill. To the team, this feels like failure. But in reality, this experience is the necessary first step toward becoming more realistic and thoughtful about their commitments. This pattern – of Scrum helping make visible dysfunction, enabling the team to do something about it – is the basic mechanism that produces the most significant benefits that teams using Scrum experience.

One common mistake teams make, when presented with a Scrum practice that challenges them, is to change Scrum, not change themselves. For example, teams that have trouble delivering on their Sprint commitment might decide to make the Sprint duration extendable, so they never run out of time – and in the process, ensure they never have to learn how to do a better job of estimating and managing their time. In this way, without coaching and the support of an experienced ScrumMaster, organizations can mutate Scrum into just a mirror image of its own weaknesses and dysfunction, and undermine the real benefit that Scrum offers: Making visible the good and the bad, and giving the organization the choice of elevating itself to a higher level.

Another common mistake is to assume that a practice is discouraged or prohibited just because Scrum does not specifically require it. For example, Scrum does not require the

Product Owner to set a long-term strategy for his or her product; nor does it require engineers to seek advice from more experienced engineers about complex technical problems. Scrum leaves it to the individuals involved to make the right decision; and in most cases, both of these practices (along with many others) are well advised.

Something else to be wary of is managers imposing Scrum on their teams; Scrum is about giving a team space and tools to manage themselves, and having this dictated from above is not a recipe for success. A better approach might begin with a team learning about Scrum from a peer or manager, getting comprehensively educated in professional training, and then making a decision as a team to follow the practices faithfully for a defined period; at the end of that period, the team will evaluate its experience, and decide whether to continue.

The good news is that while the first Sprint is usually very challenging to the team, the benefits of Scrum tend to be visible by the end of it, leading many new Scrum teams to exclaim: "Scrum is hard, but it sure is a whole lot better than what we were doing before!"

Results From Scrum

The benefits of Scrum reported by teams come in various aspects of their experience. At Yahoo!, we migrated nearly 200 teams to Scrum over three years, totaling over 2000 people. These have ranged from consumer-facing, design-heavy websites such as Yahoo! Photos, to the mission-critical back-end infrastructure of services such as Yahoo! Mail, which serves hundreds of millions of customers.

Several times each year we surveyed everyone at Yahoo! using Scrum (including Product Owners, Team Members, ScrumMasters, and the functional managers of those individuals) and ask them to compare Scrum to the approach they were using previously. Some summary data is presented here:

- **Productivity:** 68% of respondents reported Scrum is better or much better (4 or 5 on a 5-point scale); 5% reported Scrum is worse or much worse (1 or 2 on a 5-point scale); 27% reported Scrum is about the same (3 on a 5-point scale).
- **Team Morale:** 52% of respondents reported Scrum is better or much better; 9% reported Scrum is worse or much worse; 39% reported Scrum is about the same.
- **Adaptability:** 63% of respondents reported Scrum is better or much better; 4% reported Scrum is worse or much worse; 33% reported Scrum is about the same.
- **Accountability:** 62% of respondents reported Scrum is better or much better; 6% reported Scrum is worse or much worse; 32% reported Scrum is about the same.
- **Collaboration and Cooperation:** 81% of respondents reported Scrum is better or much better; 1% reported Scrum is worse or much worse; 18% reported Scrum about the same.
- **Team productivity increased an average of 36%, based on the estimates of the Product Owners.**
- **85% of team-members stated that they would continue using Scrum if the decision were solely up to them (15% said either "No" or "Undecided").**