A Scrum Introduction

By Joe Little

DRAFT – DRAFT – DRAFT

Ver 0.90

.

Table of Contents

Note on Current Draft

This is a *nearly complete* draft of this book.

We have some things to add. We have not proofread it enough yet. We are starting to add pictures.

We welcome your comments now. You might find:

- a typo
- a topic (section) you think we should add
- a question.

Please send your comments to me at jhlittle@leanagiletraining.com.

We want the book to be fairly short. Our initial goal was to get it twice as long as the Scrum Guide, but it looks like it will be a bit longer than that. Readable probably in one day.

We want the main use to be: If you are about to take my CSM or CSPO course, you can read it usefully. Or, if you have just taken the course, and want to remind yourself what you learned, it's a good summary of the basics. Of course, there are many other uses.

You can find the current version **here** for now.

Thanks in advance for your feedback.

Introduction

I built this book initially as a series of blog posts to explain quickly what Scrum is. It turns out that the basics of Scrum are very simple, and yet difficult to explain concisely.

Scrum is:

- Simple
- Hard to Master

The main purpose of this book is to give my course attendees a bit more of an introduction than just the Scrum Guide.

It is key that this book remain short. For those who have special cases or special needs, it will *not* answer every question. I am sorry that it may not directly address your specific need, but I hope you also see the value in it being short!

Scrum is incomplete. That is, it is not trying to include "everything" you will need to be successful as a Team. It is trying to include the essential things, but not everything. Add things that help your Team in your specific situation.

You are expected to add to it. And give your energy to the Team and the Game. And your skills. Come up with a playing strategy, given your overall situation. And then try to win.

It is a kind of a Game. We think, in most reasonable situations, your odds of "winning" in real life will improve notably.

What Is Scrum?

Scrum is an Agile method co-created by Jeff Sutherland and Ken Schwaber in the early 1990s.

It was known then, and has been known since, to enable teams to achieve greater productivity, to find work more satisfying and to produce wonderful products for customers.

Scrum is not a miracle, it is not a panacea, it is no guarantee, but teams regularly improve to a very impressive degree by using it.

Scrum is one of many Agile methods. Agile methods are usually defined as those methods that follow the Agile Manifesto and the Agile Principles. (See AgileManifesto.org.) Jeff Sutherland and Ken Schwaber were both present when the Agile Manifesto and Agile Principles were created. They helped create them.

One of the key things about Scrum is that it is simple.

Scrum is only providing a bare framework for a team to use to get work done. We think - with the fewest possible constraints. We want it to enable the team to pop up to a new higher level of functioning.

What is Scrum essentially? This is hard to say. It was created by two men in New England who like to express themselves practically, so you may have to experience Scrum for a while before you can answer that question.

From my experience, your answer will change over time.

Team Sport

Scrum is a team sport. Scrum tries to enable a team to be more successful. In part, to evaluate how they are doing, and then decide what to do in their situation. (Some of you may know the phrase “Inspect and Adapt”.)

I mean a real team. A real team pulls it (and themselves) together, so that the Team accomplishes more than anyone expected.

For example, all members of the team are expected to collaborate. Each person is expected to take full ownership of success.

This does not mean that everyone is exactly equal or that they all have equal skill sets or skill in all areas. Still, together, they are taking on the goal of success. They are taking on the vision as defined (usually and mainly) by the Product Owner.

I’m talking about a real team.

Virtually everyone in business has been told, “You are on team X.” Mostly those are work groups or something similar — not real teams.

Among the attributes of a real team is that each member is 100 percent dedicated to only one team, the team has a common purpose and all members of the team are invested in that purpose and in accomplishing that goal.

As you form Teams, you will see that many people have never been in a real team, as I defined it.

I think you will find that, if they become a good team (eg, not a dysfunctional team), then most people (90%) will enjoy being in a Team. But some will not. Or at least it is unclear if you will ever convince them to want to be a team member.

At some point, you will have to accept and deal with the fact that a small percentage of people will not want to be in a Scrum team.

Team Roles

Within the team, Scrum defines three roles:

- Product Owner
- ScrumMaster
- Developer (role)

Note that the 2017 Scrum Guide calls the Develop role the “team” role, which I think is confusing to beginners. It gives the impression that there is a Dev Team within the Scrum Team. I find this is not helpful. There is only one Team: the Scrum Team. The Team wins together or loses together. Everyone in the Team must contribute (to be truly successful).

The Chicken and Pig Story

Short Note: Apologies if a little story like an Aesop’s Fables offends you (we are told that some people or cultures take offense). At least you must know that the purpose is, as with Aesop, to educate, and not to offend. In my culture, it is not offensive. Far from demeaning people, the purpose is to help people become better.

Scrum lore has the “Chicken and the Pig” story. It is not used as often as it once was.

I will not give the story here, but the idea is that the pigs are committed, and the chickens are “only involved.”

The pigs refers to the people in the Team. Chickens are people outside the Team.

From the point of view of the so-called pigs, the reality that the chickens are “only involved” is ….problematic. The chickens are normally not as reliable as we (the Pigs) want them to be. Because the chickens have other things to do, different priority 1’s, so they will be less reliable — or there is a strong chance of a problem in some way.

Some think this means we think chickens are bad. No! Just less reliable than someone who is dedicated and focused on one goal. From the point of view of a chicken, he ir she is doing important work, too. It’s just that when they are helping the pigs with their work (eg, something not one of their main goals), the chickens are less reliable. To the pigs, the work of the chickens is likely to be late or the quality lower than wanted, or it is in some other way “not quite what we wanted.”

But we find chickens are always necessary. The pigs can never, in my experience, complete their work without some help from some chickens. To be fair, often the pigs can find other chickens — or do some of that work themselves in a pinch — but often enough, the Scrum Team (of pigs) finds their own success significantly dependent on the chickens.

Thus, we the Team must manage the chickens well. Everyone on the Team (Developers, SM, PO), in my view, must help manage the chickens. Because sooner or later the weak delivery by a chicken will become a key impediment.

Roles Outside the Scrum Team

I want to mention now two more roles outside of the team.

- **The Customer:** Customers are the real people who will use our product. They may be internal or external to the organization we work in. It is in delivering something wonderful to them that we get the greatest satisfaction. Typically, customers are in “pain” (in some sense or other) and we are delivering pain relief. This is urgent.

- **The Business Stakeholders:** I use this term to represent the people that must work with the team part-time, and especially be there for every Sprint Review to give good feedback. I will talk more about them later.

Why Scrum?

Why Scrum? How did it get here? How do we understand it? How do you explain it to your colleagues?

Why Was Scrum Invented?

The way I understand it, Jeff Sutherland became a software development group manager at some point in his career. The group was doing waterfall. Projects were failing. It was hard to understand what the real problems were (low transparency). People were demoralized. It was a mess, which is fairly typical for waterfall.

His reaction was that “there must be a better way”. (Not sure if he used those exact words, but essentially that.)

So, Scrum is a reaction to the pain, stupidity and unhappiness of waterfall. Or, to put it a better way, Scrum is an attempt to do several things at the same time — bring some fun back to work, give us a sense of mission, enable us to see that we have some traction and give us the transparency we need to make useful changes.

What Are the Ideas Behind Scrum?

There are many ideas behind Agile and Scrum, and I think that Sutherland and Schwaber could not accurately remember in 1999 all these ideas nor where they came from. People are easily influenced and can sometimes forget where those influences came from.

Scrum is a very interesting mixture of very simple ideas (e.g., KISS or Keep It Stupid Simple) and complex ideas (e.g., Complex Adaptive Systems ideas).

Agile Manifesto and Agile Principles

Scrum was invented before the Agile Manifesto and Agile Principles were articulated in 2001. Still, Schwaber and Sutherland were there at Snowbird in 2001 when the Agile Manifesto and Agile Principles were defined. They helped create them. They would say that Scrum “follows” those Agile ideas. Read the Agile Manifesto and the Agile Principles. Do you agree Scrum follows those Agile ideas?

The New New Product Development Game article

Scrum was also strongly influenced by many other ideas. Early on, maybe before any experimental teams, they read one particular Harvard Business Review article. So, one

set of ideas comes from "The New New Product Development Game" article by Hirotaka Takeuchi and Ikujiro Nonaka.

Here are the six ideas described there:

1. Built-in instability
2. Self-organizing project teams
3. Overlapping developmental phases
4. "Multi-learning"
5. Subtle control
6. Organizational transfer of learning

I cannot too strongly recommend that article and almost any article or book by Takeuchi and Nonaka.

Complex Adaptive Systems theory

Let me mention again the Complex Adaptive Systems ideas. See this link on Wikipedia for a start.

Ideas about People

I am convinced that Peter Drucker's idea about knowledge workers and similar ideas had a significant impact. Related are Takeuchi's, Nonaka's and others' ideas around knowledge creation.

In my opinion, Scrum is notably about people. That is, Scrum is an attempt to help get smart people working together much more effectively. So, embedded in Scrum are many ideas about people, how they work and how they might work together better.

At the same time, Scrum takes on none of the specific theories you may have read about (Theory X, Theory Y, and others.). (If you say "more Theory Y and Theory X", I might agree.) Scrum does not assume that a Team will self-organize well, that things will become the best that they can be "in this best of all possible worlds". It might be an interesting exercise to puzzle out the ideas about people implied in the existing Scrum rules.

Agile is sometimes thought of as being overly optimistic about people. Not so, in my opinion.

Scrum is *not* overly optimistic. It does not assume that people are always perfect; quite the contrary. Scrum seems well aware that humans have strengths but also have weaknesses. Examples: We are easily distracted, and we tend to procrastinate. So, Scrum does a few things to address these likely issues. On the other hand, Scrum is somewhat positive, in that it assumes that usually people can work together effectively and become more effective together.

Empirical Process Control theory

Ken Schwaber visited Tunde Ogunnaike in Delaware, working at DuPont at the time. Ogunnaike and Ray wrote the “bible”: Process Dynamics, Modeling, and Control. Schwaber was introduced to the ideas of Defined Processes (eg, waterfall) and Empirical Processes (eg, agile).

Hence the three key ideas. Obtain more transparency. Enable better inspection of the current situation (eg, how good the “completed” user story may be), and then adapt quickly. Transparency, Inspection and Adaption. Within the sprint, at the end of each sprint, and across sprints.

One can think of it this way, in part. We do not rely on what *should* happen, but rather on what *has* happened - the reality of the current situation (inputs (including people), outputs, everything).

Lean

As many of you know, Lean has been called many things over the years. J-I-T, Toyota Production System, Toyota Way, Lean Thinking, etc, etc. The ideas, especially from Taiichi Ohno, had a profound impact on Scrum.

And there are many ideas embedded within Lean. Single-piece continuous flow, small team, minimize work in process, focus on the essentials, using cards to manage the work, KISS, continuous improvement, visual management, eliminate waste, value stream mapping, the Five Whys, and on and on.

Experiments

Scrum did not arise from Sutherland and Schwaber sitting at the top of a mountain thinking big thoughts. As suggested, I think it arose from a hard, practical reality: *waterfall was not working*. And then they looked, read, sought and, then, experimented. It was the experiments that showed them they were on to something. They did not believe just the ideas — they believed the experimental results.

Note: These were experiments involving people working as a Team. The people inside the experiment are self-aware, have a will, have some relationship to the “experimenter”. So, this is not like a clean room experiment with non-conscious elements (even if animate).

One could say that at least some of the talk is an attempt (ex post) to explain why the experiments worked.

What we do know for sure is that many have played Scrum, and that a good percentage of them — if they do “all” of Scrum (or as close to all as we can reasonably expect) and

do that professionally with rigor — tend to get amazing results. Sometimes as much as five to ten times better than waterfall fairly quickly.

To the degree they do *not* do "all" of Scrum, the results come down quickly from amazing. Still, even "half-baked" Scrum tends to get them 20 percent better results.

But Wait, There's More…
There are many more ways to explain Scrum, to help others understand why a piece of Scrum helps or to talk about why Scrum works.

Here are a few ancient sayings (perhaps one not so ancient):

- "Two heads are better than one."
- "The whole is greater than the sum of its parts."
- "Many hands make light work."
- "Live and learn."
- "Go confidently in the direction of your dreams!" —Henry David Thoreau

Note that Sutherland and Schwaber both live outside of Boston, not far from Concord, Massachusetts where Thoreau lived. No doubt they were to some degree influenced by the New England and Boston culture (which of course still includes Emerson and Thoreau).

There still remain many more ideas that will help you explain Scrum. I enjoy using about half of the Yogi Berra quotes to explain Scrum, and it helps, I think, to use a humorous method for getting the concepts across. Two examples from Yogi Berra: "It ain't over till it's over," and, "When you come to a fork in the road, take it."

You will have to use all these methods and more to explain and explain. Because after a bit of time, your team will forget why they are doing this or that part of Scrum and, quite often, if they cannot explain to themselves why they are doing it, they will stop doing it. You, as the Agile advocate, must remind them again and again.

Scrum Roles

We mentioned earlier that a Scrum Team is composed of people in three roles.
A team should be seven people normally (or start off at seven):

- One Product Owner
- One ScrumMaster
- Five Developers

The Scrum Guide does not “require” you to have a fully dedicated team. I think if you read it carefully, you will see that they are urging you to have a fully dedicated team — and a stable team at that.

I strongly suggest that you do that. It is just common sense, once you pick the top priority goal to accomplish. Have a dedicated Team accomplish that goal.

The key reason: life is simpler for everyone. One team, one goal. There are several other reasons.

Scrum Team

The Scrum Team is key. They together, helping each other, accepting help, self-organizing and adapting to an ever-changing situation – they must make it happen. Build and deliver the product.

The whole is greater than the sum of the parts.

These are fundamental ideas in Scrum.

As one small example, it is the whole Team that makes (the Team’s) Velocity happen.

Note: The Velocity of a Sprint is the sum of the story points on each of the [8] user stories that the Team completed in the Sprint. Completed in accord with the Definition of Done.

The Velocity that we usually speak of is the average Velocity of that Team over the last 3 or 4 sprints. And that becomes the default expectation of what the Team will accomplish in the “average” sprints that are about to happen. And then the Team can inspect and adapt from that with good reason.

The Product Owner

The Product Owner (PO) ultimately owns the quality of the Product Backlog.

The PO must be decisive. So, the first key power: The PO must decide the order of each Product Backlog Item (aka user story) in the product backlog, quickly.

The PO should take input, within reason, from everyone. And then decide, under conditions of uncertainty. About the future impact of doing this story in this order. Based on “everything”. And future benefits and future costs are top of mind, but not the only considerations.

The PO is responsible for articulating the vision of what the Scrum Team is trying to accomplish. He or she must explain the vision in such a way that the team is inspired — not everyone can do that.

Then, the PO enables everyone to contribute to the Product Backlog and tries to do things (e.g., talk to people in various ways) to assure it is the best possible Product Backlog and will fulfill the vision.

The PO should inspire each team member. I said it before, but let’s emphasize: The PO should explain the vision and everything else so well that *of course* the others on the team feel that this product is something that they want to do.

The PO must make the “Unclear Requirements” that we have always had, and make them notably clearer.

The PO must help the company gather and convey the detailed “requirements” that go to the team, so they build the right thing, and build it sooner rather than later.

Getting all the right people to give all the right details in a timely manner, just enough, just in time — this is almost always something that the company is not good at doing at first. To be fair, it is hard to do. Even the customers do not know what they want (the full solution) and do not even explain the problem very well.

Still, the PO is responsible that the communication about “requirements” is much better so that the Team is more productive.

The PO must answer questions about the requirements. And/or get those questions that arise during the Sprint answered very quickly.

The PO must work with the rest of the team daily. For example, every day the team members will have questions about their “requirements” (at least this is very typical) and the PO must answer them (or get them answered). Ideally in 10 seconds, but 10 minutes would be OK — 24 hours should be the limit. To be fair, answering within 24 hours in a large organization can be tough. Overall, we need a much faster turnaround.

The PO must be a change agent, and change the business side (as some of us call it) so that these questions are answered more quickly and accurately.

The PO decides when we have a Minimum Viable Product, and when we release. Obviously, the PO must talk to others, but should be the final decision-makers.

The PO likes to work with the “geeks.” (I am teasing now about the divide between “the suits” and “the geeks” that many firms have now. We want them to learn to collaborate.) The PO should like learning about technology. This is important for her or him to order the Product Backlog better.

The PO is working with others outside the team on what I call Release Plan Refactoring. The Scrum Guide calls this Product Backlog Refinement. Other names of this rough type of work are backlog grooming and pre-planning.

The PO has to “herd the cats” — the Business Stakeholders. (I recommend four of them. We will discuss them later.) This is often hard.

The PO is a kind of leader, but the PO is not *the* leader of the team. The PO leads, of course, in deciding how to prioritize the Product Backlog. The PO leads by inspiring, and by articulating the vision. The PO is the final decision-maked on when to release (when do we have an MVP). But the PO does not lead in any other way.

In general, we typically want the PO to come from the business side. Several good things typically result from that. It can also be OK if he or she comes from technology (or whatever your firm calls it).

In general, we expect the PO to understand these areas well:

- The customers (internal or external)
- The customers’ problems
- The current product (or product set)
- The competition
- The business model
- The business situation
- The people on the business side (who understand all these things and the details well — so that the PO can, for example, help pull together the right details for the team).
- The product strategy
- The Team
- How this product fits in the organization’s strategy

In general, we expect the PO to not know much about technology at first. But by working with the team, the PO starts to understand technology better and better with time; we want that.

How Much Allocation?

How much should the PO be allocated? In general, always more. This is a key problem.

Some assumptions:

- The Team has an important and difficult mission or goal
- That is urgent

- The Team is seven people in total

  Then in that case, I'd recommend that the PO be 100 percent allocated to that Team (of seven).
  It makes a huge difference!

For example: One of our key problems is always unclear requirements. It is up to the PO to address this issue well. It's a lot of work.

In general, a 1 to 7 ratio of allocated hours is a reasonable ballpark. (One PO hour for every 6 hours from the five Doers and the ScrumMaster.)

Over and over and over… one of the biggest impediments is that the business side has not allocated enough time of the person who is playing the PO role. At first, this is almost universally the biggest impediment.

You can backfill some. For example, you can add a business analyst to the team or maybe a couple of part-time business analysts outside of the team, and that can make things somewhat better for a while.

You also really need more of the PO almost always. A good, highly allocated PO can improve the productivity of the team enormously and is well worth the expense.

One can imagine circumstances where it might be better to have a PO allocated part-time. But I usually feel I was forced into that situation, rather than "this is wonderful".

Still, it pays to have the PO allocated full-time if the team is working on an important project. (Well, they are working on one of the top projects in the company, right?)

Related: New POs are never very good. The person is often very good at the old job, but the PO role is notably different. The new person needs help in riding quickly up the learning curve; as I say it, to try to emulate the Wayne Gretzky of POs. This is hard to do if your company has no Wayne Gretzky.

In any case, get the new POs more training and coaching and other help to become better. It is trouble, but it is very much worth it.

The ScrumMaster

The ScrumMaster role is easy to misunderstand, and in fact it is commonly misunderstood.

- Servant Leader

We want the Scrum Master particularly to be a servant leader. (In a way, we want everyone to be a servant leader. For example, we want the Manager to also be a servant leader.) There many examples of a servant leader, not just Mother Teresa.

The SM asks the team: “What is your biggest impediment?” And then the SM asks: “May I help you with that?” Normally the Team would say: yes.

Impediments can be anything slowing down the Team or perhaps stopping them. Really, any opportunity for improvement.

As part of this, we want the SM to look at the people and the Team in a holistic way: are they well and good, as people. Not just as workers.

This servant leader idea does not mean that the SM becomes the slave to everyone. The SM is trying to help the Team to become a high-performing Team, so the SM is always working on the top impediment (typically the impediment with the best ROI).

- Teach the Team to self-organize

The ScrumMaster (SM) should help the Scrum Team learn how to self-organize.

Self-organizing is a thing we as humans all do. And we all can do more. It is odd that some people feel that they are not supposed to self-organize at work. So, you may find that your Team or some of your people do not self-organize well. Or do not do it well in some contexts. Fairly common.

Other teams struggle with this in part because other people (eg, managers) inhibit the self-organizing.

In my experience always the Team can learn to self-organize more and better. The SM helps them learn.

- Honesty and Transparency

We want more honesty, particularly in the Daily Scrum, as one example.

We want the Team and the organization to be more transparent. Not just tp tell the truth, and be more open. To share more, to allow the team members to have and use all the information available.

This can be hard.

People commonly od not want to reveal their weaknesses, shortcomings, and mistalkes. And yet people make mistakes all the time.

The SM can try to create a “safe environment”. But to do this perfectly is impossible. So, the SM must have the courage to be more honest and then to ask others to be more courageous by being more honest.

Not about everything, but about the work of the Team to build the product. And related matters.

- Build trust

The SM should build trust inside the Team. And also build trust between what is often called the business side and technology.

Each sprint the Team makes a promise and then either fulfills that promise, or not. By making promises and usually fulfilling them, we build the trust. Being honest also helps.

The SM cannot of course “make” the Team successful, but can certainly influence in these areas.

- Protect the Team from distractions

Distractions are one the key types of impediments.

The SM protects the team from distractions. Of all types or sub-types.

This, of course, is in some sense an impossible task — to eliminate all distractions. But to reduce distractions significantly is actually quite easy.

To reduce some important distractions can be challenging. Some managers in some situations want to distract the team. The manager may be senior and may feel he or she is helping the team. The SM must decide when it is not helping the team. That conversation might be challenging.

We often think of the SM as the sheepdog, protecting the Team.

- Educator

The SM teaches the team and the organization about Scrum. He or she is explaining agile-scrum all the time. They forget, they seem to actively mis-remember or they interpret it wrongly; often coming from the waterfall paradigm. So, the values, principles and practices of Lean-Agile-Scrum must be explained over and over.

- Improve the Velocity!!

By fixing impediments, we want the Velocity to get a lot better. (For now, think of Velocity as a measure of Team productivity.

Who is the key driver? The SM.

But the SM does not fix all the impediments by himself or herself.

The SM can fix some impediments (alone) where he or she has the right skill sets. The Team (esp. the Developers) have the skill sets to fix some impediments. And it is good that they take responsibility for their own problems some. And people outside the Scrum Team often have the best skill sets to fix many impediments.

The SM must mainly be making sure we are making progress on the top impediment for the Team. So, the SM if often helping or encouraging the Fixer. Or keeping the Fixers undistracted.

Sutherland says a typical SM should be able to increase Velocity 100% in the first 6 months. Longer-term, we believe virtually every team can become hyper-productive. That means, productivity has increased 5x to 10x from the baseline.

At the same time, happiness should be higher (see the Happiness Metric), Business Value should of course also be significantly up, quality should improve, and the Team should be working fewer hours (eg, in the 40/hrs/week range).

[Move to Impediment List]

An impediment is anything that is slowing the team down. The Impediment List should include the top 20 or 30 impediments – areas where we, the organization or life need to improve — so the team can be better.

An impediment is not only a distraction, not only a blocker to one story, not only a system down problem, not only… anything. An impediment can be anything that is slowing down the team: people issues, anything that a team member is not perfect at doing, problems with management, organizational issues, technical issues, etc.

One of the key impediments is that the PO sucks. OK, a little dramatic, and at least partially not true. That is, the PO is typically very good at his or her prior role. But that person is not used to this very different role.

Who fixes impediments? Well, obviously a typical SM is good at fixing certain types of impediments himself or herself, but no SM is good at fixing all types of impediments. Also, the team can and should invest some in fixing impediments, and people outside of the team (managers, consultants, whatever) can be excellent at fixing certain types of impediments.

The SM has to draw on all these different people and get them to fix the impediments in priority order. He or she has to do this with no authority, no power. The SM must convince with logic and reasonable persuasion.

[Move above to Impediment List]

- Help the Product Owner

The SM must get help to the PO until he or she becomes an excellent PO.

We have already discussed how the SM helps the Developers, the Scrum Team generally, and the Organization.

And the SM also helps the PO. This can be hard. Most beginner SMs don't understand the PO role well, have never played it, and don't know where to look for help.

But the SM must start to find help. Get the PO trained. Get a coach. Show the PO some really good POs (if you can find them), and suggest that your PO copy what the good POs are doing.

***

On a good professional Scrum Team of 7 people, we recommend the SM should be full-time.

There is no Scrum rule in the Scrum Guide that says that, but then, the Scrum Guide is silent on many things.

If the SM is one of seven people, then we are devoting about 1/7th of the team's power toward continuous improvement. This is a bit over 14 percent of the team's time. This seems reasonable and fair. Especially if, by investing in this way, we get 100 percent improvement in team productivity.

If we open our eyes and see, there are a good number of notable improvements to be made — everything can be made better in some way. In Lean, the idea is expressed as the relentless pursuit of perfection. The team members can become more skilled in all the hard and soft skills. The team's level of collaboration can always be improved. The impediments from outside the team always need more fixing.

So, there is plenty of work to do, if the SM knows he/she should be doing it.

As we hinted, there is a problem of identifying impediments at first. Often the culture is "we've been doing things this way for ages, everything is working fine, nothing to improve."

In Lean, they have the saying: “No problem is the problem.”

That is, failure to recognize opportunities for improvement is often initially the biggest impediment.

The Team, the SM, and the Manager(s) must work together and commit to doing something about almost all of the top impediments. Over time. One at a time.

I hope the SM job has become a bit richer for you now.

The Developer Role

The next role was commonly called the team role. AKA The Doers, or the Implementers.

We also had the notion of the Dev Team. The group of Developers. (Note that Developer in the Scrum Guide means or includes, for software at least, the coders and the testers.)

Scrum has evolved. We used to speak of the Dev Team within the Scrum Team. Now, I think we realize that the whole (Scrum) Team wins together or loses together.

Let's mention that most Scrum Teams can not be nearly as successful without the help of Chickens, that is, people outside the Team who help the Team in some way. These so-called Chickens (involved, not committed) are part-timers. Hence, "only involved".

Ok. But Chickens almost always are essential also.

Who Are the Implementers? What Do They Do?
Well, if the total team is seven people, that leaves five to be the Developers. The five Implementers should have all the skill sets to build and verify and validate the product. (In software terms, at least coders and testers.)

This is important and often misunderstood. A Scrum Team doing software must include testers. For a story to be completed and working by the end of the Sprint, it must be professionally tested (validated and verified) (and the bugs fixed) by professional testers during the Sprint.

Again, the Developers should have *all* the building, verifying and validating skill sets for whatever product the team is working on. And related knowledge.

Does This Ever Happen?
In my opinion — never. The team always has most of the skill sets, but not all. So, the Developers must rely on people outside the team to provide, one way or another, some of the skill sets. Or the knowledge.

How Much Are the Developers Lacking?
We should hope they have 99 percent or 98 percent or 95 percent of the skill sets. This happens. Or maybe 90 percent?

We will not bother to define 90 percent coverage, but to me, at about that level, the team is seriously compromised in terms of doing quality work in a tight timeline. Almost always, that is what is wanted. So, the Developers must be honest about where they are lacking skill sets. Again, getting humans to be this honest can be challenging.

The team must be team players, as we say. The team must help each other, and the strongest person must teach the others the skill set that he or she is strongest in. (Use common sense, which is very uncommon.) But the key thing is that it is about the team success and not about what an individual does. (We are not against recognition for individuals, but we want the team to be stronger. This is a team sport.)

Taiichi Ohno (famous for Lean) is famous for using the rowing metaphor. They must all row together in the same direction to be successful. I am not sure the rowing metaphor, as with any metaphor, is completely apt in every circumstance, but we think in America at least it needs to be repeated and discussed more. (See "Toyota Production System" by Taiichi Ohno.)

If a team has a dominant individual who will not let the team work together, then ultimately the SM must get that person removed from the team. At least *very* commonly that turns out to be the correct solution. Often that person is clearly the most talented person on the team on paper, and most people think so, and yet, paradoxically to some, once that person is removed, the team's Velocity goes up.

Should the Team Include a BA, an Architect, a DBA, an X or a Y?
It depends. It seems to work sometimes. It seems not to be necessary every time.

Overall, it really helps to have a great team. In this statement, we mean not only those in the Developer roles but also the PO and SM. And important how well they all work together.

Reminder: Scrum is a team sport.
It is remarkable how many companies or people do not understand or follow-through on this "team sport" idea. This is often a big change to the culture. It is always a notable change for many people.

The Whole Team

The team needs to be considered on its own.

The team is the whole Scrum Team. Normally we recommend about seven people — including the PO and the SM — and normally the team would be a full-time, real and stable team.

Why?
First, we assume you are working on one of the top "things" for the company or at least your department. Why work on anything less important? We assume that that top priority needs to be done quickly, both for the company's sake and for the customers.

Then, the ratios of PO to Developers and SM to Developers are better if the whole team is seven people. Also, we have enough people to cover more of the required skill sets. Lastly, seven is about as big as you can get and still have good communication throughout the team, so that everyone knows what each team member is doing.

Next, a team has a separate identity, or, has its own identity. Hence, we must consider it as a separate thing.

Now let's consider a few topics about the team.

Historically, "The Chicken and the Pig" story was part of the Scrum Guide. The main point of the story was to distinguish between the committed and those *only* involved. This is an important and useful distinction.

The Scrum Team includes the committed people, and they are responsible, as adults, for full and complete success in all the dimensions that real success requires. They should have all the right stuff to be successful.

Who Forms the Team?
Scrum does not address this question. Just as it does not answer many important questions.

But let me suggest the following.

Start with a good Scrum advocate, well-trained, knowledgeable. Maybe you!? Maybe the best Scrum person in your company, or at least your area. You may have some power or authority or rank... or maybe not. You know agile-scrum better than most.

Next, you gather three managers. They know the project or product. They know the people. They are used to making decisions about who should be in a Team.

We recommend at least three managers because deciding who will be in the Team is a hard and important task and three heads are better than one.

We also recommend that you advise those managers — who typically are not Scrum experts — on how a Scrum Team works, how the team should be set up for success, how the team will be responsible for full and complete success, what a PO does, what a SM does, what the Developers do, etc. You must help them see the importance of team chemistry if they do not see it yet.

Hopefully they then select a good team.

Once they have selected a Team, have the three managers reflect on whether the whole team will work together and truly be successful. Sometimes they are used to looking at each person "on paper", and not at how the Team will come together.

One good thing about Scrum: You find out quickly how good the team is. Usually within three Sprints you have a fairly good idea.

Once a team is chosen, the Team should evaluate: can we win? Can we do it together? Do we have everything we need to be successful? What are possible failure modes? What do we need most?

They must be adult enough to insist on getting the things they will need for success. Things needed might include: the assistance of other people, books, training, knowledge, tools, etc.

The Involved or "Chickens"

Now we must talk about the "involved," or what might be called the extended team.

We find that every Scrum Team always needs the assistance of others — always — and that that assistance is critical to success. The assistance could come from individuals, departments, other Scrum Teams, vendors outside the company, etc.

(Yes, some Teams can be successful with almost no assistance, but I find that is rare, assuming we also need to meet an aggressive timeline.)

The Team members (and managers) must recognize the *importance* of the "involved."

First, the Scrum Team must help identify the chickens and prioritize them. Then, the Scrum Team must do their best to manage each chicken. If further management is needed, the organization or organizations involved must also step in to make success more likely.

The involved are generally not very reliable vis-a-vis the mission of our Scrum Team. This trait because they are just not committed. To the involved people, the work of our

Scrum Team is not their main thing. And normally, the involved have other work that is their own top priority (in some sense or another).

Typically, some of the involved will prove more reliable than others. The team must monitor them and do what they can to help assure that the involved deliver on a timely basis and with high quality.

What might the involved do for our Scrum Team? That too can vary quite a lot. Perhaps they provide knowledge or advice or coaching. Perhaps they write some code. Perhaps they do some work by pairing with us. Perhaps they build (and test) some (small?) component of the bigger product that this Scrum Team will deliver.

Business Stakeholders

Among "the involved" we want to call out the Business Stakeholders (BSHs).

Among the key tasks of the BSHs is to come to the Sprint Review and give good feedback. The bad news does not get better with age.

Why do we only say good feedback? Because no one really knows the customers that well. They customer situation is always changing. Customers change their minds. The customer base is complex. What the competition will do is hard to predict (and how that affects our customers). So, it is hard do go higher than "good". We must be realistic.

And. The BSHs do offer unbiased feedback. That is, they don't own it, they did not create it. So, at least they are unbiased. So, we avoid the problem of confirmation bias, at least to some degree.

We recommend that you have about 4 BSHs. That number gives you many hands on the elephant, and the PO and the BSHs are more likely to get a quick and complete view of how customers and the business will like these stories and the product. This understanding is hard and important, and we do not normally recommend fewer than five (and not more than seven in total either). That is, the combination of BSHs plus PO.

When I say BSH I probably mean someone notably different than what your company culture may mean.

I mean people who can give high level feedback, about the Business Value of each story, as well as low level detailed feedback about every detail of each feature. Or, who can bring people who can. A BSH must come to every Sprint Review (as much as anyone does anything every time). This combination is rare and hard to find but essential to high success.

The Team

It bears repeating that Scrum is a team sport. It is all about the team. The Team has some chemistry, and the Team has some chemistry with the involved (chickens). Also

important: the team must rise to the occasion and take on the responsibility of delivering a wonderful product. One might call this the character of the team.

Who Leads the Team?
Scrum defines this a bit, but perhaps mostly relies on emergent leadership. (This is a phrase that Jeff Sutherland uses.) Note that we call it leadership, not “boss-ship.”

The PO is the leader in the sense that he or she can decide how to order the Product Backlog. The PO gets input from everyone, but ultimately must make the lonely uncertain decisions.

The SM is a leader in terms of servant leadership. And servant leadership might be simplified into helping the team get one impediment fixed at a time. And the SM is the lead educator about agile-scrum. The authority. But the SM cannot make them do Scrum.

In general, we recommend a team of strong players who are all adult.

Thus, it is possible that any one person could (briefly) be a leader of the team in one area or another, from one hour to the next, or that, in the heat of battle, any person could rise up and lead the team in a moment of crisis. It is this type of team, with aggressive play and risk taking along with emergent leadership, that tends to be more successful in our kind of knowledge work in developing new products.

Those are suggestions. Discover your own style of play, and see if that brings you success. Experiment.

[stopped here]

The Core Sprint Events or Meetings

The Scrum Guide defines the main events or meetings as:

- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

We start with an explanation of the first, larger time-box: the Sprint.

The Sprint

First, the Sprint is an important time-box (one of many time-boxes) where the team must build working product and then get feedback on what they have built at the end of Sprint. This concentrates their minds wonderfully.

For most teams, we strongly recommend a two-week Sprint.

Some advantages of a 2-week Sprint:

- The managers are more likely to come to the demo every time
- The Team has built enough to show and we need the feedback. So that the bad news does not get better with age.
- It gives us enough time to recover from smallish problems. And still have a successful Sprint.
- We plan the Sprint better than a 4-week Sprint.

On rare occasions we might recommend a one-week Sprint or a four-week Sprint. Even more rarely, we might recommend a different length.

Scrum has four defined events and all these events happen within a Sprint:

- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

In simple terms, you start a Sprint with the Sprint Planning Meeting and end a Sprint with the Sprint Retrospective.

Sprint Planning

Before the Sprint Planning meeting, there must exist a Product Backlog with small Product Backlog Items (PBIs) at the top of the list. The PBIs must be estimated.

Specifically, we recommend User Stories and estimating using story points. So, for at least the stories to go into the next Sprint, they have been estimated with story points. (Story points will be explained more later.)

Also, the PO with the help of others has pulled together all (most of?) the details needed to build the stories quickly. Written as much as needed. And key technical issues have been resolved. The stories are thus "ready" (some say "groomed").

We also recommend that the BSHs and the whole Scrum Team attend.

For a two-week Sprint, the maximum length is 4 hours.

I divide the meeting into two parts: Stories and Tasks.

1. Stories

The first part I call Stories. In this short part, everyone together reviews and agrees on the stories in the Sprint. Everyone can see all the information we have on each story, ask questions, as well as add or clarify any information.

More specifically, the Developers pull one story at a time from the Product Backlog into the Sprint.

The Developers can stop at any time. They are volunteering to do this work (and we hope are also motivated).

The Developers might stop at or refuse a story for typically one of two reasons:

(a) We do not have some details needed to complete the story, or
(b) There is some dependency that is unresolved, so that we think we cannot complete the story in the Sprint

There can be variations, but those are the classic ones.

Typically the PO agrees to a change in priorities, so that the next story is taken in. Until the Team has, say 20 story points. (Twenty being the expected Velocity for this Sprint. More on this later.)

While we hope this would have happened before, the Implementers can reject any story they think is inadequate (we do not have sufficient information to commit to it professionally).

As each story is discussed briefly.

The PO says to the BSHs, “Speak now or forever hold your peace.” That is, any BSH is asked to reveal or clarify or comment – so that the BSHs are satisfied that the Developers have all the information and the right understanding to build each story successfully.

So, if a story is not perfect in the Sprint Review (demo), it is partly the BSHs fault. They should have explained more. If they are unhappy in the Sprint Review, it is likely because they did not speak up in the Sprint Planning.

Again, the Developers get to volunteer, one story at a time.

For the 6th Sprint, we have the average Velocity for Sprints 3-4-5. Let’s say 18, 22, 20 for an average of 20 story points.

If some people will be absent some (eg, Doctors qappointment), then maybe less. Or if the SM has fixed an impediment, then maybe a bit more. Otherwise we expect to commit to the average Velocity (20, in our example).

It is important that all the top stories are small and about the same size. We recommend that a typical Sprint (for a team of seven) has about eight stories. With a Velocity of 20, that means a typical story would be two or three story points in size.

This first part has usually been planned before and this is a final review. So, typically, this part is done in 45 minutes or less. The BSHs can leave after this first part is complete. For those reasons, we expect part one to be done in 30-45 minutes (or maybe an hour at most).

2. Tasks

The second part is what we call Tasks.

We mean that the Developers (and others) will define the tasks or sub-tasks by which they will complete all the stories (get each story to done-done, as we say).

How do stories differ from tasks?

The story has Business Value (at least in the eye of the PO) and a task by itself does not deliver Business Value. To put it another way, we can demo a story and get feedback, but it is not useful, normally, to demo a task.

Each person creates his or her own tasks, although people can work together.

How big should the tasks be?

To us, it is a trade-off between the work of building out the tasks versus the benefit that greater visibility would give the Team. What is the right balance?

We recommend that each task be typically 2 hours.

So, a Developer in the Daily Scrum would expect to say "I expect to complete Task 1 (of 2 hrs) and Task 2 (of 2 hrs) and start Task 3 (of 2 hrs)." And the next day, after a pretty good day, say: "I completed Tasks 1 and 2, and started on Task 3." And it was good.

More broadly, having the detail enables the Team to see where they are making progress, who needs help, and where we are behind. Having that knowledge, we expect the Team to inspect and adapt better.

Another factor. The 2-hour tasks allow each person and the other members of the team to feel some completion — some traction — each day by each person.

Having 2-hour tasks is a cultural change quite often. But the visibility is worth it.

Do not forget: People should not be blamed for getting stuck. Think of this more as an opportunity for team mates to help team mates who get a bit stuck. That happens to everyone, to some degree. We help each other.

The small tasks also force people to admit their impediments in the Daily Stand-up. Sometimes the only real impediment is that I am bad at estimating, but even that is a learning experience, and we all become better at estimating the tasks.

"If you want to change the world, start by making your bed."
"One small task done leads to another small task done."

New teams are usually bad at first at creating the small tasks. But as Goethe said, "Everything's impossible until it becomes easy."

So, the new Team defines the tasks at first (badly or maybe not so badly). Then does the work. Then defines tasks for the next Sprint. And does that work. With each iteration, we learn how to write better tasks. That is, when we don't know how to do it, it is impossible for us. As we learn and practice and practice, it then becomes easy.

So, each task is described, assigned (or volunteered for) and estimated.

After the Team has created all the tasks, we enable everyone in the team to see the whole Sprint Backlog (the stories and their tasks).

Anyone can now ask to change it, describe an item better, re-assign a task (in fact that can be done at any time during the Sprint), re-estimate a task, and even add or eliminate tasks.

With the Sprint Backlog improved, the team can now commit. To the combination of the stories and the tasks. Before committing, they can consider any factor that might help or impede them. So, it is the whole team committing.

The PO must get questions answered quickly. The SM must work on impediments. Key things that affect success.

I prefer to do this with fist-to-5 voting by each Developer. Showing a fist equals zero — no confidence. Showing five fingers equals maximum confidence that the team can get all 8 stories fully completed in this two-week Sprint. We want each of the Developers to have at least three fingers showing.

If we do not get that level of confidence on the team, then either the Sprint Backlog needs to be discussed or improved, or the number of stories reduced. Or possibly increased if they feel that is reasonable.

Commit does not mean guarantee. With normal (good) stress, about 40 hours per week, with faster response to questions and a SM dedicated to fixing impediments, the team should become 70-80 percent reliable in meeting their commitments. That means for about seven or eight Sprints out of 10, the team should get all the stories (or perhaps all the story points) that were committed done-done.

If they fulfilled their commitment in 100 percent of the Sprints, that would be too high. They surely had notably under-promise to get **all** the stories done every Sprint.

Around 50 percent is probably ideal, in terms of getting the Team to the edge of what they can achieve.

A lot of good disciplines occur when the team learns to promise what they usually can complete. As one example, it forces them to minimize WIP, at least to some degree. It forces them to learn to estimate decently. Not to over-commit. It forces them to insist on at least on pretty good details before the story comes into the Sprint. They become more serious about fixing impediments.

The learn to commit and then become fairly reliable in delivering that. The Team starts to feel like “we got this”. The trust between Technology and the Business side improves.

[stopped here]

The Daily Scrum

Scrum has a daily team meeting that we call the Daily Scrum or the Daily Stand-up.

The maximum time-box is 15 minutes. If the team is seven people, the minimum time-box is 7 minutes.

The whole team attends. Others may attend, but they must be silent during the meeting. (More on this below.)

Note: The Scrum Guide implies to some that the PO should not attend. Jeff Sutherland has clarified this. You are still doing Scrum if the PO does not attend, but he recommends that the PO attends.

The team members answer three questions:

- What did I do or get done yesterday?
- What will I do or get done today?
- What is my biggest impediment?

Each person speaks for himself or herself.

The Scrum Guide mentions that the actions should be about the Sprint goal. So, for example, you do not get to talk in this meeting about your shopping yesterday.

Why the biggest impediment? We find one of the biggest problems in no problem — usually expressed as “no impediments.” People want to pretend that they themselves are perfect and that the world around them is perfect. There are some who wish to wallow in problems and worries all the time, too. But in the Daily Scrum, we want to hear the biggest impediments quickly. It is likely that one of the seven will be important enough for the SM to work on immediately today.

This brings up the problem of honesty. Humans are not always as honest as we would like. In one specific area... each person tends not be completely forthcoming about his or her own weaknesses.

Now, we don’t need to hear about everything, particularly non-work stuff, but work-related “areas for improvement” — these are important with Scrum.

Now we get to a key issue. What is the purpose of the Daily Scrum?

One answer is to enable the team to sync up so they complete all the stories during the Sprint. This is true as far as it goes.

I think a better way of putting it is this: The Daily Scrum is a meeting for all the members of the team to get the information they need to help the team self-organize, self-manage and self-direct themselves to a higher level of success**.**

The Sprint Review

This meeting happens almost always at the end of the Sprint.

We gather the team (the whole team, including, of course, the SM and the PO) and meet with the BSHs.

We want to learn the truth. What progress have we made? What mistakes have we made? What do we need to do better? How do we improve it so that it is an outstanding product?

One key phrase: The bad news doesn't get better with age.

The overall time-box for a two-week Sprint is 2 hours.

I find that most teams at first do not have that much stuff that's done; the feedback is also not that extensive. This means that often the meeting is over in about an hour.

But, you can imagine, after the SM doubles the Velocity and after the PO starts providing better "requirements" that the team will produce a lot more in two weeks, and so the meeting may need to go to 2 hours.

I think of the Spring Review in two parts.

1. A Short Review
In this short review, I recommend that the PO discusses a few basic things quickly, maybe summarized on one slide. (*Certainly not* a 50-slide presentation!)

These basics might include:

- Here are the stories we committed to in the Sprint Planning Meeting.
- Here are the stories that are done-done.
- This was our Velocity this past Sprint.
- This was our average Velocity over the last three Sprints, and with this average Velocity, we expect the current release to be delivered in X more Sprints (say, two).
- Here are what our biggest impediments were, and here are our biggest impediments today.

Then, there might be some discussion. Hopefully the discussion is about how the business side would like to support getting one or two of the aforementioned impediments fixed.

So, make this relatively quick — 10 minutes might be typical.

2. The Demo
People often call the entire Sprint Review the Demo, which is not so bad, but I think is slightly misleading.

What do they demo? Well, maybe it is obvious by now, but they demo (mainly) the new "working product" that has been built during the Sprint.

Perhaps we should add now that the demos (of each story, one story at a time, typically) are given usually in the context of the whole product. That is, all the features from any prior release as well as all the features built in any prior Sprints to build the current release.

Ok, so we want everyone to give us honest feedback, the best feedback that they can give us and the most complete feedback possible. The bad news does not get better with age.

Anyone can give feedback, but we want it mainly about two things.

1. How much Business Value does the story have now that they see it?
2. Are there any details that are imperfect?

The Demo needs to be prepared (before this meeting) quickly. The data needs to be prepared. Someone needs to think through how we will demo the new features. Which examples will we use? Are they appropriate? Etc. This is important and sometimes hard, and it must be done in a time-box.

The demo needs to be "narrated," particularly, at first, by someone who understands the BSHs. It cannot be so technical that the BSHs never want to come back. The speaker must engage the BSHs and then handle their comments and feedback. If this is not done well, then the BSHs will not come back, our feedback will not be as good and, therefore, the product will achieve much less Business Value.

Let's say again: Getting good BSHs that come every time is hard. Good luck with that problem.

Now, back to the Demo itself.

WE need to hear every imperfect detail now. So, if an (imperfect) BSH does not understand every detail of each story being shown, that BSH should bring a SME or BA or someone who does understand all those details — and bring that person today!

What we want is perfect feedback about what the customers are going to want over the whole lifecycle of the product so that we achieve maximum Business Value over that whole lifecycle. What we get is not as good than that. We get feedback from humans who are not the real customers, or at least that's what I usually see happening.

Anyone can give feedback. That is, we mostly expect the best feedback will come from the BSHs. It is they who should understand the different customer groups the best (along with the PO).

But, in fact, often the Implementers have excellent feedback. George may have excellent feedback on the story that he did not work on.

Hopefully, a majority of the time we get positive feedback. Yay! We did stuff well and the customers are really going to like it! Yes!

It is wonderful for the team to get that kind of feedback every two weeks — wonderful. (Surprisingly, in the past the Implementers would often go months without getting positive feedback. Well, I am slightly sarcastic. Often they never got any positive feedback from anyone who mattered. That is, anyone who represented the customer well, sad to say.)

Some of the feedback is negative. Sometimes the Business Value appears to be less than we originally thought. Sometimes some of the details are not right, now that we can look at them.

To be fair, it is hard to read the mind of the customer and what he or she or they will want in the future. We do the best we can. We live and learn.

Also, not everyone agrees on the feedback. One BSH will disagree with another BSH, the PO won't agree with an Implementer, etc., etc.

In any case, though, the PO must decide quickly one of the following:

1. "It's done, we think the customer will like it how it is." — Hopefully most of them fall into this category.
2. "It needs some changes and here is the complete list of exactly what those changes are." — A couple of stories can be in this category. We hope not many.
3. "It needs improvement, but we are uncertain about the details." — Hmmm. This is not a good category if we want to get the release done on time.
4. "Wow! Things are mucked up. No one is ever going to want this." — A story in this category is kind of depressing, but at least the bad news is not getting better with age. Then we have to decide: Is there anything within this user story (the three parts) that is a real need? Sometimes we realize that it was an idea, but maybe not a real need for the current release.

The PO must state their decisions clearly and also with a minimum of injury to the egos of the people whose opinions the PO is ignoring. Good luck with that.

I guess at this point we should mention: The PO is not always senior to everyone in the room. Sometimes some of the BSHs are more senior. Nonetheless, the PO must decide and get it to stick with the people in the room.

Again, sometimes the BSHs do not see things from the same point of view. Perhaps their departments or interests are rather opposed. Nonetheless, the PO must get them to express their differences and then must resolve them quickly — or at least decide quickly what to do with the current story.

I'm sure you can imagine how this meeting can be "interesting."

Sometimes it is useful to have a "wrap-up" section of the meeting and review the decisions that were made. They review the eight or so stories and say which ones ended up in which category. Then, summarize the changes to be made (to a few stories) in the next Sprint.

Well, technically, a "to be improved" story does not have to be improved in the next Sprint, but almost always that is that right thing to do for everyone.

We also strongly recommend asking these two questions of the group. These questions can be asked many ways, but here is our suggested wording.

At the beginning of the meeting, after all the stories have been quickly mentioned, ask: "Did we work on the most important stories? Now with 20-20 hindsight, should we have worked on another story instead?"

Often this question will reveal new learning. For example, a new story for this release might be identified.

And then, after all the stories have been shown, ask: "Seeing all these stories, do they make you think of any stories we should add to the Product Backlog?"

Again, sometimes some very important stories will be identified at this point.

Remember what Buddha said: *"Everything changes, nothing remains the same."*

More specifically, we are always discovering ourselves and each other. As things change, we then start to want different things in the product, or we (the builders) start to understand the life of the customer in a much different way. Whatever the reason, new important stories can still be identified. It's better to identify them before the release than have the customers tell us of our glaring omission.

Just because we discover a new story does not mean that the PO must put it in this release. That's a different decision.

The Retrospective

We usually think of the Retrospective as the last meeting of the Sprint.

What is the difference between the Sprint Review and the Retrospective? The Sprint Review is about the product. The Retrospective is about the process. I prefer to put it this way: The Retrospective is about how we become better as a team — "continuous improvement."

Who comes to the Retrospective meeting? The whole team: the Implementers, the SM and the PO. They can invite others, which does happen sometimes.

Also, if the Implementers are afraid of the PO and are not able to tell the truth if the PO is there, then that is an important impediment, especially for this kind of meeting. So, the SM may have to ask the PO to skip one or two meetings until the SM can get the Implementers to tell each other the truth. However, the Implementers must all become used to telling the truth with the PO present very soon. If they can't tell the truth with the PO there, perhaps the wrong person is the PO.

What is the time-box? According to the Scrum Guide, the Retrospective is 3 hours for a four-week Sprint. This implies 1.5 hours for a two-week Sprint. I am OK if you use 2 hours every two weeks.

This meeting is often not done well, or not done at all. (If not done at all, typically because it was not done well. It perhaps almost useless, the way it was done.)

Do this meeting well. We are about to give advice that will make it a more useful meeting.

What Is the Purpose?

The purpose of the Retrospective is to become better — measurably better — in some dimension in some way.

Typically, we want to improve Velocity (productivity). This is a good thing, but it presents problems. If you tell a team to improve productivity, they usually hear that as "work more hours."

So, somewhere, the manager and the team have to agree on this:

1. We are not working more hours. We are working 40 hours per week (or some reasonable number).
2. We will be having more fun. Fun is essential to innovation work, and we will measure this with the Happiness Metric (See Jeff Sutherland's blog).
3. We will improve Velocity a lot, 100 percent in the first six months.

Always you have to discuss these three things together. They won't believe it at first.

The purpose could be something other than increased Velocity, although whatever it is, it usually influences Velocity eventually. Maybe higher quality, maybe faster learning, maybe better motivation, maybe higher Business Value, maybe more fun, etc.

I like to divide the meeting into two time-boxes.

The first time-box is small. It consists of the following quick discussions we suggest. (There are alternate ways of doing it, and once they master the basics, we do recommend changing things some.)

1. What Went Well?
Take a moment to celebrate the things that went well, and ask everyone to do more of the good things — and to do those good things more often.

2. What are the Impediments?
I like to be blunt and honest. Always there are things in this sprint that sucked. Ok, things that could have gone better. Or, opportunities for improvement. We were dumb, managers tried to kill us (it seemed), the servers decided to crash, it rained, etc. "It's always something."[1]

It is important that we identify our own imperfections as a team, that we identify the supposed evil in others and that we complain about the world. All of these are true, and it is helpful, up to a point, to talk about them.

We are identifying the impediments as we did in the Daily Scrum and, remarkably, we identify some of the same ones (no surprise) along with some new ones.

Anything that slows down the team, in any way, is an impediment. A lack of a ping-pong table could be an impediment. Technical Debt could be an impediment. An interfering manager, a missing team member, corporate culture, a lack of babysitters — anything could be an impediment.

Let me emphasize this: It is human nature to be reluctant to speak publicly about one's own imperfections. Nonetheless, that is what we must do. These are always some of the more important impediments. Because everyone is imperfect, and we can see this daily (e.g., in the Daily Scrum), it becomes less hard to be honest.

Still, if you are the SM, good luck getting them to be completely honest, especially at first. What they are used to is this: "The beatings will continue until the morale improves." Usually the more specific version is the blame game. You have to change this.

[1] Some may enjoy the Gilda Radner reference.

We already by now (before this meeting) have created an Impediment List with the top 20 impediments. And, there are 20 things already on the list.

I suggest that each person identify 3 possible improvements, by thinking in gthis way:

1. I suck because…
2. We suck because…
3. They suck because…

It is very common in human nature to blame "them". And, indeed, sometimes it is it or they that started the problem. But we must also look at ourself and ourselves. (We have somewhat more control, also, over ourselves.)

Virtually always, most of the "things that sucked this Sprint" will not break into the top 20 list, because they are not that important, or are already on the list. But some will. One new thing might even go straight to the top of the list.

So, one value is: They got some things off their chest, they got to moan a bit (as all humans must do), they see that most of their impediments are not that important and they can live with most of them and move on. This is useful. More about the Impediment List later.

The key thing for now is that it is prioritized, based on the benefit/cost of the impediment. Or, maybe it's better to say the ratio of the benefit achieved by mitigating or fixing the impediment over the cost of fixing it.

So, again, a few of the impediments identified in this meeting will break into the top 20 list. This is useful also, because we are going to act on the top impediment.

Next, I recommend the SM Report. The SM has 10 minutes to "justify his love" for the team. That is, 10 minutes to explain what he or she has done in the past two weeks to help the team.

- Which impediments has he or she worked on?
- Which impediments has he or she taken to a manager?
- Which impediment has been fixed?
- How much has the Velocity improved this Sprint?

And the team (and the SM) expect the Velocity to usually improve every Sprint by a small amount. (Rome wasn't built in a day.) The rest of the team starts to see how the SM is useful; and they are surprised. The SM is also surprised that they did not think (before) that he or she was valuable — but now they do.

Next, the SM leads the team to quickly prioritize the top four impediments (from the top 20 list). The SM can add information (as can the PO) but allow the rest of the team (not the SM) to decide the priorities. Even if they are wrong, they are right.

Now, the whole team spends the rest of the time working on the top impediment.

Here are three things the team can do together. (These are not the only things a Team can do in the Retrospective, but common examples.)

1. Devise a Solution Together.
This is a phrase Ken Schwaber likes. They take the responsibility to define how to fix the top impediment. Again, some impediments cannot be fixed, but can only be mitigated or the impact can only be mitigated (reduced).

This might take some time. We might include root cause analysis (RCA). The Team may need to consider alternate solutions. The Team might need to weigh alternate solutions and carefully decide. (I am not recommending endless dliberations, but if a solution is expensive, it deserves some thought before spending.)

2. Plan the Execution.
We figure out how to implement the solution. Which steps are needed. Who should be perform the steps.

This might lead to the observation that the solution is costly, and maybe this isn't the impediment with the best ROI.

3. Prepare a Business Case.
We work together to prepare a business case to take to a manager and ask for a "yes" — a yes to some money or to getting some people to work on it or a yes to allowing the change to happen.

We recommend that you use the A3 approach to Kaizen, but specifically that the business case be prepared much like an A3 report would be done. You will need to include the following sections:

- Problem
- Solution
- Benefits (from the Solution
- Costs (of implementing the solution)
- Action Items (e.g., steps to be taken immediately after approval)
- Measures (how we will measure after the fact that the solution actually improved things… or maybe not)

It would be fairly typical that the business case would not be perfect at the end of the Retrospective time-box. We expect the SM will move it forward and improve it. The SM might work with others on that. And then the SM (or the SM and the rest of the team) will present the business case to the right manager.

If it is approved, it is up to the SM to keep driving it so the impediment is fixed quickly. We want most fixes to be done within two weeks; and by the end of two weeks, we want to already be starting to accrue benefits (e.g., higher Velocity).

These comments raise a couple of points.

First, the managers should be expecting these business cases. They should expect the team not to be good at presenting these business cases, and the manager should start to teach the team how to prepare a better business case.

Next, the managers should be expecting to say “yes” and expecting change. My saying: *“If you don’t change things, nothing’s gonna change.”* So now, the team is helping the managers become effective.

The Artifacts

Now we turn to the artifacts in Scrum.

First, we have to be clear: There can be many artifacts for a team, depending on the work and how you define artifact. So, the main artifacts we will discuss here are what we consider the core Scrum artifacts.

Here's my list:

- The Product Backlog
- The Sprint Backlog
- The Scrum Board
- The Sprint Burndown Chart
- The Release Burndown Chart
- Working Product
- The DOD
- The Impediment List

My list is not what you will find in the Scrum Guide. So, when they are not mentioned there, I will explain that a bit later.

Do we have to use all these artifacts every time? Well, of course not. Use common sense. If you are quite confident that an artifact won't help you in your specific situation, by all means do not fool with it. Just be careful.

"Common sense is not very common." That's a Ken Schwaber saying, I believe. What I think it means is that we are so easily captives of the old ideas, that we do not see the truth of the current situation well enough. Often, we do not make the right decision. Be careful!

Should you have other artifacts? Do I recommend others? I often get these questions. The answer is probably "yes" in both cases. Again, we are only discussing the most basic Scrum artifacts.

Let's mention one more artifact (or someone could call it an artifact). That is, the Scrum tool.

There are many Scrum tools out there — none are identical. They do many things and they vary a lot, but, typically, they hold the Product Backlog and the Sprint Backlog. Often, they do more, such as generate the burndown charts or show a Scrum Board, etc.

Well-known Scrum tools include using Excel, Rally (recently renamed within CA), Version One, Jira, Pivotal Tracker and many more. Some of the Scrum tools (other than the ones named above) used to be lame a few years ago; now most of them are pretty decent.

You probably should have a Scrum tool, even if it is only an Excel sheet. Our goal in this paper does not include addressing the Scrum tool as one of the artifacts.

OK. Now let’s discuss each artifact I mentioned above.

The Product Backlog

The Product Backlog is usually the first mentioned of the artifacts. In some sense, Scrum starts with the Product Backlog, or at least the first Sprint cannot start without a Product Backlog.

The Product Backlog is a list of all the work for the team. We also think of the Product Backlog as the list of all the new features for the current product.

User Stories
The name of the items in a Product Backlog is what we call Product Backlog Items (PBIs). We also speak of those items as User Stories, especially if they are in the User Story format.

The User Story format is:

As an <end user role>
I can <do something>
So that <explain purpose or next step or because>.

Who Can Contribute?
It is important that the Product Backlog include all the work of the team. (More on this later.)

Anyone can contribute to the Product Backlog. This is important and often misunderstood. Any team member can propose PBIs, any BSH can propose PBIs and any customer can propose PBIs.

The PO can judge an item out of scope, improve the quality of an item proposed for the Product Backlog and even re-write PBIs.

Prioritized
The Product Backlog is prioritized. The final decision maker on the priority order is the PO. By this we mean that anyone can suggest things to be considered in the prioritization or suggest new data that would affect the prioritization.

In general, we first say that the prioritization is mainly based on Business Value or the value to the end customers. Later we say that prioritization should mainly be by the expected return on investment (ROI), which is the Business Value divided by investment with investment being mainly cost or effort. In truth, there are other factors that contribute to ROI as well (e.g., dependencies are a key factor).

Product Backlog Length
The Product Backlog should go out a reasonable amount. Jeff Sutherland has said a typical length is one year for a typical product. Surely this could be less for some products and more for others. In my experience, many Product Backlogs do not go out far enough.

This is a serious problem because it means that the PO is choosing from too few items "what is the most important PBI to work on next." That means that the team is typically not working on the most important thing it could be working on.

The 80-20 Rule
In general, for a given Product, the Product Backlog should help the PO do the 80-20 rule, or something close to it. That is, the team does 20 percent of the work and delivers 80 percent of the Business Value. This is hard to do (for many reasons) but focusing on this issue is very useful. For example, if the team did 20 percent of the work and got 50 percent of the Business Value, that would be a serious and very useful improvement.

Kinds of Work
The Product Backlog should include all the different kinds of work the team must do. The Product Backlog should include any legacy bugs, meaning bugs or defects that existed before the team started this working on the product. The Product Backlog also typically includes technical debt (sometimes expressed as technical stories). So, the PBIs include stories to fix the technical debt.

If we are automating QA tests, then the Product Backlog probably needs to include PBIs to automate the existing manual tests or needs PBIs for the work to set up or improve the automated testing.

Sometimes we have a separate list of "small enhancements." Often these enhancements are small changes to the existing set of features. When we describe the vision of the next release of the product, it does not include small enhancements to the existing features. Hence, these small enhancements are often work outside the vision of the next release — except that someone feels we must do some of them.
There can be other categories.

We do not have separate Product Backlogs. Each team only has one Product Backlog. Hence, "everything" (all the different types of PBIs) must be prioritized together in one Product Backlog. This is not always an easy job.

Product Backlog Refinement
The Product Backlog must be refined or groomed over time. New items are identified later and may be identified soon as part of the current release, later release or may never be built.

Product Backlog Refinement or Grooming includes many things. Among them are identifying new stories, breaking up larger stories (stories too big to go into a Sprint well), putting story points on stories, adding or revising the BV points on stories, adding details to stories (as much as the team needs), reorganizing the order of the stories, etc.

We have a whole book to describe Product Backlog Refinement. See our book on Agile Release Planning.

The Sprint Backlog

This artifact is different than many suppose. The Sprint Backlog is the list of stories and the list of tasks for those stories that the team thinks they can get done in a Sprint. (In my mind, the normal Sprint should be two weeks in most cases.)

The Sprint Backlog is created in the Sprint Planning Meeting. The stories in the Sprint (in the Sprint Backlog) come from the top of the Product Backlog. Again, the team gets to decide how many stories to pull into the Sprint.

The Scrum Guide describes it a bit differently. The plan that the Scrum Guide mentions does not have to be a lot of tasks for each story. Having small tasks for each story is a very good discipline that most teams need. But, if they get more successful and want to experiment with that a bit, then fine.

As we said before, the tasks must be small. We want to see (or maybe not see) that each person is making some progress each day. So, the small tasks (or very small stories) make that progress clearer. This clarity enables the team to self-organize better. For example, key problems can be identified and attacked.

The Sprint Backlog becomes the Scrum Board, which is a kind of visual management. More specifically, the Scrum Board is a kind of Kanban board. So, Kanban, or a form of Kanban, is baked into every basic Scrum implementation.

The Scrum Board is usually composed of several columns and rows. The columns are often titled backlog, in process, to-be-tested and done. There is one row for each story and it is expected that only one story is “in process” at a time. Well, that level of minimization of WIP (work-in-process) is a bit tight for beginners. Normally, a team has two stories in process at any time. But in any case, the situation is usually pretty clear from the Scrum Board, or at least after a brief conversation about the Scrum Board.

Some people complain that the Sprint Backlog is micro-managing. Indeed, often the work is more broken down (for the two weeks) than you often find in a waterfall project, but the team is not being micro-managed by someone else (or at least that is not the intent).

The team creates the Sprint Backlog. The team volunteers for the stories and the tasks. The team is *not* supposed to over-promise, but only sign up for the work that that data says they have a history of doing; unless there is a very good reason to believe the team now can do more. Two examples: Now Person X is back from the vacation that happened in the prior Sprint, or the SM has fixed impediment Y and now the Velocity should be higher.

Little things are big, and the bad news does not get better with age. And so, by seeing the small problems sooner, the team is able to take corrective action sooner and the impact is greater.

Is the Sprint Backlog perfect right after the Sprint Planning Meeting? **No!**

So, we expect the Sprint Backlog to be revised and improved as the team does the work and gets smarter. At least once each day, anyone on the team can revise the Sprint Backlog. In general, sooner or later a team member should explain why the Sprint Backlog was changed, if only briefly.

The Sprint Backlog is pretty darn useful.

The Sprint Burndown Chart

Now we come to the Burndown Charts. You could start with either one, but let's start with the Sprint Burndown Chart.

We might first note that the Scrum Guide does not talk about a (Sprint) Burndown Chart. There is a section in the Scrum Guide about "monitoring progress toward goals," which mentions burndowns and burn-ups.

I will recommend specifically the Sprint Burndown Chart.

What does it measure? It gives our best guess, as of today, of the work remaining in the Sprint.

The way I recommend for beginning teams goes like this:

1. In the Sprint Planning Meeting, we create the tasks needed to complete the stories. I recommend putting hours to those tasks. I recommend the tasks be small (typically 2-4 hours each) and that a person (or persons) be assigned to each task. (This was all discussed earlier.)
2. Each day, the Implementers will get work done and learn. All of that information leads to revisions of the tasks. For example, some tasks can be replaced by other tasks. Tasks can be added. Tasks can be re-estimated. Tasks can be re-assigned to a different person who then gets to re-estimate the task.
3. Just before the Daily Scrum, the Implementers make all the changes and put them "in the pot" (by which, I probably normally mean into the Scrum tool), and then someone (maybe the SM) can then calculate the net effect and therefore how much work is remaining now.
4. That enables setting the points shown above (as an example) and this the Sprint Burndown Chart.

Why?
The key thing is that this report is for the whole team. The team wins together or loses together. So, the team uses the Burndown Chart to give them the information they need to self-organize, self-manage and self-direct themselves to greater success (or less failure).

Some of the action by the team is not discussed. It happens sub-consciously. Sometimes the Sprint Burndown Chart leads to a conversation, perhaps like the following:

> **Person 1:** Yikes, we're screwed.
>
> **Person 2:** Damn, well we have to do something. (And in that tone from experience they know that means fix an impediment.)

Person 3: I think [X] is the biggest thing to fix now.

Person 4: I think the thing to do is [Y].

Person 5: I'll get started on that. Who can help me?

In this little conversation, you see the team figuring out what to do. The report is *not* primarily for others, but for the team itself. They are the adults.

And we look for emergent leadership, people who rise to the occasion and make it happen. So, we expect the team members, possibly any one of them, to take this information and do something with it.

Transparency

The Sprint Burndown relies on the team being as transparent, honest and accurate as they can be about all the work remaining — to everyone.

For example, if the team decides they will not complete a story, they might decide to stop working on it and focus on stories they still hope to complete. This is fine (or at least understandable that this will happen sometimes). First, they must be honest with all the people who care that that story is now "dead." Then, they can take those related tasks out of the "work remaining" so other things go roughly as expected — so that day we "burndown" more.

There is no point pretending that this day went well when it did not, and there's point in pretending to make more progress than we really did. It does not help. It does not force us to make the changes we need to make it we "play pretend." Equally, this can be challenging to managers who too readily want to intervene if things do not go perfectly in one day. One day is not a problem. Even a "failed" (weak) Sprint is not a problem, so long as we eventually are successful.

Innovation work cannot be predicted with great accuracy and surprising things happen.

We recommend being willing to fail. We do not recommend forgoing all planning and all management. In fact, we recommend spending more (good) time on those things so that over time we end up being more successful; in large part by putting all our heads together to solve the problem.

The Release Burndown Chart

Now we come to the Release Burndown Chart.

What is the idea? The first idea is that we want to hit the date, but let's agree that things can be a bit more complex.

A few side notes:

- There are many different kinds of situations in Scrum. We have continuous delivery (CD) now. Scrum still helps in that situation, but it is different.
- We also have people releasing every Sprint into production.
- The word "release into production" is fairly commonly used in software. It is probably less commonly used for other products, and we think Scrum is also very suitable for any new product development. So, if you use different words, please translate.
- So, if we have CD or are releasing every Sprint, the concept of a Release Burndown Chart is not meaningful as such. The concept of a Product Burndown might be.

So, assuming you are taking two Sprints or more to produce a product (six Sprints is a fairly common number to use as an example), then why do we want a Release Burndown Chart?

Why?
Two reasons come quickly to mind.

One is to ***measure progress***. If we start at 120 SPs of work and get down to 60 SPs, then we are in some sense 50 percent done. Why is that useful? Because I think it gives us the transparency to "take arms against a sea of troubles, and by opposing, end them." It makes us cut through the crap and get stuff done on time.

This keeps managers from canceling projects that have made significant progress. (That has been done to waterfall projects, unfairly.)

So, I am suggesting that the common (not universal) practice in Scrum is to (eventually) set a date. We have a stable team — therefore budget is fixed — and the flexible part is scope (or how many stories will we get done in the time-box). Another flexible part is how much the SM will raise the Velocity of the team.

Here an example picture:

[To be added]

One axis is story points, or the total of the story points for all the stories that are "remaining" — we are measuring the "work remaining." The other axis is time, divided

into six Sprints. So, we measure work remaining every Sprint and get transparency on that.

Side note: One of the purposes of the Release Burndown is greater transparency (this benefits many things). Obviously, humans are not always honest. So, for the Release Burndown Chart to be effective, the team must be honest.

“Work remaining” means that we can also potentially *redefine all* of the work. We can add stories, we can remove stories, we can break stories up (or break them down, if you prefer), we can re-story point stories, we can redefine stories or re-write them. So then, the Release Burndown becomes as accurate as humanly possible as of that moment in time.

But why have a Release Burndown Chart?

The second reason is because this information enables the team to self-organize (around a common goal of hitting that date), self-manage (as if they were adults) and self-direct themselves to greater success — or at least less failure.

Of course success and failure are not completely defined by hitting a date. but the date is the key element. The date is important because customers care so much about the date (or getting it earlier), and business (for a variety of reasons) cares about the date.

Side note: We discuss elsewhere at more length that happiness, fun and sustainable pace are also very important. While we are increasing Velocity, we must also insist that happiness and fun is going up and hours are normal (I’ll say 40 hours per week, but we could debate the exact number of hours). More on this elsewhere.

So, anyone in the team can use the Release Burndown Chart. It is mainly for the team.

Imagine a discussion after Sprint 3, heading toward a release in Sprint 6:

> **Person 1:** Wow! We are way over by 20 story points.
>
> **Person 2:** Yes, we’re not going to make that date if we continue like this. We have to hit that date. [In real life, hitting the date is not always that important, but let’s imagine in this case that it really is.]
>
> **Person 3:** We still have some big stories. We need to break them down and see if we can move some of those story points to the next release. You know what Pareto says.
>
> **Person 4:** ScrumMaster, what can we do to increase Velocity?
>
> **Person 5:** If we could fix the [X impediment], I think the Velocity would go up five points.

Person 6: We need to stop scope creep, too. If anything new comes in, we have to tell the business side that something has to go out.

Person 7: I'll take some time in the next two days to help with the [X] problem.

This conversation all started with an observation about the Release Burndown Chart. We are assuming that the team can act like adults and, to some degree, control their own destiny. It may not happen or may not be able to happen every time, but it can happen often enough.

Notice also that there was no discussion of working overtime or on weekends.

The Working Product and DOD

Now we come to the working product. The big phrase is “potentially shippable product increment.”

What it means is first this: At the end of a Sprint, we expect every story that the team committed to in the Sprint Planning Meeting is “working” by the end of the Sprint.

Working means built and tested — fully working (at the story level). Working does *not* mean that we have built a full Minimum Viable Product yet. For the moment we are assuming that scenario where it takes multiple Sprints to build a Minimum Viable Product.

We define working product mainly through the Definition of Done. We score points in the game (we earn story points for this Sprint’s Velocity) by getting all the items on the DOD done for a specific story. It’s all or none — either the story is fully done and we get all the points, or the story is partly done (or maybe un-started) and we get zero points.

The DOD must include two key things: The product is well built, and the product is well tested.

The stronger the DOD, the better. There is strong bias toward the “quality is free” idea. That is, the sooner you build in quality, whatever that costs you, it is much cheaper (in every way) than building in quality later.

Why?
Why do we want working product at all? I mean, people will say that it is slowing down and that it is lots of trouble — and it is trouble. People will say “it is more work for me” — and it certainly will at least appear to be more work for them (although if done correctly, not more work for the team).

One answer: *“The bad news doesn’t get better with age.”* That is, identifying and fixing the bad news *now* is much cheaper than fixing it later.

Another answer: We get better feedback from working product than from the documentation. Getting better feedback is very important. It is so easy to misunderstand what the customers (end-users) want. This bad news does not get better with age. If we spend less time building what they don’t want and more time building what they do want, we usually get done quicker. (OK, a bit of sarcasm, but getting done quicker is very important in several ways.)

Another answer: We have a better gauge how completed (what percentage complete) we are. In waterfall, the schedule would tell us we are 90 percent complete and then awkwardly ask ourselves how much longer to complete the last 10 percent. (It was always a bad joke.)

Now, we identify the story point for all the work in a release (e.g., 120 story points in total) and ask how many story points are completed (e.g., 60SP). Then we would know how much of the work we have done (e.g., 50 percent) and how much longer it will take to deliver (roughly another three Sprints). This is *much* more accurate.

And what we know (so much better, with more confidence) is much more useful. For example, managers are less likely to cancel a release that is truly 50 percent done with many great features already built. Features that they can see, taste and feel — fewer stupid decisions.

What's Included in the DOD?

Here's an example we typically use for software stories; conceptually the basics are the same for about any product.

I always mention "better requirements" because that is so important. Really, better requirements are part of the ready-ready criteria (some people are calling this the Definition of Ready or DOR), but it is important to mention now. This is key input to getting a story done.

It is amazing how much more work they can get done if they know what they are doing, and how much less work they get done if they have no flippin' clue what they are doing — by this, I mean if the requirements are clear. (The saying also applies to their skill sets, but normally this is much less of a problem.)

Here's an example of starting the DOD:

[good Req (better than ever before)]

- Analysis and Design: keep this short
- Coding
- Code Review: fix problems
- Testing
    - Unit Testing: automated, fix all bugs
    - Functional Testing: automated (80 percent), fix all bugs
    - INT/REG Testing: automated, fix all bugs
    - Any other testing…
- Documentation
- PO Review: fix any problems

The DOD is used to decide whether your team scored the two story points on Story 38 or not. It's part of the "rules of the game" for the referee. There are no partial scores — it's either done-done or not done.

A Few Comments…
We want clean, good code, well-written code; code that anyone could understand and modify.

We want to fix all the bugs. OK — we do allow people to come to the PO and make a case that "this bug does not have to be fixed ever." That's the only argument, that we will never have to fix this bug or defect; never, ever. (Remember: The bad news does not get better with age, and you have to slow down to go fast.) Very rarely the PO may agree not to fix the bug or defect now. The bug is then moved to an "issues" list. Usually at least half of these bugs (so moved) must be fixed later at *much* more cost. You shot yourself in the foot.

Functional Testing is called lots of things. It is the testing done by your good QA or testing people.

We recommend that some documents are updated every time we do a story. There is a longer discussion about which ones those are and exactly what that means. No documentation is unacceptable, and no documents being updated (however much or little) each time we do a normal story is something I just cannot imagine happening with any professional team.

The PO Review is important. We want a mini-demo each time a story is done, or almost done. The PO can say, "Now that I see it, it's not quite what the customer will want," and ask for some changes. The PO cannot add all or part of another story at this point.

The Doers will say, "But we built it according to the spec," (and usually they will be correct about that) and then ask, "Why wasn't this new information in the spec?" (the enabling spec).

There is not a happy answer to that question. The Doers should have asked the question earlier and the PO should have answered the question earlier (whether asked or not). Nonetheless, we have to accept that the PO (and others) will still learn things later than they should.

This practice *does not* allow the PO to be irresponsible and learn just anything later. But, we must accept that once he or she sees the built story, the PO may discover that it must be changed. As long as the change is within the scope of that one ticket (that one story), then the change must be made before that story is done.

Have we really made progress if the customer will not be happy? No. So, it ain't over 'till the customer is happy (or until the PO thinks the customer will be happy).

Is the PO always right? No, but we have to have some independent person decide quickly. The PO (or anyone he or she designates) is that person.

There should be some thoughtful discussion, such as, “Why didn’t we discover this problem earlier?” It is not a blame game, but rather a search for the root cause and an attempt to become better as a team.

We want about one mini-demo of this sort each day. (I am assuming eight or more stories per Sprint.) So, for example, all the testing is not done at the end of the Sprint. Most of the eight or more stories should be done before the last two days in the two-week Sprint.

The Impediment List

The most important thing a ScrumMaster does is remove impediments. This is easy to say, since anything that needs to be fixed or changed we define as an impediment. Maybe a bit more accurately: The fact that it has not been fixed yet is an impediment.

What Is an Impediment?
Just about anything can be an impediment, but how do we define it? One way: Anything that is slowing down the team.

If you define them that way, then there are hundreds or thousands of impediments at any one time — hence the need to have a list and prioritize it.

Here are some examples of the varieties of impediments:

- Technical debt
- A bad boss
- Someone who does not understand Lean-Agile-Scrum
- A hurricane
- A power outage
- A server that falls over
- Lack of automated testing
- Continuous integration that is not good enough yet
- A culture that does not fully support Lean-Agile-Scrum
- The matrix organization
- Distraction to people on teams or to whole teams
- Having the team work on more than one release at the same time
- A team room that is too small
- Insufficient skill sets on the team
- A PO that is not good enough as a PO yet
- Confusions about what the story is
- Unresolved technical issues
- Inability to make decisions quickly (by the PO or by people in the team or by people outside the team)
- Someone on the team who is not a team player
- Lack of self-organization on the team
- Lack of knowledge about method X
- Lack of DBA skills within the team
- Need to refactor the architecture
- Lack of baby sitters for some team members
- Someone getting a divorce
- The company recently had a re-org
- One person distracting the team too much

So, to name broad categories of impediments, we might have a list like this:

- Technical impediments
- Blockers to specific stories
- Organizational impediments
- Culture
- Insufficient education or training on Lean-Agile-Scrum
- Insufficient knowledge or skill sets
- People issues
- Things not working that were working before
- Basic things (e.g., lack of team room)
- Things outside the company (e.g., the weather)

But really about anything could be impediment if it slows down the team.

Issues About Impediments

Impediments are not only blockers.

We mentioned blockers, which is not always a well-defined term. Typically, blocker means an impediment that stops one story, and blockers can be important impediments. The key thing to remember is that blockers are not the only type of impediment. There are many other types.

Impediments can (and always do) include “things around here that have been here for ages that no one has ever tried to identify, much less fix.” That is, we are asking fish to identify water as the problem. It is almost that bad and that hard.

So, you have to ask the team (and others) to think much differently, imagine that anything could be fixed and put those “it would never be fixed” things on the list. And then, especially if you are the SM, you must figure out how to address them.

Addressing Impediments

Some impediments are initially expressed more as symptoms than as root causes. So, very commonly, we must identify the root cause.

This means someone — probably the team — must do some form of root cause analysis (RCA). This might be done with the “Five Whys” technique or with other tools.

Some impediments can be fixed, and some impediments cannot. The ones that might be fixed, we normally recommend two weeks of fixing — if that is possible, which we find usually is if you work hard to identify the right two weeks of work — to try to get some intermediate benefit.

Once again, some impediments cannot be fixed, such as a hurricane. But even those impediments can be mitigated; that is, the impact on the team can be mitigated almost always. So, we take mitigation steps.

Product Backlog Refinement

We have a book on Agile Release Planning. In that book we discuss this topic (PB Refinement) at some length.

The key idea is that the Team is continuously refactoring the product backlog, in each sprint.

First, we recommend 2 week sprints. And a Team of 7 (including the PO and SM).

So, in those conditions, we recommend a "short-term" meeting in the second week, before the Sprint Review and the Retrospective.

In that meeting the Team gets to vote on the quality of the details provided (or organized) by the Product Owner. The key purpose of this meeting is to enable the Implementers to give their feedback on the information (details) that the PO has had prepared for the 8 stories in the next sprint (I recommend that each sprint have at least 8 stories).

Any one Implementer can blackball a story.

And it is fairly typical for one or two "last" questions to be identified, and the PO then needs to get those answered before the next Sprint Planning Meeting.

The other Refinement (or Grooming) meeting is what I call the Long-Term meeting, in the middle of the first week of the Sprint. There, we can re-do anything that we did in the initial Agile Release Planning. Find new stories, break up stories, vote or re-vote Business Value Points, vote or re-vote Story Points, re-organize the Product Backlog based on an improved understanding of Velocity or Risks, Dependencies, Learning, or MMFS/MVP.

All this is explained in more detail in the Agile Release Planning book.

You Must Self-Organize

It is fine to give you the Scrum framework, which is bare bones.

What is essential is for the Scrum team to use that framework to help them self-organize successfully around getting their work done, their goal or mission accomplished. In fact, probably more essential than Scrum is that the Team self-orgaize.

All adults, even most children, know how to self-organize.

The problem is that all adults have also learned “mental blocks” to self-organizing.

It is not that the cannot do it at all, but just that they will stop doing it, or slow down a lot, in certain circumstances.
Example One: They feel they are not supposed to self-organize, for example, they have been trained that the boss will tell us what to do.
Examples Two: They have been trained that “we cannot do X until Y has been completely done” and the signal that Y has been done has not be given. One can of course imagine that that idea fully makes sense….and one can also imagine where waiting for perfection on Y will never come, and so it does not really make sense.

Putting It Together

What is Scrum? Is it the practices? Is it the ideas? Is it mainly the values?

Many say that if you do not “get agile”, then you can do a bunch of practices, but it won’t make much difference.

I am not sure that is true, as in causation.

Certainly the more one gets agile, the more one does it well and with the right intention. And that will make a difference in terms of results.

In any case, you must put Scrum together with your people. You and the Team must figure how to make it work.

This is hard in some ways. A common way is that people will not like to tell the truth (often about themselves) or they will not see the truth (eg, in a Sprint Review). Maybe Scrum is somewhat counter to the existing company culture.

You and the Team and others must put it all together for you.

Another thing. If you are doing software, then you must meld Scrum and the Team with, really and eventually, all the other good practices in XP (Extreme Programming).

If in another context, then you all must meld it with other ideas and practices.

This is work. It can be hard. There are many to guide you (and you need to ask for help). It is quite do-able.

Managers and Scrum

Managers are essential in Scrum.

In what way does he mean that?

First, I think he would say that managers must help the Teams learn how to self-organize. Or self-organize better.

Then we must explain the new role of a manager in Scrum. Reading the old reports will not cut it. It is actually a better life, where your real skills come into play. A good manager is invaluable.

But, secondly, a manager must do something if a Team is self-destructing. If the Team is about to drive off a cliff.

***
Giving support for self-organization is not the only thing a good agile manager does.

The next key thing is to support removing impediments. For a Team, the impediment-remover-in-chief is the ScrumMaster. But the Team must always get help from other people. And often help requires that the manager say yes. Yes to giving people, or money, or just approval.

Allocating these resources is an important job of the managers.

We need to also talk about servant leadership, and leadership in general, within the firm. But that is for another day, outside this fairly brief Scrum Intro.
.

Change and Scrum

Just doing Scrum starts to change things.

And every time you fix impediments you are changing things.

So, change is part and parcel of Scrum.

Let us state that more strongly. If you are not notably changing your situation, you are not doing Scrum right. Changing things can be many things, almost anything. Whatever needs to change the most to help the Team be happier or more effective. Or both.

But we must be a bit more honest. When you start to do Scrum, it starts to change everything. At least that is what is commonly said. Mostly true.

Neither I nor Scrum are prescriptive about how fast change must happen.

Scrum Values

There are five Scrum Values: commitment, courage, focus, openness and respect.

It is through Scrum that the Team learns to live these values more fully.

It is worth thinking, from time to time, what those words mean, and specifically, what they might mean in the context of working together as a Team.

Ideas Behind Scrum

First we must mention the Agile Manifesto.

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

One could speed some paragraphs trying to explain what those lines really mean.

Then we have the 12 lines of the Agile Principles. Here is my summary of them, very quickly.

- The Customer is important. Deliver something useful fast.
- Welcome change, or at least accept it. And do the best you can with it.
- Deliver working software more frequently.
- Business people and developers must work together daily.
- We want to set the Team up for success, with motivated individuals, and then we trust them.
- Use face to face conversation more. Strongly preferred.
- Working software is the primary measure of progress.
- We all should work at a sustainable pace.
- Technical excellence and good design are key to achieving the benefits of Agile.
- Simplicity is essential.
- The best solutions arise from self-organizing teams.
- The Team regularly reflects, learns, and takes action to become better.

Then there are LOTS of other ideas behind Lean-Agile-Scrum. I will write a book that at least lists them. But let me mention two or three now.

One is the 6 Blind Men and the Elephant story. Google that. This is an ancient story, we don't know how old, but we do know that Buddha used it.

To me, the key point is that we think that no one person understands the whole elephant, and that each person touching the elephant (each team member) has something valuable to say, that will help us. And that, it is the Team's job to work and "fight" and discuss with each other, and discover or comprehend, more of the elephant as soon as possible.

Another key idea: The bad news doesn't get better with age. That is, it helps to be honest with ourselves about the bad news, and then deal with it sooner.

It is also meant to be said in a somewhat funny way, so that the fear of mentioning the bad news goes away. So that the shame of having made a mistake (and normal human mistakes are only one of many sources of bad news)… are not so hard to mention.

Another key idea: experimentation and learning.

Knowledge workers learn together. And mainly by doing experiments and seeing how they turn out. We are learning our way through a complex problem set that has multiple dimensions. (Some examples: What are the needed features now? Who understands the details? How much time can we take? What should the product look like? What would form a minimum viable product? Will this new technology work? How do I understand these new people I am working with? How can we work together more effectively?)

Experimentation of course reminds us of the famous story of Edison and the 10,000 light bulbs.

And, more generally, experimentation means that we will have successful and unsuccessful experiments. Or, to put it Edison's way, all experiments are useful but only some have positive results.

This of course brings up Yogi Berra's saying: "I knew I was gonna take the wrong train, so I left early." That is, we have to allow contingency for "mistakes" (in the experiments) and also mistakes (those things that happen with human beings). And for other things (Ex: the US recently had Hurricane Michael hit the Florida panhandle).

There are many many other key ideas that support Lean-Agile-Scrum. More to discuss later.

You should be learning and re-learning these ideas with your Team. That learning will help them do Scrum (and Agile and Lean) more effectively.

Key Questions, Myths, Mis-Understandings

- Is the SM the boss?

No. The SM has very limited power. The SM is not like the old Project Manager (or at least what people thought the PM was – the boss).

- Does each of the Developers have a personal velocity?

No. Velocity applies to the Team. Really a result of everything done by the whole Team. Obviously the Developers (these include coders and testers in software) are the main drivers of the Velocity. But they do it together, not each person in isolation. Scrum is team-oriented, collaborative (or much more collaborative).

- Does Scrum mean we stop doing estimates and planning?

To the contrary, we spend more time estimating planning. And we do it in such a way that we get more benefit (learning mainly) at less administrative cost.

Another difference is we always expect change and learning, so we never believe in a plan. Any plan. But we understand the power that planning gives us.

See another comment about estimating below.

Recommended Reading

ScrumPLOP.org — We cannot recommend this site enough. First, it is a list of patterns.

If you do not know about Pattern Languages, read the Wikipedia article here, https://en.wikipedia.org/wiki/Pattern_language. You might want to read, or at least look at, Christopher Alexander's book, "A Pattern Language". The idea has, at least in some way, been around for millenia (one imagines), but Mr. Alexander (an architect) is famous now for several books, and "A Pattern Language" is maybe the one that made the concept more well-known recently (he has other books that also helped). Many people in Agile were strongly influenced by his Pattern Language idea. Jeff Sutherland speaks of Scrum being a collection of patterns.

We strongly endorse the idea and use of patterns. Second, ScrumPLOP.org is available 24/7. Third, it is curated by Jim Coplien and Jeff Sutherland. Enjoy!

Toyota Production System by Taiichi Ohno. Strongly recommend this book. Seems to be about automobile manufacturing. In fact, it is about your work.

Extreme Programming Explained by Kent Beck and Cynthia Andres. If you are doing software, once you get the basics of Scrum going, you must start adding things from XP (as it is called).

Scrum by Jeff Sutherland.

Agile Project Management with Scrum by Ken Schwaber. Very useful because if has small stories that explain, in story format, lots of ideas and key issues around Scrum.

Some Relevant Sayings and Quotes

“I learned this, at least, by my experiment; that if one advances confidently in the direction of his dreams, and endeavors to live the life which he has imagined, he will meet with a success unexpected in common hours.” — H.D. Thoreau, Walden

“Whether you think you can or you can’t, you’re right.” — Henry Ford

“You miss 100 percent of the shots you never take.” — Wayne Gretzky

“If you don't set goals, you can't regret not reaching them.” — Yogi Berra.

“Take it with a grin of salt.” — Yogi Berra.

“Things should be as simple as possible, but not simpler.” — Albert Einstein.

K.I.S.S. (Keep It Stupid Simple)

“Do the simplest thing that could possibly work, and then test.” — Ward Cunningham.

“I've missed more than 9,000 shots in my career. I've lost almost 300 games. 26 times I've been trusted to take the game winning shot and missed. I've failed over and over and over again in my life, and that is why I succeed.” — Michael Jordan

“Everyone has a plan ‘til they get punched in the mouth.” — Mike Tyson.

“If you wait for perfection, you might wait too long.” — Joe Little.

“People are remarkably good at doing what they want to do.” — Joe Little.

“Everything is impossible until it becomes easy.” — Goethe

“The road is long
With many a winding turn
That leads us to who knows where
Who knows where
But I'm strong,
Strong enough to carry him
He ain't heavy, he's my brother.” — The Hollies (Russell, Scott)

“Most people do not really want freedom, because freedom involves responsibility, and most people are frightened of responsibility.” — Sigmund Freud

“All of me
Why not take all of me.” — Marks, Simons (from the song “All of Me”)

“Although human beings are incapable of talking about themselves with total honesty, it is much harder to avoid the truth while pretending to be other people. They often reveal much about themselves in a very straightforward way. I am certain that I did. There is nothing that says more about its creator than the work itself.” — Akira Kurosawa, a great movie director

“It is more blessed to give than to receive.” — Jesus

“Everything changes. Nothing remains the same.” — Buddha

“Bhikkhus, all is burning.” The beginning of the Fire Sermon by Buddha. The desires of our customers can never be quenched. Desire (fire) is neither good nor bad, but to avoid the pain, we must learn to step back from it.

“In theory, there’s no difference between theory and practice. In practice, there is.” — Yogi Berra

“I knew I was gonna take the wrong train, so I left early.” — Yogi Berra; the train in this quote is a subway train in NYC.

“It ain’t over ‘til it’s over.” — Yogi Berra

“90 percent of baseball is mental, and the other half is physical.” — Yogi Berra.

“Don’t believe half the lies they tell about me.” — Yogi Berra.

“The bad news doesn’t get better with age.” Source unclear; in our work, the bad news gets exponentially worse with age.

“If you don’t change things, nothing’s gonna change.” — Joe Little. I am being a bit sarcastic. It is similar to the old saying “You can’t make an omelet without breaking some eggs” – but with more of a tone of “use some common sense and stop being so stupid now”. In the end, most change is moving from stupid to less stupid – and that’s a lot of help.

“If you wait for perfection, you might wait too long.” Joe Little

“People are remarkably good at doing what they want to do.” Joe Little

Quality Is Free. Title of a book by Philip Crosby.

“I went to the woods because I wished to live deliberately, to front only the essential facts of life, and see if I could not learn what it had to teach, and not, when I came to die, discover that I had not lived.” H.D. Thoreau, Walden

“One is asked, then, to accept the human condition, its sufferings and its joys, and to

work with its imperfections as the foundation upon which the individual will build wholeness through adventurous creative achievement." Robert Greenleaf in his essay "The Servant as Leader"

Feedback

Please send feedback (pro or con) to **Joe Little**.

Thanks!