A brief introduction to Agile Product Development with Lean and Scrum

Mikael Lundgren
Introduction

Concepts such as Lean and Agile have become increasingly popular as people and organizations are inspired by the success of companies such as Toyota. These Agile or Lean frameworks grew out of a number of companies in Japan after WWII, later indirectly inspiring the likes of Ken Schwaber and Jeff Sutherland to introduce Scrum as a framework for software development in the early 1990’s.

Lean and Agile work principles share many ideals. Where Lean express good practices regarding organization and work, Agile framework such as Scrum, XP, DSDM etc. express various ways of implementing several Lean practices for the software industry. All of the Agile frameworks have three things in common- they are **iterative, incremental and time-boxed**.

- Iterative – all work is done in short cycles, where the results are analyzed, and the way we work is improved accordingly.
- Incremental – we work with scalable, modular solutions, where we can often release a product that is **done, but not complete**.
- Time-boxed – time limits and deadlines are used for many things, and always honoured, but we use strategic measures to work with scope and solution alternatives to deal with them.

The actual term ”Scrum” is adopted from the world of rugby, where a scrum is the gathering of the team in a huddle on the field to solve a problem (sometimes resulting in bruises)

Scrum is based on cycles of **inspection and adaption**, sometimes referred to as **empirical control**, where variability and unknowns are assumed to exist during development and are dealt with as they are discovered, stands in contrast to **defined** methods which assume that everything can be found out beforehand.

Actually introducing empirical methods can seem like giving up control to many of us coming from a technical background.

*Maybe we can learn how to predict the future if we just analyze the work even more beforehand?*

Rewarding quick decisions and having too much faith in the possibility to predict how development work will unfold is currently leading to huge challenges with economy and trust to many companies and organizations.

---

A *gated*, or *phased* workflow assumes incorrectly that work can be passed on as a relay pole in various degree of completeness between individuals and departments, without losing information and quality in the process. To compensate for the transition problems, energy is wasted on adding documentation and overhead to the handovers.

The agile workflow commits to assembling smaller bits of functionality with all steps from identifying the need to release included in small cycles. This way, the system is evolved in increments of being **done but not complete**. This changes the way we work and organize ourselves.
A closer view on Scrum

This is how I usually depict the Scrum framework:

From this image, all the mandatory bits of the Scrum framework can be seen:

1. **Prioritize**, through a list of high-level requirements called a *backlog*, initially sorted by increasing customer value. The backlog defines the upcoming work and the planning horizon.
2. **Plan.** The development teams are responsible for planning the work together with the product owner, and follow up the results. At the sprint planning, the goals for the upcoming sprint are agreed on.
3. **Do.** During a time-boxed period of time called a *sprint* (when trying Scrum for the first time, 30 days is a typical sprint length), development is done to realize the agreed goals. The sprint goals or priorities are not changed during the sprint, unless agreed between the team and product owner.
4. **Synchronize.** The work within teams and between teams is synchronized daily, and the plan is updated to reflect current status.
5. **Release.** By the end of each sprint, results are made available that are *done but not complete*. This could typically be a system with additional and/or improved functionality.
6. **Review** of the work done is made by all involved, and the process is improved as needed.

When collaborating with an agile team, the most obvious difference to traditional projects is the sprint – the focused work towards set goals in each sprint, goals that don’t change. Change is embraced – but only when planning the next sprint. Also, the team usually produce continuous updates on deviations in the plan, advances and quality measurements.

The fact that development work can progress at a higher speed if done in short stints instead of allowing continuous changes in priorities and requirements every day can be difficult to accept intuitively. But the positive effects are numerous – increases in focus, commitment, less waste of time to context switching and less energy wasted on worrying over unforeseen changes in course. By dealing with change as each new sprint starts, work follows a rhythm, a pulse, often making us stop and think before changing the course. Is the change necessary, and which priority should it have? What will the consequences be? In the
end, the sprint length will be determined by a number of factors, including the trade-off between the speed of focused work versus the need to change direction.

By combining the Scrum cycle with the s.k. PDCA improvement cycle introduced by Deming (see the Wikipedia entry for PDCA!) we can exploit the similarities. First we Plan a shorter work effort (sprint), which is then performed (Do). During work, we Check the results versus expected results, and as the final point of the sprint, we Act on impediments that has been exposed, and change our way of working for the better. This way, improvement is a part of the everyday work instead of a separate project.

**Roles of Scrum**

Scrum basically has three defined roles: Developer, Product Owner and Scrum Master.

**Developer**

Being the developer of an agile team does not only bring responsibility to develop a product or service, but also the responsibility to develop your way of working. That means that we coach our teams to improve over time, by setting requirements on them, and helping them fulfill the requirements. Just as the members of a band play the same song, in the same rhythm, teamwork is conducted. Even if the musicians are specialists, it’s the teamwork that counts.

**Product Owner**

The product owner is responsible for the investment in developing a product by prioritizing customer and/or business value, taking into account cost and complexity tradeoffs with various suggested solutions. The product owner “sings the song”, making the product vision clear to everyone involved. One important aspect of the product owner is that he or she is the single point of decision. This means that the steering committee is usually changed to a reference team, and the product owner needs to be available for questions and decisions throughout the development cycles.

**Scrum Master**

The role of Scrum Master has previously rarely been formalized in organizations. It is a coaching leader with no authority or responsibilities with regards to personnel. The Scrum Master helps one or several teams to keep the pace, utilizing Scrum and various tools and experience with product development to coach them to improve. The Scrum Master often becomes the coach to several roles in the organization through his or hers deep knowledge on Scrum and agile development.
Sustainable Development

The title developer is given to everyone who develops the product. Yes, not only programmers, but also people with specialities such as quality engineers, art directors, tech writers, texture designers, … In short, we use one term to denote all skills that are needed to build complete functions and qualities of the product, all the way from defined need to released function.

This does not mean that everyone should be able to do everything. In many markets, the top specialists in the field are needed to stay competitive – but we require from them that they should work in cross-functional teams, and be able to demonstrate partial results often.

We always try to avoid the situation where specialists become bottlenecks, and to some extent this is a self-adjusting mechanism of Scrum as everyone is responsible for their calendar and availability when planning the work ahead. Scrum makes it clear when projects are being slowed by specialists, burdened with trying to serve many teams and development lines, and we have to act strategically to improve the situation – either by diminishing the pressure by removing work, or increasing the amount of the needed competence.

Working in cross-functional teams, where people with different skills needed to bring requirements into features with customer value work together, quickly becomes mandatory when trying to reap the benefits of Scrum. The opposite will be to hand over half-baked parts of a product (almost done untested code, partially implemented functions, and so on), which eventually leads to rigid control systems and hyge efforts to transfer information and knowledge between groups and people. There will almost always be misunderstandings or loss of information when passing stuff on, which inevitably leads to potential quality problems further down the chain.

As problems of this kind are often discovered late in the process, teams upstream start to lose focus as they are getting interrupted by people downstream asking for bug fixes or more information. It is then easy to end up in a frustrating situation where functionality always seems to be “80% done”, and the end date seems to be impossible to predict.

Typically, two risky handovers are present in many organizations:

- Handing over requirements (and expectations) from the customer to the developers
- Handing over code from programmers to test

Cross-functional work teams will be able to build usable functions with a visible business value done. This means for instance that the quality engineers no longer work as ”inspectors” at the end of development cycles, but are now taking active part in processing requirements (by detailing acceptance criteria) and system design (by ensuring that new functionality is testable and has high quality to begin with) Choosing to do a final verification round afterwards should just be a formality, where most, if not all, quality issues have already been taken care of during development. By working in tight cycles with quality focus, time and money is saved, and we improve our chances to make predictions of the development speed.

Being Done

The concept of being done is so important it deserves its own paragraph. The definition of “done” can be tricky to say the least if different parts of development have different views on what “done” means. Cost control can be made more difficult when various parts of development states that they are “done” with stuff that has not been built with quality. A change in the way we view ”done” is needed, and a first, important, step is to establish what is known as “done criteria”. The done criteria lists everything that needs to be fulfilled in order for a task to be done. Here is a suggestion of simple done criteria that ensures that
solutions are always accepted by the product owner, that we document our work, and that we build with quality (compare with a mountaineer securing himself constantly when scaling the mountain wall – would you be confident just to be secured at just one spot?)

T - Tested
I - Implemented
R - Released (checked in, installed, etc.)
A - Accepted (product owner agrees on, and understands the solution)
D - Documented

(Pun unintended)

Embracing such an ambitious checklist can be too difficult at once. My suggestion is to establish basic done criteria reflecting the current situation, with the long-term goal to extend the done criteria to make it more and more complete. The infrastructure work needed could be planned in upcoming sprints.

**Agile requirements**

Agile requirements work differ from traditional requirements on one important point – that requirements work is considered being an integrated part of development rather than being completely done beforehand. When planning a larger project, some requirement work is needed to get establish a backlog, but it is thereafter supposed to be a part of development work to refine, detail and implement requirements in close collaboration with the product owner and other experts. This is done just in time when the requirement is selected to be worked on in the sprint.

Instead of performing a full requirements analysis and detailing before starting the project, the assumption is made that a number of requirements will change or be reprioritized during the project, and additional requirements will emerge. To deal with this, the most time and effort is put into detailing requirements with high priority, that will be implemented in the near future. This will eventually give the backlog a kind of pyramid shape (or an iceberg as Mike Cohn likes to call it) with high-detail requirements near the top, and lower detail requirements near the bottom (in the future).

As development progresses, the requirements a couple of sprints ahead will be further detailed, thus constantly shaping the backlog, detailing requirements in an incremental and iterative fashion.

From my experience, **User Stories** is a technique to formulate requirements that suits agile requirements work quite well as it helps keeping focus on customer value and users even when detailing the requirements further. User Stories have been described in detail by Mike Cohn.

The requirements on the product- or release backlog are owned by the product owner. When the team commits to a sprint backlog, the ownership of these requirements are formally transferred to the team. With the ownership comes the responsibility to detail and implement the requirements, and the authority to get help in doing this from the product owner. Often, this is done through many short meetings (perhaps over a coffee) where the developers present various ways to implement the requirement for the product owner to choose from.

All priorities on the product backlog are made by the product owner.
From requirements to a plan

A common myth about agile frameworks is that no or little planning is performed. This is wrong, and in fact Scrum often replaces the illusion of control with actual control and insight into the development progress. One thing contributing to this is the fact that you as a developer is expected to check your progress towards the goals and the quality of your work every day. Scrum focus on remaining effort and development teams constantly strive to reach the set goals – for the next sprint, and the next release. Just as we work to release often, the plan and prognoses are constantly updated. This assumes an organization that are set to act strategically on deviations instead of trying to change reality to reflect the plan, as often becomes the case.

Scrum works with content and scope to meet set target dates. This means that if the current prognosis reveals that more time than available will be needed to meet the goals, the product owner together with the teams work with changing the solutions or scope to meet the target date. By having a daily routine to check how much more work is needed on remaining tasks, early warnings on deviations can be found and acted on.

Agile development also focus on giving the product owner opportunity to start using the product early (high priority functionality is shipped early) instead of awaiting one, huge, final release. This gives the opportunity to start earning back invested money earlier.

The product owner is typically responsible for the long-term planning of the product (through the product backlog) and takes part in release- and sprint planning. The team also takes part in release- and sprint planning and is responsible for planning the daily work (through the Daily Scrum). The overlap in responsibilities creates deeper collaboration between the roles.

Larger projects can also be planned this way, with knowledge build-up and backlog creation performed in sprint form, where sprint goals can be formulated as better insights to what the project will make, and if it will be worth undertaking. Often, actual development is started early to try out concepts and design ideas.

All planning sessions are iterative and time-boxed, with participants using a proportionally larger amount of time to detail requirements of high priority, and less time on the lower end of the backlog.

It becomes clear that all skills needed to create complete functionality with customer value have to be present. In the world of agile development, phrases such as ”the GUI is done”, or “the database tables are done” have no relevance to the product owner. A functionality is either done (offering value) or not. It also becomes important to have a clear picture of the current quality. Time estimates lose relevance if it is not clear whether the solution will bring 30, 300 or 30 000 defects to the product, making it close to impossible to manage the project. Test- and quality engineers work in the development teams to build quality in and design testable functions to help measuring quality constantly.

Visual planning

Visual planning has become popular as agile frameworks have gained popularity, often inspired by Kanban systems on Toyota, giving visual feedback on the development flow.

In the purest form, it’s about minimizing the need for complex plan- and status tools, instead lettings teams and management work with simple tools such as whiteboards and sticky notes. In this way the plan becomes available for everyone and the urge to keep it updated increases. A simple example of a visual roadmap can be seen on the next page.
In this example, the team together with the product owner has set up a tentative plan a few sprints ahead. To this, a sprint backlog board can be added, containing the goals and requirements of the current sprint.

A visual planning board should at least contain remaining work, a clear formulation of the goal, and the current prognosis. These can be represented by sticky notes with activities that are marked or moved according to whether the work is waiting, in progress or done, a sprint goal, and a prognosis curve, often called burndown chart. It is usually an advantage to list the done criteria as a reminder of what is expected to be able to mark a task as done.

Typical gains with visual planning are simplicity and access.

**Anatomy of the sprint**

**Sprint planning**

The first day of the sprint starts by planning the work, and the development team commits to as much work from the backlog as they can finish during the sprint, considering available time and development velocity, using their Scrum Master to coach the time-boxed session if needed. The requirements are detailed as needed and estimated. Once the team and product owner agree on the scope, a sprint goal is formulated, and the team sets up a sprint backlog with tasks to realize the sprint goal based on requirements from the product backlog. When working with a longer project plan, long-term prognoses are updated when planning the sprint.

**Sprint work**

After planning the sprint, the team starts working towards the sprint goals. By re-estimating the remaining work every day, the team tries to give early warnings on deviations, in worst case leading to a re-negotiation of the sprint goal with the product owner.

Work is synchronized daily by team members briefly informing each other on completed tasks, tasks that will be started, and identified risks or impediments to the work (see format later). The Scrum Master helps the team by removing impediments they can’t address themselves.
Sprint review

The final day of the sprint is spent reviewing the results of the sprint. The team demonstrates new functionality and qualities for all stakeholders. After the demo, the team is gathered to review the actual work flow of the sprint, agree to continue good practices and find improvements to things that did not go well. Among typical improvements can be things like moving to a common work room, improve the done criteria, detail requirements in a different way, changing the sprint length, etc.

Improvement suggestions the team can’t act on themselves are collected on an *impediments backlog* by the Scrum Master, who then use it with management to improve the organization.

Agile organization

By combining working organizational structures and meeting techniques from Lean organizations with the Scrum framework, larger efforts as well as the organization as a whole can be managed in an agile way.

Large projects

A practical upper limit for group size governs the fact that Scrum tries to keep teams small, often less than ten people. My rule of thumb is that every team member should be able to grasp what everyone else in team is working with. When this starts to get difficult (and the synchronization meetings such as Daily Scrum starts to take too much time) it’s time to suggest to the team that they need to split, if they don’t realize it themselves.

It is important to remember that a split Scrum team still is parts of a larger unit, usually by having Sprint planning and review together, even when working in individual teams during the sprint. This is a technique that works with geographically separated teams.

The demand to build a fully integrated system every sprint is made more difficult with multiple teams. To ensure that the code does not deviate from the architectural guidelines, technical infrastructure is needed to help the developers get immediate feedback on how changes impact the system a a whole, but the work efforts also need to be synchronized. The synchronization can be performed through a meeting called *Integrating Scrum* that adds to the Daily Scrum.

Integrating Scrum

*Integrating Scrum is a meeting held daily or multiple times a week that follows the format of the Daily Scrum. The main purpose is to ensure that the architecture is held together, and that development teams do not deviate from the guidelines, keeping the system integrated. To keep the meeting efficient, it consists of participants from the respective teams – not Scrum Masters! These ”ambassadeurs” can be different from meeting to meeting (team members that at the time have important integration issues), or same members (informal technical leaders). By having team members sort out the issues directly, time and effort is minimized.

The meeting synchronizes the teams through three statements:

1. What my team has done since last meeting
2. What my team plans to do next
3. Integration issues or risks we have identified

If no risks are identified, the meeting is very short and only serves as brief synchronization and information. Otherwise, a solution meeting with relevant team members is immediately scheduled.
To this meeting, we need an Integrating Scrum Master! This could be one of the teams’ Scrum Master, but an even better idea is to call in a stakeholder in quality, architecture or similar, such as an architect, quality engineer, integrator, development manager etc. He/she is responsible to ensure that the done criteria are used and communicated!

**Pulse meetings**

The technique of using short meetings focusing on synchronization and finding deviations through visual planning and signal systems is sometimes called *pulse meetings* to borrow a Lean term. These are used successfully by several organizations (in Sweden Scania is one of the forerunners) as a way to distinguish status meetings from work meetings. When these types of meetings are mixed up, they tend to become lengthy and only concern a few participants at the time. The basic idea of pulse meetings is to have a very tight time-box for the meeting which means that participants need to come prepared and that as little time as possible is perceived as waste. Whenever deviations without solution ideas are reported, involved persons immediately plan a work meeting to find solution suggestions. This is sometimes done by having the involved persons stay after the pulse meetings. Sometimes parts of, or entire teams stop their work to bring forward solution suggestions to the integration/architecture problem.

Here are suggestions on some pulse meetings inspired by Daily Scrum, but typically held with lower frequency:

- Line meetings where line- or functional managers meet to briefly check that personnel with proper skills are available for ongoing development, and help each other solve temporary shortages
- Product portfolio meetings where product owners meet to briefly each other on what their teams are working on (earlier sprint results, sprint plans) and priority decisions that need to be taken
- Scrum Master meetings where Scrum Masters meet to review their impediments backlogs to find common points to bring to the attention of management, and share good practices from their teams

And so on!
Contents

Introduction ................................................................................................................................. 2
A closer view on Scrum .................................................................................................................. 3
Roles of Scrum ............................................................................................................................. 4
  Developer .................................................................................................................................. 4
  Product Owner............................................................................................................................ 4
  Scrum Master ............................................................................................................................ 4
Sustainable Development ............................................................................................................. 5
  Being Done ................................................................................................................................. 5
Agile requirements ......................................................................................................................... 6
From requirements to a plan ........................................................................................................... 7
  Visual planning .......................................................................................................................... 7
Anatomy of the sprint .................................................................................................................... 8
  Sprint planning .......................................................................................................................... 8
  Sprint work ................................................................................................................................. 8
  Sprint review ............................................................................................................................... 9
Agile organization .......................................................................................................................... 9
  Large projects ............................................................................................................................ 9
    Integrating Scrum ..................................................................................................................... 9
  Pulse meetings .......................................................................................................................... 10
About the author ............................................................................................................................ 11

About the author

I have worked as an employee in product developing companies since 1994 as a developer, project manager, Scrum Master and development manager with a broad spectrum of product areas, including online gambling, telecom and biotechnology.

In 2004 I started as a management consultant at Citerus, where I work today as a mentor and consultant to my clients to increase their competitive force. I also give courses in leadership and agile management.

I have a tight collaboration with leading organizations within Lean product development to help “our industry” – software development – learn from their experiences and views.

I became Sweden’s first Certified Scrum Trainer together with Tobias Fors 2006 and take active part in research- and knowledge networks on Lean Product Development together with leading Swedish and international companies in the field.

I can be reached at www.citerus.se or mikael.lundgren@citerus.se. I blog at www.leanspired.com.

Second publication 2010-02-25 © Mikael Lundgren, Citerus AB