The EXIN handbook for Scrum Masters and Product Owners

By Johann Botha

Title: The EXIN Handbook for Scrum Masters and Product Owners

Author: Johann Botha

Publisher: EXIN Holding BV

ISBN: 9789076531007 (eBook)

Edition: September 2021

Index

List of figures

Introduction

This book is the official companion to the EXIN Agile Scrum Master and EXIN Agile Scrum Product Owner or Product Owner Bridge certification scheme.

As such, this book is written with the explicit goal to help candidates prepare for the EXIN Agile Scrum Master, EXIN Agile Scrum Product Owner, and / or EXIN Agile Scrum Product Owner Bridge certifications.

As these certifications are ***Advanced Level*** certifications, we will assume that if you are reading this book, you are already familiar with the basic concepts of agile and Scrum as an Agile method. If you are not, we recommend that you also read through the ***Agile Scrum Handbook*** by Nader Rad and Frank Turley, The ***Scrum Guide*** and the ***Nexus Guide***.

Although EXIN Agile Scrum Foundation is not a prerequisite for doing the certifications that are within the scope of this publication, we would recommend that you do so, unless you are already very familiar with the basic concepts of Scrum.

Other EXIN Agile certifications are also available. Please check the EXIN website for the most updated information at: https://www.exin.com/certifications.

Foreword

My involvement in agile goes back a long way – using another Agile method, Extreme Programming (XP), to manage software development projects some two decades ago. We used XP when I headed up an internet start-up for a large IT conglomerate, with a bunch of maverick programmers. Since then, I have used Lean, Scrum, ABC/DSDM, Kanban and DevOps methods and concepts in many projects, and in consulting and coaching engagements.

The natural result of using adaptive and empirical methods (like Agile Scrum) is that one develops preferences and biases based on one's own experiences. I would argue that since being agile requires that you can think and act beyond one method, drawing on your experience and the resultant preferences is perfectly fine. I have however tried not to let my personal bias and preferences come across in this book and have hopefully succeeded in doing so. This book is primarily about Scrum, and I have tried to stick to this topic as best I could.

That being said, Scrum quite often benefits from the use of methods and techniques that are not from Scrum. Two quick examples would be the fact that we will use Kanban as a means to manage iterations visually and achieve flow, and we use MoSCoW from ABC/DSDM as a possible means to order the product backlog.

I hope you enjoy this book and find it practical enough to use it not only as a means of preparing for your EXIN exam, but also as a reference guide in your task as Scrum Master, Product Owner or Agile Coach.

Sincerely,

Johann Botha. November 2020 – Johannesburg

1 Setting the Agile scene

1.1 Agile thinking

Big 'A' Agile (noun): relating to or denoting a method of project management, often used for software development, that is characterized by the division of tasks into short phases of work and frequent reassessment and adaptation of plans.

Small 'a' agile (adjective): able to move quickly and easily; able to think, understand and respond quickly.

The simplicity of these definitions, to be found on PM-partners website in Australia, set the scene perfectly.

1.1.1 Agile as a 'project' approach (Agile with a big 'A')

Agile project management approaches are ideal for empirical projects like software design and development, and that would be true for any project in a complex system. The number of variables and unknown factors in these projects makes a traditional Waterfall approach impractical.

Agile project management approaches emphasize the notion of discovery as a consequence of experimentation and an acceptable means to deliver an evolving project.

The concept of *agile* was first defined on 11 February 2001 at the Snowbird ski resort in the mountains of Utah. The outcome of this seminal meeting was the Agile Software Development Manifesto, known today as the Agile Manifesto.

Present at this historical gathering were representatives of many of the popular Agile methods still in use today. All of the members were convinced of the need for an alternative to the document-driven, heavyweight software development processes used at the time.

The reasons for developing the Agile Manifesto by the seventeen visionaries included the frustration with long delivery cycles, vast amounts of planning that proved worthless, and the continually changing requirements as the world changed around them during a project. These frustrations led to the creation of the Manifesto and the twelve associated principles that explain how Agile will help organizations to deliver better value, faster and with exceptional delivery quality.

It was soon realized that what works well for software development projects could work well for any project where requirements are unclear or constantly evolving.

This level of uncertainty in projects is pervasive. Virtually all projects, to a greater or lesser extent, have to deal with ambiguity, changing requirements and learning from mistakes as the project progresses.

Agile and Agile methods are therefore the only way to go if you face an environment where the future is unclear, deliverables cannot be clearly defined from the beginning of a project, and where you need to learn as you go along.

Why? Because an empirical approach is the most sensible approach when dealing with uncertainty, ambiguity and the constant change that happens on complex adaptive systems (CAS) like organizations and projects.

em·pir·i·cal (Ɛm-pîr-î-kl) adj. – Relying on, or derived from, observation or experiment or verifiable by means of observation or experiment. They are guided by practical experience, not theory.

What makes using conventional (so-called Waterfall) project management so limiting is the assumption that detailed planning could and should precede project execution.

There seems to be a general misconception that Agile methods do not pre-plan, when in fact the opposite is true. It would be difficult to find a single Agile project where some time was not spent pre-planning.

The difference is that with an Agile approach, you plan based on what you know, you record what you don't, and you allow people who need to deliver, to answer the questions while they are busy with delivery. Usually, by that stage, they have learned a lot during previous stages in the project and understand the context in which things need to work. Often a better understanding of the context emerges as the project progresses and unfolds.

To those who assume that they can make a perfect, detailed plan at the beginning of a project, take note: no-one is smart or lucky enough to be able to plan and predict everything, especially not up-front.

The next thing that we need to remind ourselves of is *that projects are much more about customer expectations than specifications*. No matter how well we try to define specifications up-front, we will never be able to capture and quantify the expectations of customers and users.

Sometimes the customer, the user, or the consumer must see something tangible before they can accept or reject it because it either meets their expectations or it doesn't.

Using Henry Ford's words to illustrate the point: If I had asked my customers what they wanted they would have said: a faster horse.

We quite often say that the output (what you deliver) doesn't matter to customers; it is the outcome (what they are able to do or achieve with what you deliver) that does. This means that customers are satisfied when 'what' the project delivers (the output specified), enables them to achieve what they want or need (the outcome).

Value is, therefore, always co-created by the supplier and the customer – something worth remembering.

Consider the following question:

How many times were you involved in a project where the delivery was very close to the specification agreed with the customer, but they were still unhappy?

The answer to this question is probably many, if not most of the times, before we started doing things the Agile way.

There are two reasons for higher levels of customer satisfaction when Agile is used:

- Agile engages users and customers in making them part of the project, which gives them the opportunity to provide frequent feedback early on.
- Agile gives us the ability to deliver against a two-week-old expectation, rather than a two-year-old one.

The world changes, and so do the needs of our customers.

1.1.2 agile as a way of thinking and acting (agile with a small 'a')

It is worth remembering that much of the initial discussion at Snowbird was around a different way of working and how using techniques from TQM (the Plan Do Check Act (PDCA) cycle) and Lean (or the Toyota Production System) applies to software development.

It is no wonder that when we look at the twelve principles of Agile, there is a striking resemblance to the principles found in Lean.

The intent was not to develop one way of doing software development. Instead, the focus was on how project teams should value certain things (this OVER that) and translate this into a set of principles to apply in software development.

It is only natural that Agile methods were developed as a result because practitioners needed to move from principle to practice, and all the Agile methods today use the twelve principles as guidance or a moral compass.

Now is maybe a good time to remind you of the four things that agile teams value and the twelve defined principles. Agile values…

Value 1.	**individuals and interactions**	OVER	*processes and tools.*
Value 2.	**working software**	OVER	*comprehensive documentation.*
Value 3.	**customer collaboration**	OVER	*contract negotiation.*
Value 4.	**responding to change**	OVER	*following a plan.*

Just looking at the values, it is clear that it is more important to help customers get value than all the other things we do in projects. Please note that these statements do not say the processes, tools, documentation, contracts, and plans are not important. The values merely state that talking and working with people, giving them something that satisfies their needs, involving them in the process of delivering value and knowing when their circumstances and needs have changed, is more important.

The Agile principles build on the stated values and make them actionable. They are:

Principle 1. **Our highest priority is to satisfy the customer through early and continuous delivery of valuable software** because customers are happier when they receive something they can use to reach their outcomes sooner, and often, rather than having to wait for long periods between releases.

Principle 2. **Welcome changing requirements, even late in development.** Agile processes harness change for the customer's competitive advantage. As the customer environment and landscape changes frequently, so do their needs. This is not scope-creep, it is reality. Deal with it, otherwise you will fail. Just remember that for every new requirement there is most probably an old requirement that is no longer valid.

Principle 3. **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference for the shorter timescale. This touches on principle 1 but is not exactly the same. Here we say that we should not only deliver something sooner that customers can use but also at a predictable rate or cadence (also see principle eight).

Principle 4. **Businesspeople and Developers must work together daily throughout the project.** Involve stakeholders early and often. If you can, even try and make them part of every delivery iteration beyond just the sprint review. We often make users of software part of Scrum teams. Who knows the requirements better than the person doing the work?

Principle 5. **Build projects around motivated individuals.** Give them the environment and support they need and trust them to get the job done. This is your hint that if you try and use Agile without cultural change within the organization, you will most probably fail.

Principle 6. **Face-to-face conversation is the most efficient and effective method** of conveying information to and within a development team. Remember a full 80% of communication is non-verbal! We are learning to deal with a less physically connected world, but in a development team, face-to-face interaction is king. Note: in a post-Covid-19 world, working in virtual teams is the new reality. It is however still recommended to use face-to-face communication when possible and as such, virtual team meetings should be done using video.

Principle 7. **Working software is the primary measure of progress.** Can it do the job? Can it be used? And does it produce the results intended? If not, you have failed.

Principle 8. **Agile processes promote sustainable development.** The sponsors, Developers and users should be able to maintain a constant pace indefinitely. Once again, a principle closely linked to principles one and three. In Agile, there will never be a “when” for the whole project, but there should be a “when” for what the team is busy working on right now. As the environment in which the business constantly changes, so too will requirements. That means Agile projects are never finished!

Principle 9. **Continuous attention to technical excellence and good design enhances agility;** this means that ensuring that teams have the right

skills, that they follow the rules of good design, and not allowing the build-up of technical debt is vital.

Principle 10. **Simplicity – the art of maximizing the amount of work not done – is essential.** This principle can also be called the Goldilocks principle: everything has to be just right. It is intended as a balance for principle nine, where we say focus on quality and detail. Yet, we need to remind ourselves that more is not better. We need to do whatever it takes to make it work, not less, but also not more.

Principle 11. **The best architectures, requirements, and designs emerge from self-managing teams.** This principle is the most difficult for organizations starting on an Agile journey, as it implies a change of management style, performance measurement, and sometimes organizational structures, including a re-definition of roles and responsibilities. The point here is to leave the detail to the specialists doing the job, namely the team members – so do not micro-manage. Most of today's employees are knowledge workers and don't need supervision; they need motivation and support!

Principle 12. **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.** This implies a level of maturity in teams and a willingness to be the master of your own destiny. To get this working, organizational leadership must tackle organizational change and build high-trust environments.

By now you should be able to see why it can be said that agile with a small 'a' is a mindset, guided by principles of which customer centricity is probably the most important. Agile with a small 'a' is about developing a deliberate set of cultural, organizational, and behavioral changes, that should start at the leadership echelon in an organization.

No amount of using the techniques in this book, producing the artifacts, renaming roles or practicing the rituals described here will make Agile with a big 'A' succeed in your organization, unless you become agile with a small 'a', and that is not an easy thing to do. If we consider why so many Agile initiatives fail – the definitive answer is that you must be agile with a small 'a' before Agile with a big 'A' could work for your organization.

It is human nature to get fixated on a model or method that seems to work, and then start advocating that method. These methods are all Agile with a big 'A'. However, if you use the twelve principles in your organization, intending to achieve business agility regardless of method, you are practicing agile with a small 'a'. This is more likely to bring success and is closest to the spirit of the Agile Manifesto.

You may then ask why this book is necessary at all.

The Agile and Scrum methods, accountabilities and techniques described here are extremely valuable and will yield great results for your organization as long as you don't become a slave to them. They are helpful aids, and they will serve you in good stead; but never let any Agile method dictate what and how you do things because

then you have lost true organizational agility. Agile with a big ‘A’ should always serve agile with a small ‘a’, and not the other way around.

Since this book is focused on practice and how to get the most out of Scrum as an Agile method, Agile will be written with a big ‘A’ from this point forward, except in specific circumstances where there is a reference to ‘agile behavior’, an ‘agile mindset', or ‘agile principles’ in the broader sense of the organization.

1.2 Making a case for Agile

Selling the benefits of Agile is difficult at first and it is best done on a personal level. In this short chapter one possible approach will be presented for you to consider. It has proven to work well.

The approach taken is in the form of an interactive story or roleplay with the skeptic and the advocate as the two main characters. This story is an IT project story, but it can quite easily be adapted to any environment by changing the scenario and some of the parameters used. As most people have seen online initiatives both succeed and fail, it seems to be an appropriate example to use. Some background to set the scene:

XYZ Inc. is a thriving retailer in the fast-moving consumer goods (FMCG) market. They have achieved success in the last ten years and have grown their retail operation from 2 to 15 outlets situated in the nation’s largest ten metropolitan areas.

Recently XYZ has come under pressure from online retailers and, although the company has an extensive online marketing presence, management realizes that they rapidly need to develop online retail capabilities: a webshop. If they can do this successfully, they can leverage their existing market position and relationships to out-compete the new competition.

Two options can be presented for consideration to achieve the desired goal.

Option 1 – Conventional Waterfall project approach

- Research what competitors do (2 months)
  - Done by IT
- Research technology available (2 months)
  - Done by IT
- Define detailed specifications for an online retail capability and create a business case (4 months)
  - A Business Analyst from IT collects requirements from the business
- Design an online retail capability (4 months)
  - Done by IT
- Develop that online retail capability (12 months)
  - Done by IT silos: development, testing, release management, IT operations
- **Total project time** **24 months**

Questions to the business

Would you say the above is a reasonable representation of projects of this nature in your business?

- Normally the answer is **yes**.

In the competitive scenario described above, could you wait two years for a solution?

- Normally the answer to the above is **no**.

What would happen if you had to wait two years?

- Normally the answer spells out doom and gloom.

What do you consider a reasonable time to be able to compete with this new entrant?

- Answers normally range from 4 to 6 months.

Do you think the requirements defined and agreed in month 5 of the scenario above would be the same 19 months later, when the project is hopefully completed?

Normally the answer to the above is **no**.

Do you think the business case submitted for approval in month 8 of the project would be valid 16 months later?

- Normally the answer to the above is **no**.

2. In your experience, do projects of this nature complete within budget and on time?
   - Normally the answer to the above is **no**.
3. What compromises would you rather settle for?
   - Answers here vary, but in general the following crop up (not in order of importance):
     a. A less extensive product in a shorter time
     b. Outsourcing the shop to another party
     c. Utilizing an existing platform
     d. Spending more money to get it done sooner

Although b. and c. are valid options and should be explored, they both have distinct disadvantages – for instance, loss of control over customers, and lower margins. Spending more money may be feasible if the company is cash-rich, but experience shows that this option is often not viable or at least not popular.

Option 2 – An Agile project approach

- Identify general high-level requirements based on the organization's own needs and what competitors do (1 month)
  - Done by the business, with help of IT
- Identify one or two product ranges that would most likely sell well online (2 weeks)
  - Done by the business, with help of IT
- Identify and investigate the minimum capability required to offer these products online (2-week design sprint)
  - Done by IT with the help of the business
- Build basic shopping cart capability (4-week sprint)
  - Done by IT
- Build & test user interface (4-week sprint)
  - Done by IT
- Refine solution (2-week sprint)
  - Done by IT
- Launch basic capability
- **Total time to have online capability (4½ months)**
  Enhance capability of the online shop to eventually include all products deemed viable to sell online over the next year (4-week sprints)
  - Done by IT based on feedback from the business on buyer/customer behavior
- Time until a refined and feature-rich online capability **ca. 18 months**

<u>Questions to the business</u>

1. Are you happy that on the launch date the online shopping capability did not meet all your requirements?
   - Normally the answer to the above is **no**.
2. Would you rather have all the requirements met if it meant waiting for another 19 months?
   - Normally the answer to the above is **no**.
3. Do you think requirements, or your understanding of requirements, would change in the 18 months following the launch?
   - Normally the answer to the above is **yes**.
4. Have all of the upfront requirements defined in the past for typical two-year projects, proven to be required or valid?
   - Normally the answer to the above is **no**.

Thinking back to your experiences with past projects, after the two-year period between project definition and project delivery:

5. Have you asked for the system to be updated and why?
   - Normally the answer to the above is **yes** – new requirements emerged, changing markets, customers, competitive situation, etc.

6. How many new requirements emerged during the two-year period?
    - This answer may vary from industry to industry, on average the answer is **quite a few**.
7. How many of the current features would you consider to be essential?
   This answer may vary from industry to industry, on average the answer is **most, but not all**.
8. Of the two scenarios given, if you were the management of XYZ Inc., which option would you have chosen?
    - Most of the times the answer is **Option 2**.

This brings us to an illustration, simplified as a practical illustration. Like money, time also has value and if one compares the value/time equation, Agile is a no-brainer.

Agile is potentially suitable for every organization. Some will face more challenges in changing organizational mindsets and ways of working than others, but everyone can. Those who do not, will most likely not survive to fight another day in this fast-changing digital age.

Culturally, however, Agile will not work in all environments and **unless there is at least some willingness by top management to embrace Agile concepts**, it is best not to pursue them.

Figure 1 Agile creates more value sooner than traditional project management

Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

The question top management must ask themselves is: Can we survive if we don't embrace the concept of business agility? Answering 'yes' implies a willingness to face the facts. Becoming Agile is not an easy task, nor is it an easy decision; it takes boldness and bravery to see it through. But fortune favors the brave.

1.3 Critical Success factors for getting Agile right

Agile is an approach to deliver value to customers using short iterative delivery cycles, with high customer involvement by cross-functional teams who take ownership of the quality of deliverables. To describe anything more than this would be to start delving into a specific Agile method.

Generally speaking, some critical success factors can be defined regardless of which Agile method you use.

1.4 Leadership and behavior, culture, ethics, and trust

Although any (part) of an organization can use Agile methods, Agile works best if the right environment is created through leadership.

Agile, like Lean, is a cultural and behavioral phenomenon before anything else, and in essence, the changes in behavior Agile advocates work best if the leadership of the organization understands them and commits to enable them. These changes in culture are impacted quite significantly by the leadership style in the organization.

In the last 20 years, people have talked a lot about servant leadership but very little has changed in the way management behaves, acts, and manages. What makes the implementation of Agile successful in organizations more than anything else, is a change in the attitudes of managers at all levels of the organization.

The reason why this is so important is that autonomous, self-regulating, cross-functional teams execute Agile projects. For this to work, a fundamental shift in the management and leadership style has to occur.

In the words of Michael Shota: the leadership must understand that it is not about Doing Agile, it is about Being Agile, and being Agile starts with the leadership of the organization.

Leaders in organizations where successful adoption of Agile occurs show a genuine interest in how employees execute their tasks, and inspire employees to develop and act as coaches, teachers, and enablers rather than supervisors.

Successful Agile implementations require **trust** and **openness**, meaning an unwavering commitment from management towards all staff and a genuine desire to enable people in the organization to become the best that they can be.

It also requires communication to be **open, frank, and fair**, and employees being able to trust when engaging with leadership and its structures in the organization.

When asked: how do you create an environment of trust and openness? We say that management must learn to trust their people to do the right thing.

Employees will never exhibit the required open and frank communication to make Agile and Lean work unless they know that they can trust management, and leadership will not punish them if they try something new, speak the truth, and are honest about what happened.

Ernest Hemingway defined it as: The best way to find out if you can trust someone is to trust them.

An example of why high-trust environments are so much more beneficial to organizations than low-trust ones is as follows:

Low trust
Management doesn't trust staff to do their work well and therefore they prescribe to employees how to do their jobs. Often the methods specified by management are impractical, outdated, or cumbersome, and to get their work done, staff do the work in a manner that facilitates the required output.

The danger in this scenario is that the safety net and controls that formed part of the prescribed method, process, or procedure, now also fall by the wayside.

If something goes wrong, no one will acknowledge that they did not follow the process and mistakes are often hidden by employees trying to 'escape' punishment for their misdeeds.

The result is that the environment becomes even more unstable until, eventually, a major breakdown occurs. As part of root cause analysis, a witch-hunt is started to find and punish the guilty party.

The problem is that even if the guilty party was discovered and punished, the organization and management lose, the damage is done and what's lost is lost. Surprisingly, this toxic behavior is the modus operandi in many organizations.

Richard DiPilla said: It does not make sense to hire chess players and then to treat them like chess pieces. It does not make sense to hire smart people and then have them follow stupid rules.

Now let's explore the behaviors expected in an Agile environment in contrast to the above.

High trust
Management employs professional people and expects them to use their skills and talents for the betterment of the organization. Staff working together in small teams, typically between three to nine people, are given a goal and decide amongst themselves how it will be achieved, and how the work will be structured to accomplish the goal within the set time frame.

Because the team members are closer to the work, they have a much better understanding of how things work practically and what must be done.

They design their own work making use of heuristic controls[1], defined by management, to ensure risk is minimized while optimum performance is achieved. If something goes wrong, the team openly communicates and tries to find a permanent fix to ensure reoccurrence does not happen.

Management supports the team with advice, by rolling stones out of the way and becoming a facilitator and intermediary between teams.

Because failure is accepted as part of life, but not tolerated if it continues, problems are solved continually and hardly ever build up to catastrophic failures. In this scenario constant, organizational learning occurs and performance continually improves.

- Which of the two work environments do you regard as the most desirable?
- Where would you rather work?

From the above example, you may have realized that one of the significant cultural and behavioral shifts that must occur in an Agile organization is that of becoming a **learning organization**. Transformation to a high-trust environment is needed, where people are not afraid to tackle problems, when they see them, and solve them before they become an issue for or in the organization.

The idea of a learning organization was popularized by Peter Senge in his book *The Fifth Discipline* (1990) and it is well worth taking a leaf out of the book. Senge proposed the following five characteristics of a learning organization:

1. **Systems thinking**, meaning that everyone should understand the organization as a system. The better everyone understands the system, the more they will learn and the better the organization will become as a consequence.
2. Systems thinking also implies an individual commitment to the process of **learning and personal mastery** becoming the norm in the organization. Learning becomes more than just acquiring knowledge. It becomes a focus on

[1] Heuristic controls commonly define rules of behavior in certain circumstances and not actions. If you know how to behave and what outcome is expected you can decide on the most appropriate course of action. If, however, the control defines an action, the action will always be taken, whether appropriate or not!

the ability to be more productive. Individual learning is the responsibility of the individual and not exclusively that of the organization, and personal mastery should be practiced daily.

3. Assumptions held by people (**mental models**) must change. These assumptions can only change if an open culture that promotes inquiry and trust replaces confrontational attitudes.
4. The development of a **shared vision** is an essential self-learning motivation for staff, as it creates a collective identity that provides focus and energy for learning. Applying the practices of a shared vision creates a suitable environment for the development of trust through communication and collaboration within the organization, and encourages the staff to share their own experiences and opinions, thus enhancing the effects of organizational learning.
5. **Team learning** occurs through sharing accumulated individual knowledge. The benefit of the team or shared learning is that staff grow more quickly, and the problem-solving capacity of the organization is improved through better access to (cross-functional) knowledge and expertise. Individuals are engaged in using dialogue and discussion to achieve shared meaning and understanding. The individuals must have concrete ways of sharing knowledge.

1.4.1 Agile and its impact on organizational structure

Theoretically, Agile can be used in any organization. If done well, Agile will inevitably start to change the fabric of the organization, impacting structure and how roles and responsibilities are defined the most.

Because Agile requires that autonomous cross-functional teams execute projects, this inevitably leads to flatter organizations and multi-skilled, multi-talented employees who require less supervision.

There are strong reasons to believe that the organization of the future will be just like that. As more and more people in the workplace become knowledge workers, the need for complex management structures is becoming less important.

2 Adopting Agile

These days, many Agile methods exist. Scrum is the most commonly used method. You can find some of the other more popular methods in Appendix A of this book.

Lean is the common ancestor for all Agile methods and techniques. They all share common elements, and although the intent here is to focus on Agile and Scrum, from time to time methods and techniques will be included that are not unique to Scrum.

As you start using Agile Scrum in your organization, you should explore other methods also, because you may find some gems there that you can also use.

The practices and principles and the use of artifacts defined in this book are the bare minimal to be able to say you are using Agile.

That being said, at least initially, it is a good idea to stick with one method. Scrum is a very suitable place to start because of its simplicity.

The reason for starting and sticking to one method for a while is because you need to focus on getting the people and organizational change elements to move towards getting Agile embedded into the culture first.

Focusing too much on artifacts, tools, and techniques that may be useful diverts focus from what really matters. Agile is a mindset change first, then an organizational change and behavioral change in people.

Everything in Scrum was specifically designed to facilitate progress on this journey. Don’t get distracted too quickly; that distraction will prove destructive and set you up to fail. It is advisable to stick to the basics until the general mindset has changed in the organization and resistance to the new way of working has fizzled out, before introducing methods and techniques from other frameworks and experimenting with these.

Getting Agile to work in your organization really needs dedication and focus.

In the Appendix to this book, other methods will be explored. For now, the focus will be on Scrum as the Agile method of choice.

2.1 The challenge in adopting Agile

Like in Lean, it is important in Agile to understand that the entire organization creates value together. It is not about a single product, service process, or team. The entire organization must combine all its processes across functions to create value for a customer in the form of a product or a service.

In Lean, value streams are mapped across business functions and processes. It is important not to focus exclusively on products or the product backlog, but to put these in the wider context of the business.

Similarly, in Agile, it is implausible that you will be able to create the type of value that customers want unless the teams creating the solutions are multi-functional and multi-disciplinary.

However, successful cross-team delivery challenges organizational hierarchies, structures, and vested interest.

The creators of Scrum said: Scrum is easy to understand but difficult to master!

The difficult part is the fact that to use Agile and Scrum successfully and effectively, a lot must change in the organization, and these changes are really difficult. The introduction of Agile or Scrum is a massive change management initiative that should not be underestimated and must be carefully planned and executed.

There are various change management methods and techniques, and it is not possible to prescribe which one to use; if a method works for you, keep using it. Hard experience has proven that evolutionary change methods succeed much more often than revolutionary change.

So, part of an agile mindset is to be willing to be patient, and not to rush the change – slowly, slowly, catch the monkey.

Business change management is not the focus of this book, so the main-stream change management approaches will not be covered here. It is however useful to describe at a high-level, two possible alternative approaches to introduce Agile in an environment for the first time.

2.2 Managing change

Resistance to change is futile as change is inevitable, and it is no different when adapting an Agile method of delivering value incrementally.

If organizations don't deal with the *people* side of change, the implementation of Agile will fail.

Jeffrey M. Hiatt and Tim J. Creasey in their book *Change Management: The People Side of Change* (2012), defined change management as a process that follows a repeatable cycle and uses a holistic set of tools; and a competency, because it enables change and creates a capability to increase organizational effectiveness.

They defined five tenets of change management that help people deal with and accept change.

1. We change for a reason

We are changing to achieve a future state that involves a specific and desired outcome. Reasons for change vary broadly from organization to organization and may include cost reduction, improved customer satisfaction, and in Agile's case, better value for customers.

The need for change is always driven by an opportunity or a problem to solve.

2. Organizational change requires individual change
New tools or processes are not enough to achieve organizational change, you need individuals in the organization to adopt new values and begin working in new ways.

3. Organizational outcomes are the collective result of individual change
Change is an individual event, so there are human factors to consider. That means the more adoption an organization gets from employees, the closer it is to achieving the desired outcomes.

4. Change management is an enabling framework for managing the people side of change
Resistance to change is the norm in organizations, especially in change-saturated environments. Managing the people side of change drives a higher speed of adoption and improved proficiency.

5. We apply change management to realize the benefits and desired outcomes of change
The primary objective of change management is to drive and support the realization of the desired future state and the achievement of the expected outcomes.

2.3 ADKAR® and ADAPT

Hiatt and Creasey defined a model to help facilitate change at an individual level which they called the ADKAR Model, an acronym for Awareness, Desire, Knowledge, Ability, and Reinforcement. Change is successful only when every employee has achieved all five milestones.

Of the methods described in this book, the ADKAR or ADAPT method is most probably the closest to the more conventional change management approaches.

The second is Lean management principles and methods, but more on this later.

ADAPT[2] is based on the ADKAR model[3], a popular change management method developed by PROSCI®, and with only a subtle change from the original approach.

ADKAR is an acronym standing for:

- Awareness
- Desire
- Knowledge
- Ability
- Reinforcement

Scrum guru Mike Cohn adapted the ADKAR model to ADAPT because he argued that in an environment where most of the work is knowledge work, Knowledge and Ability are inseparable. As such, he combined these under the heading Ability.

---

[2] Cohn, M. (2010) *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley. Chapter 2.
[3] More information on the ADKAR model can be found at https://www.prosci.com/methodology/adkar

Cohn also argues that Reinforcement is too vague a term, and he ended up replacing it with two steps, namely Promotion and Transfer.

ADAPT, therefore, stands for:

- Awareness
- Desire
- Ability
- Promotion
- Transfer

To follow a high-level description of ADAPT:

Awareness
Only once you realize that the status quo is no longer desirable can real change begin, but becoming aware that what worked in the past is no longer working can be extremely difficult.

People, in general, are usually slow to develop an awareness of the need to change. This is often due to a lack of exposure to the big picture.

The need to adopt Scrum may be the result of a confluence of factors not visible to everyone. And if the need for change is only apparent to a few (e.g. those who have seen the decline in sales to new customers), rumors about the change will start spreading.

It is vitally important that everyone in the organization is aware of the facts that drive change to ensure that changes don't lead to negative behaviors and consequences for the organization.

To ensure awareness is based on fact and not speculation, make sure that you clearly communicate the problem and provide evidence with objectively measured facts and metrics.

Explain to your audience how the change will solve the problem and maybe it is a good idea to do so practically by running a pilot project, proving that the change indeed solves the problem.

There may be various reasons why your organization may want to introduce Agile but keep the message clear and simple – focus on the most important reasons, two or three and not more.

Desire
Moving from an awareness that the current process is not working for the organization or producing the desired results for customers may be hard for people to accept.

When it comes to project management, that is undoubtedly true as most of us have either been taught or have experienced the traditional way projects are done.

It has always worked, why change now?

The fact of the matter is that if you spent a bit of time with people and talked about their negative experiences of projects, it soon becomes apparent that it has not always worked.

If you contrast the negative outcomes experienced and explain or showcase how Agile can help negate these, it starts creating a desire for more positive outcomes.

For instance, if you ask people if what they need now and what they needed two years ago is the same, they will tell you it is not. With Waterfall projects, they will also acknowledge that what is delivered is often out of date, no longer needed, or not aligned to their current needs at all.

It is then relatively simple to contrast the old way, which delivered what is no longer needed, to an Agile approach. The use of Agile, where the project team always focuses on current requirements and delivers against current needs in short cycles will do a lot for them to say; yes, this is what we want, and this is better than the old way of working.

To create a desire to change, show stakeholders that there is a better way and that the organization needs better results urgently. Stakeholders will agree if you make a good case for the change to Agile.

Be warned though: Don’t act as if the past was all wrong; focus on only the new benefits that really will make an imminent or immediate difference. Even better, do a pilot to show that it works and works well. Make sure that it does!

As things start changing, help people through the difficulty of the change to the new way. Be patient and help people to let go. Show stakeholders that the change is not so scary and very safe to make.

Involve stakeholders as broadly as possible and sometimes consider incentives to change also. Incentives as a driver of change are, however, dependent on the context and culture in which the changes take place. Be sensitive to these.

Tom Peters famously said that the greatest difficulty in the world is not for people to accept new ideas, but to make them forget old ideas.

The only way to get people to let go is to show them something better to replace it with!

All of the awareness and desire in the world won’t get a team anywhere if it does not also acquire the ability to be Agile.

Ability
Like in the Tom Peters example before, succeeding with Scrum requires team members not only to learn new skills but also to unlearn old ones.

Some of the larger challenges Scrum teams will face include the following:

- Learning new technical skills
- Learning to think and work as a team
- The concept of self-management
- Learning how to create working products and services within short, fixed time cycles of between one and four weeks (timeboxes).

Much emphasis should be placed on providing the appropriate training. Training, however, is not enough; in the beginning, teams will need some handholding and coaching as well. For this, it is advisable to get someone from outside the company with a high level of experience and a track record of successfully implementing Scrum to be the Agile Scrum coach for the team.

Don't expect too much too soon, and although you want to keep people accountable for doing things the new way and doing them correctly, don't set unreasonable targets.

Inspect, measure, provide feedback, and involve stakeholders in solving issues and problems.

Just remember that no amount of training will make you and your team ready to do it; you have to start as part of the learning is learning on-the-job and learning from making mistakes. When the Scrum team is new to Scrum, it is better to strictly keep to the rules until there is more experience with Scrum and Agile working.

Promotion
You know that your company will not sell if you don't market your wares. Why should changes be any different?

We continually have to sell the benefit of the new method or the benefits of letting go of the past to ensure that the new approach, in this case, Agile Scrum, is appropriately embedded and institutionalized in the organization.

There are three goals during the Promotion phase:

1. As you will most probably begin small, the pilot project must lay the foundation for the next part of the introduction of Agile Scrum. Promoting current successes and creating awareness for the new approach will jump-start on the next round of improvements.
2. The second goal is to reinforce agile behavior in existing teams by spreading the news of the good things those teams have achieved.
3. The third goal is to create awareness and interest among those outside the groups directly involved in adopting Scrum.

Promoting means publicizing success stories and creating a vibe that others want to be part of. Sometimes doing showcases that involve audience participation can also be a great way of promoting the change.

Getting stakeholders to take part in activities such as games and simulations, gives them a first-hand experience with none of the pain and negative consequences associated with changing in real life.

Transfer
Once you have done the pilot and it was a roaring success you must spread Agile through the organization. You will not be able to maintain the momentum if the rest of the organization does not follow because Agile involves not only doing things differently but also a different way of thinking. If the old way of thinking remains in the rest of the organization, it will eventually stifle even the best start that any organization can make.

If the implications of using Scrum are not transferred to other departments, the organizational immune system will constantly attack the change and will eventually win the battle.

It does not mean that the rest of the organization must start using Scrum, but that the rest of the organization must become at least compatible with Scrum.

Here changing metrics, HR- and other policies, and even governance and control structures should not be overlooked. You cannot measure the success of an Agile team in the same way you measure a traditional hierarchical organization. You have to replace the old with a new way of managing, measuring, and structuring teams. The controls need to change, and the most fundamental are project and program management, HR policies and roles and responsibilities, management structures, performance management, product management, and even funding, budgeting, and financial practices in the organization.

The introduction of Agile Scrum will impact the whole of the organization; the implication is that eventually, everything will change in the organization, and ideally sooner rather than later.

The final word on ADAPT
Like Scrum itself, ADAPT is an iterative approach. It begins with the realization that change is required and that the current way of working is no longer producing acceptable results. As awareness spreads, some individuals, the early adopters, develop the desire to try Scrum in an attempt to improve the situation.

Through trial-and-error, these early adopters within the organization develop the ability to be successful with Scrum. A new status quo may emerge with a small number of teams successfully using Scrum within a broader organization that does not.

As these initial Scrum teams continue to improve their use of Scrum, they begin to promote their successes – sometimes informally, as might occur over lunch with friends on another team, other times more formally as in a department-wide presentation or planned change programs.

This helps individuals on other teams begin their progression from Awareness to Desire to Ability, and then soon, these other teams begin to Promote their successes as well.

2.4 What we mean by products

In Agile Scrum, the word *product* is an inclusive one. Some people already use the word product in this way, saying products can be either goods or services and in fact in many cases products are both goods and services.

Others contrast services to products, quite often talking about products and services; in which case product is not an inclusive term.

As this work is based on accepted Scrum practices, the position taken is that in Scrum, product is meant to be an inclusive term and as such, services are also products.

3 Lean management

ADAPT can be seen as a change management method. The other form of facilitating change is much more organic and evolutionary.

Adopting Lean as a way of managing and running an organization means a radical change in management style, employee behavior, organizational structure, and the way roles and responsibilities are defined.

Lean, however, is an evolutionary management approach rather than attempting to facilitate revolutionary organizational change. That means it takes time and therefore unwavering commitment from the leadership of the organization.

All Agile methods are based on the foundations laid by Lean, and building organizational agility and becoming a Lean organization is the same at its core.

The aim of this book is not to teach Lean, but some core Lean principles and practices that form the foundations for successful Agile implementations will be shared.

3.1 Lean as a strategy and management approach

For the most part, Lean can be interpreted as an operating strategy and a way to design, transform, and run a business, focusing on two main topics:

- maximizing customer value
- reducing waste

After World War II, Japan was faced with significant resource shortages, and it occurred to the leadership of Toyota that a series of innovations was the only way to provide both continuity in process flow and a wide variety in product offerings.

Toyota focused on minimizing the raw materials required to produce cars whilst improving the time between customer order and vehicle delivery.

The ideas and methods used became known as the Toyota Production System (TPS). This system shifted the focus of the manufacturing engineer from individual machines and their utilization, to the flow of the product through the entire process, based on customer demand.

Toyota soon discovered that factory workers had far more to contribute than just muscle power; the people running and operating machinery are the most likely people to come up with innovative ideas and improve the manufacturing process.

Two fundamental principles form the basis of Lean:

- **Just-in-time production**: only doing what is necessary when it is necessary
- **Jidoka**: standardization and automation

As Toyota improved, TPS became part of the all-encompassing *The Toyota Way*, which was published in 2001. Today, the Jidoka and Just-in-Time principles remain;

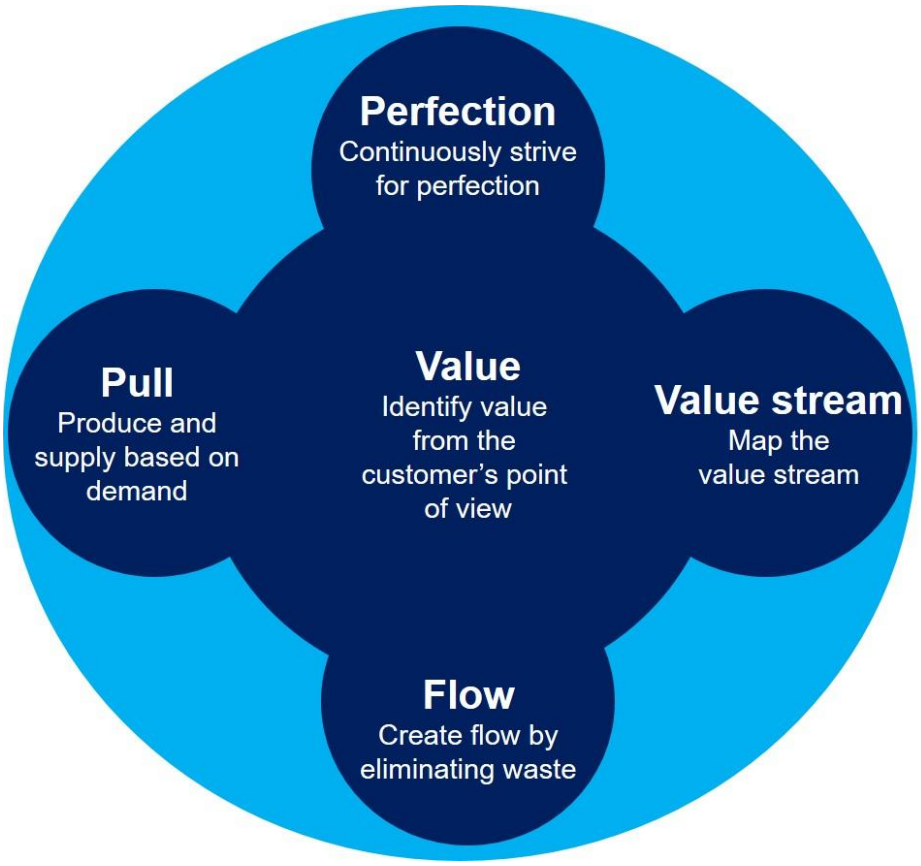
however, the Toyota Way philosophy within which they now reside has included two higher principles:

- Respect for People
- Continual Improvement

Lean offers fundamental improvements for today's business: from design and initial set-up to operations and transformation. These concepts revolve around establishing long-term results for customers and stakeholders, enterprise-wide process capability building and alignment, cultural enablers for sustainable success, and continuous improvement involving everyone, every day and everywhere.

Lean companies are more successful than non-Lean companies. They create better business value and find it easier to retain talent. They also find it easier to sustain success over time.

Figure 2 Lean Principles



Picture created by EXIN based on: Botha, J. (2021). *Lean fundamentals for IT [Courseware]*. GetITright.

Firstly, Lean helps to focus on customer value, and by doing so, organizations add more value to their products and services while reducing sources of waste and increasing their agility and ability to adapt. Being better connected and improving dialogue with customers and end-users, enables an organization to improve customer satisfaction and also improve customer loyalty substantially.

Secondly, the Lean organization continuously improves process performance. Their services are of higher quality, their delivery times are shorter, and their project and operational efficiency keep increasing.

Lean organizations also realize that most workers today are knowledge workers and the most important asset of an organization is its people. The level of involvement of employees is higher in Lean organizations, and so too are the motivation and job-satisfaction levels of employees.

Thirdly, there is financial value to be expected from **reducing process waste, optimizing value-adding work that frees up time to add even more value**. Organizations cannot conceivably do this if they don't understand the 'process' used to create value right across the organization. A technique called **Value Stream Mapping (VSM)** provides the necessary insights to not only improve but also to eliminate waste.

One way of reducing waste is the reduction of the duration between order intake and delivery, by streamlining how work flows through the system and changing all processes and systems to a **pull** rather than a **push** system.

Improving workflow also improves cash flow, as it reduces the need for raw material stockpiles. It must be stressed that improved profitability is not the primary goal of Lean. Although it can be expected, it is a secondary effect of improving and reducing the effort and time spent on non-value-adding activities.

Poor quality can also be a driver for the adoption of Lean. Poor quality has its effects both within and outside the organization. Aspects such as reputation damage and loss of customer trust may cause the organization to incur substantial costs or penalties. Also, waste of talent or an unexpected number of defects causes stress within the organization. Again, costs will increase due to inspections, rework, or demotivation.

Lean shifts paradigms. A traditional paradigm, for instance, is that knowledge is power. An organization may find itself depending on a few critical resources that know how processes work. It is better to ensure that all employees are developed and capable rather than depending on a few.

All the Agile and Scrum principles and ways of working are based on Lean principles and practices, including the concept of self-managing teams, the devolution of decision-making power, creating safe environments in which everyone can contribute and speak their mind, and the drive for all to learn by any means possible, including learning from failure, without punitive results.

4 Scrum and continuous improvement

Agile and Scrum are by nature about continuous improvement; that's why an Agile approach is iterative.

Central to continuous improvement methods like the Deming PDCA cycle, are the idea of empiricism and the concept of constant and continual learning, and the improvement that results from learning. They are also amongst the central themes of Lean and Agile.

Agile can be seen as a way of realizing continual improvement rather than only as a way of executing projects.

New requirements from businesses, customers, or users drive a never-ending program of continual improvement and creating better value for stakeholders.

Improvement is driven by what is important now and the flexibility to tackle immediate concerns – and even changing direction if need be!

This, however, should be done in a planned and predictable manner. It is about creating and maintaining a cadence of change and value delivery in the organization.

The maintenance of a cadence of positive change using Scrum as a continual improvement method means that Scrum also creates predictability and stability in environments where it is used.

5 Scrum basics

5.1 A quick summary of Scrum

Scrum is a high-level framework that was created to help organizations to deliver complex projects incrementally. It was deliberately high-level as the intent of the authors was that organizations would develop their methods to fill in the blanks, in line with what worked best.

Suppose you compare Scrum with other Agile methods like SAFe, Disciplined Agile Delivery (DAD), and ABC/DSDM; this lightweight structure is evident, but rather than making it less useful, the opposite is true. In that case, the freedom to develop your Agile method makes Scrum a potent tool.

Because Scrum is a high-level framework, it is not prescriptive and not detailed. However, the fundamental concepts in Scrum were carefully designed and tested over the years, and today forms the foundation of successful Scrum implementation. The best way of using Scrum is to *adopt* and *adapt* while keeping the basics in place.

It is best to start with the given guidance and to follow that for a few months before you start modifying things. It is often not self-evident why the Scrum method is suggesting a certain way of doing things, and it can take some time before it becomes obvious why doing something based on the Scrum method is a smart move.

Scrum is often defined as:

> A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value. Scrum is: light-weight, simple to understand, and difficult to master.

The last part of the statement is interesting. If Scrum is so high-level and straightforward, why is it so difficult to master?

There are several reasons, but the following stand out:

1. Because it is not the way organizations are traditionally managed.
2. Because it is not the way organizations are traditionally structured.
3. Because it relies on behaviors of the people involved which are not commonly seen in traditional organizations.
4. Because there is no single way of doing Scrum. Situational awareness is therefore vital.

In short, it is difficult to master because it requires a significant change in organizations, and these changes are difficult at first. But once an organization (and the people within it) embraces the change and sees the benefits for the organization, customers, and employees, it is impossible to go back to the old ways.

The first major organizational shift is the fact that in Scrum, all the work is performed by self-managing teams, which is a very foreign concept for organizations with traditional management structures.

In Scrum, there are only three accountabilities, and not one of these is a traditional management role.

The second thing that causes much doubt at first is that there isn't a perfectly detailed plan of the whole project before the start, so there is no end date and Gantt chart to put on a wall.

At first, management may be skeptical about this, as it seems to be a blank page in which no-one is willing to commit.

The problem with the traditional method is that even if you made a commitment and worked out a perfect plan, the reality would not be in line with the plan, and promises are often not kept. We all know that this is the reality – we just have to acknowledge it.

Imperfect plans are not the result of a lack of diligence on the part of project participants. Instead, the assumption that one can perfectly plan the future state of changes in a complex adaptive system (CAS) like an organization is flawed in the first instance.

Scrum acknowledges the impossibility of perfect planning up-front, and rather than trying to do something impossible, it offers a better solution: to continuously work on the things that are most important to the business and to do so within a broader (lightweight) plan. That enables us to focus on business value realization rather than timelines.

5.2 A different way of working

The basics of the framework and associated approach are the use of self-managing teams with only three simple accountabilities (roles), a few planned events and tools (artifacts), and straightforward rules.

These basic building blocks are essential elements of Scrum and necessary for Scrum to work. Although organizations using Scrum have much freedom to adjust and change the way they do things, experience has shown that if you mess with these fundamental components, you break everything.

It is advisable to stick to the basic elements of the framework and associated 'rules'. Each component within Scrum serves a specific purpose and is essential to the framework's success.

Scrum was developed for managing software projects, but many organizations use it equally effective for projects in different contexts.

If iterative value delivery and incremental knowledge transfer are the order of the day – Scrum is the way to go.

In the Scrum vocabulary, the team doing the work is called the Developers. The term Developers remains as one of the three Scrum accountabilities, but this does not mean that the team is formed with only software developers – in this broader context one should instead interpret this as the team which develops, builds, and delivers whatever is the deliverable for that specific sprint or iteration. The Scrum team encompasses the Scrum Master, the Product Owner, and all Developers together.

It is immaterial if your organization is a software development company, makes industrial products, provides services, or even if it is a school. Learning is iterative and incremental – therefore Scrum is used in progressive schools as the primary teaching method. Remember, Scrum is about small, autonomous, and cross-functional teams creating value perpetually. In a broader context these teams don't lose their autonomy to become large teams, they instead network, collaborate, and interoperate, whilst each remaining an autonomous team. Therefore, Scrum impacts significantly on traditional hierarchical organizations which adopt the framework.

5.3 Some essential theory behind Scrum

Note: This book was written as a guide to further Scrum studies and it is assumed that you already know the basics of Scrum. If not, please read the *Agile Scrum Handbook* (Rad, 2021).

The intent here is to help you be a better Scrum Master or Product Owner. In this book, attention will be given to the accountabilities of, and practices used by, Scrum Masters and Product Owners.

Note that Scrum Masters should know about Product Ownership and vice versa.

The ideas behind Scrum are not new and can be traced back to empirical process control and principles and practices used in Lean.

The idea is that we develop knowledge and experience by doing things and making decisions based on what we know – not what we assume. It also means that sometimes we don't know something and the only way to find the answer is to employ an empirical approach.

In science, the basis of an empirical approach is that you develop a hypothesis, and then you create a test in order to test the hypothesis. After testing the hypothesis, you either prove or disprove the hypothesis – either way, you will know more than you knew before.

This is one of the significant benefits of using Scrum as opposed to Waterfall methods, where all of the project planning is done up-front and where most of this is based on assumptions that more often than not prove to be false.

We never hide the fact that we don't know something in Scrum; we acknowledge it, and we try to find the answer as quickly as possible using an empirical approach.

This is one of the three pillars of empirical process control: **transparency**. The experiments and lessons we learn when trying to understand what must be done are the remaining two: **inspection** and **adaptation**.

5.4 The three pillars of Scrum

So maybe it is time to look at the three pillars again:

- Transparency
- Inspection
- Adaptation

Transparency
Transparency means that processes that are used to do the work, and what we know about the work that must be done, must be visible to team members, as they are accountable for the work done and its outcomes.

It becomes much easier for everyone to understand if common standards, language, methods, tools, metrics, and a common vocabulary are used. Work must also be made visible – so that everyone can see and understand the work to be done.

Inspection
Inspection means that we must continuously check if what we are doing is working, and the assumptions (hypothesis) made are true and valid. Once the work is done, the way of working and associated Scrum artifacts must be frequently inspected and validated, or changed, fixed, or improved, which is the last of the three pillars.

Adaptation
Adaptation means that if any element of the process, or work, deviates from the set norm, the process, or work that is done, must be adjusted to ensure a swift return to the set norms and quality delivered.

5.4.1 Scrum events

Scrum only has five events or rituals, and each of these presents an opportunity for inspection and adaptation.

They are:

- The sprint
- Sprint planning
- Daily scrum
- Sprint review
- Sprint retrospective

and although product backlog refinement is not defined as an event it is an important activity.

Many organizations underestimate the cultural change needed to adopt Scrum, or other Agile methods or frameworks. There are significant cultural implications.

We all know that organizational culture can’t be changed overnight, but for Scrum and Agile to work, organizations need to subscribe to a set of values that underpins the use of Scrum.

It is most probably in this single regard that most organizations that abandon their aim to become more Agile and to use Agile methods, fail.

Remember that at the core of Scrum is empiricism and Lean thinking. It is based on the belief that knowledge comes from experience, and that making decisions should be based on what is observed and what is learned. Lean thinking helps us to focus on what works and keep things simple. Lean's approach to reducing waste was not made only for resources consumed but also for the process of building new and valuable products and services.

Because Scrum, Lean, and other Agile methods depend on a set of values to work effectively, it is wise to ask yourself: Are these compatible with current organizational values and culture? The further removed these values are, the more difficult the change and the adoption of Scrum or Agile will be for your organization.

But what are these values?

5.5 Agile Scrum values

Scrum teams operate as self-managing, autonomous units that are responsible and accountable for what the team is doing. When the team fails or makes mistakes, for their learning and growth, they should learn from this and be able to fix it without external interference, as long as they deliver value to customers. The delivery of value in turn means that the team will interpret and validate customer requirements and make sure that these and associated outcomes are achieved by breaking requirements down into tasks to be performed to deliver the value.

One of the key issues here is that the team must feel safe and know that as long as they learn from their mistakes, punitive actions will not follow. This single issue is most probably the biggest reason why Scrum and Agile fail in organizations.

The Scrum values are:

- Commitment
- Courage
- Focus
- Openness
- Respect

If these values are not lived and realized, the three pillars of Scrum will collapse.

Furthermore, everyone must believe and trust that everyone that they deal with, both in the organization and their customers, believe in and support these values.

In essence, team members must not only live and believe in the values; they need to ensure that they are aligned in the organization.

Let's see what the values mean in practice.

Commitment
Commitment means that every team member is committed to the team and also to the work that will be undertaken.

Courage
Team members must show the courage to speak up, and other team members must listen. The team must address issues as quickly as possible to be able to deliver their promises.

Focus
The team members must individually and collectively focus on getting the job done so that they can deliver what was promised within the constraints set for delivery.

Openness
We all make mistakes, and not all of us can do everything. Team members must be open and admit if they made a mistake, if they are facing an obstacle, or don't know how to do a task assigned to them. Hiding mistakes has a knock-on effect that the team cannot allow. For this to happen they must, in turn, believe that they will not be punished because they spoke up. Unfortunately, this is still the norm in many organizations.

Respect
Team members must therefore treat each other with mutual respect and help each other where and whenever they can.

5.6 A summary of Scrum accountabilities

5.6.1 The Scrum team

The fundamental unit of Scrum is a small team called the Scrum team. The Scrum team consists of one Scrum Master, one Product Owner, and Developers. Within a Scrum team, there are no sub-teams or hierarchies. It is a cohesive unit of professionals focused on one objective at a time, the product goal.

Scrum teams are self-managing and cross-functional; they can therefore agree and choose how to best get their job done with no interference from outside the team. Because the Product Owner represents the customer and the business, the priorities of the business are already known by the Scrum team via the Product Owner.

Scrum teams should consist of cross-functional Developers, the individuals of the team should have all the needed skills to complete the chosen part of the work in a timeboxed event called a sprint, and the team should have a reasonably low reliance on skills from outside the team. This means that external Developers should be avoided as much as possible; however, in some moments when there is a gap of a specific required skill, an external Developer can be used for that occasion. This also means that working in a small team makes it easier to live by the Scrum values. Larger teams tend to create more defects compared to small ones.

The Scrum team should be small enough to remain agile and large enough to complete a significant amount of work within a sprint. A Scrum team size of 10 or fewer people is advisable.

Why so small? It is proven that smaller teams communicate better, harmful over-specialization is less likely to occur, and such teams are more productive.

If teams become too large, you should consider reorganizing the large team into multiple cohesive Scrum teams, each focused on the same product. Because the teams are focusing on the same product, they should share the same product goal, product backlog, and Product Owner.

Scrum teams are responsible for all product-related activities. This specifically includes stakeholder collaboration, requirements verification, maintaining team operations, and the maintenance of team practices, whilst continually experimenting on how these practices can be improved. To achieve this the team must invest time in research and development activities, and anything else that might be required to improve the team and their way of working. Scrum teams are empowered by the organization to manage their work. The team should, however, consult broadly to ensure that they understand what must be done and what constitutes value for customers and users. This agreement serves as the basis of defining what is called the definition of done (DoD), a critical metric of the Scrum team's performance.

The entire Scrum team is accountable for creating a valuable, useful increment in every sprint.

A sprint is an event that spans between one and four weeks during which the Scrum team will develop one or more increments of value.

A useful increment means something that can be deployed and used by stakeholders and creates value for them.

Note that one sprint may include multiple useful increments of value.

Scrum defines three specific accountabilities within the Scrum team:

- The Developers
- The Product Owner
- The Scrum Master

Attention will be given later in the book to talk about effective teams, team makeup, and how to constitute a team in more detail. The primary method to ensure that a Scrum team consistently works at a sustainable pace is the use of sprints. Sprints are therefore a fundamental building block of Scrum.

Here are some basic points for a *good* Scrum team:

- **Scrum teams are self-managing.** No one tells the Scrum team how to turn product backlog items (requirements) into increments that create value (products/services).
- **The Scrum team is cross-functional**, with all the skills a team may need to create a product increment. This is the ideal, but not always possible.
- **Scrum recognizes no titles or positions**, regardless of the work being performed by the person. The Scrum Master, the Product Owner, and Developers are not as strict as traditional 'roles', but rather accountabilities. Having a certain 'role' at a certain time simply means that someone must make sure certain things get done.

- **Scrum recognizes no sub-teams**, regardless of domains that need to be addressed like testing, architecture, operations, or business analysis; and,
- Individual Scrum team members may have specialized skills and areas of focus, but **accountability belongs to the Scrum team as a whole**.
- The accountabilities of the **Scrum Master and the Product Owner cannot be combined in one single person**, in this instance segregation of duties is important to avoid conflicting responsibilities.

5.6.2 Developers

Developers are committed to creating any aspect of a usable increment in each sprint, and have the specific skills needed to develop the increments.

The skills, capabilities, and even levels of skills of Developers, often vary broadly and will be primarily determined by the domain of work involved in the sprint.

Developers are always accountable for:

- Creating the sprint plan and the sprint backlog
- Promoting quality by adhering to a definition of done (DoD)
- Adapting the plan, and way of work, as and when needed, to ensure that progress is made towards the sprint goal
- Holding each other accountable as professionals

5.6.3 The Product Owner

The Product Owner is the person, not a committee, responsible for maximizing the value of the product by working with Developers and the Scrum Master.

Being a Product Owner is preferably a full-time job; attention must be given to this point to avoid multi-tasking or potential bottlenecks in the projects. The Product Owner represents the needs of all stakeholders in the product backlog and they must be trained in how to fulfill the accountabilities of Product Ownership. If you want to change the product backlog you can do so by convincing the Product Owner based on the merit of the changes.

- The Product Owner is accountable for creating, managing, and maintaining the product backlog, managing the product budget, and product launch coordination.
- **The Product Owner develops and explicitly communicates the product goal and clearly defines a product backlog** consisting of requirements that will ensure that the product goal is achieved. Requirements are defined as product backlog items.
- Product Owners order the product backlog items in line with organizational goals.
    - Note: ordering the backlog is largely defined by business priority, but it is not the only used criteria.
- **Product Owners attend Scrum meetings where they act as the voice of the customer (VoC)**, explain requirements, and help Developers to better understand product backlog items and their order in the product backlog.

- **Product Owners must coordinate between different Scrum teams** to make sure that efforts are not duplicated, that dependencies are made visible and clear to everyone, and work and dependencies are aligned and well-orchestrated.
- **Product Owners must ensure that the product backlog is refined and that this is done frequently.** Backlog refining includes re-ordering, and the breakdown of requirements to the appropriate level of detail, and getting feedback and help from Developers to do so.

Because of the importance of this accountability, Product Ownership must not be assigned to someone who has no or little authority and is not respected amongst customers.

The Product Owner is accountable but may delegate responsibility to others in the team.

Although not explicitly stated, it is advisable to combine Product Ownership in the business (strategic, tactical, and budgetary) into a single accountability.

The Product Owner should not only be the representative of the business but also a business representative. That means that they should come from the business/customer side of the fence, not the project or technical side of the fence. The Scrum Guide (Scrum.org, 2020) specifically states that for Product Owners to succeed, the entire organization must respect their decisions.

The decisions of the Product Owner are made visible in the content and ordering of the product backlog, and through an increment that can be inspected and reviewed during a sprint review.

If the Product Owner makes a decision, it is the business that made the decision. Product Owners with this level of authority and sway prove to be the most effective.

5.6.3.1 Product Owner as the voice of the customer (VoC)

Voice of the customer (VoC) is a term used in Lean to describe the customer's expectations, preferences, and needs.

As the representative of the customer, the Product Owner should have an in-depth understanding of the wants and needs of customers and users. It is therefore fair to say that the Product Owner is the voice of the customer and of the user.

That being said, Developers still need to engage with users and customers during an iteration to better understand and refine requirements and needs. The Product Owner should never become a gatekeeper that hinders end-to-end communication.

As the representative of the customer, the Scrum team must live out the following **agile principles**:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
    a. Remember: in your context, it may be any product or service.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Businesspeople and Developers must work together daily throughout the project.
4. Simplicity – the skill of minimizing the amount of work done whilst maximizing value – is an essential Agile skill.

5.6.3.2 Attributes of good Product Owners

Good Product Owners all share the same attributes. Here is some guidance on some of the characteristics of an effective Product Owner.

1. Product Owners are **strong communicators** who can communicate well with every audience. They adapt their language and communication methods to best suit the audience. But most of all, good Product Owners are great listeners; they listen, observe, ask probing questions, and reflect on what they have learned.
2. Product Owners should have an **intimate understanding of the business** so that they can effectively translate business needs into actions. They need to come from the business side and not from the project side, to have the right level of insight and to be able to make complex decisions and compromise when stakeholder needs are not clear, or even worse, in conflict with each other.
3. Product Owners must be senior enough to **have the respect of the business** when they make decisions on behalf of the business.
4. They also need to have the **ability to distinguish between what is needed and what is wanted**: Stakeholders will dump many requirements on the Product Owner when given the opportunity. Product Owners must be able to distinguish between what is needed and what is a nice to have. Here the MoSCoW technique[4] can be advantageous at the beginning for Product Owners.
5. **Solution mindset**: Many requirements or needs of the business are expressed in negative terms (problems) and Product Owners must have the ability to facilitate a discussion that does not focus on what is wrong but how the problem can be solved.
6. **Results-driven**: output is not essential; it is the *outcomes* (what the user, organization, or customer will be able to do after an increment has been delivered) that must be achieved.

Some other skills and attributes contribute towards making an effective Product Owner, but the above six are absolutely essential.

5.6.4 The Scrum Master

The accountability of a Scrum Master is nowhere close to that of a traditional project manager.

[4] For a full explanation on MoSCoW, please see the section MoSCoW.

The Scrum Master is much more like a Scrum-half in rugby, where the term *Scrum* originates from. Their job is to see that the Scrum gets the ball and can fulfill their role to get the ball to the backline (create value for the business), who then must use the ball to score a try (goal). If you are not familiar with the game of rugby, you luckily don't have to learn it first to become Scrum literate, so don't stress too much about this analogy.

Scrum Masters help to coordinate work, arrange meetings, help members overcome difficulties, and track progress with the use of visual tools so that everyone knows exactly how much progress was made, which issues remain, and how members can help others when required.

Scrum Masters are accountable for promoting and supporting Scrum as a method and process. They achieve this by helping everyone to understand Scrum theory and practice. We can say that a Scrum Master has a three-part role:

- Trainer
- Facilitator
- Coach

As part of the role as coach and facilitator, attention should be given not only to the technical aspect of Scrum. The people aspect, especially in the beginning, is important as well. The Scrum Master then acts as a change agent, creating a cohesive team and dealing with destructive team dynamics.

Scrum Masters are leaders who serve the Scrum team (servant leaders). They do so by:

- ensuring that goals, scope, and products/services are understood
- by finding ways to improve the management of the product backlog
    - They do not manage the product backlog but come up with suggestions to do it better.

- Scrum Masters help the Developers to **understand product backlog items**, order, priorities, and value.
    - Although this is the primary accountability of the Product Owner, the Scrum Master assists.
- Scrum Masters help the team members to **understand Scrum and all its parts**, and help them to best use and adapt Scrum to the context of the team and the organization.
- Scrum Masters are **facilitators of Scrum events**.
- Often overlooked, the Scrum Master must **protect the team from outside interruptions and distractions**, **remove obstacles and impediments** to increase/support the team performance, and facilitate communication.
    - In this role, they also need to establish an environment that is best suited to work as a team within the context of what the team must do.

The Scrum Master should preferably not be a Developer as well. This practice leads to a lack of attention to the accountability of Scrum Master and may lead to other impediments.

In other Agile methods, a Scrum Master may be called a project manager (which is not a good idea), an iteration manager, an Agile coach, or a team coach.

In their role as coaches, the Scrum Master, by virtue of their experience with Scrum, helps the team to figure out the best way to apply agile principles and Scrum methods and techniques to their specific context and situation.

Be very careful about using existing project managers without the proper training in the Scrum Master role. If you apply the typical command and control structure usually used in Waterfall projects, you will fail, and fail spectacularly.

Also note that using former Tech Leaders in the role of Scrum Masters may have adverse effects, as in their role as team leader they were used to providing structured directions about what must be done based on their experiences; however, the Scrum Master does not make decisions on behalf of the Developers. In case you are considering who should be appointed as a Scrum Master, it is advisable to give the preference to those that have been trained for the role and also are volunteering for it. Especially when starting with Agile Scrum, it is best to bring in a competent coach onboard, with a high level of experience to get things going. Agile Scrum is not only new in practice but also a radical departure from the traditional way of working in most organizations.

You will face many obstacles, and an experienced coach can tell you what works, what does not work, lessons learned, and things to avoid at all costs.

Experienced coaches are masters of organizational change management first, then Agile Scrum experts second. But they do need to be Agile Scrum experts too – so that makes them super-experts at helping your organization through the difficult transition period.

5.6.4.1 Attributes of good Scrum Masters

We have already touched on the accountability of the Scrum Master and the implicit skills required to be a great Scrum Master.

Here are some of the attributes and requirements for an effective Scrum Master. Some of these characteristics are ‘softer’ than others; however, let’s start with some of the hard skills needed by Scrum Masters.

Scrum Masters are coaches and teachers, which implies that they must have a solid knowledge of Scrum and Agile theory.

- **Scrum Masters must be perpetual learners** and constantly expand their technical and theoretical repertoire. This does not only include the theory and practice of Scrum and Agile, but also the latest tools and techniques used to create a great effect by other teams and demonstrating when and how the organization or their team can benefit from these.

- **Scrum Masters are organizers but not supervisors**; they need to build and manage the system used by the team to iterate and do so inclusively in a consultative manner. This means that the Scrum Master should be an organized person and be able to keep their cool when everything around them goes wrong.
- **Scrum Masters must know the tools the team uses intimately** as they will usually be responsible for the maintenance and upkeep of tools and data, like the Scrum board and burn-down charts.
- **Scrum Masters must be good trainers and coaches**. A great Scrum Master must not only know what to do but also be able to explain how and why things are done in a specific way in Scrum to everyone that should be involved.
- **The Scrum Master initially must make sure that the team is mastering Scrum**; this skill is an absolute must. Whilst the Scrum Master is part of the team, their role is also to coach and encourage team members to improve and work together. No one should know every team member's strengths and weaknesses better than the Scrum Master, and he or she must be able to help individuals to maximize their contribution to the team.
- Although the Scrum Master does not have to be technical, **it helps if they have core technical skills** or at least a good understanding of the technical context in which the team functions. This also means that they should have a good understanding of the environment, key role-players, what to do, or who to talk to when things go wrong.

It is relatively easy to check which Scrum Masters have the required hard skills to function both as a Scrum Master and as a Scrum Master for a specific team.

Other skills are not that easy to check up-front.

- **Excellent Scrum Masters are fast learners** and will pick up the required soft skills to function in a specific environment at a rapid pace.
- Whenever people work as a team, there will be conflict and **Scrum Masters must be able to facilitate conflict resolution** and get the team to work as a unit and not as individuals.
- As already mentioned, **Scrum Masters are leaders that serve**. Leaders lead by example and are willing to contribute and do their fair share of work. Servant leadership is about putting the needs of the entire team before the individual needs and helping others perform to the best of their abilities.

In summary, a good Scrum Master must be:

- Responsible
- Humble
- Collaborative
- Committed
- Influential
- Knowledgeable

5.7 Overview of Scrum events

Talking about Scrum is always a chicken and egg scenario – if you talk about how things work, there are questions about roles or accountabilities. If you speak about accountabilities first, there are bound to be questions about how things work.

It is advisable to talk about roles first before diving into how things work because this is perhaps the lesser of the two evils.

In general, Scrum's planned activities are called **events**, and the resources used or generated are sometimes called **artifacts**.

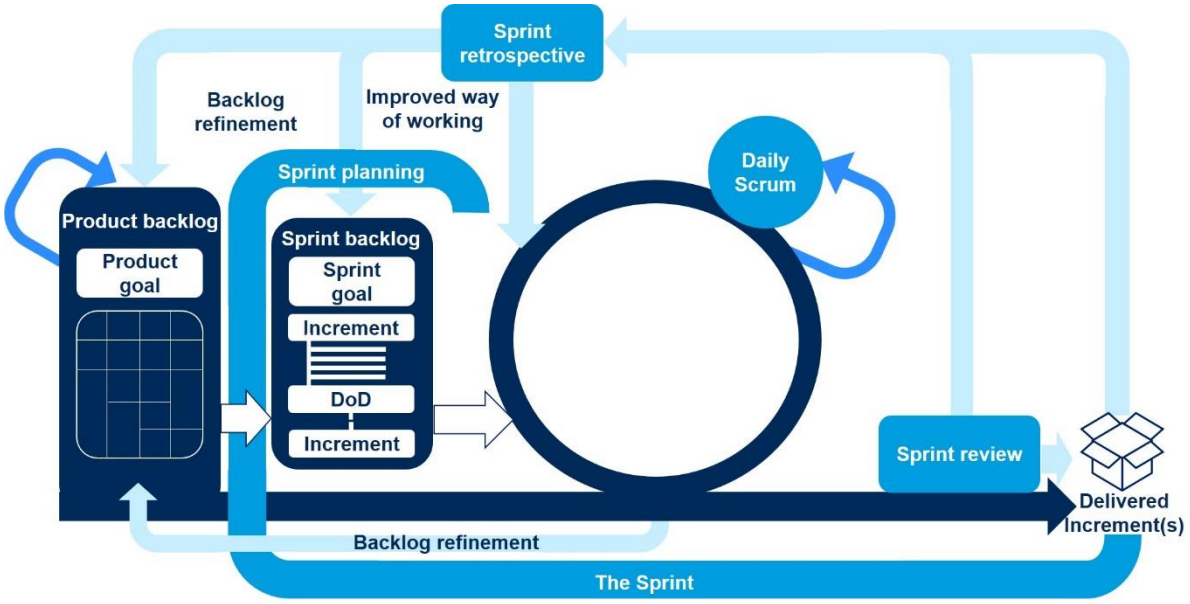
Stakeholder requirements are gathered and recorded as user stories, which populate one or more product backlog(s). These will be described in more detail later.

Each product has its own product backlog and a Product Owner who is accountable for maximizing the value of the product resulting from the work of the Scrum Team. The Product Owner is not a project manager, but preferably the business owner of the product or the product portfolio in question.

Stories in the product backlog are ordered by the Product Owner, who will engage with other members of the Scrum team to deliver stories in the product backlog within the shortest possible time based on their priority. This will be done continuously until all requirements gathered are addressed.

The delivery against requirements is done in short, small, project-like, iterative, planned events, called **sprints**.

Figure 3 A high-level view of everything in Scrum



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

A Scrum team would consult with their Product Owner and select requirements they can complete within a set timeline (sprint). The set time limit is usually the same for all sprints. Sprints are **timeboxed**, or fixed timeframe events, typically between one and four weeks long.

Some organizations set a fixed time for all sprints, say four weeks, while others allow them to vary depending on what must be done. Generally, the length of sprints is not prescribed, although especially in the beginning, there is value in defining fixed-length sprints.

As a starting point, it is advisable to work with fixed four-week sprints. The use of timeboxes, when planning and doing work, allows for better predictability, which makes it easier to better understand the sprint's deliverables and product or feature launch dates.

When recurring problems prevent you from finishing a sprint that has a timebox of less than four weeks, by consulting the Product Owner you may extend the length of the timebox, but it may not exceed four weeks.

The Scrum team functions autonomously. It chooses and plans its own work and is jointly accountable for deliverables. If the sprint fails, everyone fails.

We already indicated that requirements are defined in the product backlog as user stories that also define the outcomes to be achieved by the team. More detail on user stories will come later.

The Scrum team breaks down requirements into smaller requirements that can be delivered during a sprint. The Scrum team, as part of the **sprint planning**, then breaks down the smaller stories into tasks, from which team members can select the most appropriate for action.

As part of sprint planning, a definition of done (DoD) is defined for each increment being delivered in the sprint. Definitions of Done serve as an assurance measure, to check if what was planned and committed, was done, and delivered. Note that a sprint must deliver at least one increment.

Sprint planning is also timeboxed, and a sprint planning meeting for a four-week sprint should take no more than eight hours.

The team then tracks progress towards delivery. The primary event used to do this is called **daily scrum**. During this timeboxed event that takes no longer than 15 minutes, team members report progress, discuss difficulties and inform other members on which piece of work each person is busy with. Sometimes dependencies also mean that work between team members should be closely coordinated during the daily scrum or superstructures. One such method named Nexus will be addressed in detail later.

At the end of the sprint, the team demonstrates the product or feature that is potentially usable and deployable to customers or users. Functional or deployable products or features are sometimes called *shippable products* when the product has

completed the following tests: issues, integration, performance, and usability. This event is called the **sprint review**, and for a four-week sprint, it won't take longer than 4 hours.

A shippable product can be used by users and will unlock value for the user and the business. It may not have all the requested features, but validating that it works and provides desired outcomes must be done. A shippable product may be one or more increments as long as it is delivered in a single sprint.

After the sprint review, the team evaluates their own performance during the sprint, seeing how they can do things better in the future. The event used for this self-evaluation is called the **sprint retrospective**. Sprint retrospectives are timeboxed events that should take about three hours for a four-week sprint.

Although not officially recognized as a Scrum event, Scrum teams often spend a certain amount of time, usually also timeboxed, to help Product Owners in doing the product backlog refinement.

Note: Product backlog refinement used to be called 'grooming', but because of the negative connotation associated with grooming in some cultures, the term was changed to 'refinement'.

Product backlog refinement helps to ensure that product backlog items are in the right order before the next sprint starts.

Please note that although teams often take specific time to do product backlog refinement, this is an ongoing activity, as the team is continuously learning and can discover new dependencies. As soon as this happens, the product backlog is immediately updated with the latest information or insight.

Please note that the intent here is not to fully describe each sprint event, but rather give an overview of the flow from event to event in Scrum. The following chapter will elaborate on Scrum events specifically.

5.8 Scrum events

In the previous overview, five Scrum events were mentioned. They are:

- The sprint
- The sprint planning
- The daily scrum
- The sprint review
- The sprint retrospective

We will now briefly describe each of these events in this chapter. Note that Scrum events are exceptionally simple and straightforward.

5.8.1 The sprint

Sprints are the heartbeat of Scrum, where ideas are turned into value.

They are fixed length events of one month or less to create consistency. A new sprint starts immediately after the conclusion of the previous one.

All the necessary work to achieve the product goal, including sprint planning, daily scrums, sprint review, and sprint retrospective, happens within a number of sprints.

During the sprint:

- No changes should be made that would endanger the sprint goal.
- Quality and quality requirements will never decrease.
- The product backlog may be refined as needed.
- The scope of work may be clarified and renegotiated with the Product Owner as the team learns more about product backlog items (PBIs) included in the sprint.

Predictability is ensured by inspection and adaptation of activities whilst progressing towards the product and sprint goals. When sprints are too long, the sprint goal may become invalid (and therefore misaligned with changes in the product backlog), complexity may rise, and risk may increase. Shorter sprints can be employed to generate faster learning cycles and limit risk. You may see each sprint as a short project.

Many tools can be used to forecast progress, like burn-down charts, burn-up charts, or cumulative flows. While proving useful, these do not replace the importance of empiricism. In complex environments, what will happen is unknown. Only what has already happened may be used for forward-looking decision-making.

A sprint could be canceled *if the sprint goal becomes obsolete*. Only the Product Owner has the authority to cancel the sprint.

However, when the team realizes that a sprint cannot be completed for whatever other reason (usually when Developers notice that work cannot be concluded in the timebox due to unexpected complexity) it is better to revise the order of the product backlog to check which work should be prioritized.

5.8.2 Sprint planning

Sprint planning initiates the sprint by choosing the work to be performed in the sprint. The resulting plan is created by the collaborative work of the entire Scrum team.

The Product Owner ensures that attendees are prepared to discuss the most important product backlog items and how they are mapped to the product goal. The Scrum team may also invite other people to attend sprint planning to provide advice.

Sprint planning addresses the following topics:

Why is this sprint valuable?
The Product Owner proposes how the product could increase its value and utility in the current sprint. The whole Scrum team then collaborates to define a sprint goal

that communicates why the sprint is valuable to stakeholders. The sprint goal must be finalized before the end of the sprint planning.

What can be done in this sprint?
Through a discussion with the Product Owner, the Developers select items from the product backlog to be included in the current sprint. The Scrum team may refine these items during this discussion, which increases understanding and confidence.

Selecting how much work can be completed within a sprint may be challenging, especially if the Scrum team is new to Agile and Scrum. In this situation it is advisable to be careful in sharing velocity expectations with stakeholders; however, a good understanding of their expectations is useful in order to be able to manage them.

The more the Developers know about the team's past performance, upcoming capacity, and the definition of done (DoD), the more confident they will be in their sprint forecasts.

How will the chosen work get done?
Developers need to plan the necessary work for each selected product backlog item for an increment, which meets the definition of done (DoD) of that increment. This is often done by decomposing the product backlog items into smaller work items of one day or less. This is done at the sole discretion of the Developers. No one else tells them how they should turn product backlog items into valuable increments.

The sprint goal, and the product backlog items selected for the sprint, plus the plan for delivering them, are together referred to as the sprint backlog.

Sprint planning is timeboxed to a maximum of eight hours for a one-month sprint. For shorter sprints, the event is usually shorter.

5.8.3 Daily scrum

The purpose of the daily scrum is to inspect progress towards the sprint goal and to adapt the sprint backlog as necessary, adjusting the upcoming planned work.

The daily scrum is a 15-minute event for the Developers of the Scrum team. To reduce complexity, it is held at the same time and place every working day of the sprint. If the Product Owner or Scrum Master is actively working on items in the sprint backlog, they participate as Developers.

The Developers can select whatever structure and techniques they want, as long as their daily scrum focuses on progress towards the sprint goal and produces an actionable plan for the next day of work. This creates focus and improves self-management.

Daily scrums improve communications, identify impediments, promote quick decision-making, and consequently eliminate the need for other meetings.

The daily scrum is not the only time Developers are allowed to adjust their plan. They often meet throughout the day for more detailed discussions about adapting or re-planning the rest of the sprint's work.

In previous versions of the Scrum Guide, three specific questions were notated that each team member should answer. They were:

- What am I working on today?
- What did I complete yesterday?
- What can't I do because of some impediment?

Many interpreted this list as the only questions that should be asked and answered, and that's why they were removed from the 2020 Scrum Guide.

Scrum teams should ask, answer, and discuss whatever is relevant to ensure tasks are completed and flow is maintained.

It is entirely up to the team how they use the 15 minutes to best plan and execute work for the day and deal with impediments and issues.

5.8.4 Sprint review

The purpose of the sprint review is to inspect the outcome of the sprint and determine future adaptations. The Scrum team presents the results of their work to key stakeholders and progress towards the product goal is discussed.

During the event, the Scrum team and stakeholders review what was accomplished in the sprint and what has changed in their environment. Based on this information, attendees collaborate on what to do next. The product backlog may also be adjusted to meet new opportunities. The sprint review is a working session, and the Scrum team should avoid limiting it to a presentation.

The sprint review is the second to last event of the sprint and is timeboxed to a maximum of four hours for a one-month sprint. For shorter sprints, the event is usually shorter.

5.8.5 Sprint retrospective

The purpose of the sprint retrospective is to plan ways to increase quality and effectiveness.

The Scrum team inspects how the last sprint went with regards to individuals, interactions, processes, tools, and their definition of done (DoD). Inspected elements often vary with the domain of work. Assumptions that led them astray are identified and their origins explored. The Scrum team discusses what went well during the sprint, what problems were encountered, and how those problems were or were not solved.

Traditionally teams talked about three items specifically, but the discussion, as with daily scrum meetings, is not limited to these topics. It may, however, be helpful to still ask these questions during a sprint retrospective:

- What should we stop doing?
- What should we continue to do?
- What should we improve during future sprints?

Note that the questions just help to get a conversation going, and the Scrum team should identify the most helpful changes to improve its effectiveness. The most impactful improvements are addressed as soon as possible. They may even be added to the sprint backlog for the next sprint.

The sprint retrospective concludes the sprint. It is a timeboxed event of a maximum of three hours for a typical one-month sprint, and usually shorter for shorter sprints.

5.9 Scrum artifacts

All the artifacts mentioned here will be covered in detail, namely:

- The product backlog
- The sprint backlog
- An increment

Although not defined as artifacts, these concepts are closely linked to the Scrum artifacts:

- The product goal
- The sprint goal
- The Definition of Done (DoD)
- User stories (and the variations of these including epics and features)
- Tasks breakdown

The link between the three artifacts and the second list can be described as a commitment to create or maintain an artifact:

- The commitment for the product backlog is the product goal
- For the sprint backlog, it is the sprint goal
- For the increment, it is the definition of done (DoD)

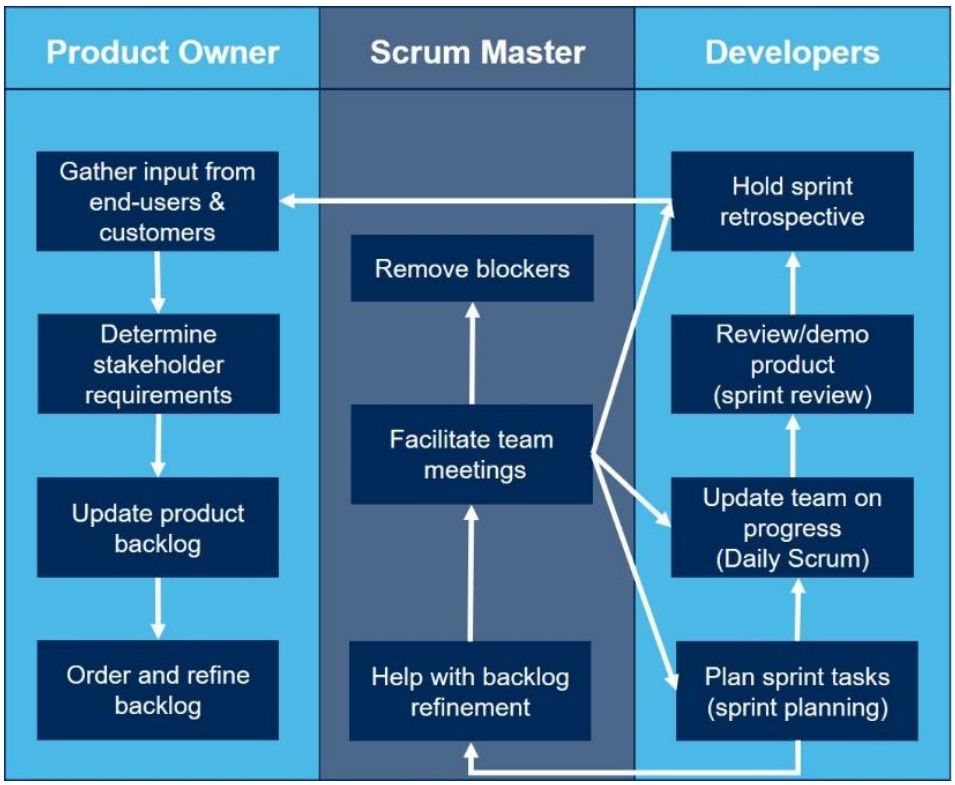
Although no commitments, user stories are used to describe product backlog items and during sprint planning they are broken down into associated tasks.

You may have realized that the defined roles in Agile Scrum are exceptionally simple. It would be pointless to have another chapter on what a Scrum Master or Product Owner must do and then describe only their role in an event or in the use or creation of an artifact, or in fact some other activity.

Instead, the approach chosen for the rest of this book is to talk about a concept or artifact and while doing so, describe the role of the Developers, the Scrum Master, or the Product Owner in a specific event or as part of the concept or artifact discussed.

In summary, the graphic below serves as a quick reminder about the roles in Scrum and the associated activities. This is not a process flow, see this rather as a relationship map of activities and Agile Scrum roles.

Figure 4 Highlighting relationships between Scrum activities (This is not a process diagram.)



Picture created by EXIN based on: Botha, J. (2021). *[Courseware].* GetITright.

6 Other activities of Scrum teams

6.1 Portfolio, products, and roadmaps

Strictly speaking, Scrum gives no guidance on the issue of portfolio management; the guidance provided by Scrum starts at the product level. The reality is that products do not appear out of thin air.

There is a cascade of event horizons where key decisions need to be made in any organization, including the associated planning to ensure decisions are appropriate within the context of the organization, and decisions made are informed decisions.

There is a cascade of planning activities and related decisions within organizations. Planning starts at a strategic level, and then at a product management level, and then eventually to where work is done in operations (Developers). Note that the operations environment can be divided into *operations run* (ongoing activities) and *operations change* (projects). Scrum mainly concentrates on the latter.

For Scrum to work and be scaled, it is necessary to understand the basics of the event horizons, or the level of the cascade from strategy to operations run. In a Scrum context it may look something like this:

- Organizational goals
- Portfolio decisions
- Backlog creation
- Backlog refinement
- Scaled delivery planning
    - Nexus, Scrum@Scale, or even release planning
- Sprint planning

Note that although it is useful to talk about decisions, in an Agile environment these decisions are not sacrosanct, and changing decisions or plans should never be an issue (responding to change over following a plan). Ensure that no time is wasted on unnecessary planning. Do the minimum amount of planning at each level of the cascade. By doing this, it will be so much easier to change plans as you learn more (empiricism) about circumstances, customer needs, or as your understanding of reality changes over time.

Former US President Dwight D. Eisenhower once said: In preparing for battle, I have always found that plans are useless, but planning is indispensable.

Remember Eisenhower's words; planning is indispensable as it is part of our journey of learning and understanding. Focus on planning, not the plan, encourage change and learning, and make it as simple as possible to change the plan.

6.2 Portfolio planning

For many large organizations, the work that must be done is managed in a broader portfolio. The portfolio in question could be around products, or systems, or value streams, or supply chains, or investments, or even programs.

The goal of portfolio management and planning is to determine which products support the organization's goals and objectives.

Based on organizational priorities, we can then understand in what order the organization must work on products and under which conditions they are met.

Portfolio planning should involve a broad stakeholder landscape and should also involve Product Owners. The planning horizon for portfolios is longer-term and could be for the next 6 to 12 months.

The question should be: How will products in the portfolio bring the organization closer to its goals and objectives? And sticking with Agile methods, the organization's view of the portfolio can be expressed by creating a portfolio backlog, consisting of product roadmaps for each product in the portfolio.

6.3 Envisioning your products

Envisioning the essence of a potential product and creating a rough idea of how a product can be created is the starting point. The Product Owners and stakeholders gather to envision a new product at a high level, with a planning horizon that can stretch over more than a year. It is recommended to stick to shorter planning horizons if you can because the world changes far too quickly and frequently.

Participants should consider the purpose or goal for the product and ensure that the discussion always links back to the organizational goals or objectives which the product will underpin.

The *primary outputs for envisioning* should be to define a product goal, to develop a high-level product backlog, and to create a view of how the backlog will be converted into value by defining a high-level product roadmap (a living and evolving document). These outputs become inputs for the higher-level portfolio planning.

The initial high-level product backlog provides a broad starting point for Product Owners to identify the overall and large/broad features, or benefits stakeholders are expecting from the product so that detailed requirements gathering can commence.

Although a product roadmap is an effective tool and mechanism to communicate the purpose of the product, as well as the incremental nature of how the product will be built and delivered, it often turns out to be very different from what was initially imagined.

6.4 Products and product goals

Scrum teams should use the product goal as a planning reference, as the product goal describes a future state of the product and indicates what the superordinate organizational goals and objectives are that it supports. The product goal is part of the product backlog. It is normally written down before you start defining product backlog items and is the primary “check” that items added to the product backlog will bring the organization closer to reaching the product goal.

To define the product goal and a product backlog, it is first necessary to define Scrum’s view of what a product is.

Scrum defines a product as a means or a vehicle to deliver value to someone**.** It means that you not only need to understand which ‘value’ is delivered from the product, but you also need to know who the ‘someone’ is.

Clear boundaries need to be established. Knowing who the stakeholders are, what are their roles, and what constitutes value for them, is a great place to start.

In Scrum, there is no distinction between products and services; products can therefore be physical or very abstract.

The product goal is the long-term objective of the Scrum team. Every sprint goal should be a step towards fulfilling the product goal, and the team must fulfill or abandon one objective before taking on the next.

6.5 Product goals and business value

The product goal is the output of envisioning products and the value they create, and a high-level statement of what a product is.

But how does business value relate to a product goal?

The business will define goals and objectives that the organization must realize to achieve its strategy. Products represented in the organization’s portfolio sometimes enable the organization to achieve these goals. My use of the word ‘sometimes’ is deliberate; other means to accomplish some of the set goals also exist within organizations.

Therefore, the difference between organizational goals and product goals is that the first is related to business strategy and the latter to product management.

What makes the *product goal* in Scrum different than the general product management use of the term ‘product goals’ is that in Scrum, the word is used in a more specific way. In older versions of Scrum, there was a reference to the *product vision*, and the use of the term product goal is closer to this term than the general use of the word in product management circles.

The product goal is a short description of the long-term objective or future state of the product. One can say that the product goal is the WHAT of the product, sometimes supplemented with a WHY.

One can also say that the product goal reflects the overall commitment of Scrum teams working on a product backlog; it provides focus.

Enabling the organization to achieve its strategic intent in itself is valuable, which is the main focus of the product goal.

However, it is important to consider that generally, products enable the organization and/or their clients, and therefore they are valuable to the business in various ways. So products often create both internal value, by making organizational processes and work methods better; and external value, by enabling end-user customers to do something they could not do before, or do something they did before, better. Products do this in various ways, and the same product or even feature can create different value for different users of the product. Delivering value, as useful/valuable increments of a product, goes beyond the scope of what is defined by the product goal.

Although delivering valuable in increments is related to the overall product goal, the product goal does not describe all the product's value.

One should be careful not to make the product goals too complicated by capturing all forms of value the product creates. A complicated product goal becomes less useful as it does not facilitate focus.

Consider the product goal as a description of the most important strategic goal or objectives the product enables.

The next question one should ask in this regard is: What is value or valuable?

Value is determined by the constituent whose problem the product or feature solves. Although value is a very subjective term and difficult to define, attempts should be made to do so.

The main reason is that creating and delivering value consumes resources and, therefore, has a tangible cost. Therefore, the question about value cannot be answered unless one can answer the following question: “Is the value created worth more in monetary terms than the cost of creating the value?”

And this is where it gets interesting in Agile environments.

6.6 Measuring value in real terms

Don't expect traditional financial ratios to measure the gains you made by being Agile or the value of products in real terms.

It is instead recommended to focus on improvement and metrics that are lead-indicators. Financial reports hardly ever positively change organizational performance – but focusing on performance on a day-to-day basis does!

To a large extent, a traditional industrialized worldview drives the makeup and use of conventional financial metrics, an ill-fitting match for Agile environments. This challenge is not new to Agile environments but quickly became apparent when organizations started embracing Lean in the 1990s.

Why are traditional financial ratios so problematic, and more often misleading in Agile environments?

Although some evidence of the use of some financial ratios dates back to 300BC, the proliferation of investing in companies as a business highlighted the utility of financial ratios. The problem they solved was simple. The use of ratios allows investors to compare investments in companies that are vastly different in makeup, industry, risk, and market. Using ratios, an investor can consider if they should invest in a company manufacturing wooden boxes in India, a car manufacturer in the USA, or a financial institution in the EU. Looking at the key ratios makes it easier to make the call.

The ratios we talk about here include all the traditional accounting ratios:

- **Liquidity ratios** measure a company's ability to repay both short- and long-term obligations.
- **Leverage ratios** measure the amount of capital that comes from debt.
- **Efficiency ratios** (also known as financial activity ratios) measure how well a company is utilizing its assets and resources.
- **Profitability ratios** measure a company's ability to generate income relative to revenue, balance sheet assets, operating costs, and equity.
- **Market value ratios** evaluate the share price of a company's stock.

It is not the intention here to state that none of the traditional ratios are useful in an Agile environment. However, many of these ratios will tell quite a different story if measured over a month, six months, a year, five years, or even ten years, depending on the window of the view.

The problem with traditional accounting practice and the resulting financial ratios is that accountants think about financial reporting periods as something with a beginning and an end. It is also simple to use the traditional measurements and metrics for Waterfall projects – after all, the definition of a Waterfall project is that it has a beginning and an end. In this instance, it is easy to calculate return on investment (RoI) for a project by comparing the cost versus benefits realized.

The way one looks at the performance of the business in an Agile environment is very different.

Here the focus is on improving all performance measures continually – there is no beginning and end. Therefore, most traditional financial ratios and accounting practices do not make sense when using Agile or Lean.

In fact, using traditional ratios may lead to short-term improvements with negative long-term results. In an Agile environment, it is instead recommended to make long-term improvements even if it means negative short-term results.

You can apply the lessons learned from Lean very effectively when considering accounting practice in Agile organizations. Companies soon found that radically different financial practices were needed that involved a much broader systems view of the organization and what constitutes 'good financial performance'. The moment this realization dawned Lean accounting was born.

The challenge with using return on investment (RoI), internal rate of return (IRR), and net present value (NPV) as measures to inform Agile investment decisions becomes tricky.

Here are some reasons why old assumptions may not be valid anymore:

- In Agile, there is **no start or end date of a project**. In reality, one could argue that there is not a true project, but only short increments that deliver value.
- As requirements are allowed or even encouraged to evolve over time, there is **no upfront or finite view of what will be done** over time.

RoI, IRR, or NPV become problematic metrics in an Agile environment if one wants to make investment decisions. A different set of measurements is therefore needed.

What makes it more complex is that continual improvement, which traditionally was exclusively an operations activity, is expected to form part of every sprint, thus blurring the lines between 'operation projects' and 'operations run environments' even more.

As with Lean environments (from Lean accounting), the most appropriate measures of success now become:

- Improving flow
- Improving or maintaining quality
- Improved performance (delivery, lead times, productivity)

and as a consequence:

- Improved profitability

The focus of Lean (Agile) accounting is a broad, system view. You may report on improvement in a specific window to course-correct, but it is important to understand that these reports are an interim measurement and do not necessarily reflect the company's health long term.

This changed view means that the practice to fund a department or a project no longer makes sense. Since an organization’s income directly relates to its products' performance, it is more prudent to fund products, where the funding is both project/change and operations run. You have to think about the entire value-stream that creates, sells, services, and maintains the product – yes, end-to-end. This systems view is a significant departure from what accountants learn at university!

The new Lean/Agile way of looking at RoI is more holistic. To effectively measure investment performance, it is necessary to contrast the total of everything that consumes resources across the value stream, with gains achieved by selling a product to customers. Reporting must become a time-sliced instance compared to performance, over time, on a continuum, rather than reporting that reflects company performance as a series of un-related instances.

You may argue that the total cost of the value stream that delivers a product versus the income derived is still essentially RoI, and yes, it is. But how does that help you in the future? Past RoI is a very unreliable measure of the future RoI of a product.

The shortcoming of ratios like RoI is that they are lag indicators – it may tell you that you did not achieve the desired return on investment, but it is useless as a measure to course-correct. It is for this very reason that Lean accounting focuses on improvement as a lead indicator of performance.

Interesting side note: One of Toyota's goals is to improve everything in the company overall, every year by 2%. It sounds small, but they have achieved this since 1950 and the compound effect is huge!

If improvements were made this month, and last month and the month before, you would be well on your way to improved RoI. If you have not improved this month, you need to do something immediately to correct the course.

Members of the Agile Alliance ran projects between 2012 and 2016 to address Agile accounting standardization, but judging by the initiative's output, they largely failed to look at agility as an integrated, value-stream-wide, continuous approach to evaluate financial performance in Agile environments.

Lean accounting, for now, is the most appropriate way to look at investment decisions in an Agile environment, but it requires quite a substantial shift in your understanding of what accounting is and how it should be done in an organization.

So, what is the conclusion from all of this?

Focus on improvement and metrics that are lead-indicators. Financial reports hardly ever positively change organizational performance – but focusing on performance on a day-to-day basis does.

6.7 More on managing the product backlog

Without a decent product backlog Scrum just won't work. The product backlog is the heart of all Scrum planning activities – it is the Scrum team's single source of truth. It describes all the work that the teams need to do, and the relative priority of the work. If it is not in the product backlog, the requirement and its associated activities do not exist.

Product backlogs describe not only customer needs but also all the technical tasks that need to happen in the background to ensure the fulfillment of the customer’s needs. One can therefore describe the product backlog as a “prioritized list” (or more correctly: an **ordered** list, as will become apparent below) of both functional and non-functional requirements that need to be delivered during the project. Once a requirement is fulfilled, the related item is removed from the product backlog.

Some companies also include tasks that need to be done to be delivered against requirements, like planning, resourcing, and other preparatory steps to be taken to get the ball rolling.

Every backlog is owned by a Product Owner, who is accountable for managing that specific product backlog. It is their job to ensure that the backlog is up to date and ordered. They do this, however, with the help of the rest of the Scrum team.

A common practice is to ensure that the product backlog meets four specific criteria. Defined initially by Mike Cohn, this concept is now in wide use in the Scrum community. The criteria are represented by the acronym DEEP:

- **D**etailed appropriately
- **E**stimated
- **E**mergent
- **P**rioritized

It has already been indicated that Scrum shies away from the term *prioritized* as it carries different meanings in different contexts and organizations. Scrum prefers the use of the term **Ordered**. DEEP in a Scrum context thus becomes DEEO:

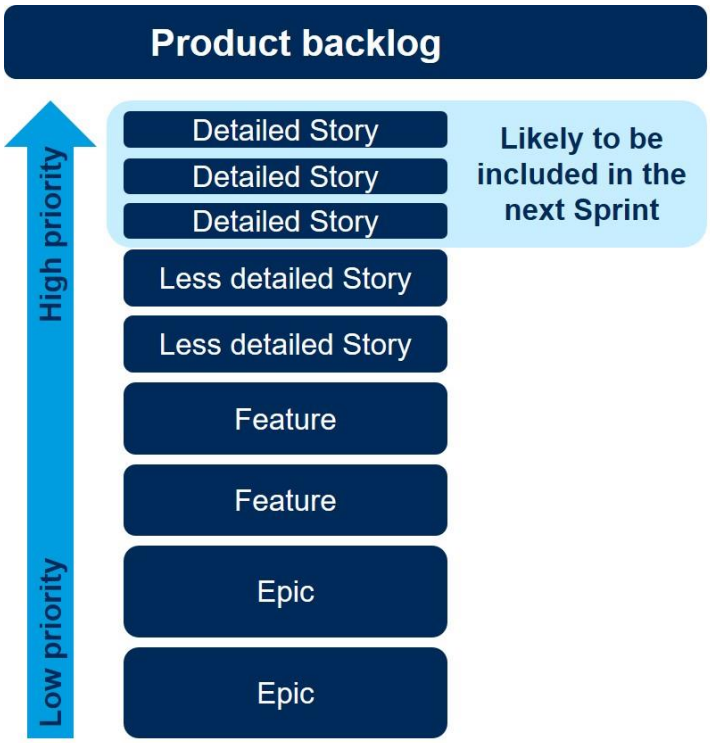
- **D**etailed appropriately
- **E**stimated
- **E**mergent
- **O**rdered

6.7.1 Detailed appropriately

Although the product backlog is not the place where you break down requirements into detailed activities, it must be detailed enough to be able to have a view of how long it would take to complete the work associated with a requirement.

Requirements are normally written as stories, but stories can also have different levels of detail.

Figure 5 Level of detail increases as stories move to the top of the product backlog, larger stories are decomposed into smaller (more actionable) stories.



Picture created by EXIN based on: Botha, J. (2018). *Scrum Masters and Product Owners [Courseware].* GetITright.

Items that are high priority need to be broken down to a detailed enough level to be able to determine the required effort and the skills to meet the requirements.

Important items that are likely to be worked on during the next sprint or two, therefore, need to be decomposed to the correct level during product backlog refinement, and during sprint planning.

This means that as projects progress, the work of decomposing stories or requirements to a greater level of detail will continually happen. It is however only done when necessary and not upfront for all stories or requirements, as when the story is first added to the product backlog it is not even certain if this item and all the related requirements are important, or even if it will remain in the backlog. Often lower priority requirements are removed from the backlog, or other requirements may emerge later that may even replace these.

In Agile, the effort is not wasted on things that would not be used – this is what is meant by doing "just enough" work to be successful.

Epics and features are **complementary practices** that help describe the size and complexity of stories; the Scrum Guide does not differentiate between different story types.

6.7.2 Estimated

All items in the product backlog must have some estimate concerning the required effort to complete the story. Obviously, the larger the story, the more difficult it is to estimate. Estimation on epics will therefore have a low level of accuracy, but that does not matter; stories left as epics are normally of lesser immediate importance.

Estimating is an essential activity to be able to do sprint planning. The definition of done (DoD) must be considered when estimating the efforts to accomplish the feature as an increment. This implies that every increment has a DoD, as there can be more than one increment in a sprint.

Using cross-functional teams can provide a great benefit when estimating tasks due to their ability to provide missing information during the activity realization. Cross-functional teams have all competencies needed to accomplish the work without depending on external resources. Since there is always at least one team member with the knowledge and skills to complete the task, there is always someone able to estimate it.

6.7.3 Emergent

What is meant by emergent is that the product backlog is not a definitive statement of what must happen during the lifecycle of a product. Often customer requirements change during the product lifecycle, and new requirements are added to the backlog. This is, however, not the only time that items are added to the backlog.

Often during the incremental creation of a product, new requirements are discovered that were not previously known. These are often dependencies that other

requirements depend on to work, and many times these are not functional but rather non-functional requirements.

So, when it is stated that the product backlog is emergent, the point is that it will evolve.

6.7.4 Ordered

All items in the product backlog will be presented by ordering items, with those that should be done first being represented at the top of the product backlog. Order is an indication of the importance or implementation order or dependency on other important stories for the completion or implementation of the story in question.

6.8 Product backlog refinement

The product backlog is an ever-changing artifact, and because items continuously enter, exit, or change in the product backlog, it must be carefully maintained.

The DEEO activities already spoken about are tasks involved in the product backlog maintenance. This activity is known as product backlog refinement.

The Product Owner is accountable for refinement, but this is not an activity performed by a single role. The Product Owner is assisted in this endeavor by Developers. They contribute with their specific technical insight to re-order and re-estimate existing entries in the product backlog or new entries that are added to the product backlog.

Developers specifically contribute because they have the technical know-how to gauge how long a story will take to complete and to recognize if an important story depends on other stories and therefore inherits the importance by virtue of the dependency. Scrum team members will also know how large stories can be broken down so that they are easy to action in a sprint.

Even though the Product Owner is accountable for product backlog refinement, it is a highly collaborative activity.

Although backlog refinement is not a defined Scrum event, Scrum team members have to plan time for this activity.

Because it is not a planned (timeboxed) event, product backlog refinement happens when it is required, and often during other Scrum events. If issues arrive, like changing requirements, dependencies, or miscommunication about unearthed requirements, it should be addressed there and then.

Some level of refinement will inevitably happen during the scaled or sprint planning events. And it is only natural that the same will occur during the sprint reviews and sprint retrospectives. Completed stories need to be removed, and stories need to be re-ordered in the light of what is now known. Stories may even be decomposed and new stories added if dependencies or new requirements were unearthed during the just-completed sprint.

It is suggested that you take time each day to review the Product Backlog and Sprint Backlog. Any problems that arise, for example as part of the Daily Scrum, should be discussed as soon as possible.

As a recommendation, teams should spend as much as 10% of a Scrum team's time for refinement. Most teams however spend much less time doing product backlog refinement.

Product backlog refinement normally includes the following activities:

- **Newly discovered items must be added to the backlog**. These can either be new requirements from customers or missing functional and non-functional requirements and dependencies discovered during the previous sprint.
- **Backlog items are ordered** with the most important items at the top of the list.
- **Backlog items need to be sized appropriately** to make it easier to do sprint planning and estimation.
- **Large, or vague important items should be decomposed** into small user stories that could be used in upcoming sprints.
- **Important items are refined** (described better) to make sprint planning easier.
- **Dependencies need to be noted** and the order they need to be performed is recorded.
- **The backlog must be given a once-over** to ensure that all new items are added, the order of execution determined, sizing and completion estimates are appropriate, and items no longer required are removed from the backlog.

6.9 Creating product backlog items

As noted before, requirements gathering is not a one-off event in Scrum. Although a lot of requirements gathering will happen at the beginning of a project, this activity should not focus on getting detailed requirements and spending lots of time in decomposing Epics into fine-grained user stories.

Decomposing large stories is, therefore, an ongoing activity that is best done just before the work is done. This way, teams can ensure that the requirements worked on are the latest and current requirements and not a wish that is two years old. In short, by doing this, you are ensuring that what you are focusing on is current and creates maximum value.

It was highlighted in the previous section that requirements do not remain static and new requirements continuously emerge, often replacing older requirements – so spending too much time decomposing stories that will not be worked on during the next sprints is actually wasteful; you may do a lot of work that will never be used. This is aligned with the Agile principle of doing 'just enough'.

In general, it is fair to say that when you first start gathering requirements, you need to understand the requirement and the outcome, not the detail of how you will get to it.

A great way to think about requirements is to define a high-level view of what would be done in the project first by creating a roadmap, and then to start thinking of the requirements for each stage of the roadmap.

Creating a roadmap will also compel you to think about core requirements, which can be delivered soon and would immediately start adding value, and then focus on the periphery and making the product better.

Gathering these high-level requirements can be done in a brainstorming session where all the key stakeholders are involved. Start the session by defining what you are trying to do and creating a product vision. What problem are you solving or what value will you be unlocking?

The product can then be broken down into high-level components, and the development of these can be the basis of your product roadmap.

Don't get involved in the details at this time – this is talking in broad strokes. It is much like first defining what the minimal viable product (MVP) should be. The focus should therefore be on the core functionality without which the product will not work.

Remember that the product backlog will be refined over time and more detailed and less important requirements can always be added later.

DSDM talks about a concept called MoSCoW – which may be appropriate to mention here.

MoSCoW
MoSCoW is a prioritization method developed by Dai Clegg, one of the contributors to the development of the Dynamic Software Development Method[5], and intended to be used in ordering items in a sprint. Using MoSCoW during your requirements discovery process can be very helpful and provide the Scrum team with valuable insights.

When defining a requirement, you ask if the requirement identified is a

- Must-Have
- Should Have
- Could Have
- Won’t Have

In this instance, it is recommended that you concentrate only on the Must Have and maybe some Should Have at first, as it is often difficult to distinguish between the two initially.

[5]DSDM, which is now also called ABC, short for Agile Business Consortium.

The difference between Must Have and Should Have can be defined as follows:

- **Must-Have** requirements are critical and should be realized soon, to deliver the required value. If you don't deliver against these, the project would be a failure.
- **Should Have** requirements are important but may not be that urgent. Generally, Should Have requirements should be delivered by the end of the project to ensure the product is fully functional or easily usable.
- **Could Have** requirements are nice to have features and Won't Have requirements are specifically excluded because there is not a justifiable cause to include these items as part of the project.

6.9.1 Decomposing non-functional requirements

There is one exception to the rule of not decomposing requirements immediately, which is non-functional requirements.

Non-functional requirements are the parts of an initiative that the rest of the initiative will depend on, but the customer doesn't ask for. It would be best if you had a good idea of what is involved in delivering against these requirements, as non-functional requirements are the foundation for the rest of the project.

Non-functional requirements are therefore always decomposed as best as possible by the team as soon as they are known.

6.10 Requirements gathering – outputs and outcomes

Understanding which outcomes users and customers seek is seen as very important in Scrum. Understanding these outcomes is a fundamental part of the way Scrum gathers requirements from the outset.

So, a minimum requirement for entry into the product backlog is a good understanding of WHICH stakeholders will rely on the requirement (this provides context and helps to identify dependencies), then WHAT is required and WHY it is required.

It is for this specific reason that user stories are structured the way they are:

> As a <stakeholder ROLE>,
> I want to <the WHAT of the requirement>,
> so that <the WHY of the requirement>.

If you are working from already collected requirements, be very careful. It is recommended that you do not translate already gathered requirements into user stories at all.

Traditional requirements gathering techniques aim to do precisely the opposite of what is described above. These techniques attempt to define fine-grained and detailed requirements.

Just translating or copying these to your product backlog will result in a very complex backlog, with very little insight into prioritization and value created.

If you have to work from traditional detailed requirements, it is recommended to create high-level requirements from these, by grouping requirements (the opposite of item decomposition) to end up with fewer items, that can be more easily sequenced, ordered, and estimated.

A technique to use here is to write all the detailed requirements on sticky notes and to create an affinity map with groups of related requirements. The group descriptions can then be added as coarse-grained requirements into the product backlog.

Using this technique will also result in getting rid of or de-prioritizing requirements that would fit into the Could Have and Won't Have categories.

There are lots of different ways that user and other stories can be represented in the product backlog. Some people prefer a pre-defined format, and others don't. A disadvantage of using a pre-defined format is that it automatically forces one to think about requirements in a certain way.

Sometimes user stories exist that are literally just the line shown above, and at other times they can include lots of details, like stakeholders, priorities, dependencies, related activities, or even who should work on the story in a sprint. In most cases, however, recording all of this is a bit excessive.

The user story format shown above ensures that all the absolutely required information is recorded in the backlog item. Using the story format, you answer WHAT must be achieved, for WHOM, and WHY. Also, note that the WHAT and WHY are both performance or acceptance criteria.

Note that Scrum does not prescribe that requirements should be defined in a user story format, but it is highly recommended to use this method to record requirements.

6.11 More about user stories

User stories are called stories because the stakeholder with the requirement is supposed to tell you about the requirement not in clinical terms but rather in the form of a story. The story of what they do or what to do and why it is important for them to do it.

This last bit is actually really important! It was previously stressed that iterative delivery is not about output only, but about helping stakeholders to achieve their outcomes, often and as soon as they can. The WHY of the person's story is the outcome.

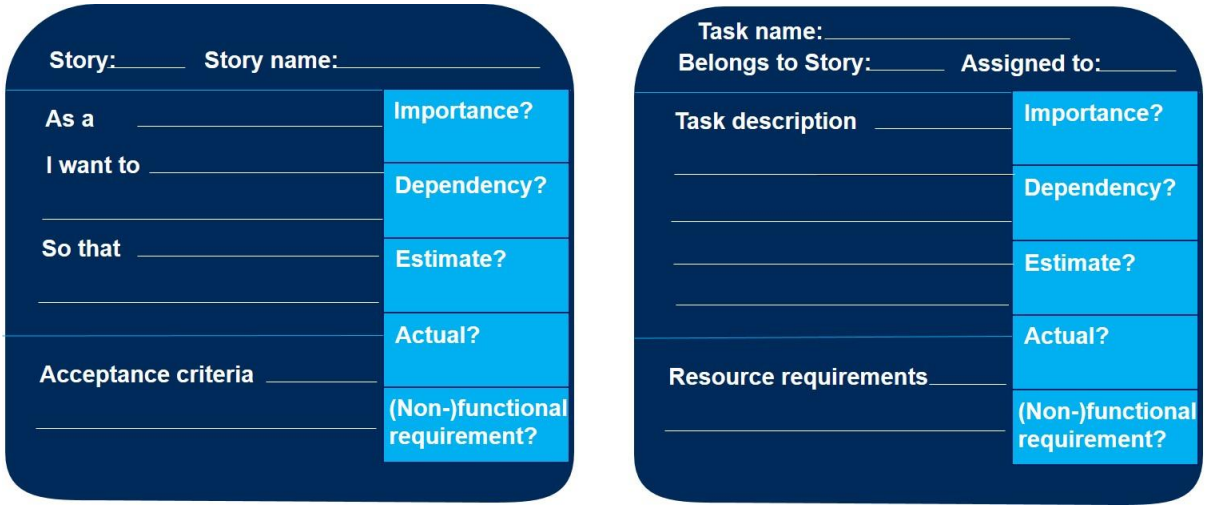
While telling a story about the how and the why it helps the person recording the story to evaluate if it makes sense. Just taking down requirements in the traditional manner does not.

If it does not make sense, the recorder can ask for clarification – they can also ask why and how.

How exactly do you do it? Why do you do it in that way? Why do you do it at all? Will it be okay to do it in another way if you achieve the same result?

Conversation leads to understanding, and understanding, in turn, leads to better knowledge on how to create value for stakeholders, most of the time users and customers.

Figure 6 An example of a Story and a Task ticket



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Using the story format is an exceptionally powerful way of recording and understanding requirements.

> As a <stakeholder ROLE>,
> I want to <the WHAT of the requirement>,
> so that <the WHY of the requirement>.

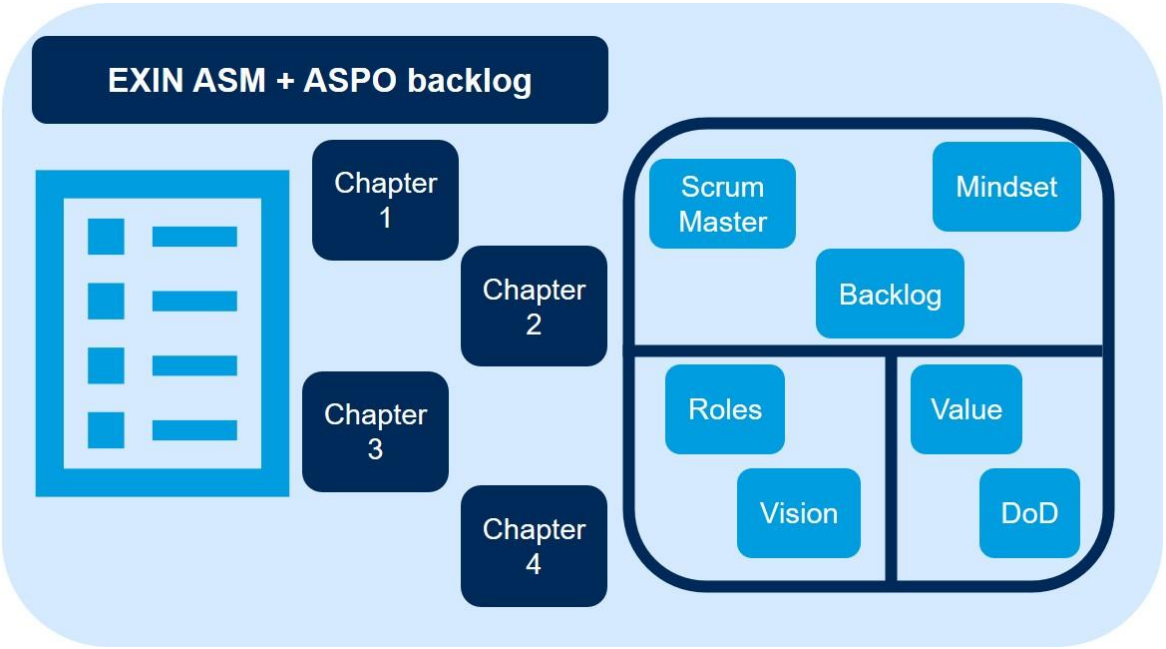
You may elect to store your requirements in a tool, but it is recommended to record it on physical cards or sticky-notes and plot the delivery progress on a Kanban board.

The Figure below is an example. It is actually the Kanban for writing this book – with the product backlog on the left side. You will see in this example that the stories are very simple and only record the outcome. That is sufficient information to enable the product (in this case, the book) to be written – in other words, it is *just enough*.

Even in this personal example, there is value in creating a backlog, defining requirements, and refining requirements along the way.

In this instance, the light blue "sticky notes" are decomposed stories belonging to one of the higher-level Epics, here represented by a larger darker blue sticky. And each of the different color has its own unique meaning, with additional information often recorded on the note. This is not the norm; it is a highly customized way of working. Each Scrum team will adapt their own way of working over the course of time.

Figure 7 The product backlog



Picture created by EXIN based on: Botha, J. (2021). *[Courseware]*. GetITright.

Remember that recording the requirement is just the beginning of a journey; requirements will be refined and refined until they are done and delivered.

6.12 User stories and the constituent task-breakdown

The following chapter will cover estimation techniques and doing task-breakdown. It is important to note here that stories can only be worked on once the work required to deliver on the story is known.

Doing task-breakdown and task estimation is the last step in the sprint planning. The Scrum Master and the Product Owner should not influence the Developers on the estimation. The Product Owner must be present during the activity to clarify any issues that might arise when estimating.

Task-breakdown is not a single event and although it must be done at a high-level when starting the sprint planning or any scaled implementation planning, it must be comprehensively done before starting work in a sprint. Just remember it may even happen again during a sprint if required, so like product backlog refinement, it reoccurs until it works and delivers results.

Theoretically, tasks can take as long as a sprint, but clearly, that would be a bad idea and not practical. So, this raises two questions:

- How long should tasks take?
- If they take longer, what do we do about it?

It is always best if tasks match the cadence of the daily scrum, as during this meeting the team typically talks about working on something for the day.

Inevitably, you will find tasks that will take longer than a day to complete, but if you look closely, you should be able to break them down into subtasks and make an estimation for each of these.

Breaking down tasks to a too low level is also unproductive – the rule is, task breakdown must be at a level where a task can easily be accepted by a team member based on their skill set and not lower.

A recommended maximum time to complete a task should be no more than 8 hours. Sometimes it is tempting to add more stories into a user story that has been selected to be worked on. It is not recommended to combine them for the next sprint as this may make the story too large to be manageable.

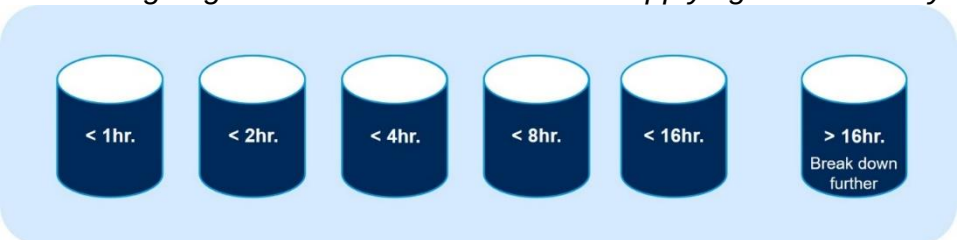
If you cannot do task breakdowns to the 8-hour level, you can consider introducing additional feedback at the point when team members report progress during daily scrum meetings. This can be a very helpful practice. In cases where work remains in a task after a day, the team member doesn't report how much of the time allocated they have spent on the task, but rather, how much of the estimated time remains to complete the task.

This practice has two major benefits:

1. **It forces people to think about work** and helps them learn how to do better task estimations in the future.
2. **It helps the Scrum Master to update progress** measurement like a burn-down chart, reflecting the time that remains to complete the iteration. Here, purists will say that a burn-down does not reflect half-completed tasks. In theory, this is correct, but the benefits of reflecting on uncompleted tasks help the team to plan better and also serves as a motivator for the team-member that is busy with the task, as no-one wants to say, "I have not achieved anything today".

When performing planning and task-breakdowns, you can use the following 'buckets' to estimate task durations.

Figure 8 Assigning estimated effort to activities applying the bucket system



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

6.13 Creating and maintaining the product backlog and roadmap

In the previous section, we talked about requirements gathering and the need to first start with coarse-grained (high-level) requirements.

The Scrum Guide describes the product backlog as an ordered or sized list of all known product requirements. It is the single source of any and all changes to be made to the product to eventually meet these requirements that should all support the product goal and organizational objectives.

Note the word "everything". It is important to make sure that you also include non-functional requirements in your backlog. It may even be a good idea to include key issues that still need to be fixed, discovered to be non-optimal during sprint reviews or retrospectives.

You can start a product backlog with as few as four and five coarse-grained user stories (Epics), and then decompose these in order of priority, with each iteration done. A good rule of thumb is to not have more than 100 items in your backlog – with a maximum of about 300.

Anything more than this will make the product backlog refinement virtually impossible and would make managing the backlog extremely difficult.

It is advisable to stick to the 100-item rule, and as soon as you see this number going up, see which items can be dropped from the backlog. Go through the exercise again to combine detailed low-priority items into Epics, as described above when talking about working with pre-existing requirements gathered traditionally.

If you start with your four or five high-level requirements, it is also easy to create a product roadmap – simply assign an order of estimated execution to each, and you are done.

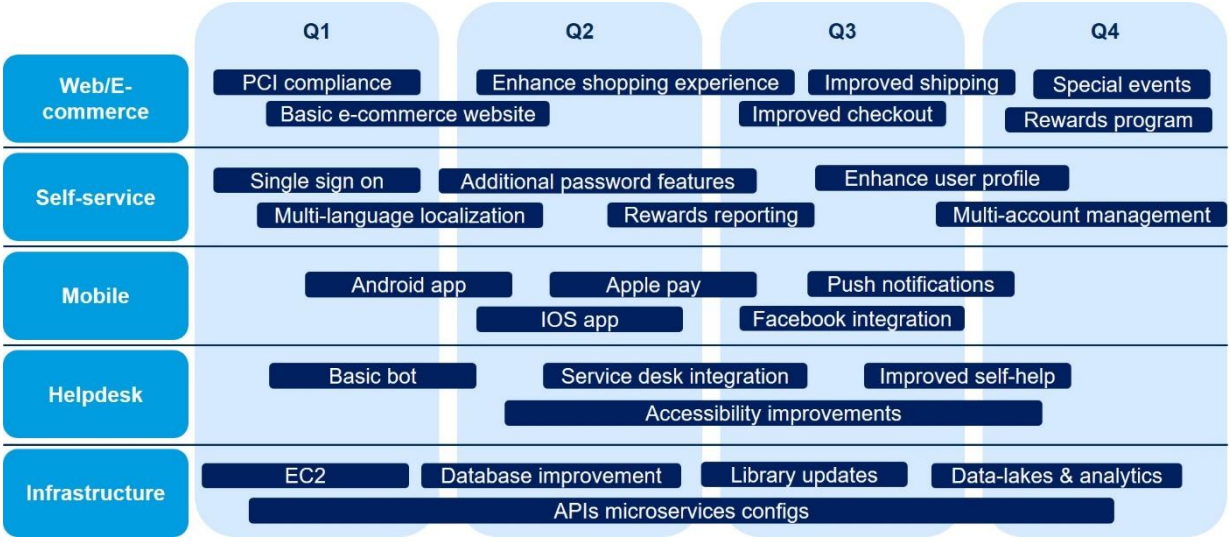
Most of the time, however, it is not as simple as that.

To create a roadmap, it would require that you group even high-level requirements into themes and use these as a guideline to create a roadmap. Generally, themes can be described as a group of requirements that relate to products or product categories, business use (including supported processes), or functionality.

The reality, however, is that themes are not the perfect way of creating roadmaps in complex projects, especially when products, product categories, or business use are used to create themes. It is easier and more effective to create themes based on clusters of related functionalities.

This, however, is also not always ideal as the purpose of a roadmap is to give broader stakeholders an idea of how value would be created. Most of the stakeholders would not be able to understand a roadmap based on themes if they are too technical; it only makes sense if it relates to things they do or want to do – therefore requirements.

Figure 9 An example of a product roadmap



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

When defining themes, be very clear about why you are doing it and also how creating themes may help you or hinder you. Always ask yourself: Will the users and customers understand what we mean by the theme? Will they understand which features belongs to which theme?

There is no simple answer on how to create a product roadmap. Let's explore some of the reasons why.

A product roadmap is used to share the vision, direction, and delivery sequence of a product. It is not a committed timeline like a traditional project plan or the related Gantt chart.

The intent is to inform and update stakeholders on what is happening and give them a sense of comfort.

Product roadmaps are therefore also fluid, and they change as what is seen as business-critical evolves and as Scrum teams start working on items and identifying dependencies.

Product roadmaps are also often used as an in-project resource or visual communication tool. Here, in addition to the mentioned items communicated, the team also wants to see the progress of a product over time. If used like this, it then becomes an action plan and a way of reporting the progress.

Do not use the same roadmap for both audiences. If you use the team view of the roadmap for other stakeholders, especially management, you turn the roadmap into a commitment to customers. This is a bad idea because in three months the roadmap may look quite different.

Note that dimensions used by Scrum teams as performance indicators must never be used as a progress or performance view for general stakeholders.

6.14 What criteria should be used for ordering items in the backlog?

Steven Covey said in 'The Seven Habits of Highly Effective People' that priority is the product of urgent and important, but unlike the way that most people handle these two terms, priority should be assigned as follows:

1. Important and urgent
2. Important
3. Urgent
4. Unimportant and not urgent

But what do these terms mean?

- **Important** means that the business will suffer significant loss of value, or there will be significant consequences if it is not done and done quickly.
- **Urgent** means that there is someone who has a vested interest in getting it done, for example, a salesperson made a promise that a feature will be in the next release.

One can say that the two criteria listed are value and risk. Value and risk are, therefore, the two primary determinants of importance, and of urgency to some extent.

Clearly, it is extremely important to deliver on promises made to customers, but if it means by doing something first the organization will suffer loss, it is a no-brainer to decide which to do first. Value and risk are more important than short-term promises. It is better to work with stakeholders to determine the value of things to be done than to make ‘cheap’ short-term promises that may be difficult to keep.

If too many items are seen as important, it constitutes a significant business risk – because, in effect, it means that nothing is important. Use this analogy to help stakeholders to understand how determining the importance and the order of items in a product backlog should be done.

Another way to consider is using the **Pareto rule** – only 20% of items in the product backlog will be Important. Ask stakeholders to decide which 20% of the items in the product backlog it is.

Although Product Owners ultimately assign the order of items in the backlog, it is best to involve a broad range of stakeholders to understand and decide on backlog item order.

Use the assigned relative importance of items, based on agreed criteria, to order items in the backlog. What should be done next should be at the top of the product backlog.

Also, ensure that top-level items are suitably deconstructed into fine-grained requirements including accurate effort estimations associated with the items.

Remember, these are the items that are most likely to be in the sprint backlog of the next sprint.

You may find that during the decomposition effort, some of the fine-grained stories are clearly important and others are not. Re-order the product backlog in such a way that only items that should be done in the next few sprints remain at the top of the product backlog. The other items may be placed lower in the backlog even though they are fine-grained, or stakeholders may decide to delete them entirely from the backlog.

It may also be helpful to consider if there are other similar requirements in the backlog that may largely or fully enable the same outcomes so that if this is the case, it is easy to motivate the removal of duplicate items from the backlog. If you remove items on this basis, make sure that dependencies and requirements are mapped and understood.

Don't spend too much time deciding the order of lower important items; it really does not matter until it becomes clear that they have now become more important.

Essential items should be well understood and documented. If needed, use separate documentation in support of backlog items. Often this is done by going back to stakeholders and asking more questions or by involving stakeholders in the decomposition of Epics.

If some of the very fine-grained items are difficult to order, you may decide to roll them up into a single story or item that is easier to understand and allocate a position in the backlog. This practice helps because it is easier to understand dependencies. The Scrum team can break it down again during sprint planning.

Re-prioritization must take place every time the product backlog is refined. It is the easiest way to keep the backlog clean and relevant.

You will also notice that soon after the first results of the iteration are made available for use, "new" requirements will start coming in. Don't assume these are new; many of them already exist as part of Epics or higher-level stories.

You may need to re-order backlog items if demand is high and these items become more important.

Non-functional requirements are the exception as they often constitute higher dependencies and therefore require a higher place in the product backlog. Once identified, they should be decomposed immediately and listed as fine-grained requirements from the start. Risk is also a good indicator of order.

It is better to deal with risk and uncertainty as soon as possible to avoid extra work later. Classify items with a high level of risk and uncertainty as high and move them up as close to the top as you can.

This is early experimentation to validate assumptions, which by nature is uncertain and often risky. By doing that, you conform to one of the principles of Scrum – using an empirical approach.

Unaddressed risk is a major millstone around the team's neck in Scrum, and it will drag you down later. If anything, that was already done depends on the item that

may ultimately prove to be unsustainable, untrue, or unusable, it means that all the associated efforts are wasted.

When listing items in the backlog, you may choose to already group items with dependencies, or items that are needed to deliver a "done" that is usable by users. A Scrum Scaling method (Nexus) will be covered later that specifically uses this approach.

Just be careful not to assume that all of the related stories are necessary for the output of a sprint. Often a sprint can deliver some core elements and exclude one or two features and still be perfectly usable and deliver business value.

When talking about sprint planning, you may remember that it was advised to choose Must, Should, and Could features, as Should and Could feature may not be a necessary part of the definition of done (DoD), and are therefore used to create a ‘buffer’ in a sprint.

This approach is often used when introducing Scrum, as it is of course undesirable for sprints to fail. This could lead to a situation whereby you need to publicly deal with the traditional means used to deal with failure in the organization (punitive actions).

Remember, to unlock early value for stakeholders is the aim. Always try to deliver real value as soon as you can.

6.15 Communication with stakeholders

One of the biggest challenges for any project-like environment is knowing what is seen as valuable and knowing what creates value.

Unfortunately, value is much like beauty – it is in the eye of the beholder. What is valuable for one person may not be seen as valuable for another.

It is also important to note that stakeholders often describe something as valuable to them, but on further investigation, it is found that not everything is equally valuable. Here the MoSCoW technique helps to steer the conversation and helps stakeholders to differentiate between what is absolutely non-negotiable (a Must Have), what is important and greatly facilitates efficiency or productivity (a Should Have), what is a nice to have as it would make things better or easier (a Could Have) and what should be a future consideration but is not currently critical (Wouldn’t Have Now).

Time is a very important part of the discussion when designing and building new products and services, as some features form the core of a product or service and others are enabling or enhancing features. It is self-evident that the core should be built first, and Scrum teams should then focus on what is the next most valuable thing to do.

Building new products and services using Agile Scrum is very similar to the Lean Startup approach described by Eric Ries. It is an iterative process and one must quickly learn what stakeholders think of the core of the product or service being delivered.

Scrum teams can define a **minimum viable product (MVP)** and validate with stakeholders if the MVP is something they can use and will unlock value for them. If so, all the stories that make up the MVP should be fine-grained and at the top of the product backlog. If you do this, you may find that everything at the top of the backlog are Must Have items. But if this is true, how will MoSCoW be helpful?

Use the just-as-well principle when the Scrum team refines the product backlog and look for work that can easily and effectively be done with the Must Have work, even though it may be a Should or a Could.

Warning: It is recommended to not use this principle when discussing features with other stakeholders, as you will end up with unrealistic expectations. Only use it in the sprint (or Nexus) team. And only do it if you want to use the MoSCoW technique.

You will find as you break down high-level user stories (called epics) that there will be natural clusters of features that automatically fall into the Must, Should, and Could categories.

Often MVPs can be tested and used, but it would be very difficult to sell these on the market because they are still not complete enough to entice customers to use them. Developing commercially offered products and services needs more.

The alternative is developing **minimal marketable products (MMP)**. The idea of an MMP is based on the Lean principle that less is more. The MMP includes only the smallest possible feature set that addresses the needs of the initial users – they would be willing to pay money for it. The MMP is a way to reduce time-to-market, by offering a stripped-down version of the product or service, and over time add additional features to make it a more complete and feature-rich product.

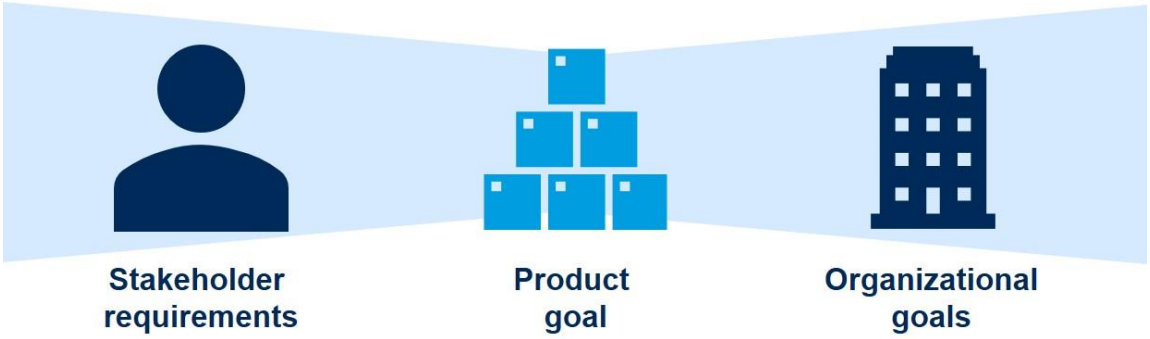
6.16 Defining a product goal

The product goal is a summary of the business objectives the product supports, and all the stakeholder requirements, evaluated and ordered in a product backlog. Only with a complete picture of the product, the requirements of customers and other key stakeholders, and how it supports organizational goals, can one define an effective product goal.

Remember, for all Scrum teams executing against a product backlog, the product goal is a representation of the True North. Every sprint goal must align with, and be a stepping-stone on, the journey to realize the product goal.

Product goals don’t just emerge by themselves as they are superordinate to requirements, so they need to be specifically developed.

Figure 10 What is a product goal?



Picture created by EXIN based on: Botha, J. (2021). *[Courseware].* GetITright.

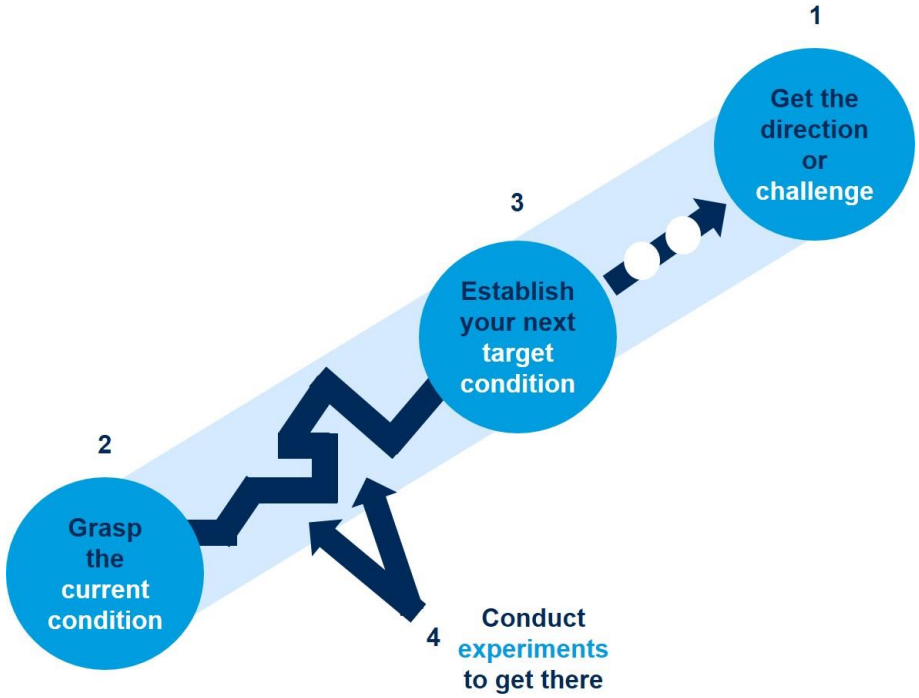
One useful technique is the **Toyota Improvement Kata** approach, to clarify and refine not only the product goal but also the product backlog, especially at the beginning.

Toyota Improvement Kata, like Agile, is a scientific and empirical approach to understand problems, goals, and objectives. Traditionally, it is not used to clarify goals and objectives; however, it can be argued that a badly defined goal or objective is a problem.

Toyota Improvement Kata generally follows these four steps:

1) **Get the direction or challenge**: Understand the sense of direction, the larger, likely time-distant vision, and be clear on what that is.
2) **Grasp the current condition**: Examine where you are now and be able to define the current situation factually and clearly.
3) **Establish the next target condition**: Determine a good next step goal that will stretch the limits of current knowledge and capabilities, move towards the challenge, and will be accomplished by a certain date.
4) **Conduct experiments**: Experiment methodically and scientifically to get to the next target condition.

Figure 11 The Toyota Improvement Kata steps



Picture created by EXIN based on: Rother, M. (2018). *The Toyota Kata Practice Guide: Practicing Scientific Thinking Skills for Superior Results in 20 Minutes a Day.* Mcgraw-Hill Education.

You can use familiar techniques like stories to define your product goal. It is, however, best if you have had ample input from stakeholders. You need first to understand organizational goals, as well as the principles and purpose of the organization that drives the primary, normally longer-term, organizational goals.

You may start with something like this.

- **Define superordinate organizational goals**: In <x years' time> <your organization> will be/achieve/have <the thing you aspire to>.
- **To achieve the set goal**: We need to <the things that you need to do or get done, defined measurably – by when>.
- Then **evaluate where you are now** so that you can identify the gap that must be closed. You may choose to close the gap step by step, for example
    - "We will <what you will do> by <when>".
    - If not too ambitious apply the same approach for the overall set organizational goal.
- **Actively managing the product backlog** represents the experiments conducted in the kata.

Often the way and order in which we do things are influenced by the requirements and priorities of stakeholders. Their needs must be considered when defining the high-level steps to get to the target condition.

Purists would say that Toyota's Improvement Kata is not to be used in this way. However, this has been proven to work effectively. A lot of resources and guidance

are available for the Toyota Improvement Kata approach, and this makes it a suitable method to use as a possible way of defining good product goals.

Much like Agile, the Toyota Improvement Kata further relies on communication, teamwork, and collaboration.

6.17 Gathering requirements

The primary way of gathering requirements in Agile is by recording user stories.

The user story format can gather coarse-grained and fine-grained requirements by defining the expected output and outcomes of a requirement in a single statement.

Stories are therefore often gathered when a product and product backlog are first defined and at this stage these stories are coarse-grained and large, known as epics.

As work progresses towards the delivery of requirements, user stories are broken down into finer-grained user stories. These fine-grained user stories can, in turn, be broken down into tasks, to easily be worked on as an increment.

Use the INVEST[6] method when defining User Stories:

- **Independent**: Each user story should stand on its own, independent from other stories.
- **Negotiable**: Stories are not contracts, rather opportunities for negotiation and change.
- **Valuable**: Every story should add value for users and stakeholders.
- **Estimable**: Every story's time and budget costs should be calculable, based on domain and technical knowledge.
- **Small (or simple)**: User stories should be small enough to estimate and implement simply.
- **Testable**: Make sure you can test the user story through criteria the story itself explains.

People often find this way of requirements gathering and the constant refining a bit chaotic and, in these cases, it may serve the organization well to supplement user stories with other information and documents.

Additional information is quite often either technical or relating to non-functional requirements and may include compliance or process standards. In fact, it is recommended that all technical and non-functional requirements should also be defined in a story format where possible.

Because requirements are constantly refined, it is important to involve stakeholders regularly in conversations or even in refinement activities.

---

[6] Bigelow, S. J. (2020). *7 techniques for better Agile requirements gathering.* https://searchsoftwarequality.techtarget.com/tip/7-techniques-for-better-Agile-requirements-gathering

Also involve other stakeholders, specifically users, when requirements are not clear enough to act upon. Stakeholders are the experts when it comes to requirements, and they could share ideas, make suggestions, and even help to document requirements in each iteration.

Constantly refine the product backlog and update requirements as time goes on. Requirements are not static and therefore have to change over time. Involving customers and users from time to time in product backlog refinement helps to identify new requirements as well as changing business needs and priorities.

As Developers start working on building increments in a sprint, they often discover information or dependencies previously un-known. This information should be used to refine both the product and sprint backlogs.

Refining the sprint backlog means re-planning work in the current sprint, but often the newly discovered facts have consequences that cannot be dealt with in a sprint, in which case product backlog refinement must be done. Remember that product backlog refinement is an ongoing activity and not an event!

7 Agile planning and estimation

In this chapter, several approaches to estimation and planning will be discussed, but at a high level, two types of estimation and planning techniques are used in Scrum: Velocity-driven and commitment-driven. Proponents of both are very passionate about their choice.

Velocity-driven estimation and sprint planning attempts to assign work to a sprint that is equal to the work they completed in the previous sprint (story points and T-shirt sizes, for instance).

The typical steps when using velocity-based sprint planning are:

- **Calculate the team's velocity**. It may have changed since the last time it was done – especially at the beginning of the Scrum journey.
- Based on the product goal – **define a sprint goal**.
- **Select stories** from the top of the product backlog equal to the team's velocity.
- **Split stories into tasks** and determine how long each will take to complete.

When doing commitment-driven sprint planning, the team must commit to an item before adding it to the sprint backlog. Once the total of items in the sprint backlog equals the timebox's time, the team stops and will not add any further stories to the backlog.

Because the team here naturally translated work to how long it will take, you will often find that teams prefer to use the ideal days or hours estimation technique.

A sprint planning session will usually run like this.

- **Define the sprint goal** based on the product goal
- **Select a product backlog item** from the top of the product backlog
- **Split the story into tasks** and determine how long it will take to complete
- **Get the team's commitment** that completing the story is feasible and add the story to the sprint backlog
- See how much time is still left in the sprint timebox
- **Repeat** the process until you cannot add any more items to the sprint backlog because the sprint timebox is full

As mentioned before, which approach to take and which tools work best to do different types of estimation is a very divisive topic. The major techniques will be described here and it is up to your team to decide which works best for you.

Why plan if we say that Agile is not about up-front and detailed plans?

Planning at the beginning of the Agile cycle is for one purpose only – you need to know enough to be able to add the product to your portfolio, to effectively perform portfolio and product management.

The level of required planning is such that it is possible to make proper decisions for the next six months to a year, and at least have an idea of what could happen beyond that timeframe and its required investments. You want to be able to allocate funds and resources to achieve goals and objectives as unveiled in the enterprise portfolio.

So the answer to upfront planning is that it should be ‘just enough’ to be able to make decisions on business priorities and to be able to allocate funding and other resources to the initiatives that will have the most significant positive impact on the business.

Initial planning is always flawed; we know that from Waterfall projects – but it does provide insights and it becomes a catalyst for discussion, discovery, and tactical plans of the business as a whole.

One of the main objections against an Agile approach and specifically the use of Scrum is that the organization does not have this high-level view to make proper strategic and tactical decisions. The objections are partly correct if you look at Agile and Scrum as a purist. However, every Agile method these days has a way of overcoming and correcting these initial objections.

When covering scaling later in this book, some of the current methods considered as complementary to the Scrum approach will also be mentioned.

However, the approach followed by all the Agile methods is to do just enough planning at any level, to be able to answer the questions answered at that level of planning and/or execution.

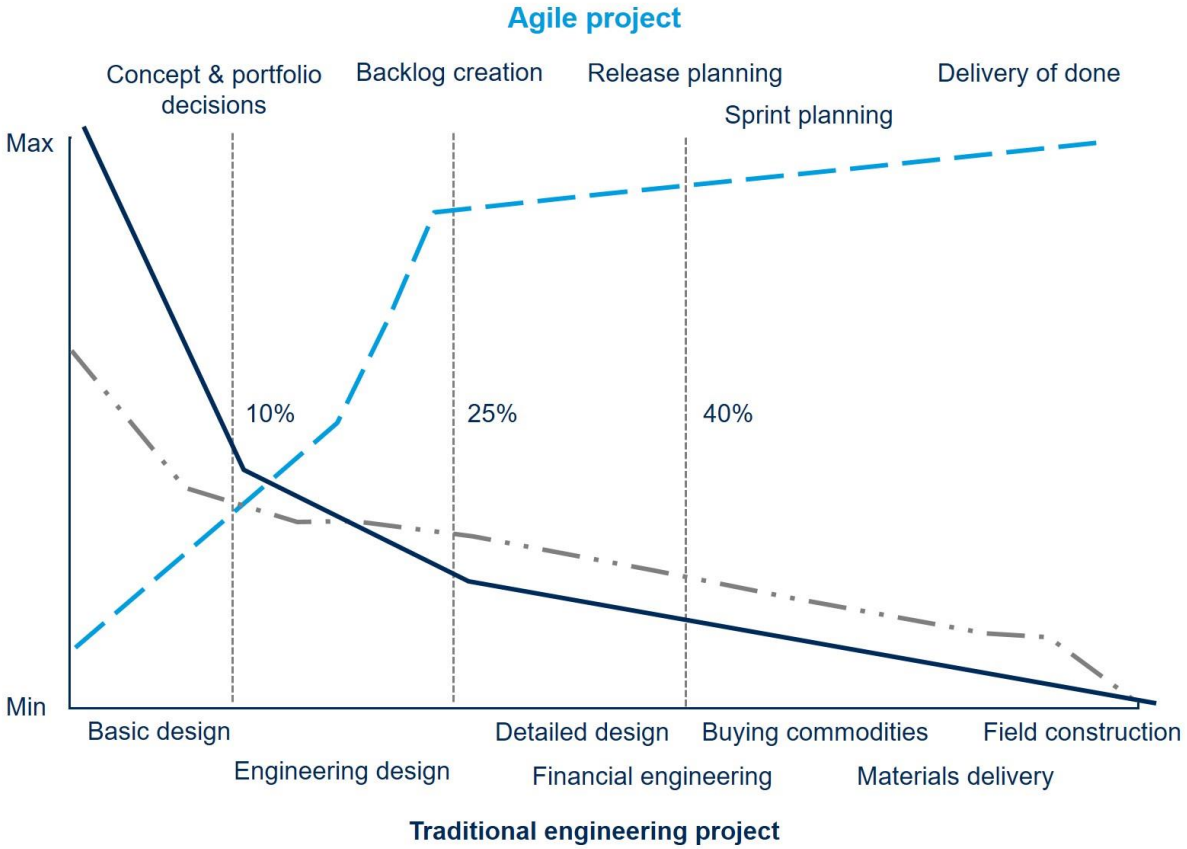
So, make peace upfront if your initial plans and estimates turn out to be wrong – this is true for Waterfall projects as well as Agile projects. Initial plans are a stake in the ground, a baseline; a place from which you can learn more, know more, and become better the closer you get to the delivery of the real value to customers.

Bruce Martin, in his 1980's article *The goal: to improve credibility in the reporting of engineering progress*, in the PMI's Project Management Quarterly, drew a chart that is still valid today. This article did not talk about Agile or Scrum, but Waterfall projects – once again proving that the basis for most of the objections against Scrum are flawed and based on a very erroneous assumption.

The assumption is that detailed, Waterfall planning is better than high-level Agile planning at the start of a project (see the adapted picture below).

So why bother you may ask? The more frequently you do high-level estimation and refine your portfolio, the better you will get at it over time. This is one of the benefits of using Agile estimation techniques – because you often compare a new project and deliverables with a known, older project. The more you do it, the more you learn and the better you will get at it.

Figure 12 The reliability of detailed planning



Picture created by EXIN based on: Martin, B. A. (1980). *The goal: to improve credibility in the reporting of engineering progress.* Project Management Quarterly, 11(2), 14–22.

The reliability of detailed planning is questionable early in the process; the closer we move to work being done, the more reliable the planning and estimation and the less need for cost and schedule controls

The process of learning and improving means that creating and maintaining the product backlog and Roadmap will also become more accurate over time. As a consequence, the level of confidence the organization has in the view of the future it presents will increase.

Creating this high-level view of all projects including estimates provides the inputs into portfolio management, which in turn helps us to focus our efforts on what is vital for the organization and relate strategic initiatives to projects that will support these initiatives.

It helps answer the questions – what should we build, when, and does it matter?

Planning also helps direct sub-projects, like marketing, product branding, sales, and advertising, and again often kick-off training and upskilling initiatives in preparation for project delivery.

A good plan has enough detail to do planning at the level where you want to use the plan. That means that Agile planning is iterative, with indicative typical planning horizons where the use of the planning done is imminent.

7.1 What makes Agile planning different?

Practitioners should be careful not to approach planning traditionally. Conventional project planning is linear and activity-based.

In Agile, the delivery of value does not happen in a linear fashion, but rather by constant reprioritization of which features will unlock the most value for the customer.

In addition to the drawback of linear planning and execution, by having stop-start performance whenever a delay impacts the critical path, the Waterfall approach assumes that perfect up-front knowledge of the future is available – which is not valid.

So, in Waterfall projects, every delay in the delivery of something on the critical path means a delay in the project delivering value to customers.

Another problem is that according to Parkinson's law, project teams will fill the "free time" or "wait time" caused by delays. This phenomenon most probably occurs because the primary measure of projects of this nature is that you are at least doing something! Downstream idle time is consequently not utilized for non-dependent tasks because they are not the next thing to do on the list. As a result, the impact of the delay is almost always passed downstream.

Remember the four things Agile teams value:

Value 1.	**individuals and interactions**	OVER	*processes and tools.*
Value 2.	**working software**	OVER	*comprehensive documentation.*
Value 3.	**customer collaboration**	OVER	*contract negotiation.*
Value 4.	**responding to change**	OVER	*following a plan.*

These values should be part of everything you do, including the way you plan.

Please note that planning takes place in Agile just before you do the work. It is the best time to do it, as much would have been learned since the initial plan and the creation of the product backlog. The iterative nature of Agile and Scrum means that the lessons learned will also help you to ask the correct questions and come up with the best possible plan.

Work as teams, with different teams at different levels as highlighted by the planning horizons above; and include a broad segment of stakeholders as it makes sense at every level of planning. Converse, ask, enquire, and come up with something that best represents the wisdom and understanding of the parties involved.

Don't try and define what is not known; leave that for later. Every iteration of the plan will emerge and become more detailed as you get closer to work being done.

Plans should be light – using the Lean principle of ‘just enough’ to be able to make a good decision. Not more, not perfect, not comprehensive – just enough.

When planning, it is recommended to use participative and visual techniques. If the wall is not covered with sticky notes, and the boards full of scribbles, it may indicate that you are not trying hard enough.

Besides, make sure that planning is properly transferable to the next level of participants. Don't throw something over the wall and hope for the best. If the party that must do the next activity is not in the room, get them in the room, explain, discuss, debate, and reach a consensus or agreement.

Agile planning is based on the value created by features, and assigning priority based on the creation of value. Detailed planning is not done upfront in Agile; activities do not even exist in high-level Agile plans.

The approach has four significant benefits:

1. There is **no activity-based planning** – thus avoiding the pitfalls of the approach.
2. The nature of the work done is left in the hands of the specialist that will execute the delivery of value; they are by far the best equipped to decide what the best way to do the work is.
3. Because you don't have to worry about activities, you can **elevate the view of planning** to think about the best possible sequence to unlock value for customers and the organization.
4. Because you don't have a detailed view of all the activities, you can **accept that you have an imperfect idea of the future** and continuously, at each step of the journey, attempt to have a better understanding of what must be done and reduce uncertainty.

The last point above also means that you need to communicate with stakeholders that the portfolio or roadmap view of the projects is not a commitment but an indication of planned progress and that everything may change at the drop of a hat when organizational or customer circumstances or priorities change.

You may ask if this approach will not create a lot of unease with customers; customers want certainty.

The honest answer is *yes*. But it is: *yes, initially*.

Why?

Because customers must remain involved in planning throughout the cycle of delivery of value, and it is the Product Owner's job to keep them on-board.

It was previously mentioned that Scrum teams are involved in backlog refinement and the reprioritization of items in the product backlog. The Product Owner, as the representative of the business, also must have conversations with business counterparts about business priorities when doing backlog refinement.

When discussing the tasks of the Scrum team, it was stated that they need to plan the work for the iteration or sprint. It was also noted that the Product Owner must plan and order the work in the product backlog (product level), to best plan and execute releases and iterations. Customers should update the Product Owner if any

of their priorities change, and the Product Owner should ensure that there are no changed customer priorities.

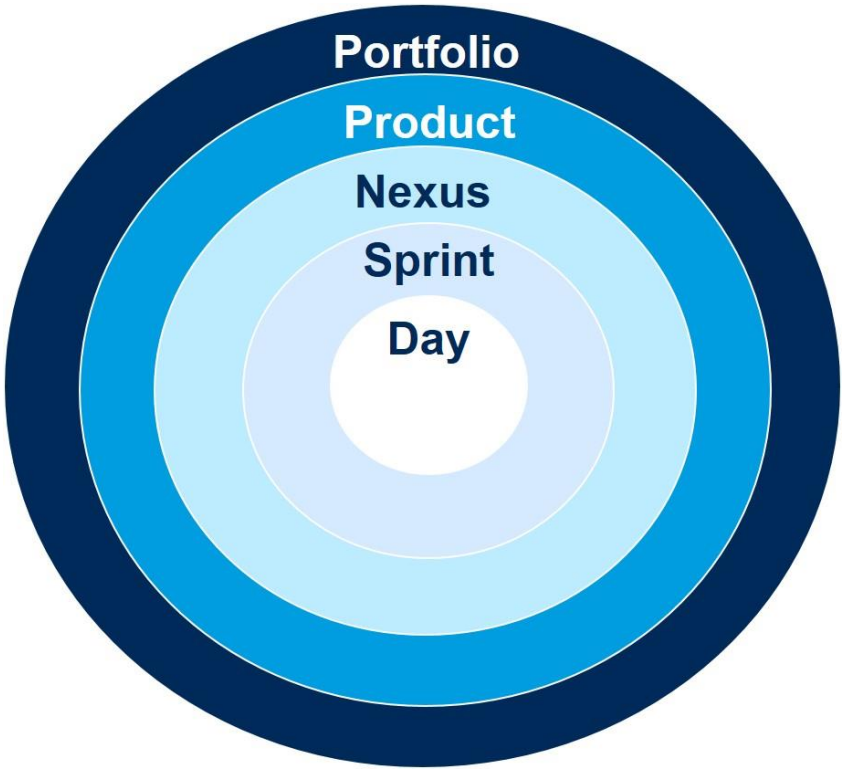
Inclusion of a project in the organization's portfolio requires an estimate of the amount of effort, budget, and resources needed to complete the project. This information is also used to assign a priority to the initiative; viewed in the context of other initiatives in the organization.

This represents at least four levels of estimation and planning required throughout the lifecycle of a project. In many organizations and with complex projects, this may even be more – four, five, or even eight levels of planning and estimation. The key point is that you should only do the level of planning and estimation at each level which is necessary to answer the specifically asked questions.

The levels of refining planning and estimation usually are: What do we need to know to determine if a project should be added to the portfolio, to create a backlog, to refine a product backlog, to plan scaled implementations or releases, or to plan and execute a sprint?

Planning and estimation also occur within a sprint: What are we going to do in this sprint and what are we going to do today?

Figure 13 Layers of planning become more detailed as we get closer to work being done. Agile uses the principle of "just enough" planning at every layer



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Planning and estimation on a product, release, and sprint level have previously been addressed, but what about planning and estimation from a portfolio perspective?

To answer this question, you need to know what the questions are that need to be asked when creating and updating the organizational portfolio.

The purpose of an organizational portfolio is to manage the attainment of the organization's goals and objectives. The question is essentially: What (initiatives) need to be executed to reach our goals and objectives? Initiatives identified then need to be evaluated and ordered.

- What do we need to do first?
- What will have the most significant impact?

Unfortunately, these questions are not that simple to answer because you are always faced with constraints, usually resources and budget.

The question then becomes:

- What combination of things that we must do, will have the most significant impact?

To answer this question, and perform estimation and planning at this level, you need to have a good idea of the value being created by the initiative, how long it will take to get it done, and the cost and used resources. Secondarily you also need to look at interdependencies.

It is understood that the answer to this question is an educated guess at best, but making comparisons to previous initiatives and known work means that an estimate at this level can very quickly get to 80% plus levels. For portfolio planning, an estimate of 80% is good enough!

Did you notice that answering these questions not only deals with the nature of the work to be done? The velocity of the team involved will also have an impact!

If you don't know who will eventually do the work, it is recommended to work on the average velocity across teams.

7.2 Who gets involved in planning?

That is a very good question, and the answer depends on what level of planning is referred to.

For portfolio planning, one would expect Product Owners (managers) together with their business sponsors (the customer) and senior and executive managers to be involved in the discussion.

Product backlog creation should include Product Owners, business sponsors (if possible), and initially a broad range of user representatives. The audience must have a good enough understanding of the stories that go into the backlog and their relative order.

The choice of the word order is deliberate; product backlogs do not only use importance and priority to determine the order in which the work is done. Other factors like risk, dependencies, and many others may also play a role.

Product backlog refinements can include the same audience, but the reality is that a broad range of users is most probably not necessary. In addition to these roles, involve Developers currently working on the backlog as well as Scrum Master(s). Input in terms of lessons learned by Scrum teams is vital.

A similar audience is appropriate for scaled implementations and sprint planning. The involvement of the customer and users becomes less as planning progresses towards a specific sprint. That does not mean that they are not involved at all. Scrum teams with a high level of user and customer involvement throughout their sprints do much better than those without. You don’t have to wait for a sprint review to get user and customer feedback.

Product Owners may need to involve multiple teams of Developers and Scrum Masters in product backlog definition, refinement, and planning activities.

Planning becomes progressively more technical the closer the planning gets to an actual sprint. Here you need to understand how different teams may work on the same backlog, the dependencies and the consequences of these, how to sequence work and activities, what communication is required, and eventually in a sprint who is going to do what. This is why a cohesive scaled approach is so important in larger and more complex environments.

Although it is not a good idea to view all types of planning as timeboxed events, it is often executed as a timeboxed event.

Both scaled implementations and sprints should have goals, objectives, and success or acceptance criteria for deliveries to achieve; however, it should be a bit more fluid at a scaled implementation level. A sprint, however, has a clearly defined sprint goal and a definition of done (DoD) for each increment, which may include further and more detailed associated acceptance criteria.

7.3 Estimation techniques

There is clearly a need for estimation when doing planning, but the very nature of the word already tells us something important.

Estimation is not 100% accurate. The trick is to know what level of estimation is good enough.

The two factors to consider are **accuracy** versus **effort**. As with most things in life, if different attempts are plotted, it will most probably show a typical bell curve. One way of getting better estimates is to have more meaningful data so that teams will naturally get better at it the more they do it.

The more team members actively participate and gain experience at estimating, the better the team will collectively get at doing good estimates. You will notice that all Agile estimation techniques rely on the shared input of the team.

Suppose a story seems to be challenging to estimate. In that case, the team should consider splitting the story into smaller stories that prove easier to estimate and then assign the sum of the smaller stories as an estimate for the larger story. This technique is called disaggregation and aggregation.

But what if the team has no idea?
You may elect to ask an expert or several experts; however, this will not result in high-fidelity estimates because experts outside the team have no insight into the skills and capabilities of the team. It is, however, better than guessing.

Teams can also attempt to compare the current work with the work done before. Working with ideal days or story points is actually just that. But if there is no project to compare to, it leaves getting an outside opinion as to the only viable option.

7.3.1 Re-estimation

When using story points and ideal days, it is implied that once work-breakdown occurs, re-estimation must be done. Estimation using story points and ideal days are used to simplify the estimation of a feature or a story, and all its associated complexity. It is not necessary to do work-breakdown to the level of the tasks in the feature or story.

By implication, it is evident that once task-breakdown happens in sprint planning, re-estimation is in order as you can now aggregate the sum of the associated tasks back to the feature or story.

You may argue that that is superfluous as you are no longer concerned about the story or feature at this level. However, aggregation proves meaningful as the team can learn from the delta between the original estimation and the now more detailed estimation, and that will significantly benefit the team's ability to estimate work of this nature in the future.

One of the topics of discussion in a retrospective can be cases where a significant delta was discovered between past estimates, detailed estimates, and actual time to complete the work.

This is specifically encouraged as an agenda item, considering that the team will most probably help the Product Owner with product backlog refinement immediately after the sprint completion.

Including this agenda item therefore immediately helps with improving estimation in the future.

Also, remember that partially completed stories need to be added back to the product backlog immediately after a sprint. These stories need to be re-estimated, as some of the work may have been completed, but not enough to say it is done.

In the section on sprint planning, some estimation techniques will be introduced that can also be used when doing the estimation of product backlog items.

7.4 What else is or can be done in sprint planning?

Just as a reminder, the following questions should be answered during sprint planning:

- Why is this sprint valuable?
- What can be done in this sprint?
- How will the chosen work get done?

At the end of the planning activity, the following must be clearly defined:

- The sprint goal,
- The product backlog items selected for the sprint, and
- The plan for delivering items in the sprint backlog.

Although an overview of sprint planning was given earlier, this section expands a bit on techniques that can be used during sprint planning.

7.4.1 Sprint planning

Sprint planning is a timeboxed event that lasts roughly 1 to 2 hours for every week of a sprint.

In sprint planning, the Scrum Master will coach the Developers on the estimation of features but never decide on the estimation, whilst the Scrum team agrees to complete a set of product backlog items that will bring the organization closer to achieving the product goal. This agreement is reflected in a defined sprint goal and by items added to the sprint backlog.

How do you know how much to take on?
In the beginning, you will have to experiment with the estimated time of backlog items until there is some data available regarding the rate at which the Developers can work. This is known as the team's velocity and will be covered in some more detail later. For now, let's concentrate on the timeboxed event called sprint planning.

The preparation for a sprint starts before sprint planning. The Product Owner must ensure that the product backlog is appropriately refined, decomposed, and the essential items are ordered and sized as best as possible to facilitate a productive discussion during the sprint planning session.

The entire Scrum team is involved in sprint planning. The aim is to reach an agreement between the Product Owner and the rest of the team regarding what work to include in the sprint.

Scrum practitioners often treat the sprint and the term increment as the same but according to the Scrum Guide, a sprint can include several increments. An increment is defined as a concrete stepping-stone towards the product goal.

Each **increment** is additive to all prior increments and thoroughly verified, ensuring that all increments work together. To provide value, the increment must be usable.

Multiple increments may be created within a sprint.

The sum of the increments is presented at the sprint review, thus supporting empiricism. However, an increment may be delivered to stakeholders before the end of the sprint. The sprint review should never be considered a gate to deliver value.

Work cannot be considered part of an increment unless it meets the definition of done of that increment (DoD).

This definition implies that each increment must define what 'done' is for the increment, while the sprint goal is rather the overarching deliverable.

Sprint planning starts with a conversation about the ideal, and broadly defined outcomes, of the sprint. The Product Owner will give the team an overview of the items at the top of the product backlog and remind them of the product goal to set the scene.

This sets the background for defining an initial and broadly defined objective for the sprint. The term broadly defined is deliberate, because if it is done too narrowly, the team will have no choice in what they have to do during the sprint – which contradicts the Agile principle of self-managing teams with the autonomy to decide what work they do and how they do it.

This conversation results in the defined sprint goal for this sprint, and the sprint goals should support the overall product goal.

The sprint goal is an objective set for a specific sprint, and the team can meet the goal by implementing items listed in the product backlog. Sprint goals are the result of a negotiation between the Product Owner and Developers and should be specific and measurable. Developers need to ensure that the goal is realistic, as it is what they commit to get done.

To determine the sprint goal, Roman Pichler[7] offers three questions to consider:

- **Why do we carry out the sprint?** Why is it worthwhile to run a sprint? What should be achieved?
- **How do we reach its goal?** Which artifact, validation technique, and test group are used?
- **How do we know the goal has been met?** For instance, at least three of the five users carry out the usability test successfully in less than a minute.

It is also important to answer the following questions:

- **Why is this sprint valuable?** The Product Owner proposes how the product could increase its value and utility in the current sprint. The whole Scrum team then collaborates to define a sprint goal that communicates why the sprint is valuable to stakeholders. The sprint goal must be finalized before the end of sprint planning.
- **What can be done in this sprint?** Through a discussion with the Product Owner, the Developers select items from the product backlog to include in the

[7] Overeem, B. (2016). *The 11 Advantages of Using a Sprint Goal.* https://www.scrum.org/resources/blog/11-advantages-using-sprint-goal

current sprint. The Scrum team may refine these items during this process, which increases understanding and confidence. Selecting how much can be completed within a sprint may be challenging. However, the more the Developers know about their past performance, their upcoming capacity, and their definition of done (DoD), the more confident they will be in their sprint forecasts.

- **How will the chosen work get done?** For each selected product backlog item, the Developers plan the work necessary to create an increment that meets the definition of done (DoD). This is often done by decomposing product backlog items into smaller work items of one day or less. How this is done is at the sole discretion of the Developers. No one else tells them how to turn product backlog items into increments of value.

Between these two sets of questions, a well-crafted sprint goal can be created. Everything the Scrum team does from then onward will be measured by how it contributes to the sprint goal.

The next step is to evaluate if the team can realistically complete the work needed to be done to achieve the sprint goal in the timebox defined for the sprint. Often the discussion about this sprint will start highlighting the next steps that could be in future sprints immediately following the current sprint.

This means that Product Owners may have to re-order items in the product backlog as the items in the sprint backlog should reflect the top of the list in the product backlog, and possible items for the next sprint should ideally be next. This also helps to minimize time spent on choosing items for the next sprint as the input of Developers is by then already reflected in the product backlog.

The amount of product backlog items included in the sprint depends on the capacity and the velocity of the team. When first starting with Scrum, this will be a guess as the team will not have available data to calculate the team's velocity.

Any items chosen for inclusion in the sprint must be already broken down into small, estimable, and testable items. They should be decomposed before the team adds these items to the sprint backlog.

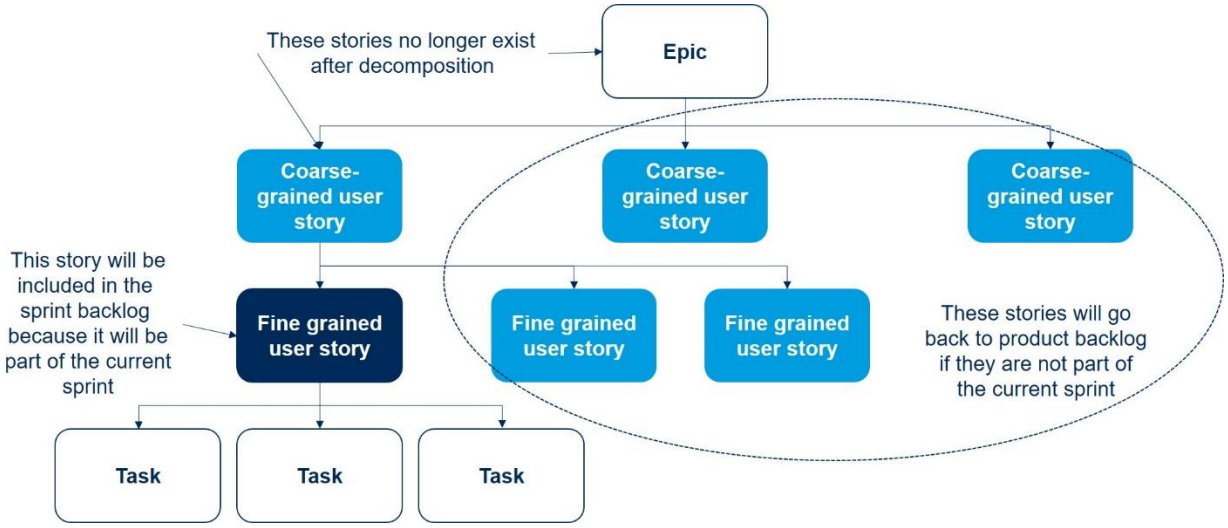
Items will be re-estimated before being added to the sprint backlog as different teams of Developers may have different levels of skills, capabilities, and velocity.

Try to keep items as small as possible, definitely not more than a few days. It will help to keep work-breakdown into simple and easily manageable tasks. It also limits work-in-progress (WiP) and makes it easier to track progress using a burn-down or burn-up chart.

Note that user stories may be decomposed multiple times and over multiple sprints. It is not necessary to decompose all the branches of a higher-level story in this sprint unless they form part of the work done in the sprint – only decompose those worked on in this sprint.

If the team is not working on a branch of a decomposed story in this sprint, they can add it back to the product backlog with an appropriate priority and estimation.

Figure 14 From Epic to Task – breaking down requirements



Picture created by EXIN based on: Botha, J. (2018). *Scrum Masters and Product Owners [Courseware].* GetITright.

Also, consider current issues in the context of the project that may affect performance during the sprint. Highlight dependencies and consider the implication of those.

Quite often, there is a reliance on a party that is not one of the Developers. Although it is best to avoid this situation at all costs, it happens. Consider how you can negate the dependencies, and if not possible, how best they can be managed.

An example of this would be a situation whereby Developers need a cloud environment to work in. Requesting this may be simple but getting it can be unpredictable. Can anything be done to make sure it won't impact the sprint's performance? If so, what should be done and can you take the appropriate steps?

Once the team reaches an agreement on the items to be included in the sprint, the team should compile a sprint backlog with the items to deliver. The sprint backlog should be updated regularly; it is recommended to perform updates once per day to keep it as up-to-date as possible.

The team then must agree on a definition of done (DoD) for each increment that forms part of the sprint, and it will be the team's commitment to work on the items that need to be completed for the increment and sprint.

The DoD is the list of requirements that must be satisfied to reach the increment's success. Some teams of Developers include the completion of all backlog items in their definition of done (DoD), and others not.

The definition of done (DoD) provides everyone with a shared understanding of what work should be completed as part of the increment. If a completed product backlog item does not meet the definition of done (DoD), it cannot be made available for use or presented at the sprint review. Instead, it must be returned to the product backlog for inclusion in future sprints.

If the definition of done (DoD) for an increment includes part of the standards of the organization, all Scrum teams must follow it as a minimum, e.g. a non-functional requirement defined as a standard security or quality requirement. If a requirement is not an organizational standard, the Scrum team must create a definition of done (DoD) appropriately for the product and product requirement.

Deliverables of Developers are required to conform to the definition of done (DoD), and if multiple Scrum teams are jointly working on an increment, they must mutually define and comply with the same definition of done (DoD).

The moment a product backlog item meets the definition of done (DoD), an increment is born.

Side note: Teams using the MoSCoW technique often choose items to include in a sprint from Must, Should, and Could Haves. Usually, 80% will be Must Have (items of high importance to get done) and Should Haves (useful and valuable but not essential) and 20% should be Could Haves. By following this approach, a buffer is created in the sprint if something unforeseen happens.

However, in a scenario where the team includes buffer items (Should and Could Haves), usually only the Must-Have stories are included in the definition of done (DoD).

The standard practice in Scrum is to only include items at the top of the product backlog in the sprint backlog. Many Scrum teams, however, have adopted the DSDM practice and if the approach works for your organization, it is your choice.

When selecting items to include in the sprint backlog, it is vital to consider dependencies and interdependencies**.** Dependencies are often technical or infrastructural in nature. If a dependency exists, also remember to include the stories it depends on in the sprint.

When is a story complete?
When it meets the definition of done (DoD) for the increment!

Stories and associated tasks should have acceptance criteria too. They are set at the start of a sprint when stories are decomposed into tasks. Activities and related criteria may change as development progresses, and when you discover more about a feature or requirement, but all work must be tested against acceptance criteria before it is made available for use.

The above description sounds very much like the XP test-driven development approach used in software development. In this case, the final test of 'done' would be to ask if the software is deployable, usable, and safe.

Done is not only meeting the individual criteria set but also means that what the team completed in the increment is ready for use and usable.

Side note: If the work being done is the development of software, the term *done* means: what is delivered can enter the continuous integration/continuous delivery (CI/DC) pipeline and the increment can be made available virtually immediately.

CI/CD are concepts used when using pre-defined tools and automated workflows to deliver software code to the live environment as quickly as possible. The overarching discipline used to get the software tested and properly deployed in as automated a manner as possible is often also called DevOps. Integrating DevOps and Scrum is a wonderful way of creating value for users and customers sooner rather than later.

Once the team has agreed on what should be included in the sprint backlog, the rest of the sprint planning should be spent doing the work-breakdown, task estimation, and acceptance of work items belonging to each user story.

The role of the Product Owner in the latter part of sprint planning is to clarify questions and elaborate on acceptance criteria. The Product Owner has no role in evaluating how the team is doing their job; their input is limited to the relative importance of backlog items.

Scrum Masters need to ensure that new issues and concerns raised during the meeting, or identified assumptions or dependencies discovered during planning, are recorded, and better understood, sooner rather than later.

7.4.2 Sizing items

Estimation techniques were referenced earlier in the section dealing with the creation and refinement of the product backlog. Similar techniques are used in breaking down stories in both the sprint and the product backlog.

Although backlog items are estimated and sized when doing product backlog refinement, items included in the sprint backlog need to be re-evaluated in the light of what is now known. Consider changes in the requirements, the environment, and the capability and capacity of the Developers when re-evaluating backlog items.

Sizing items appropriately and realistically is very important; otherwise, the team may not be able to deliver against the definition of done (DoD) at the end of the sprint.

Often items are included in the sprint backlog, but while doing task-breakdown, it is found that the estimated time or effort was inappropriate. The team can revisit the composition of the sprint backlog and negotiate with the Product Owner to put items back in the product backlog if they cannot realistically be completed during a sprint. The gap left can be filled by including other, smaller product backlog items in the sprint backlog.

Just remember that when doing this, the sprint goal and DoDs may be affected and may need to be revised. You should preferably only make changes that do not affect the sprint goal, but since you are essentially still busy with planning, inevitably the need to revise a sprint goal may arise.

Once the sprint planning is completed, do not make changes that will impact the sprint goal. There are various ways of sizing items and these methods will be briefly described here.

7.4.3 Story points

Although there is no official definition of what a story point means, in general terms a story point is a relative unit of measure, comparing work to be performed to similar work done in the past.

Story points are, therefore, relative and need to be considered together with the velocity of a Scrum team. This estimation method can be more robust against changes in the work environment.

You might say that compared to some of the work done before, the new work item will take half as long or three times as long. The unit used for comparison, however, must remain the same. It is best to choose the unit of reference as midscale or lower midscale and not as the smallest number. So if for example, you use a scale from 1 to 10, the unit of reference should ideally be 3 or 4. Even though the regular scale would be 1 to 10, you may from time to time have a story with 20 story points; this just shows that it is an epic that would need some serious decomposition.

At a product backlog level, it is much easier for Product Owners to use this technique as they are often not technically aware enough to use other methods like hours.

Variants of story points include the use of T-shirt sizes. You can use the table below as a reference for a typical 1 to 10 story-point scale:

Table 1 – T-shirt sizes compared to number of story points

T-shirt size	Number of story points
XS	1 story point
S	2 story points
M	3 story points
L	5 story points
XL	8 story points
XXL	13 story points
XXXL	21 story points

Note that the Fibonacci sequence was used here as a relative reference.

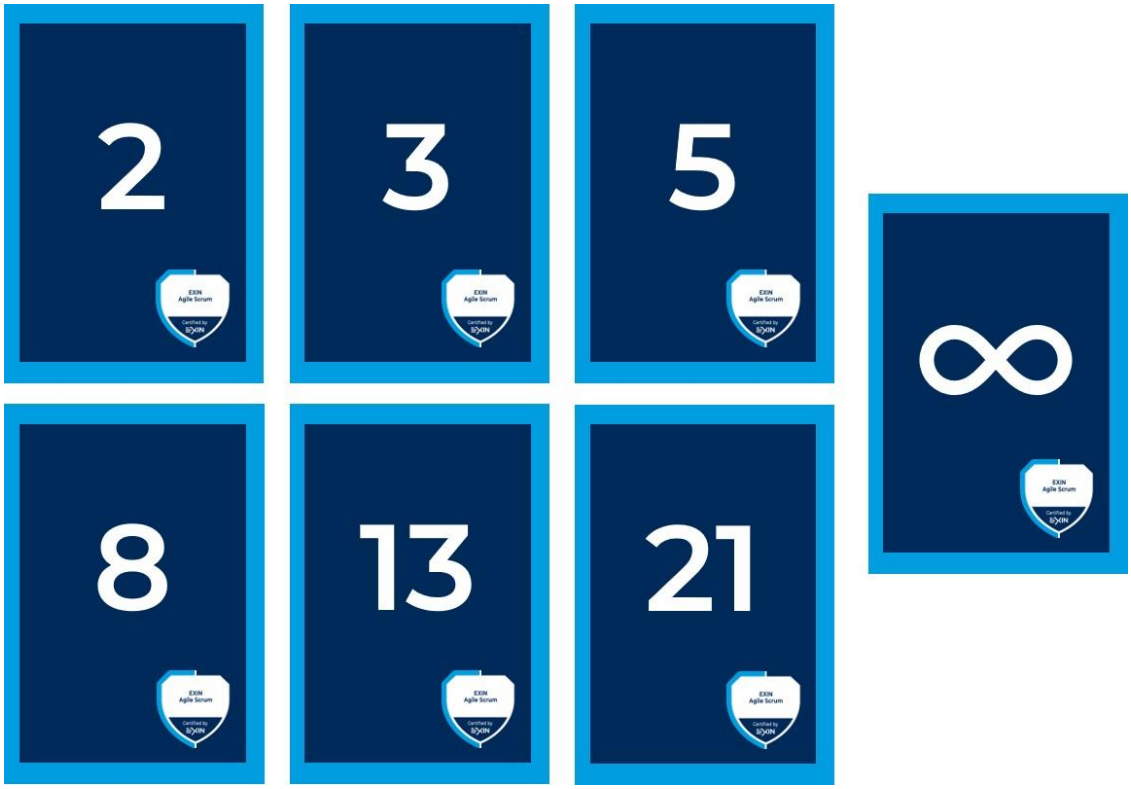
7.4.4 Planning or Scrum poker

Planning poker is actually more like playing Uno or Snap than playing poker.

Each Developer is given a deck of cards with the Fibonacci sequence. When asked for their estimate of the size of the work, each member places the card with the value that is either the same as their estimate or the next higher number, on a table.

The Scrum Master and Product Owner should not be involved in estimation here; it is about getting input from Developers. Product Owners do estimates when adding items to the product backlog; this is an opportunity for them to get better insight and input from the people doing the actual work.

Figure 15 Scrum Poker cards



Picture created by EXIN based on: Botha, J. (2021). *[Courseware].* GetITright.

The aim here is to gauge the level of disagreement between team members on what effort will be required to complete the activity or story.

If the field is closely spread, it is reasonably easy to reach consensus – often choosing the higher number to err on the side of caution.

The real value, however, is in outliers. For example, if most of the team said either 3 or 5 story points, but someone in the group said 20, two possibilities exist. Either the person with the outlier card on the table doesn't understand the scope and context of the work to be done, or they know something that the rest of the team doesn't know.

Outliers are always addressed by asking: *Why do you say that?*

Addressing the first issue is simple; by explaining the context, the person will probably decide to revise their estimate in the next round of play, and if so, play another round.

The second possibility quite often leads to a very interesting discussion and new insights that neither the Product Owner nor the other team members knew or understood. This conversation usually leads to unearthing new dependencies and risks, and often to the creation of additional stories which should be added to the backlog.

Keep playing rounds until a tight grouping of values is all that is on the table.

As stated before, if there are ten people around the table and eight estimated the task as 3 and two as 5 – then go for 3. If half say 3 and the other half say 5, go with 5. Don't waste time trying to get everyone to agree on the same number; as long as the estimates are similar, it is an acceptable outcome.

7.4.5 Ideal days or ideal hours

Estimation in ideal days or hours uses real-time as an indicator of effort. To be considered an ideal day, the following conditions must be met:

- You have everything you need on hand when you start the work.
- The story being estimated is the only thing you will work on.
- There will be no interruptions.

Once again, the estimation is not only for the piece of work but for the piece of work **for that team**. Not all teams have the same capabilities, capacity, or velocity.

Incidentally, the numbers on the poker cards could just as easily represent days or hours, so playing Scrum poker can also be used with ideal days and ideal hours.

7.4.5.1 But why ideal days or hours?

Here the estimation is based on the number of days/hours of effort that it would take the team to get a story done if the team worked with no interruptions. So, it represents the ideal situation, with no distractions, interruptions, and no task switching. Such interruptions usually account for approximately 2 hours per day, meaning the ideal day has an average of 6 hours.

The reality is that even if you use story points, at some stage you need to work it back to time. Some may argue that if the team's velocity is measured by story points, you don't need to do that; but it is semantics. Even if you say that a team can do 60 story-points in one sprint, a sprint is a timeboxed event so in effect, you have just worked it back to time in any case.

The real value in using story-points, at least in the beginning, is to change the culture in the organization. In most organizations, any timeline given is by default seen as a commitment. So, by using story points, you avoid the situation where someone will say: "but you said it would only take 8 hours to do this".

Never use ideal days or hours when speaking to customers. They will take that as a commitment.

7.4.6 Fast estimation using sticky notes

A technique described by Roman Pichler can be used if the teams are pressed for time to use one of the other estimation techniques.

Write the Fibonacci sequence on a board with a fair amount of space between numbers, give each team member a few items to estimate, and ask them to write it on a sticky note and put it under the number they think is appropriate.

Once everyone is done, allow all team members to look at the board and spot items they think may be problematic and not estimated correctly. Either allow the member to move the sticky immediately or revise the estimate.

An alternative is for the team to have a quick conversation about it before moving it to an appropriate spot on the board.

Instruct team members not to be over-critical about the estimations; this may lead to every sticky being either moved or debated. That is not the intention here.

Understand that this technique will be less accurate but will allow the team to get a lot done quickly.

You will most probably find that using this technique from the start will progressively yield better results as the team matures.

7.4.7 Other things you should know

No matter which technique you will use for estimating stories or tasks, in the beginning, you will most probably be wrong. But don't worry, the more you do it, the better your estimates will become.

Some say that only those with intimate knowledge of a task or story should be allowed to estimate, but involving the whole team often yields new insights. Secondly, it helps the team to have a better understanding and ultimately contributes to learning and growth.

If a team member really feels that they have no clue of the effort involved, allow them to abstain from putting an estimate forward. You don't have to play your hand every turn.

7.4.7.1 But what if you just don't know how to estimate effort?

You need to find out more – it is as simple as that. Situations like this often arise when a team has to deal with a new type of work.

Finding out can become a new story that must be completed first before the work on that specific item can continue. Add a new story to the backlog to find out.

This is often done by creating some kind of mock-up or prototype, and by doing research and talking to others that may have some experience.

7.4.7.2 What about non-functional requirements?

What is a non-functional requirement? People often view these as non-user requirements or technical or infrastructure requirements. But that is not how they are defined in Scrum.

A non-functional requirement is a system-level or product-level constraint that doesn’t relate to functionality. It instead refers to attributes, such as performance, accuracy, portability, reusability, maintainability, interoperability, capacity, etc., and is required for product backlog items to be fully accepted by the stakeholders*.* Non-functional requirements often relate to quality criteria, but not always.

Many times, the acceptance of functional requirements depends on a non-functional requirement. When estimating functional requirements, all the dependent non-functional requirements should also be considered.

When placing a non-functional requirement on a Scrum or Kanban board, it will seem as if you keep having to put it back on the board. So, when you find yourself with a requirement that keeps on occurring because others depend on it, it is most probably a non-functional requirement.

You have to include some of these non-functional requirements in every sprint, as the product or service is unusable or deficient when *done* does not meet these requirements. It is important to ensure that dependent non-functional requirements are included in the Definition of Done (DoD).

7.4.8 Other stories are just user stories written for other stakeholders

Technical requirements, things that users or customers don't ask for but are necessary to put in place for other requirements to work, are just a different type of user story. Here the users are internal staff who are part of the organizational system that facilitates user or customer outcomes.

In Lean these requirements would probably be considered as non-value-adding but necessary because the customer did not ask for them and is most likely not willing to pay for them – but if left out, you will never be able to deliver what the customer asked for.

Some people like to refer to these technical stories as process stories, infrastructure stories, and a host of other names. In Scrum they are all user stories – it is just the user that is changing.

7.5 Improvement activities as part of sprints

7.5.1 Roadblocks & Theory of Constraints (TOC)

Part of the Scrum Master’s responsibility is to help resolve roadblocks the Scrum team encounters. These roadblocks can be anything that prevents a team member from working at full capacity.

The Scrum Master does not need to wait until someone informs them about a roadblock or constraint. Good Scrum Masters make use of the POOGI technique explained below to proactively identify constraints.

People often think of roadblocks as something that prevents work from happening and whilst that is true, most roadblocks don’t stop work completely but, it may be an obstruction or bottleneck. Scrum teams and specifically Scrum Masters can learn a lot from the Theory of Constraint (TOC) as defined by Dr. Eliyahu M. Goldratt in his 1984 book entitled *The Goal*.

The premise put forward by Dr. Goldratt is that organizations and teams can be measured by the variations of three measures: throughput, operational expense, and inventory. Throughput can be seen as the rate at which a system generates value, and operational expense and inventory combined can be defined as Lean would define work-in-progress (WiP). WiP will be covered in some more detail later.

Before any goal can be reached, necessary conditions must also be met, like safety, quality, and legal obligations.

For most businesses, the goal is to make a profit by improving throughput, inventory, and operating expenses.

So, what does this have to do with managing roadblocks and impediments in Scrum?

7.5.2 Focus on five steps

The theory of constraints is based on the premise that the rate of goal achievement by a goal-oriented system is limited by at least one constraint. If no constraint exists, the system's output would be infinite, which is impossible.

Only by increasing flow through the constraint can overall throughput be increased, and the best way to do this is to follow five (easy) steps:

1. Identify the constraint(s).
2. Decide how to exploit the system's constraint(s).
3. Subordinate everything else to the above decision(s).
4. Alleviate the system's constraint(s).
5. If in the previous steps a constraint has been broken, go back to step 1 immediately.

The five focusing steps aim to ensure ongoing improvement focusing on the constraint, which is known as the process of ongoing improvement (POOGI).

But what is meant by words like exploit, subordinate, and alleviate?
In step two the idea of **exploiting** a constraint means that without a new investment, understand how the best possible output can be achieved, given that the constraint remains in place.

Subordinate means that you need to design work being done to work within these limits. It is essential to ensure that work keeps flowing – even if it means slowing down other parts of the system. Evaluating the effectiveness of your efforts is easily observed by looking at flow and can be done by limiting WiP.

Doing this will buy you time to identify the true root cause, not the symptoms, and come up with a way of removing the root cause/s permanently. Once these measures are implemented the constraint will be **alleviated**, and it is very likely that a new constraint will immediately pop-up, therefore requiring that the cycle starts again.

It is important to remember to remove the limits imposed in step two to exploit the constraint, as the constraint now no longer exists.

The likelihood that teams often operate for quite some time with constraints just being exploited and not alleviated is quite high, as fixing some problems requires investment and may be complex and large enough to require that the solution is implemented as a project or improvement initiative.

7.5.3 The continuous improvement backlog (CIB)

Although Scrum doesn't specifically acknowledge an improvement backlog, the best Agile organizations all have one.

Scrum states that improvements should be defined as product backlog items in a product backlog, but what happens to improvements that do not belong to a specific product backlog?

Many Scrum experts propose that a general continuous improvement backlog has its place in Agile organizations. If an improvement obviously belongs to a product, the best place to record it would be in the product backlog.

But if product-related improvements are recorded in a specific product backlog, which items belong to the general continuous improvement backlog?

There are three sources for improving product backlog items, they are:

- **Results of audit, CSI, or Kaizen** (empirical learning and improvement) initiatives that are non-product specific
- **Exploited constraints** that function cross product or system (e.g. many non-functional requirements)
- **Identifying technical debt** in the organization, its systems, and processes and defining initiatives to pay-off debt (done in both Lean and DevOps, and appropriate also to be applied in Scrum environments)

It is not the intent of this book to explore these initiatives; however, it is useful to clarify what is meant by the last item on the list.

7.5.4 Technical debt

If you google the term, you will find a definition that closely relates to software development:

> Technical debt […] is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer.[8]

However, all organizations have technical debt. Simply put, technical debt is the total sum of all those things that are known to be wrong in the organization, where people usually say: we will fix it once we have time.

[8] Technical Debt. (2021). Retrieved on February 14th, 2021 from https://en.wikipedia.org/wiki/Technical_debt

The reality is that this often means that it will never be fixed.

The problem is that all of these half-baked, sub-performing, constraint-loaded issues, slowly accumulate until they become one giant constraint that can have a significant negative impact.

It is better not to say you will fix it later, but to make time to fix it later.

This is where the continual improvement backlog (CIB) comes in. If you say it, you record it. Like any other product backlog, the CIB must have a Product Owner and every sprint cycle something must be done to address issues in the CIB.

7.5.5 Improvement increment

As mentioned before, a rule in Scrum is that at least one increment must be delivered in every sprint. This is also true for the CIB.

Many companies however elect to have all Scrum teams deliver increments from the CIB. Some companies have dedicated Scrum teams working on CIB product backlog items exclusively. Regardless of the approach chosen, delivering improvement increments is one of the ways in which you build sustainable Agile businesses.

8 What else happens during a sprint?

As mentioned, sprints start with a sprint planning. Then, the work to complete one or more increments is done during the bulk of the sprint. This is followed by a sprint review and a sprint retrospective. During the sprint, the Scrum team hold daily scrums, in which they can update each other and adjust the planning.

Significant attention has already been given to sprint planning, and additional insights can be found in the sections about planning and estimation.

The following section provides information relating to the remaining sprint activities and also some thoughts on the use of Scrum with DevOps.

8.1 More on the daily scrum

Every day at the same time, a 15-minute timeboxed meeting takes place where Developers plan the work for the next day. The intent is to optimize team collaboration and performance by inspecting the work done since the start of the Sprint, to plan work, and to do some forecasting on the work that remains.

As previously mentioned, the daily scrum allows every Developer to answer three questions:

- Yesterday I…
- Today I'm going to…
- I could not because…

Although these questions are good to answer and can provide useful insights, it was misconstrued by many that this is all that is done at the daily scrum.

It is not required to ask these three questions, and it is also fine if you ask other questions which are more relevant for your team. This is entirely up to the team; the questions are merely a means to get a conversation going.

Inspection and adaptation are at the heart of Scrum and the daily scrum is about just that.

Developers should use the daily scrum to inspect progress towards the sprint goal and to inspect how progress is trending towards completing the work in the sprint backlog. It is really about the Scrum team keeping their eye on the ball and being able to react if reality and planning do not align.

Remember that the Scrum team is self-managing, and much of this is a result of the team appropriately reacting or responding to changes, challenges, or problems during the sprint, and being able to do so as soon as possible.

The Developers can respond immediately and define the appropriate response.

Note however that the daily scrum is not a detailed planning meeting; it focuses on identifying issues, dependencies, challenges, and problems. Often it is possible to immediately define an appropriate response, but if that is not possible, formulating

the response becomes part of the daily work of one or more of the team members. Scrum team members often meet immediately after the daily scrum for detailed discussions, or to adapt, or re-plan, the rest of the sprint's work.

Daily scrums improve communication and are one of the primary means of inspection and adaptation in Scrum.

Daily Scrums also eliminate or minimize the need for other meetings and identify impediments to progressing towards the sprint goal. They also keep everyone informed and ensure that decisions are made quickly, and action follows soon.

The daily scrum is owned by Developers and they set the agenda and format. Although the Scrum Master is often tasked to facilitate the meeting, as well as to keep a record and update progress on a Kanban or Burn-down chart, it is not their meeting.

That being said, it is the responsibility of the Scrum Master to ensure that the daily scrum takes no longer than 15 minutes. If the Developers identify an issue without a clear solution, then the Scrum Master should ensure the meeting moves forward. The Scrum Master should ask who must be involved in finding a solution, and when the identified resources are available to reach some conclusion. Once these questions are answered, the daily scrum should move forward. This is the only time that someone other than a Developer may interrupt the meeting.

8.2 Doing reviews and retrospectives

8.2.1 More on sprint reviews

The sprint review is held at the end of the sprint to inspect the increment and make changes to the product backlog if needed.

Please note that the sprint review is not a release or deployment stage-gate, or a status and approval meeting. The meeting intends to elicit feedback and foster collaboration.

If a sprint has more than one increment, the increment should be deployed or made available as soon as it is done.

The sprint review once again deals with the ability to inspect and adapt.

During the sprint review, the Scrum team and stakeholders discuss what was done during the sprint and if anything changed during the sprint that was not according to plan.

What was done is demonstrated and it is shown how increments meet the definition of done (DoD) for all increments included in the sprint.

Stakeholders then collaborate to identify the next probable things that could be done to optimize value.

The Scrum Master ensures that the event takes place and that attendees understand its purpose. They should also ensure that the meeting stays within the timebox.

The participants of this event are the Scrum team and other relevant stakeholders, typically invited by the Product Owner (e.g. users, customers, management, etc.)

It is up to the organization to decide what format the sprint review will take based on their unique needs, but here are some proposals by experienced Agile Coaches on what could happen during a sprint review:

- The Product Owner explains which product backlog items have been Done and which have not been Done
- Developers discuss what went well during the sprint, what problems they ran into, and how those problems were solved
- Developers demonstrate the work that has been done and answer questions about the increment
- The Product Owner discusses the product backlog as it stands. He or she projects likely target and delivery dates based on progress to date if needed
- The entire group collaborates on what to do next so that the sprint review provides valuable input to subsequent sprint planning
- Review of how the marketplace or potential use of the product might have changed, what is the most valuable thing to do next
- Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated releases of functionality and capability of the product

The result of the sprint review is a revised product backlog that defines the probable product backlog items for the next sprint. The product backlog may also be adjusted overall to meet new opportunities or needs.

8.2.2 More on sprint retrospectives

The sprint retrospective occurs after the sprint review and before the next sprint planning. It is a meeting where the Scrum team discusses what went well, what could be improved, what should be avoided in the future and what will be the likely areas that may form part of the next sprint. For a sprint of 4 weeks, a sprint retrospective typically lasts 3 hours. If the sprint is shorter, the sprint retrospective should also be shorter.

The Scrum Master ensures that the event takes place and attendees understand its purpose. This is once again an opportunity for the Scrum team to inspect and adapt by identifying improvements that should be made in the future.

All members of the Scrum team must attend.

The Scrum Master encourages the Scrum team to improve its process and practices to make them more effective and enjoyable for the next sprint. During each sprint retrospective, the Scrum team plans ways to increase product quality by improving work processes, or by adapting the Definition of Done if appropriate and not in conflict with product or organizational standards.

By the end of the sprint retrospective, the Scrum team should have identified improvements that it will implement in the next sprint. Implementing these

improvements in the next sprint is the adaptation to the inspection of the Scrum team itself.

Although improvements may be implemented at any time, the sprint retrospective provides a formal opportunity to focus on inspection and adaptation.

Many Scrum teams create a continuous improvement backlog for items that cannot be fixed easily and include improvement items in every sprint, thus having product and improvement increments in each sprint.

9 Complex, large-scale product backlogs

Remember that every product has only one product backlog? But what if it is large and complex, and it will take many Scrum teams to get the product vision fulfilled?

Keeping the efforts of multiple teams working in one coordinated backlog is complex, to say the least.

9.1 Methods to scale

There are various ways Scrum can scale and many other Agile methods focus specifically on scaling Agile.

Many attempts have been made to deal with complexity and scale, including:

- SAFe
- LeSS
- Nexus
- Scrum@Scale

We will not attempt to describe SAFe and LeSS here, as these methods are quite complicated and overhead intensive. Nexus and Scrum@Scale are relatively simple and light-weight, and they use all Scrum practices virtually ‘as is’.

What are these other methods?

The Scaled Agile Framework® (SAFe®) is a set of organization and workflow patterns for implementing ‘Agile’ practices at enterprise scale. The framework is a body of knowledge that includes structured guidance on roles and responsibilities, how to plan and manage the work, and values to uphold.

Large-Scale Scrum (LeSS) can be viewed as a scaling framework for product development, but one can also regard LeSS as a scaling framework for organizations. It attempts to remove organizational complexity by solving problems in different and more straightforward ways in product development. Note that more straightforward doesn't mean easier, especially in the short run. Even though LeSS is simple, it is tough to adopt initially.

Nexus is a framework for developing and sustaining scaled product and software development initiatives. It uses Scrum as its building-block.

Similarly, **Scrum@Scale** relies on the foundational building blocks of Scrum, but some practices used here may reflect older thinking and a heavier reliance of longer-term pre-planning, which naturally makes Nexus a simpler choice to scale.

Software development is complex, and the integration of that work into working software has many artifacts and activities that must be coordinated to create a done outcome. The work must be organized, sequenced, the dependencies resolved, and the outcomes staged.

Many software Developers have used the Scrum framework to work collectively as a team to develop an increment of working software. However, if more than one Scrum team is working in the same product backlog and the same codebase for a product, how will they communicate when they are doing work that will affect each other? If they work in different teams, how will they integrate their work and test the integrated increment?

These challenges appear when two teams are combining work, and it becomes significantly more complicated with three or more teams.

This is also true for many other types of projects where Scrum is used, so what is discussed here equally applies to the most complex and interdependent project environments.

In this book, Nexus will be presented as the primary means of scaling, as it can be considered the least complicated approach of all. Of course, you are free to choose the approach that works best for your organization.

Like an exoskeleton[9], a Nexus is a framework around the Scrum framework. A Nexus combines the work of multiple Scrum teams into an integrated increment. Nexus is consistent with Scrum, and its parts will be familiar to those who have worked on Scrum projects. The difference is that more attention is paid to dependencies and interoperation between Scrum teams, delivering at least one done integrated increment in every sprint.

9.2 Agile Scrum's view of scaling

As it is a fundamental rule not to have multiple backlogs for the same product, multiple teams working from the same backlog need to establish effective communication and coordination.

One technique is to think about the backlog content at a higher level than functionality. The breakdown of product backlog items was previously described as epics, themes, and functions. It is possible to allocate the work to different teams based on themes. This approach would lessen the likelihood of conflict and inter-team dependencies.

One should be careful here as we describe a theme as a placeholder for product functionality, which helps us to structure the backlog and help with prioritization. Note that this is not the same as the Strategic Themes described in SAFe, which are differentiating business objectives that connect a portfolio to the strategy of the enterprise.

In Agile Scrum, methods were defined to help with scaling large and complex deliveries and backlogs. Later in the book, a more detailed introduction will be

[9] Exoskeleton. (2021). Retrieved on September 23, 2021 from https://en.wikipedia.org/wiki/Exoskeleton

provided to Nexus, which somewhat conforms to the traditional Scrum view of scaling but introduces some subtle changes in its approach.

Nexus introduces some new terms and terminology. Using this terminology consistently could prove valuable and reduce the chances of misunderstanding. Later more detail will be provided on scaling and Nexus.

10 Visual management, the Scrum and Kanban boards

Kanban is originally a Lean concept. Since Agile, and, therefore, Scrum is built on Lean principles, the use of Kanban as a method is absolutely appropriate when using Scrum. This is also shown by the creation of Scrumban by David J. Anderson, which combines Kanban and Scrum techniques.

Visual management is a key concept in all Lean and Lean-related management techniques. Visual management tools convey messages quicker and in a more engaging way than written information or verbal communication on its own.

If you are unconvinced, let us use the following illustration. If you are traveling by car to work every day, can you say with certainty at how many stop signs or traffic lights you stop? But you do stop – otherwise you would have caused many accidents by now. That is the power of visual management and information radiators!

It is much easier for teams to spot an issue and take action when this type of information is represented visually. The use of information radiators also means exposing issues, defects, and problems as they occur, allowing the team to take fast and more effective action.

A common method to visualize progress and track a team's progress towards derivable increments, and to expose issues and defects quickly, is the use of a Scrum board.

Teams often refer to the Scrum board as a Kanban board, which strictly speaking is correct, since Scrum boards are the simplest version of a Kanban board. However, Kanban as a method involves more than just tracking progress. Therefore, a distinction is made between Scrum boards and Kanban boards, where a Kanban board is generally a little more complex than a Scrum board.

Here is a quick overview of the differences. The choice of which method to use should be made by your team.

10.1 The Scrum board

In its simplest representation, a Scrum board is a visual representation of stories and related tasks that are present in your sprint backlog. This is a great technique to reduce the feeling of isolation or disconnection from the Scrum team regarding the whole sprint or Nexus plan or product goal.

The board normally has four (or more) columns, usually marked:

- User stories (or sprint backlog items)
- To do
- In progress
- Done

A software development board may look something like this:

- User story
- To do
- In progress
- Test
- CD/CI (DevOps) pipeline
- Done

Remember that this is just an example of a software development board and if you are delivering value using Scrum in a different area than software development, your board will definitely look different! The Developers in your sprint should choose the appropriate steps you would like to track progress from ‘To-do’ to eventually ‘Done’.

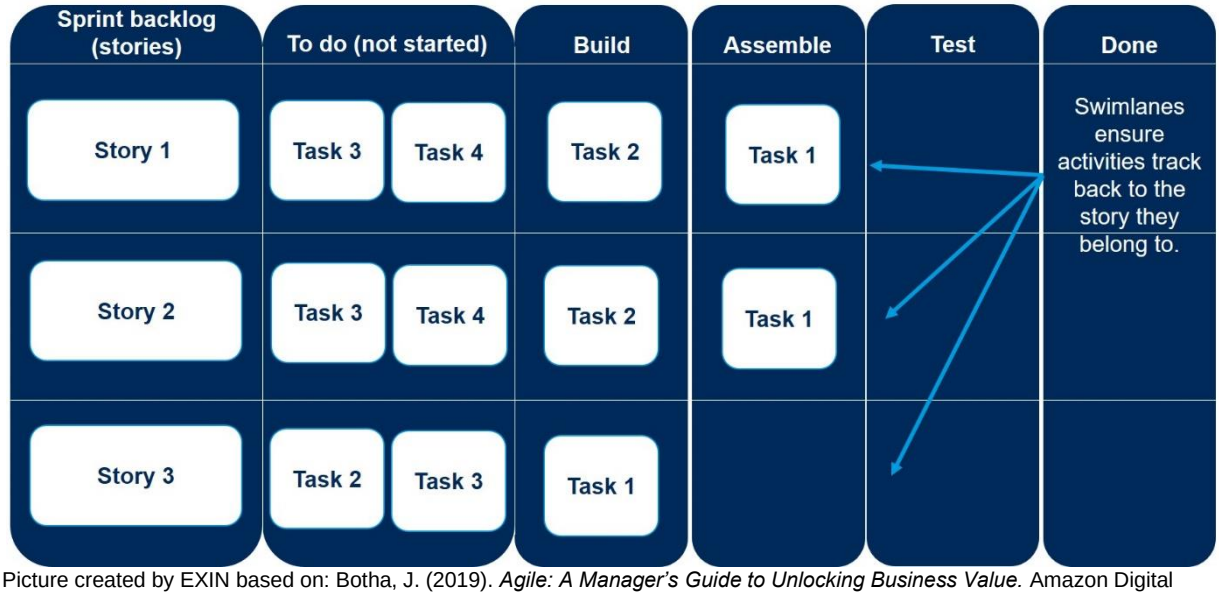
Both the story and the constituent tasks are usually written on cue-cards or post-it notes of different sizes.

Tasks are usually assigned to someone to perform and depending on the capabilities needed, may be re-assigned as they move from the left-hand side of the board to the right-hand side of the board.

Note that not all stories are customer requirements, but some may also reflect dependencies and non-functional requirements or even improvement stories and their associated tasks.

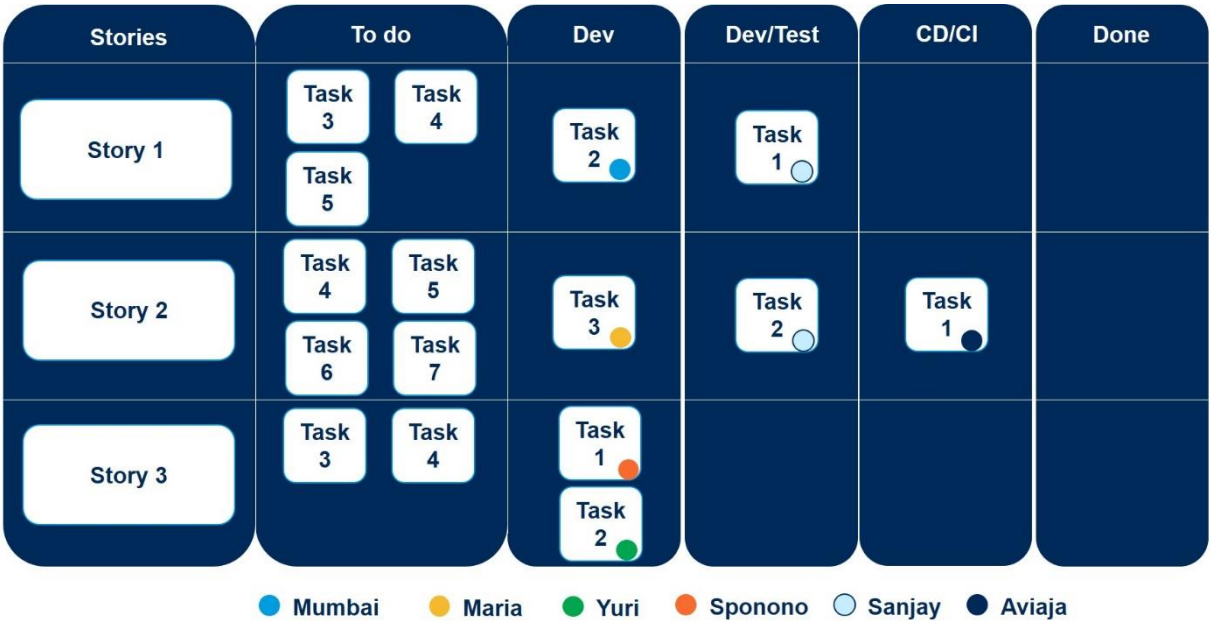
The example below shows a very basic Scrum board with swimlanes for different stories that form part of the sprint. That way it is easy to see if a task belongs to a story and to make sure that the work is completed and does not bounce around between team members.

Figure 16 A simple Scrum board (the example here is for making steering-racks for motor vehicles) and using swimlanes to keep tasks linked to the associated PBI (Story)



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager’s Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Figure 17 A typical Scrum board for a software development project, using colored sticky notes to indicate who the task is assigned to



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Colored stickers have been used to indicate the assigned person who should be working on it.

You don't have to use the stickers in the example; often only the assignee is written on the task card or sticky note. Other information that may be on a card is the estimated effort required to complete a story or a task, or which tasks are a high priority, or if there are any dependencies between stories and tasks, or tasks and other tasks. Dependencies are not shown in the picture, but this will be covered later specifically when discussing scaling using Nexus.

10.2 Why use a Scrum board?

Scrum boards help promote visibility and facilitate better communications. They also permit team members to see where issues are so that they can help each other (swarm on issues) or change plans if they were based on the wrong assumptions. A Scrum board, therefore, helps to promote inspection, adaption, and transparency and promotes teamwork as a result.

It also helps the team to commit to a sprint and the related work in the sprint, because everyone can see everything that must be done.

The board is created as part of sprint planning. In the sprint planning, the team first decides which items should be included in the sprint backlog. Then the team breaks the stories down into tasks necessary to get every story delivered.

The great thing about Scrum boards is that you can immediately start working visually. Remember, Scrum doesn't prescribe to use a Scrum board, but using one certainly is a huge benefit for the team.

10.3 Using a Scrum board

It is best to create your Scrum board as the last activity of sprint planning. Although it is often said that the Scrum Master is responsible for creating and maintaining the Scrum board, this statement is NOT TRUE.

The Scrum board belongs to the Developers and although the Scrum Master may be tasked with its upkeep, this is neither a requirement nor a given.

The Scrum board belongs to the Developers and helps them to optimize their work.

So how does one create the Scrum board?

Developers need to create a board that best reflects the type of work that they will be doing. Two examples have already been given of what boards can look like, but as a minimum, they need to include columns for Not Started (or To-Do), Doing (Something), and Done.

As product backlog items (PBIs), usually in the form of stories, are added to the sprint backlog, they should also be added to the Scrum board. PBIs will be decomposed during sprint planning, into tasks, usually with a number of attributes such as duration, dependencies, assignee, time to complete, etc. Place all associated tasks in a swimlane next to the PBI they belong to, or by using the same color sticky notes or cards. In the examples above both have been done.

By doing this your initial Scrum board should be built and ready to use as sprint planning is completed.

During the daily scrum, the discussion revolves around the work represented on the Scrum board and how it changes from day-to-day. If you still use the three questions (this is no longer defined in Scrum guidance but may still be useful), they will help determine which card/sticky note is in which column and who is assigned to the task at hand. The board will also reflect issues or impediments, commonly called blockers, which prevent task completion. Remember, Developers work on the task they choose, and tasks are not pre-assigned during sprint planning.

In software development projects the Scrum board can also be used to track and report on bugs. When one is found, it can be added to the Scrum board to be solved in a timeframe that will be analyzed by the Product Owner according to the priorities.

Moving cards and assigning tasks is a team responsibility and **not** that of the Scrum Master; however, teams often ask the Scrum Master to be the custodian of the Scrum board and to ensure that it always reflects the current status of a sprint or related iterations in the sprint. You will therefore often find that the Scrum Master performs Scrum board-related activities.

10.4 How is Kanban different from using a Scrum board?

The word Kanban roughly translates as “card you can see”. Kanban started as a stock planning system in the Toyota Production System and Lean. This led to the technique being used as a work scheduling system. The power of its simplicity and

visualization works very well together with Scrum or any other Agile method, and culminated in the earlier mentioned Scrumban.

The original Kanban was a card that indicated when a stock item was low at a workstation so that it could be supplied just-in-time for the next bit of work to be done.

Similarly, a Kanban task scheduling board gives an idea of when the next task must be done so that work can be done 'just-in-time'.

And this is where the main difference between a Scrum board and a Kanban board comes in. A Kanban board not only tracks progress but also **flow**: how work progresses through a system.

10.5 Why is flow important?

Obtaining the correct flow (tempo) of work is just as important as getting work done. In fact, you cannot achieve optimal performance unless there is flow in a process. Just imagine if all the traffic lights in your city were removed overnight – what would happen to the traffic flow?

The amount of traffic would not have changed, but the flow would be significantly impacted, and eventually, everything would grind to a halt.

Kanban as a method focuses on the optimization of workflow throughout a process and by doing so, it improves the overall predictability, efficiency, and effectiveness of the process.

Remember, Scrum is founded on empirical process control theory. Central to empirical process control is a frequent cycle of inspection and adaptation in a process that is transparent. Kanban practices focus specifically on the improvement cycle by regular inspection not only of the quality of work done but also of how work flows through the system or process.

The Kanban board provides the necessary transparency for inspecting work and the flow of work. The team can frequently adapt work or working methods to achieve quality outcomes and do so effectively and efficiently.

Now back to the Scrum board. Did you notice that in Figure 17, Sanjay has multiple tasks? Now if Sanjay could easily complete the one in the morning and the other in the afternoon, it would not be a major issue, but what if both these tasks take longer than a day?

It would mean that testing work would build up in front of Sanjay and the next task in the line could not be done. Aviaja would have to wait for Sanjay to finish testing before she could deploy the software.

In this scenario, Sanjay then becomes a bottleneck, and it does not matter how much work others do before or after Sanjay. The deliverables of the team will be determined by the bottleneck in the system, which in this case is Sanjay. His ability to complete testing in synch with the rest of the team's work-tempo is therefore essential to maximize the flow of work across the Scrum board.

What normally happens in situations like this is that Sanjay tries to task-switch between both tasks because he does not want to create a bottleneck. Task switching means that he cannot concentrate on one thing at a time and ensure that he gets it right, and instead of getting two things done, he either gets two things done poorly or nothing at all.

Note: it is a core principle in Lean and Agile to prevent task switching at all costs!

Now if Sanjay is the only tester in this team, work must be planned in such a way that no more than one task is in the Dev/Test column at a time.

By doing that, you will ensure that Sanjay can concentrate on one thing at a time, get it done well, and try to ensure that he only gets another task once the test he is busy with is completed and handed over to Aviaja to deploy and make available to users.

This concept is called limiting work in progress, by creating work-in-progress limits (WiP-limits). By limiting the amount of testing that is done at any one time, the overall delivery speed of the system is made faster.

If Sanjay is the only Tester, then the DevTest Column must have a work-in-progress limit (WiP-limit) of one (1), which applies to the Testing column of the Scrum board, to ensure the overall system works well. Or another Tester must be added to the team, and then the WiP Limit can be increased to two (2).

This may in fact mean that some of the Developers will stop doing development if WiP-limits are reached.

It may sound counterintuitive that it is a good thing to stop someone from doing work. The reality is that they are allowed to finish what they are busy with, but once that is done, they will not start with new work because that will only make the bottleneck worse! Yes, it may mean that sometimes a team member will not be working on something, but they can always offer their help to others to get work done, or even better be deployed to get work flowing again (in this case Maria may offer to help Sanjay to do testing, provided she is not testing her own work). If the team member doesn’t have the skills to help resolve bottlenecks, they can start learning how to do that and develop a new core skill.

In short: bottlenecks should be avoided in planning so that work can flow. One of the main objectives of Lean, on which Scrum is based, is to eliminate waste. Let’s use the traffic example again. Traffic lights stop and start traffic. In essence, the traffic light makes traffic slower; but in doing so, it makes traffic flow better, and therefore, the result is faster traffic.

How does this work?

10.6 Understanding the theory of flow

There are five Kanban (Lean) metrics that Scrum teams should measure to make sure that work is done better and faster; they are[10]:

1. **Work-in-progress (WiP)**: WiP is all work that was started but not completed. It is not important why work is not yet completed. What is important is that other tasks cannot be completed or even started before the WiP is completed, either because they are dependent on work items that are in progress, or because the Developers are working on the WiP.
2. **Cycle time**: The amount of elapsed time between when a work item starts and when a work item finishes.
3. **Work item age**: The amount of time between when a work item started and the current time. This applies only to items that are still in progress.
4. **Throughput**: The number of work items finished per unit of time.
5. **A service level expectation (SLE)**: It forecasts how long it should take a given item to flow from start to finish within the Scrum team's workflow.

If you have ever done Value Stream Mapping, these metrics will be familiar. It is highly recommended to study Lean as a Scrum Practitioner – it will make you better at Scrum and help to increase your understanding about the reason why so many things are done in a specific way in Scrum.

So why are these metrics important?

10.6.1 Little's law

If you want to understand flow, you have to know Little's Law, which states that:

$$\text{average cycle time} = \frac{average\ work\ in\ progress}{average\ throughput}$$

What does this mean?

In simple terms, it proves that the more things you work on at any given time, the longer you will take to complete the overall job.

Logically this makes sense. When you switch between tasks all the time, so when there are multiple work items in progress, it takes longer to complete any one task (work item age). In addition to this, you lose focus, and the result is not only getting less done (throughput) but also getting it done less well.

Kanban helps Scrum teams to increase cycle times, and therefore delivery speed, by better managing WiP.

If from the sample Scrum board, we can infer that only Sanjay can do testing and only Aviaja can do the deployments, then both these columns should have a WiP-limit of one (1).

---

[10] Scrum.org., Vacanti, D., & Yeret, Y. (2021). *The Kanban Guide for Scrum Teams.* Scrum.org. https://www.scrum.org/resources/kanban-guide-scrum-teams

The ‘Development’ column should have a WiP-limit of four (4) Mubai, Sponono, Yuri, and Maria. But even if four people can develop code, the amount of code developed must never exceed what can be tested and deployed. That is what the bottlenecks can handle, and by doing this the whole system becomes faster.

One can argue that the development work normally takes much longer than testing and using historical data this can be used to tweak the theoretical WiP-limit imposed on the Development column. In reality, it may mean that the WiP-limit for Development may be three (one less than the people who can at any time do development), but it will never be more than the number of (human) resources available to do the work.

If Sanjay and Aviaja determine the rate of work that can be done, this is called a pull-system. In pull-systems, work is only performed if the next workstation in the line can accept work when the previous station is done.

Pull-systems prevent the build-up of work in front of any given step in the workflow.

What the team ideally wants is what is called single-piece-flow in Lean. The team aims to ensure that work flows through the sprint with no hold-ups or waiting time before any given step. This then also implies that planning workflow in a sprint should be part of sprint planning.

By adding WiP-limits and planning flow, we turned our Scrum board into a Kanban.

Kanban also gives the Scrum team a clear view of skills and dependencies and how best to balance skills required in the Scrum team and to optimize work. Using WiP-limits on your Scrum or Kanban board will quickly show up potential bottlenecks in the system and allow the team to adapt work in such a way as to optimize flow[11].

What normally happens in Scrum teams and which is part of the required adaptation, is that team members start developing cross-functional skills as they learn from each other. Developers turn into T-shaped professionals by virtue of working in a cross-functional team.[12] That means that in the future a Developer could become a tester and remove bottlenecks in the system by switching roles. Remember though that a Developer can never test their own code/work.

This, however, is a longer-term solution. A more immediate solution could be to increase the number of testers or to see if more of the testing cannot be automated and made part of the CI/CD pipeline so that Sanjay and Aviaja can do more in a day without task-switching.

Another short-term solution is a behavior called **swarming**, where one or multiple team members stop working on their own tasks to remove an impediment to flow.

---

[11] You can read more on the theory of constraints in Dr. Eliyahu Goldratt’s seminal book *The Goal* (1984).

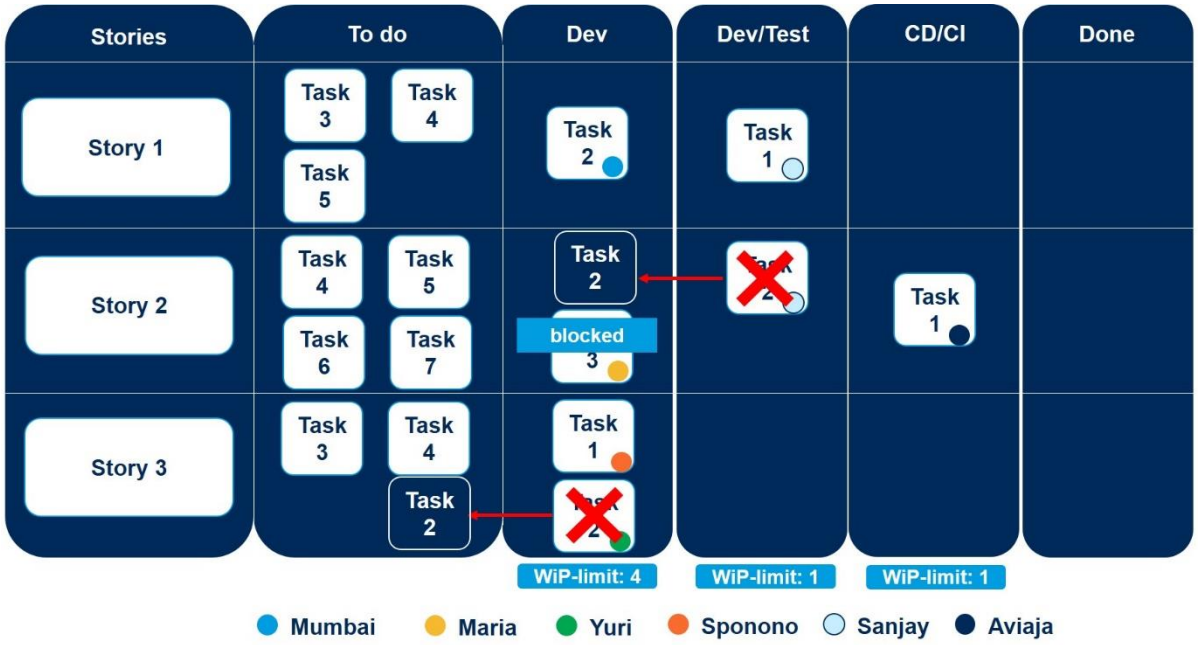
[12] More information about T-shaped professionals can be found on https://www.exin.com/about-career-path-certifications.

10.7 How will your Scrum board change when using Kanban techniques?

The main difference between a Scrum board and a Kanban board is the focus. A Scrum board is a visual representation of work being done, and a Kanban board is a tool to manage the said work in such a way as to maximize flow.

In its most basic form, by just introducing WiP-limits to your Scrum board, you have already implemented the central tenet of Kanban.

Figure 18 How WiP-limits help you avoid bottlenecks and create a Pull-system



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Let's assume that the WiP-limits discussed earlier are the correct measure to implement, your board will now look something like this (see Figure 18).

You will notice that Task 2 of Story Two should not be handed over to testing (Sanjay) and as a result will move back to the Dev Column. As it now occupies a space in the Dev Column, one of the other Dev tasks should not start as the WiP-limit for Dev is four (4).

Yuri should not start developing the code that was assigned to him yet. So, what should he do, nothing?

The correct answer is that Yuri should see how he can help Sanjay to get his tasks done faster while maintaining the same quality. They need to collaborate and focus on the team's commitments. If he does not have the skills to do that, it would be better to work in the system if he actually does nothing.

In this example, there is nothing else that Yuri can do; but in most sprints, it would be possible for Yuri to find a task that is not dependent on the bottleneck and work on that.

Scrum recommends that Developers should also work on improving the system they use to deliver increments; in essence, Scrum recommends that continual improvement should be part of the work planned in any sprint. In this example, if Yuri can't help testing, he should start with an activity that belongs to an improvement story, that is not dependent on the testing skills of Sanjay.

By doing this, the Scrum team ensures that work items are only pulled into the workflow at about the same rate that they leave the workflow, and as a result that work items aren't left to age unnecessarily.

When a bottleneck occurs, team members can also respond quickly to blocked or queued work items as well as those that are exceeding the team's expected cycle time level. This is called the **service level expectation**, or SLE.

The Scrum team uses SLE to find active flow issues and to inspect and adapt in cases of falling below those expectations.

The SLE consists of two parts: the period of elapsed time, usually days; and the probability associated with the forecasted period (for example, 70% of tasks should be done by sprint day seven).

The SLE should be based on the Scrum team's historical cycle time, and once calculated, the Scrum team in question should write it down and make it visible for everyone to see. It is recommended to leave a spot on your Scrum board for the SLE and also for your product goal, sprint goal, and DoD.

If it is a new team and there is no past data to base this on, the Scrum team should make its best guess.

10.8 What are blocked items?

You may have noticed that Task 3 of Story 2 in the graphic above is **blocked**.

Although it is a good idea to indicate issues that prevent work items from progressing on a normal Scrum board, it becomes especially important to use these when combining Scrum and Kanban.

In the example above, some dependency or problem is preventing Maria from completing her task. As the task has started, it cannot be moved back to the To-Do column; here, the team must see how they can help Maria to deal with the dependency. If the dependency is internal, the team may decide to adapt and change the order of work, so that Maria's task can be completed, or team members may swarm on the problem to help Maria to understand the cause and fix the underlying cause of the problem.

If the dependency is external, the Scrum Master will usually take up the issue with external parties to try and resolve it.

Although it has previously been stated that work items with a blocked status (a blocker ticket) could not move back to the previous column, the reality is that if nothing can be done, the team may decide, in consultation with the Product Owner, to remove the task from the sprint backlog to the product backlog, provided the sprint goal can still be achieved. Or as a last resort, it may be moved back to the to-do column (not any of the others) in the hope that it can be tackled later in the sprint.

If the team cannot do anything about a dependency or cause of a problem, and it means that the sprint goal and definition of done (DoD) cannot be achieved, the sprint fails, and work items should be returned to the product backlog.

10.9 Comparing a Scrum and Kanban board

The table below serves as an example of how a planning approach using Scrum and Kanban *may* differ.

Table 2 How using a traditional Scrum board may differ from Kanban

Using a Scrum board	Using a Kanban board
Work is performed based on priority	Work is performed to optimize the value stream flow
Tasks are usually not added mid-sprint, but many tasks can be in progress.	The ability to get work done determines if a task is worked on, or not.
Push system is possible	Pull system only

10.10 Using the Kanban method

Using the Kanban Method, not only the Board, to deliver value in the form of products or services is not the same as Scrum. There are some fundamental differences between the two methods.

In addition to the above differences in the two boards, using the Kanban method will also mean differences in:

- Approach
- Roles
- Reporting
- Ways of working

The following table lists some of the additional differences between the methods.

Table 3 Differences between Scrum and Kanban in more detail

Using Scrum	Using Kanban (method)
Timeboxed events (sprint, daily scrum)	No time boxes: the team works on tasks as they come
Very specific accountabilities (Product Owner, Scrum Master, Developers)	No set roles or accountabilities: they are assigned as need
Work in a sprint is normally limited to the sprint backlog	There is no separate iteration backlog
Tasks are usually not added mid-sprint	New work items can be continuously pulled from new customer requirements, so tasks may change at any time
Team may measure performance by how much is done and how much remains to be done (velocity and burn-down charts)	Because work is not set in a timeframe, it makes more sense to measure what was done (cumulative flow)
The Scrum team holds a sprint retrospective at the end of the sprint to discuss what went well, what went badly and how they can improve in future	There is no separate event after an iteration to discuss process change. However, teams in a Lean environment will usually do something similar (Kaizen)

10.11 How do sprint tasks change when using Kanban with Scrum?

10.11.1 The sprint

When using Kanban with Scrum, Scrum rules supreme. It means that sprint cadence remains the same as it was before. If you worked on two-week sprints, you would still do two-week sprints, if you used four-week sprints you will keep on using that cadence.

10.11.2 Sprint planning

A flow-based sprint planning meeting uses flow metrics as an aid for developing the sprint backlog. Throughput of the Scrum team in past sprints will be used to plan the following capacity of a team for work to include in the next sprint.

If you say this sounds pretty much like velocity metrics you would be correct, but now you will use the formula for SLE instead.

10.11.3 Daily scrum

Remember that the daily scrum is the Developers' meeting. Although the Product Owner and Scrum Master usually attend, they do not influence the discussion in the meeting unless consulted, and this remains the same when Kanban is used.

In the daily scrum, the focus will shift from getting things done to ensure that work flows through the system. Bottlenecks become a major category of impediments that must be addressed in the daily scrum.

In addition to understanding what is being worked on, and what will be completed during the day, Scrum teams also need to talk about:

- **Have any of the WiP-limits been broken** and if so, how will the situation be corrected? Specifically, how can blocked work items be unblocked?
- **What work is flowing slower than expected?** Here the team must look at what is the Work Item Age of each item in progress, which work items have exceeded or are about to exceed their SLE, and what can the Scrum team do to get that work completed?
- **Are there any factors or work items not represented** on the board that impacts the Scrum team's ability to complete work today?
- **Will any of the lessons learned today have an impact** on what to work on tomorrow? Should the order or priority of planned work be changed?
- **Has anything new been learned** that might change what the Scrum team has planned to work on next?

10.11.4 Sprint review

Sprint reviews don't have to change when using Kanban methods.

Although a review of Kanban flow metrics, especially throughput, may create opportunities for new conversations, it is better to talk about this in detail during the sprint retrospective.

10.11.5 Sprint retrospective

Inspecting flow metrics and the analytics thereof helps to determine what improvements the Scrum team can make to its processes. The Scrum teams should also inspect and adapt the definition of "workflow" to optimize the flow in the next sprint.

The team could use a cumulative flow diagram to visualize the team's WiP, average cycle time, and average throughput. It can be used to improve processes or when planning how much work will be done in the next sprint by the team.

Note, however, that the activities described should be part of inspection during the course of the sprint and not only during the retrospective.

10.11.6 Increment

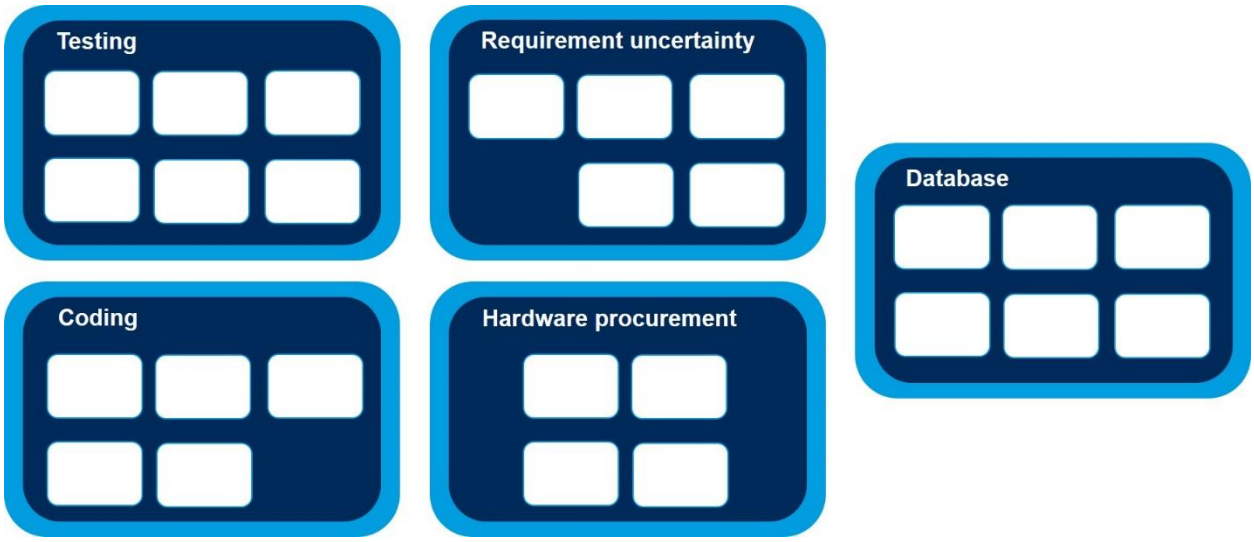
Scrum requires the team to create at least one potentially deployable increment of done every sprint. Scrum's empiricism encourages the creation of multiple increments during a sprint and to deploy these increments once work is completed by the team and not only at the end of the sprint.

Obtaining constant flow, therefore, means that customers or users continually benefit from the work of the Scrum team.

10.12 What else should be on a good Scrum board?

The sprint or Kanban board is only one of the information radiators found where Scrum teams meet. In addition to the sprint or Kanban board, other tools may be used to make all the artifacts and related work visible that will facilitate good communication and coordination, and execution of a sprint. Some of these tools have already been discussed, such as the sprint goal and increments that will be worked on during the sprint, and the associated work and records like Definitions of Done and blocker tickets. But that may not be the only useful information to make visible to the team.

Figure 19 Using the KJ method to group blocker tickets



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Although blocker tickets and how they are used have been discussed, it is very useful for teams not to discard blocker tickets once an impediment has been removed. Keeping track of resolved issues or impediments is extremely useful when performing sprint retrospectives. Some Scrum teams keep an area to 'park' resolved blocker tickets for later reference (during the retrospective) and some even use affinity diagrams to organize them into logically related themes if many occur during a sprint.

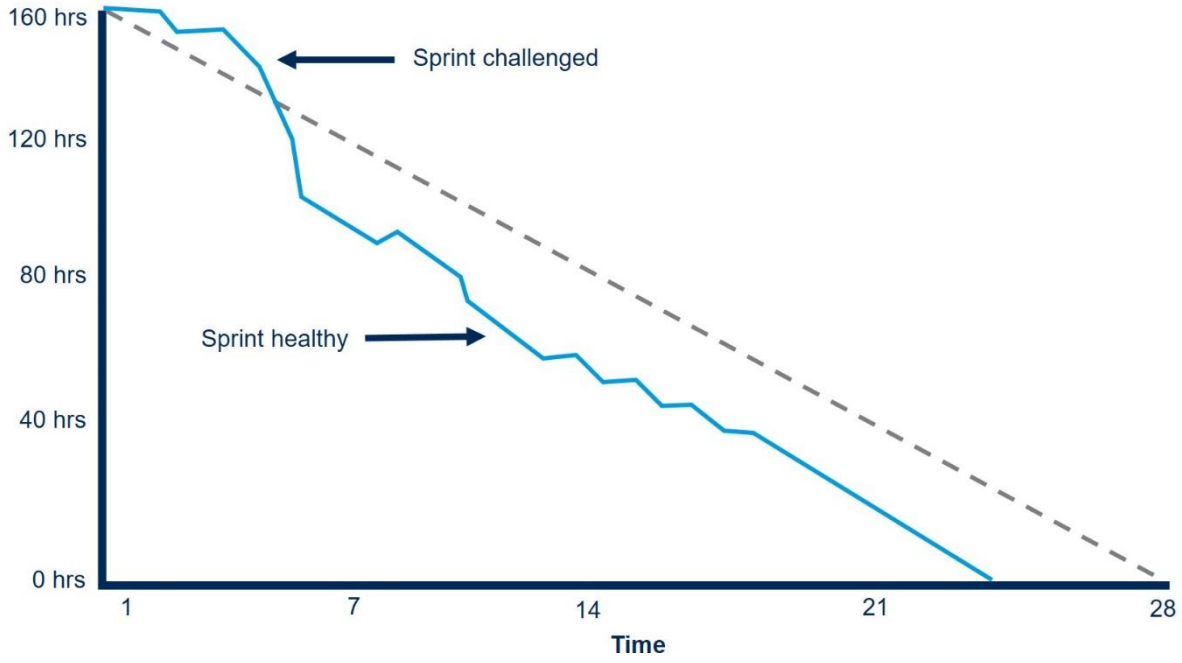
An affinity diagram, also known as the KJ Method after its inventor, the Japanese anthropologist Jiro Kawakita, is a graphic tool designed to help organize loose, unstructured data, in this case, blocker tickets. Blocker tickets are grouped into meaningful categories called affinity sets. These sets tie different tickets together around an underlying theme, clarify issues and provide a structure for a systematic search for one or more solutions.

10.12.1 Burn-down/Burn-up chart

Burn-down or burn-up charts are visual management tools specifically for tracking progress, but they can also measure velocity, the speed at which things are done in a sprint.

Sprints are planned so they can be completed in a fixed timebox, usually two to four weeks[13]. In Agile, you have to stick to the budget and the allotted time (timebox) for the sprint. The variable is how much value you deliver. Since the definition of done (DoD) considers this, a sprint must deliver at least one definition of done (DoD).

Figure 20 A typical burn-down Chart



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

As tasks are completed and checked against estimates, the remaining effort is calculated and displayed on a chart. The angle can either be downward, working towards zero effort remaining, which is called a burn-down chart; or the angle can be upwards, working towards several tasks completed within a two- to four-week timeframe, called a burn-up chart.

Most customers prefer burn-down charts as they are more interested in how much work remains to be done according to the allotted timebox rather than how much work has been completed (the Kanban board shows that in any case).

During planning, the chart template is created with a straight line from day 0 to 28 using the number of tasks. The example given is a four-week sprint – with zero tasks

[13] In practice, you may encounter longer sprints. However, according to the Scrum Guide the maximum length of a sprint is a month.

left. This line represents the estimated velocity at which tasks should be completed (the dotted line represents the estimated velocity).

Track and chart real progress daily, and if the line moves above the dotted line, it means that the sprint is not progressing as planned and the team must intervene. If the day-to-day plot remains below the dotted line, the sprint should complete on time and deliver all that was planned.

10.12.2 What is velocity?

Velocity is a measure of the amount of work that a Scrum team can complete during a single sprint. There is no industry standard for this. Velocity is calculated at the end of the sprint by totaling the points for all fully completed user stories and then comparing these to previously recorded totals. Unfinished work cannot be counted or taken into consideration when measuring the velocity.

Knowing the velocity of a team is a key input to sprint planning so that teams do not commit to too much work that they cannot complete in a sprint.

Note that team velocity in any given sprint will also depend on what work they are doing and how well the work is matched to the skills and capabilities of the team.

Evaluating velocity trends of teams may be a valuable input into the sprint retrospective and may be used to identify areas of improvement.

In general, a team's velocity should increase over time and a good metric to use here is a 10% improvement each sprint. In inexperienced Scrum teams, it is common that during the sprint it will become clear that the velocity is lower than expected. In such cases, it is recommended to identify ways to speed it up, or where this is not possible, to re-estimate the velocity.

Product Owners may decide to re-order product backlog items after a sprint, considering changes in velocity, increasing predictability, and flexibility.

10.12.3 What is the difference between velocity and SLE?

The concept of service level expectations (SLE) was introduced earlier as a predictive measure, similar to the ideal velocity charted on a burn-down or burn-up chart.

What makes SLEs different is that prediction is not only based on how much should be done in an entire sprint but also considers the complexity level of what must be done and how long it will take to get tasks done. Remember that on the burn-down chart you can only mark an activity as done when it is complete. If a task takes 5 days and it is 80% done, it is still not shown on the burn-down chart, but that skews the view of progress and velocity.

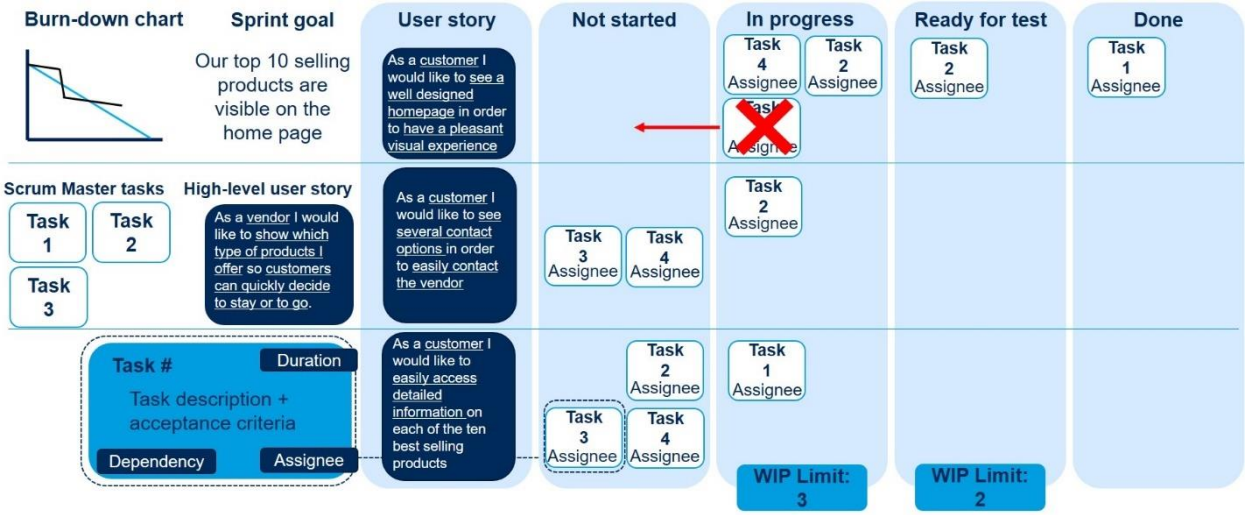
Some teams use SLEs to plot the ideal progress line on their burn-down chart as it reflects progress more accurately.

10.13 Information radiators and meeting places

Information radiators should be permanent fixtures and all team activities should be organized around these to ensure they remain highly visible at all times to all team members.

A view of a typical wall where a Scrum team meets may look something like this:

Figure 21 A typical Scrum team space



Picture created by EXIN based on: Botha, J. (2020). *Scrum Lego-game workbook [Courseware].* GetITright.

A typical Scrum team space will include other information radiators in addition to the Scrum board, here you can also see a burn-down chart and the beginning of a collection of blockers that may be analyzed using the JK method during the sprint retrospective.

11 The traditional view of scaling Scrum

As the original Scrum framework did not concern itself with issues of scale, users of Scrum very quickly needed to work out how Scrum could be used in complex and large environments.

11.1 Scaling the product backlog

Regarding the rule defined in Scrum *one product, one backlog*: What do you do when backlogs become large and complex?

The first answer here is to resist the temptation to make it large and complex. However, that is much easier said than done.

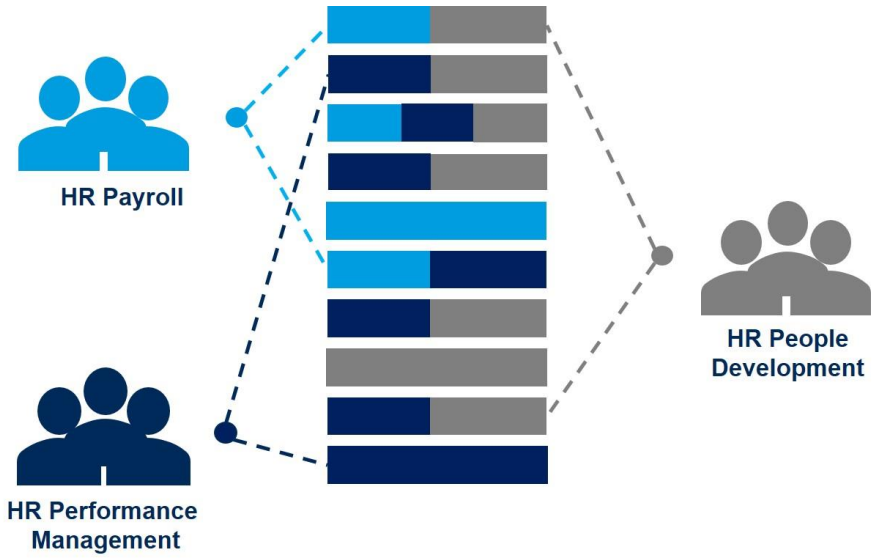
The general rule of thumb is to have about one hundred backlog items in the product backlog at any one time.

So, what do you do when you have three hundred or one thousand? And yes – this may well happen in very complex environments.

The first trick of the trade with large and complex backlogs is to make it less complex by combining backlog items into epics. Obviously, this would be counterproductive for items at the top of the backlog as they need to be worked on soon, but for items at the bottom of the backlog, this can be done with very little effect on the effectiveness or efficiency of product management activities.

Look for lower priority items that form a cohesive theme and recombine them into an epic.

Figure 22 Creating a Product Owner team



Picture created by EXIN based on: Botha, J. (2018). *Scrum Masters and Product Owners [Courseware].* GetITright.

In many environments, products are used broadly in organizations, and although it is one product, the view, use, and understanding of how the product unlocks customer value may be significantly different for different stakeholder groups.

In an environment such as these, different views of the product backlog may be created to accommodate the need for different stakeholders' views and priorities that may not be the same. This difference of perspective on the use and value of products is one of the biggest drawbacks in conventional Waterfall projects where the project team focuses on delivering a feature against a specification with no or very little regard of the outcomes and value expected by different stakeholder groups who will use the feature.

Although not the only reason to do so, this practice may also lead to scaling Product Ownership by introducing new roles into the product management structures.

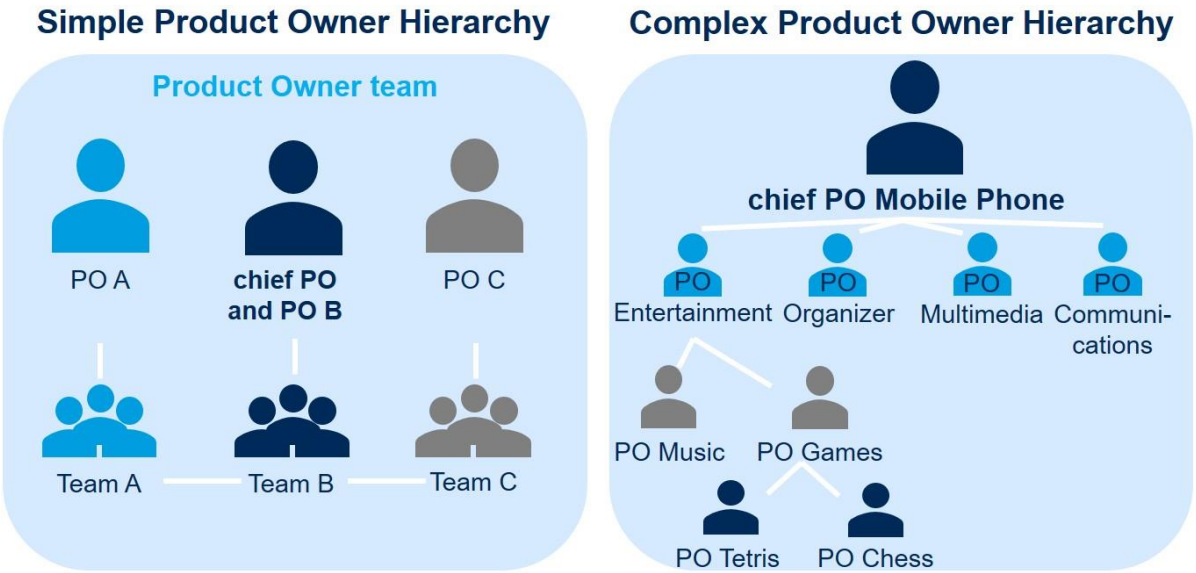
What is meant by that?

Remember that the role of a Product Owner is to represent the business and its users. In complex environments, this may lead to a lot of work for one person and to better serve the needs of the stakeholders it may be prudent to have multiple Product Owners – each focusing on a constituent or stakeholder group.

However, this would inevitably lead to one product with multiple product backlogs which is a fundamental no-go in Scrum.

So, what is the solution?

Figure 23 Structuring Product Owner teams



Picture created by EXIN based on: Cohn, M. (2010). *Succeeding with Agile*. Addison-Wesley.

A scaled Product Ownership team working with one backlog can be devised in either a flat or hierarchical fashion as in the picture above. In both cases, a single Product Owner must take the accountability, called the chief Product Owner. How work is

divided is up to the chief Product Owner, but it is recommended that the work is divided based on stakeholder group needs, similar to the multiple views created above of a single product backlog.

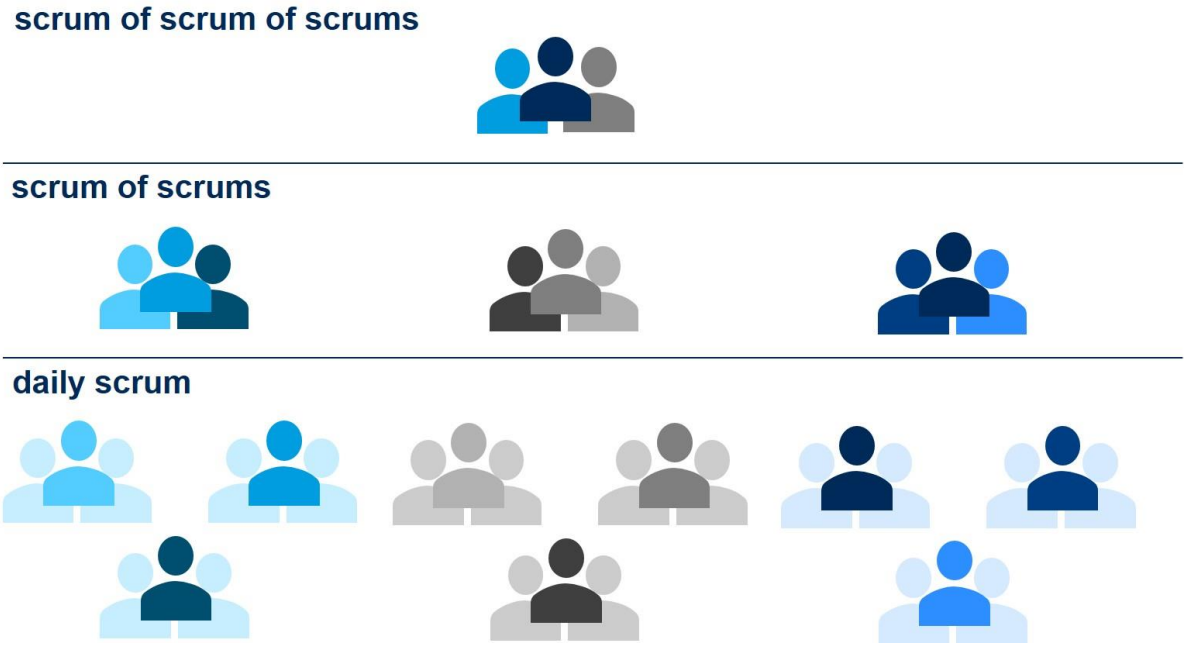
Scrum explicitly states that a Product Owner is one person and not a team or committee. Clearly the above approach, therefore, presents difficulties.

11.2 Scaling the work done

The likelihood that you will have to contend with multiple Scrum teams working on items from the same product backlog is much higher than needing to have multiple Product Owners supporting the same product backlog.

The traditional scaling approach of Scrum for the coordination of the work of several teams consists in transferring completed increments that result from one or more sprints into a coherent release (version). The necessary coordination takes place in a release planning or at the sprint level in a Scrum-of-Scrums.

Figure 24 Scaling by using scrum-of-scrums

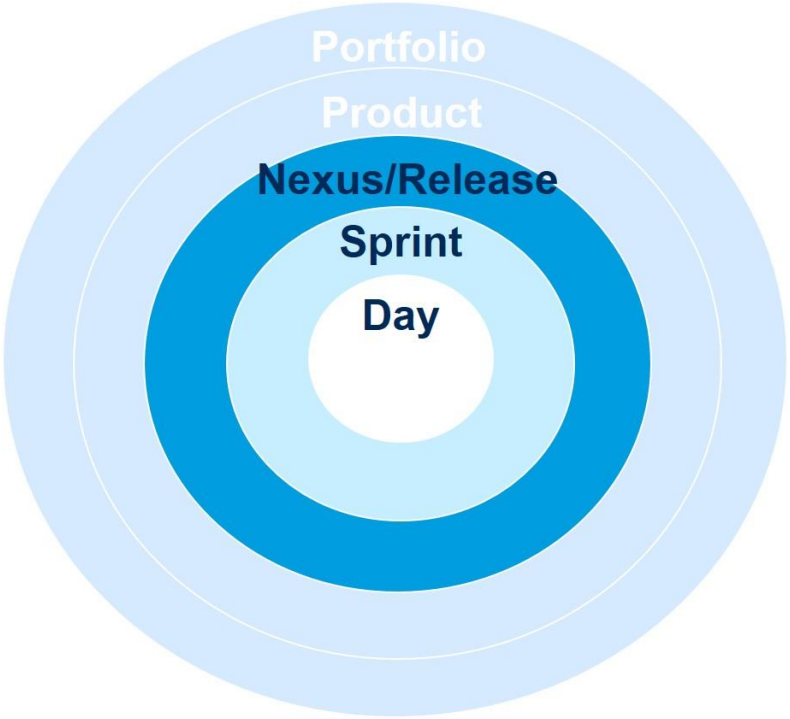


Picture created by EXIN based on: Botha, J. (2018). *Scrum Masters and Product Owners [Courseware].* GetITright.

Agile teams are primarily concerned with the three innermost levels of the planning cascade: Nexus, sprint, and day (see also the next Figure).

Release planning considers the user stories or themes that will be developed for a new product or service release. The goal of the Nexus or release planning is to determine an appropriate answer to the questions of scope, schedule, and resources for a project.

Figure 25 The three inner layers of planning



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

We will introduce the concept of the Nexus in the next chapter and here we will deal with a more traditional view of incremental delivery commonly used in IT environments, called release management within a Scrum context.

11.2.1 Release planning

A good release plan should be updated throughout the project. Updates are usually done at the start of each iteration. The release plan should always reflect the then-current expectations about what will be included in the release.

At the next level is sprint planning. Because this is a closer horizon than with release planning, the components of the sprint plan can be smaller.

The longer planning horizon of release management can be a significant drawback, because it presupposes that you know a lot at the beginning of a project. Therefore, you need to beware that it doesn't end up looking like Waterfall project planning all over again. A more detailed explanation of this problem can be found in Appendix B.

Another problem with this approach is that every organization pretty much developed its way of doing it. Some efforts were very successful and some not. Those who failed, failed mostly because they turned Scrum into WaterScrumFall.

Amongst progressive Scrum advocates, the release planning approach has now largely been replaced by the Nexus approach, which now sets a definitive reference or set of guidance for scaling Scrum.

These different methods will not be explored further, as Nexus provides definitive guidance on scaling the work done when using Scrum.

12 Nexus and scaling

12.1 What is a Nexus?

Nexus is a framework for executing scaled product development in a sustainable manner and it uses Scrum as its basic building block.

None of the definitions change, and nor does use of the Scrum roles, events, artifacts, or rules – Nexus adds to Scrum rather than changing it.

In this sense, Nexus is a scaled development framework, and a Nexus sprint is a unit of development that consists of many iterations delivering a combined 'increment', delivering value to customers.

In an iteration or sprint, all efforts focus on the outcome of that sprint, and this is defined by the goals and DoD of the Nexus sprint.

Practically, although we say that every sprint must deliver a potentially shippable product, due to the complexity of most development environments, this is often not possible.

Work must be sequenced, executed, and delivered, taking interdependencies and the skills-sets of various teams into consideration to satisfy real customer outcomes. The DoD of a sprint is therefore often not usable; the increment is not potentially shippable or even understandable by customers when they consider their needs and the outcomes a product or service must facilitate.

What does this mean in reality?
Often multiple Scrum teams are delivering value from the same product backlog. In software development and many other products, this means using the same building blocks (codebase) and confronting the same dependencies and interdependencies that resultantly develop between the work of these different teams. The increments delivered by different teams also often have to be tested and deployed together for customers to unlock value.

What makes it even more complex is that increasingly, Scrum teams are not even located in the same building, city, or country. An un-coordinated effort will spell disaster.

Teams need to be aware of how the work they do impacts others, how other teams depend on their deliverables, and what the consequences are of not completing work. Further, a common understanding of requirements, use, and application of what is developed by each team need to be understood by the others.

The more teams enter the fray, the more complex it becomes.

Because of these dependencies, a method to coordinate efforts, understand and manage requirements is therefore needed if Scrum must scale.

This method should specifically cater to:

- **Requirements.** Making sure that an overall understanding is created amongst members of all teams involved and how working on one requirement will affect work done on another.
- **Domain knowledge.** Work to be done is mapped to teams based on their technical and business skills, knowledge, and capabilities.
- **Product, test and integration, and quality assurance artifacts.** Ensuring a unified approach to testing the constituent parts as a unified deliverable.

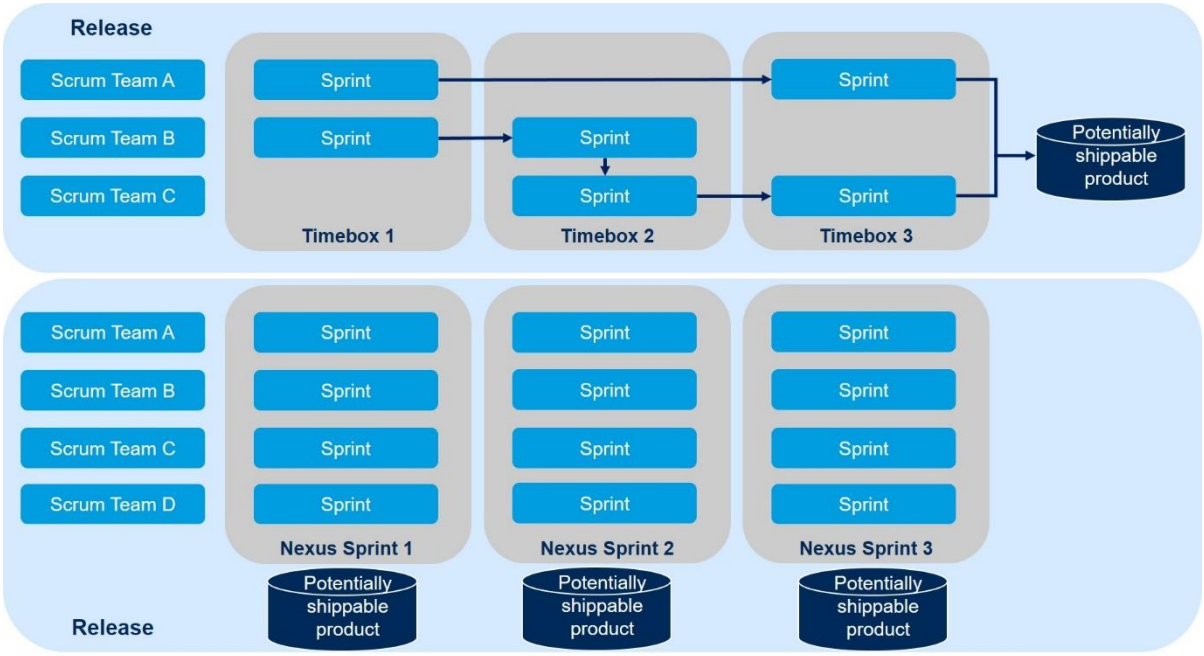
Using an approach like Nexus will ensure that work assignment does not bounce around between teams like a ping-pong ball and that work allocation takes into consideration not only domain knowledge but also the experience gained by teams in developing related or dependent deliverables. Not only is an integrated approach required, but also an optimized approach.

Nexus achieves all this by providing a basic framework in which teams can fill in the blanks.

12.2 The difference between Nexus and a release approach

Although Nexus sounds very similar to what was previously known as Agile releases, it fundamentally differs in that Nexus will only focus on one sprint or timebox at a time. This is a major difference and point of departure from the release approach, as it is specifically acknowledged that planning releases at the beginning of a project is based on incomplete knowledge – and it was this specific fact that prompted the departure from the Waterfall approach.

Figure 26 Nexus versus a release approach



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Many would argue that release planning is high-level, not detailed and that it changes during a project, adapting as requirements and understanding changes. This is certainly the intent and how it should be. The reality though is that because the approach is so familiar to what most participants in projects are used to doing in Waterfall projects, release plans change far less than what proponents suppose or advocate.

Many releases include sequential iterations or sprints, whereas a Nexus always focuses on only one iteration, and one could argue that there are merits to both approaches.

Just remember that the release approach is not essential in Scrum and it was adopted from other methods, whereas Nexus is designed for Scrum.

From the graphic immediately above, you will see that a release looks suspiciously like a Gantt chart used in Waterfall projects.

Sprints that form part of a Nexus sprint are parts of a single iteration, and the combined deliverable of the iterations that form part of the Nexus sprint is defined in the DoD for the entire Nexus sprint.

12.3 The New way of scaling with the Nexus framework

You can see Nexus as a framework that uses existing Scrum methods, roles, artifacts, and so forth, but adds to Scrum to achieve the creation of an integrated, value-delivering increment to the business. The focus of a Nexus team is, therefore, more on dependencies, interoperability, business outcomes, and the value it creates, than would be the case if only Scrum, or Scrum with an internally developed release approach, were used.

The Nexus team delivers one *done* for the whole Nexus and does so for each iteration in the Nexus.

To be able to do this, Scrum Roles need to augment. A new role is created: the Nexus Integration Team. The Nexus Integration Team’s role is to coordinate, coach, and supervise the application of Nexus and Scrum, to maximize valuable outcomes for the customer. Please note that they don’t supervise the work of the Developers as that would contradict the principle of self-management.

The Nexus Integration Team is a new role description for a new team that includes a Product Owner, a Scrum Master and one representative from each of the Scrum Teams. The representatives are the members of the integration team who represent Developers of a regular Scrum team.

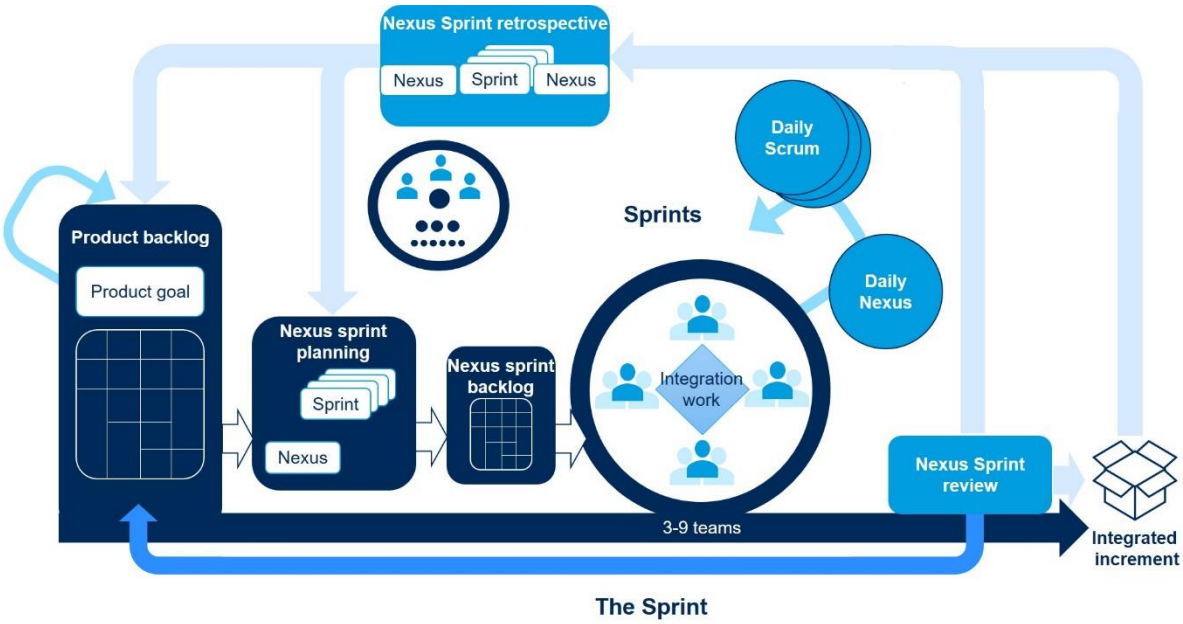
All Scrum teams should work based on the same product backlog, and refinement activities need to ensure that all team members have a good view and understanding of which items should be worked on next. It is best also to ensure that the ordering of items uses a visual means of understanding the immediacy of business needs.

Once Scrum teams have selected items, they will execute a sprint in the usual manner and work on their own sprint backlog.

None of the existing Scrum events is replaced. Instead, some events will now become a combined activity of all Scrum teams participating in the Nexus, or at the very least by representatives of all teams.

The graphic below outlines the exoskeleton created by the Nexus framework.

Figure 27 Using Nexus with Scrum



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us. and Schwaber, K., & Scrum.org (2021). *The Nexus™ Guide – The Definitive Guide to Scaling Scrum with Nexus.* Scrum.org. https://www.scrum.org/resources/nexus-guide

12.4 The Nexus process

Note: This section is presented here, as is from The Nexus Guide.

All work in a Nexus sprint can be done by any member of the sprints that form part of that Nexus sprint. All team members from Scrum teams that are part of the Nexus can work as part of a broader cross-functional team. Members working on backlog items will therefore be chosen based on their skillset and the interdependency of work items to minimize the need to decompose work needlessly.

During Nexus planning, backlog items should be refined, and the teams likely to work on them should be identified as early as possible. That means that this is a new activity to be added during product backlog refinement.

Next, an integrated planning session would be held called Nexus sprint planning. Representatives from each Scrum team should meet to discuss and review the refined product backlog, select items from the backlog for each team and place these into each team's sprint backlog. A combined Nexus backlog is also kept, to facilitate an integrated approach to reviewing Nexus outcomes and doing an integrated retrospective.

The specific reason for keeping an integrated Nexus backlog is to highlight dependencies and interdependencies!

It is evident from this description that the Scrum Master of a team is not the ideal candidate as a representative, as it will conflict with the principle of self-management and the role of the Scrum Master as a servant leader.

Each team then plans its own sprint, integrating work with others as required. Teams ensure that sprint goals are aligned with the overall Nexus goals as defined and agreed upon during Nexus sprint planning.

Development and testing are done as usual in the sprint, but some activities may require coordination between teams ensuring integrated testing can also be done where dependencies exist.

It is therefore preferable that a shared test and integration platform and approach is agreed, used, and maintained.

Teams still perform their daily scrum, but representatives of all teams also meet daily to discuss integration issues and impediments specifically, and feedback is given to each team in turn. Note that the Nexus guide does not say that the team's representative is always the same person. Yes, there may be a benefit in doing this from a continuity perspective. Still, the best person that can address the issues currently experienced at the Nexus should be the representative at the time.

Nexus daily scrum should therefore be done before the individual daily scrums so that feedback and a greater awareness forms part of every team's day-plan.

Sprint reviews are done as a Nexus. Nexus sprint reviews, therefore, show an integrated Nexus 'Done' increment. The Product Owner can then also use this information to update the product backlog and make appropriate adjustments.

A Nexus sprint review is done with representatives of each team. Challenges, issues, and improvements are then taken back to each team who in turn do their own sprint retrospective.

Representatives of each team then meet again after team retrospectives to address shared lessons learned and actions to be taken by all teams.

Special Note: The complexity of modern projects, changes, or improvement initiatives is high. It is prudent to also think about how work can be automated or how complex workflows can be optimally managed.

12.5 Nexus roles in more detail

The role of the Nexus Integration Team is to ensure that an integrated increment is produced every iteration. Scrum teams develop increments that, when combined, meet the Nexus Goal.

The Nexus Integration Team is a Scrum team consisting of:

- A Product Owner
  - remember: a single product backlog and a single Product Owner
- A Scrum Master
  - usually one from the participating teams
- Nexus Integration Team members
  - representing participating teams, at least one per participating team

If Nexus Integration Team members work as part of the individual sprints, priority must be given to work for the Nexus Integration Team. The work members perform as part of the Nexus Integration Team always takes precedence over the work in individual Scrum teams. This is important because resolving issues related to how teams work together or dealing with issues that may affect the Nexus outcomes or affect many teams, has priority due to the impact and risk to everyone involved.

Although Nexus Team membership is not necessarily the same from Nexus to Nexus, there may be benefits in keeping Nexus Integration Team membership stable – for example, understanding how work flows from Nexus to Nexus. However, team members must also consider the unique requirements of the specific Nexus, and therefore the team more than likely will change over time.

The Nexus Integration Team owns all integration issues, and their job is to ensure that Scrum teams always consider their role as a part of a bigger picture. Constraints and dependencies that may impede the Nexus should be an area of focus and attention.

The team also plays a role in coaching their own Scrum teams on their role in a Nexus.

12.6 Product Ownership

Teams based on Nexus work with a single product backlog, therefore they work with a single Product Owner who determines the makeup of the product backlog.

The traditional role of Product Owner does not change, except for the participation in Nexus activities that did not exist before.

Special note: Using a Product Ownership Team structure as described before is generally frowned upon amongst Scrum purists. However, there can be value in combining the two approaches.

12.7 Scrum Master in the Nexus integration team

As mentioned before, a Nexus Integration Team requires at least one Scrum Master as a team member. Although this Scrum Master may also be a Scrum Master for one of the constituent Scrum teams, this is probably not a good idea in complex environments, especially when organizations first start using Nexus.

The Nexus Scrum Master is responsible for coaching, facilitating communication, and enabling Nexus outcomes, as is the case in a sprint.

12.8 Nexus integration team members

A scaled approach to development and value delivery may be quite different from what Scrum teams are used to and will most probably require additional tools and techniques to be applied to make it work. Choose membership of the Nexus Integration Team (NIT) based on a potential member's ability and skills in these areas, because, in addition to ensuring the effective use of new tools and techniques, the NIT is also responsible for implementing the tools, processes, and practices needed to succeed.

As a specialist in the way that a Nexus will work within the organizational context, NIT members also become coaches and guides on the use of Nexus and associated tools and techniques.

A frequently asked question is how the role of the architect changes in Scrum. Architects in Scrum only define architectural principles and not detailed operational models which need to be implemented. In Nexus, the definition and implementation of these operating models become the responsibility of the NIT.

NIT members, because of their higher-level view of what is happening in a Nexus sprint, also fulfill the quality management role – specifically with regards to the integration of Nexus elements during the iteration.

12.9 New events introduced in Nexus

As a higher-level exoskeleton for the sprints that form part of the Nexus, new events that ensure coordination and integration are introduced, namely:

12.9.1 Nexus sprint planning

What will happen in this Nexus, who will do it, what are the interdependencies, what are the priorities? These are all valid questions to ask during the Nexus sprint planning.

The Product Owner guides participants in the selection of product backlog items to be selected for each Scrum team but does not dictate what will be worked on by whom.

But before this can happen, representatives of different Scrum teams work with the Product Owner on the refinement of the product backlog, specifically adjusting the ordering of items by applying the domain knowledge they bring to the party. Often the discussion is between what the priorities for the business are and what the dependencies are that need addressing first to get it done. Product backlog refinement is a balancing act between business priorities and technical or practical requirements that must be fulfilled to achieve the business requirements.

If possible, all Scrum team members should participate in refinement events to maximize understanding and communication and to minimize misconceptions and assumptions.

The purpose of the Nexus sprint and the outcomes to be achieved by the combined teams involved in the Nexus is defined in the Nexus sprint goal during the Nexus sprint planning. Once finalized and understood by all teams, each Scrum team will proceed with their own sprint planning.

It is best if this can take place in a co-located space, enabling teams to continually collaborate and share new dependencies, issues, and risks as they are discovered.

Because teams choose items from the same backlog, teams frequently also negotiate and re-evaluate what should be done by their team and what makes sense instead to have another Scrum team doing it – even if dependencies exist.

Although a well-defined product backlog will minimize the emergence of newly discovered dependencies or requirements, some are inevitably discovered during Nexus sprint planning.

Newly discovered requirements and dependencies should be made visible immediately and using a common visual planning tool to present them may be extremely helpful. Minimizing dependencies by moving backlog items between teams, and re-sequencing work, are commonplace during Nexus sprint planning.

The Nexus sprint planning improves the product backlog refinement, and because Scrum teams are now exposed to a bigger context, a benefit of using Nexus is a marked improvement in product backlog refinement over time.

12.9.2 The Nexus daily scrum

The purpose of the Nexus daily scrum is to inspect the progress of the currently integrated increment and to identify cross-team dependencies or integration issues as they are discovered.

Each Scrum team should be appropriately represented, and here the deliberate word choice indicates that the person who best understands the issues should attend the Nexus daily scrum. Do not confuse participation in the Nexus daily scrum with the membership of the Nexus Integration Team – Nexus Integration Team membership remains constant, at least for the Nexus sprint, but Nexus daily scrum representation may vary during the Nexus sprint.

12.9.3 Nexus sprint review

The Nexus sprint review is held at the end of the sprint to provide feedback on the integrated increment that a Nexus has built over the sprint. All individual Scrum teams meet with stakeholders to review the integrated increment. Adjustments may be made to the product backlog.

12.10 Nexus events

The duration of a Nexus event is guided by the length of the corresponding events in the Scrum. They are timeboxes in addition to their corresponding Scrum events.

12.10.1 Refinement

Refinement of the product backlog at scale serves a dual purpose. It helps the Scrum teams to forecast which team will deliver which product backlog items, and it identifies dependencies across those teams. This transparency allows the teams to monitor and minimize dependencies.

Refinement of product backlog items by Nexus continues until the product backlog items are sufficiently independent to be worked on by a single Scrum team without excessive conflict.

The number, frequency, duration, and attendance in refinement are based on the dependencies and uncertainty inherent in the product backlog. Product backlog items pass through different levels of decomposition from very large and vague requests to actionable work that a single Scrum team could deliver inside a sprint.

Refinement is continuous throughout the sprint as necessary and appropriate. Product backlog refinement will continue within each Scrum team, for the product backlog items to be ready for selection in a Nexus sprint planning event.

12.10.2 Nexus sprint planning

The purpose of the Nexus sprint planning is to coordinate the activities of all Scrum teams for a single sprint. The Product Owner provides domain knowledge and guides selection and priority decisions. The product backlog should be adequately refined with dependencies identified and removed or minimized before the Nexus sprint planning.

Effective Nexus sprint planning consists of three distinct steps:

1. As in Scrum, **teams in the Nexus select their own work for the sprint** they will do. It is best if this is a collaborative activity with representation from other Scrum teams to avoid duplication of work between teams.
2. **Each team performs its normal sprint planning process**. This occurs per Scrum team and can happen in parallel.
3. **Teams then refine work as dependencies are discovered**, and work may be re-organized between teams to minimize unnecessary work-breakdown and inter-team dependencies.

During the Nexus sprint planning, appropriate representatives from each Scrum team validate and adjust the ordering of the work as created during the refinement events. Refinement events should demand the participation of all members of the Scrum teams in the Nexus, to minimize communication issues.

The Product Owner discusses the Nexus sprint goal during the Nexus sprint planning. The Nexus sprint goal describes the purpose that will be achieved by the Scrum teams during the sprint. Once the overall work for the Nexus is understood, Nexus sprint planning continues with each Scrum team performing their own separate sprint planning. The Scrum teams should continue to share newly found dependencies with other Scrum teams in the Nexus. Nexus sprint planning is complete when each Scrum team has finished their individual sprint planning events.

New dependencies may emerge during Nexus sprint planning. They should be made transparent and minimized. The sequence of work across the teams may also be adjusted. An adequately refined product backlog will reduce the emergence of new dependencies during the Nexus sprint planning. All product backlog items selected for the sprint and their dependencies should be made transparent on the Nexus sprint backlog.

12.10.3 Nexus sprint goal

The Nexus sprint goal is an objective set for the sprint. It is the sum of all the work and sprint goals of the Scrum teams within the Nexus. The Nexus should demonstrate the functionality that was done (developed) to achieve the Nexus sprint goal at the Nexus sprint review, to receive stakeholder feedback.

12.10.4 Nexus daily scrum

The Nexus daily scrum is an event for appropriate representatives (individual Developers) to inspect the current state of the integrated increment and to identify integration issues or newly discovered cross-team dependencies or cross-team impacts.

During the Nexus daily scrum, attendees should focus on each team's impact on the integrated increment and discuss:

- Was the previous day's work successfully integrated? If not, why not?
- What new dependencies or impacts have been identified?
- What information must be shared across teams in the Nexus?

The teams use the Nexus daily scrum to inspect progress toward the Nexus sprint goal. As a minimum during every Nexus daily scrum, the Nexus sprint backlog should be adjusted to reflect the current understanding of the work of the Scrum teams within the Nexus.

The individual Scrum teams then take back issues and work that were identified during the Nexus daily scrum to their individual Scrum teams for planning inside their individual daily scrum events.

12.10.5 Nexus sprint review

The Nexus sprint review is held at the end of the sprint to provide feedback on the integrated increment that the Nexus has built over the sprint and to adapt the product backlog if needed.

A Nexus sprint review replaces individual Scrum team sprint reviews because the entire integrated increment is the focus for capturing feedback from stakeholders. It may not be possible to show all completed work in detail. Techniques may be necessary to maximize stakeholder feedback. The result of the Nexus sprint review is a revised product backlog.

12.10.6 Nexus sprint retrospective

The Nexus sprint retrospective is a formal opportunity for a Nexus to inspect and adapt itself and create a plan for improvements to be enacted during the next sprint to ensure continuous improvement.

The Nexus Sprint retrospective occurs after the Nexus sprint review and before the next Nexus sprint planning.

It consists of three parts:

1. The first part is an opportunity for appropriate representatives from across a Nexus to meet and **identify issues that have impacted more than a single team**. The purpose is to make shared issues transparent to all Scrum teams.
2. The second part consists of **each Scrum team holding their own sprint retrospective** as described in the Scrum framework. They can use issues raised from the first part of the Nexus retrospective as input to their team discussions. The individual Scrum teams should form actions to address these issues during their individual Scrum team sprint retrospectives.
3. The final, third part is an opportunity for appropriate representatives from the Scrum teams to meet again and **agree on how to visualize and track the identified action**s. This allows the Nexus as a whole to adapt.

Because they are common scaling dysfunctions, every retrospective should address the following subjects:

- Was any work left undone? Did the Nexus generate technical debt?
- Were all artifacts, particularly code, frequently (as often as every day) successfully integrated?
- Was the software successfully built, tested, and deployed often enough to prevent the overwhelming accumulation of unresolved dependencies?

For the questions above, address if necessary:

- Why did this happen?
- How can technical debt be undone?
- How can reoccurrence be prevented?

12.11 Nexus artifacts

Artifacts represent work or value to provide transparency and opportunities for inspection and adaptation, as described in the Scrum Guide.

12.11.1 Product backlog

There is a single product backlog for the entire Nexus and all of the Scrum teams working in the Nexus. The Product Owner is accountable for the product backlog, including its content, availability, and ordering.

At scale, the product backlog must be understood at a level where dependencies can be detected and minimized. To support the resolution of work, the functionality described in PBIs is often granular or *thinly sliced*. Product backlog items are

deemed *ready* for the Nexus sprint planning meeting when the Scrum teams can select items to be done with no or minimal dependencies on other Scrum teams.

12.11.2 Nexus sprint backlog

A Nexus sprint backlog is the composite of product backlog items from the sprint backlogs of the individual Scrum teams. It is used to highlight dependencies and the flow of work during the sprint. It is updated at least daily, often as part of the Nexus daily scrum.

12.11.3 Integrated increment

The integrated increment represents the current sum of all integrated work completed by a Nexus. The integrated increment must be usable and potentially releasable, which means it must meet the definition of done (DoD). The integrated increment is inspected at the Nexus sprint review.

12.11.4 Artifact transparency

Just like its building block Scrum, Nexus is based on transparency. The Nexus Integration Team works with the Scrum teams within a Nexus and the organization to ensure that transparency is apparent across all artifacts and that the state of the integrated increment is widely understood by all team members.

The state of Nexus artifacts will determine many decisions made during the Nexus and will only be as effective as the level of transparency about the state of artifacts. Partial or incomplete information will lead to incorrect or flawed decisions, which in turn can be magnified at the scale of Nexus.

Increments must be developed so that dependencies are detected and resolved before the technical debt becomes unacceptable for the Nexus or ongoing operations. A lack of complete transparency will make it impossible to guide a Nexus effectively to minimize risk and maximize value.

12.11.5 Definition of done (DoD)

The Nexus Integration Team is responsible for a definition of done that can be applied to the integrated increment developed each sprint. All Scrum teams of a Nexus adhere to this definition of done (DoD). The increment is Done only when integrated, usable, and potentially releasable by the Product Owner.

Individual Scrum teams may choose to apply a more stringent definition of done (DoD) within their own teams but cannot apply less rigorous criteria than agreed for the increment.

12.12 The core of Nexus is the Nexus integration team

The Nexus is made up of between 3 and 9 Scrum teams, with a shared Product Owner and a single product backlog. The Nexus Integration Team (NIT) consists of

- ***the*** Product Owner,
- ***a*** Scrum Master,
- and Nexus Integration Team members from teams that form part of the Nexus.

The Product Owner
Although the Product Owner is part of the NIT, they also fulfill the Product Owner role on every Scrum team which is part of the Nexus. The Product Owner ensures (accountable) that the product backlog is ordered and refined by maximizing value for stakeholders.

A Scrum Master
The Scrum Master in the Nexus Integration Team is responsible for ensuring that Nexus as a method is understood and working. They also typically facilitate Nexus level events like Nexus daily scrum, Nexus sprint planning, cross-team refinement, Nexus sprint review, and Nexus retrospective, or at least they support those who facilitate them.

Nexus Integration Team members
Nexus Integration Team members are made up of the Developers of Scrum teams within the Nexus. The NIT coordinates all activities within a Nexus. Team members, like Scrum Masters do for Scrum, become coaches in their own teams and jointly coordinate and supervise the implementation, application, and use of Nexus to ensure optimal outcomes.

NIT members are accountable for ensuring that an integrated increment, the combined work completed by a Nexus, is produced at least every sprint cycle.

The composition of the Nexus Integration Team may change over time, depending on the current needs of the Nexus.

NIT members are commonly drawn from Scrum teams within the Nexus and generally, NIT membership is part-time; but some NIT members may well be full-time members, especially in complex environments where coordination of dependent work is vital.

NIT members should, however, always prioritize their role as an NIT member above their role as a Scrum team member. The whole is more important than its parts.

Accountability
The Nexus Integration Team is accountable for ensuring that an integrated increment is produced at least every sprint. This ensures that the integrated product represents focus across the teams, rather than just the efforts of individual teams.

Scaled software development requires tools and practices that individual Scrum teams may not frequently use. Therefore, the Nexus Integration Team should consist of professionals who are technically proficient. However, integration is broader than

coding. How teams work together and collaborate is a vital part of the integration. For instance, how teams react to breaking builds or practice collective code ownership all form part of successful integration. The Nexus Integration Team must have members with both 'hard' (technical) and 'soft' (people) skills.

Activities

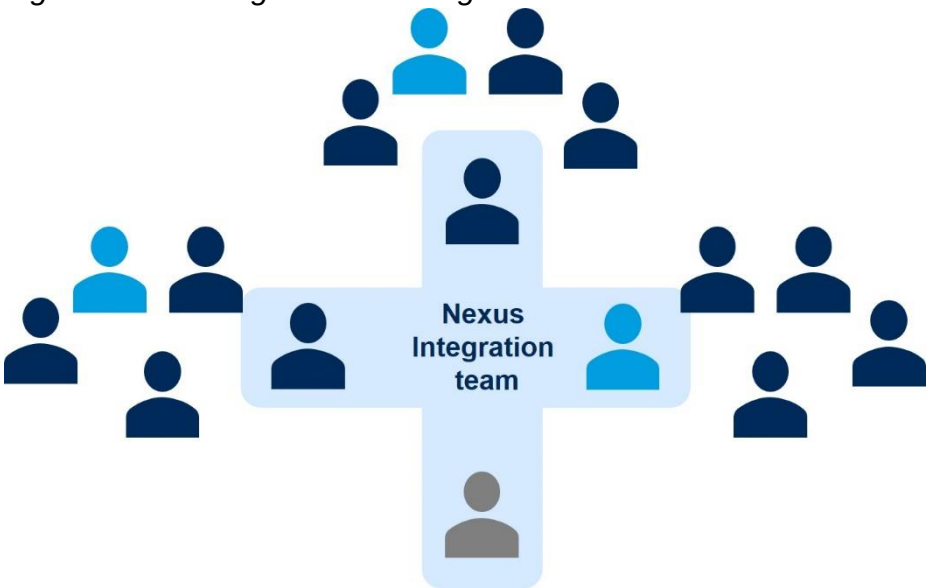
Nexus Integration Team members function as coaches on member Scrum teams. As such, Nexus Integration Team members should possess the required skills and traits to enable the Scrum teams within the Nexus to improve the state of the integrated increment continually.

Activities that the Nexus Integration Team may perform daily include:

- Helping coordinate work between the teams
- Raising awareness of dependencies as early as possible
- Ensuring integration tools and practices are known and used
- Serving as consultants, coaches, and communication links
- Sometimes assisting with the work, as appropriate or necessary
- Facilitating shared architecture/infrastructure
- Constantly providing transparency of integration.

It is important that the people serving on the Nexus Integration Team are *servant leaders* and have a coaching approach to improvement. This is not the 'all-star' team. By staffing the Nexus Integration Team with members from the Scrum teams within the Nexus as shown in Figure 28, an "us and them" divide is avoided.

Figure 28 Creating a Nexus Integration Team



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

The Nexus Integration Team consists of members from the Scrum teams. The Nexus Integration Team typically does not work as a Scrum team together, pulling from the product backlog. Instead, they are a community of coaches and guides

providing service to the Scrum teams within the Nexus. As they are also members of those Scrum teams, the Nexus inherently provides its own leadership.

The Nexus integration team as a Scrum team
If there are serious integration concerns or issues, as a last resort the Nexus Integration Team may decide that they need to pull work from the backlog and work as a Scrum team. In this case, they move from their existing Scrum teams into the Nexus Integration Team full-time, until the issues are resolved.

They may choose to do this because:

- The work requires skills that only members on the Nexus Integration Team possess
- The product is in an unknown state
- The Scrum teams within the Nexus are temporarily unable to integrate successfully

This is not a sustainable working model as it means that the Nexus has failed to scale Scrum successfully and the work is slowing to the pace of a single team – the Nexus Integration Team. Ultimately, if only the Nexus Integration Team can do the work, then scaling has failed.

However, the decision to work this way should be triggered by the Nexus Integration Team, which includes the Product Owner, as a temporary last resort in order to fulfill their accountability.

Having the Nexus Integration Team working as a Scrum team is considered “failure mode.” This should be a short-term tactical choice. In general, it is a good idea for members of the Nexus Integration Team to work with the Scrum teams within the Nexus to transfer scarce skills and knowledge through pairing and coaching.

Membership
Other than the role of the Product Owner, membership in the Nexus Integration Team does not have to be permanent. In fact, it should be situational. The types of skills that are needed should change over time, leading to changes in team composition.

The Nexus may decide to invite people from outside the Nexus to join the Nexus Integration Team. These may include representatives from areas that are critical to the success of the Nexus. If they share the same integration accountability, then their participation in the Nexus Integration Team can lead to positive outcomes.

The final word on Nexus Integration Teams
The Nexus Integration Team is not a management team, nor should it be a team of people with sliced expertise and experience. It is a team of professionals, usually from the individual Scrum teams within the Nexus, who ensure that practices and tools are implemented, understood, and used to detect dependencies. They are accountable for ensuring that an integrated increment is produced at least every sprint. Their focus on integration should include dealing with both technical and non-technical issues within the Nexus.

The Nexus Integration Team acts precisely as a Scrum Master acts to a Scrum team, not as a hierarchical manager “in charge,” but through servant leadership and coaching. They should use bottom-up intelligence from the teams to resolve issues and improve. Success at scale results from every team being able to contribute to increased product development continually.

12.13 Visualizing the Nexus sprint backlog and cross-team refinement

The purpose of Nexus sprint planning is to coordinate the activities of all Scrum teams that take part in a Nexus for a single Nexus sprint. The Nexus sprint backlog is created during Nexus sprint planning and is a visualization of the work across the Nexus. It specifically highlights dependencies.

Combined Nexus sprint planning must be done in a structured way, ensuring that work is appropriately coordinated and considers all dependencies.

Constant refinement would be needed in a complex environment as new dependencies are often only discovered during a Nexus.

A cross-team Nexus refinement board proves to be a useful tool to validate and update work done by Scrum teams in the Nexus.

12.14 Cross-team refinement in Nexus

Product backlog refinement is an ongoing activity in Scrum; it is, however, not a mandatory event. Due to the complexity of many teams working together on a single product (product backlog) in a Nexus, refinement becomes an official and required event.

Shared backlog refinement is focused on decomposing product backlog items (PBIs) sufficiently for teams to understand the work they could deliver and the sequence of delivering the work over upcoming sprints. The event also allows the teams to focus on minimizing and removing dependencies between teams.

Special note: For those who are wondering what replaces release planning in Nexus, the answer is *nothing*. However, shared backlog refinement creates a mutual understanding of the importance, dependencies, and sequence of PBIs to be delivered over multiple upcoming increments or sprints. See the questions below.

In a Nexus, there are several Scrum teams pulling work from a single product backlog. Therefore, new questions about refining the backlog must be answered:

- Which teams pull what work?
- How can we best sequence the work, across sprints and teams, to balance early delivery of value against risk and complexity?

When scaling Scrum, it is recommended that these questions are best answered by the Developers themselves through cross-team refinement.

Representatives from each team attend a cross-team refinement event, and representatives should be selected and attend the workshop based on the work being refined instead of the role that they play inside their team. It may be common to have different people attend these workshops based on the skills required.

But how does one determine which teams pull what work?

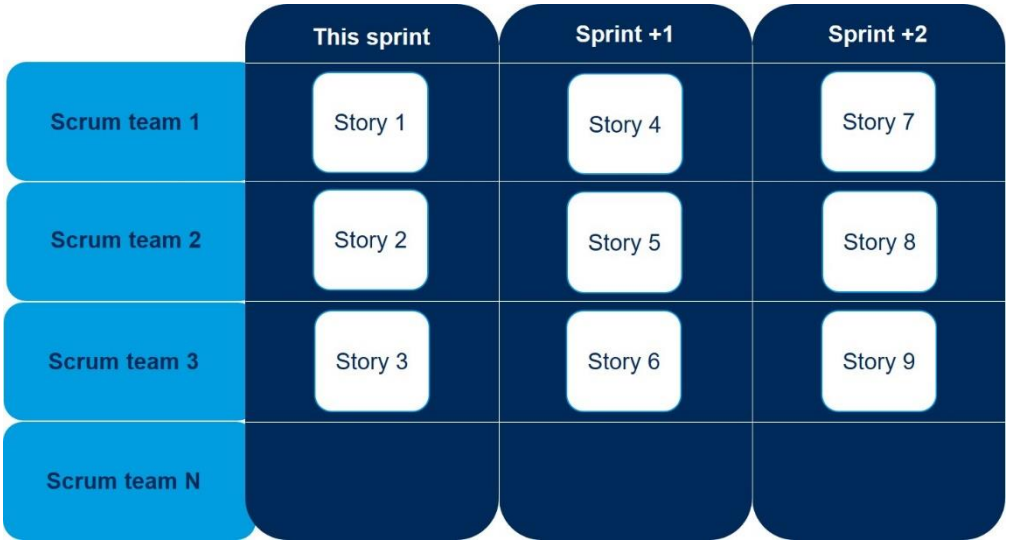
Decompose and sequence product backlog items

The Product Owner should explain the PBIs to be refined. These are generally large items that have not yet been decomposed or sliced to a size that will fit inside a sprint. This diverts from conventional wisdom where PBIs at the top of the product backlog will always be refined before sprint planning starts. Delaying this breakdown allows the opportunity to decompose the items based on skills and dependencies that emerge during refinement.

Initial conversations will be very fluid and should focus on which teams have the skills necessary to undertake the work. Existing constraints around specialist skills that are not available in each team should be carefully considered.

Once representatives from the Scrum teams start to understand the PBIs, they can decompose them into smaller items, which can then be taken back to their teams for normal product backlog refinement and sprint planning.

Figure 29 Creating a Nexus Board



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Note that the initial refinement will also consider dependencies, and although similar to traditional release planning, it will identify what can be done in this sprint and what should wait for the following sprint. Dependencies drive the conversation and not what was promised to customers months earlier. It is also not a good idea to try and look for dependencies beyond three sprints.

Understanding the flow of work for the upcoming sprints and visualizing it per team is, therefore, part of the initial refinement activity, but once this is done the NIT members need to ask: “How can we best sequence the work across sprints and teams to balance early delivery of value against risk and complexity?”

This is done by visualizing dependencies on the refinement board and then considering how best to manage the dependencies.

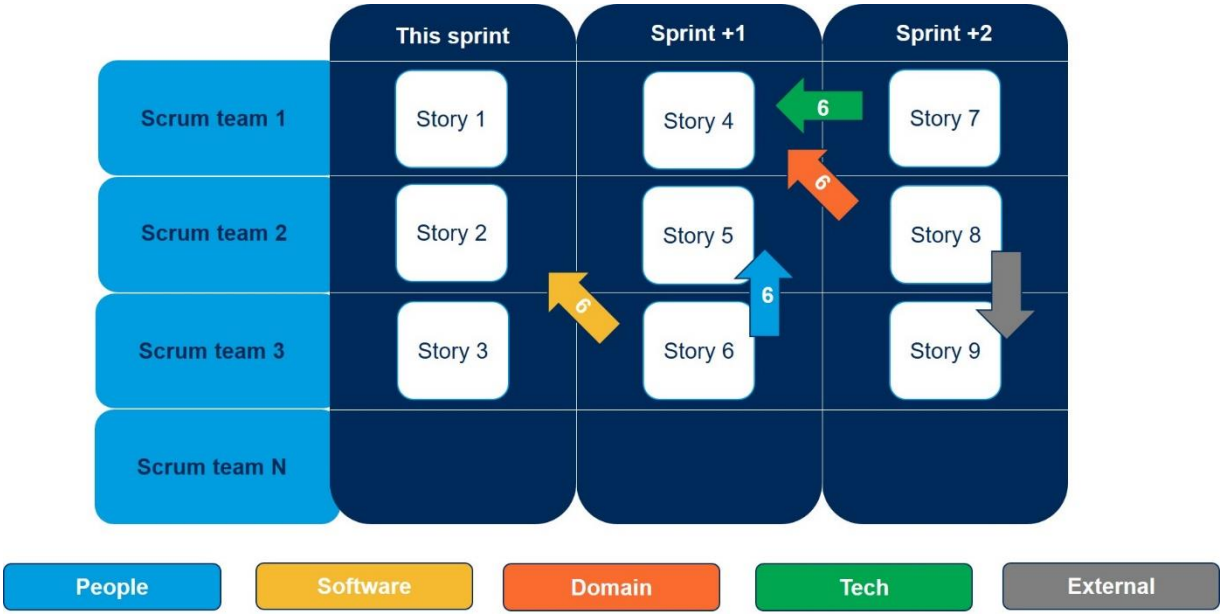
Not all dependencies are of a similar nature and teams should also categorize dependencies whilst defining them.

In general, the following categories and sub-categories can be used as a starting point:

- **Build Sequence** – An item cannot be completed until its parent is complete (can include technology, domain, software)
- **People/Skills** – Only certain people/teams can complete an item
- **External** – The parent item is being delivered outside the Nexus

Dependencies are indicated by arrows, and the color of arrows helps to differentiate between types of dependencies. The direction of arrows indicates parent-child relationships, and the parent story number may also be included on the arrow for clarity purposes.

Figure 30 Dealing with dependencies of tasks distributed across Nexus Scrum teams



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager’s Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us. and Schwaber, K., & Scrum.org (2021). *The Nexus™ Guide – The Definitive Guide to Scaling Scrum with Nexus.* Scrum.org. https://www.scrum.org/resources/nexus-guide

At a minimum, consider identifying external dependencies. You may want to add additional color as makes sense in your context.

The more arrows, the higher the risk and the more important it is for clear planning and coordination to take place. Dependencies highlight the critical path of work to come and refinement. Refinement, in turn, should minimize dependencies or at least the impact of dependencies.

The following general convention can be used:

← A horizontal arrow indicates dependencies in a single Scrum team (generally low risk).
↖ An upward diagonal arrow indicates dependencies across time and teams
(generally a medium risk).
↑ A vertical arrow indicates a cross-team dependency but during the same sprint cycle
(this is considered a high risk as the delay in one team will put the entire Nexus at risk).
↙ A downward diagonal arrow indicate external dependencies across time
(considered a medium risk).
↓ A downward vertical arrow represent another sprint cycle dependency but external to the Nexus team
(this is considered a high-risk dependency).

Visualizing dependencies helps the Nexus Team to understand complexities and the NIT can then attempt to re-allocate work items between teams, keeping dependent work in one team and thereby negating or at least minimizing cross-team dependencies and risk.

Exceptions to this practice will be if the NIT decides to distribute work across teams to facilitate the delivery of a specific item during this sprint, which is too large to be completed by a single team.

Here are some ways in which dependencies and risk can be minimized:

- Moving work between teams in order to minimize handover and cross-team dependencies.
- Moving people between teams so that there are fewer cross-team dependencies. This may significantly reduce delivery risk if certain skills are rebalanced across teams for a sprint or two in order to minimize skills dependencies in teams.
- By splitting items in different ways or by reshaping the work to be done, it may be possible to eliminate dependencies.
- When plotting work on the board, try to highlight all of the risks as early as possible in the planning of activities, and try to make cross-team in-sprint dependencies visible as early as possible.

Refinement and tracking dependencies is not a one-time event and must be revisited throughout a Nexus and across sprints.

12.15 More on Nexus sprint planning and Nexus daily scrum

Cross-team refinement information provides input to Nexus sprint planning; Nexus sprint planning should consider current sprint dependencies.

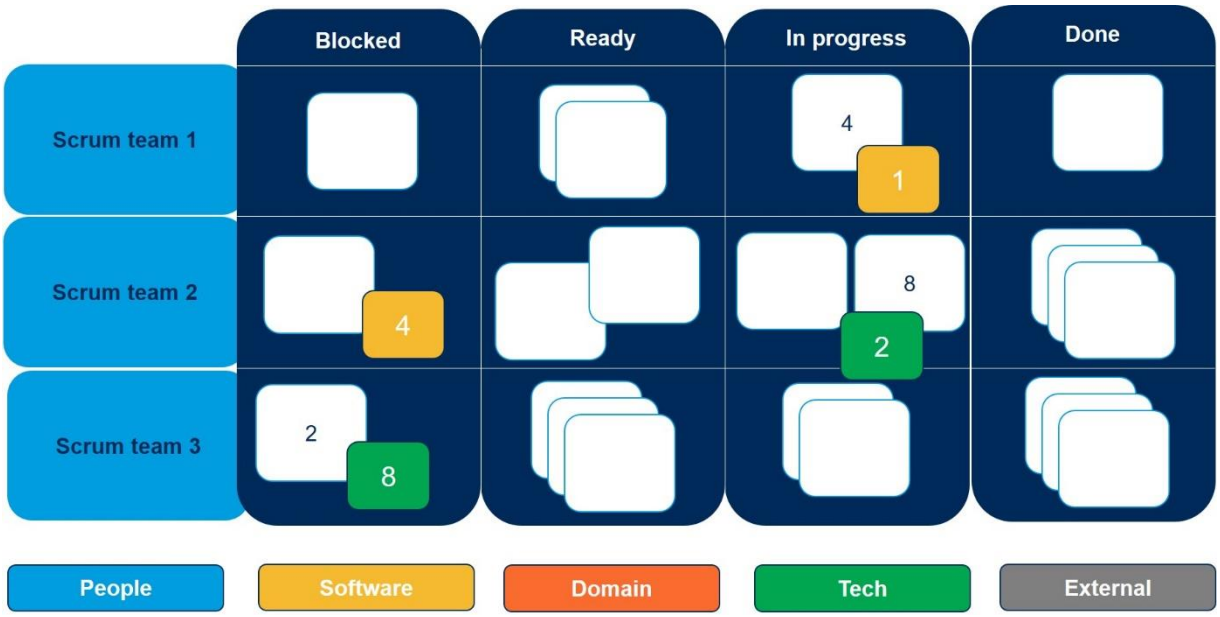
Information about dependencies should also be used as a focal point for daily cross-team synchronization and management of risk and progress during the Nexus daily scrum.

12.15.1 Nexus sprint backlog

Remember that one way of visualizing a sprint backlog is to create a Scrum board. Sprint dependencies between the Nexus teams can be managed to create a Nexus Scrum board as in the example below.

Here PBI 1 is being delivered by Scrum team 2, but it depends on PBI 4 being delivered by Scrum team 1, and PBI 2 being delivered by Scrum team 3 depends on PBI 8 being delivered by Scrum team 2.

Figure 31 Working with inter-team dependencies



Picture created by EXIN based on: Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us. and Schwaber, K., & Scrum.org (2021). *The Nexus™ Guide – The Definitive Guide to Scaling Scrum with Nexus.* Scrum.org. https://www.scrum.org/resources/nexus-guide

Using a Blocked column on the Nexus Scrum board, it is possible to indicate inter-team, in-sprint dependencies.

This visualization is important because it highlights dependencies and associated risks inside the sprint; it also encourages a focus on coordination and correct work sequencing between teams.

During the Nexus daily scrum, the Nexus sprint backlog (Scrum board) is normally a focal point and drives conversations during daily scrum.

If a Nexus does not have in-sprint dependencies, teams may alternatively choose to represent the Nexus sprint backlog through the current sprint column within the cross-team refinement board. In this case, the refinement board becomes the focal point for the daily scrum.

13 Implementing and succeeding with Agile Scrum

Research indicates that while Agile initiatives exist in almost all organizations, the large majority of organizations consider their level of Agile maturity to be below par.

Even so, the benefits of Agile are clear; and they include faster software delivery, an enhanced ability to manage to change priorities better, and increased productivity and alignment between the business and IT.

Scrum is by far the most popular Agile method being used by organizations, so you are in good company.

There is, however, a more somber side to the research findings.

The highest-ranked challenges to adopting and scaling Agile continue to be related to organizational culture. These include general organizational resistance to change, inadequate management support and sponsorship, lack of leadership participation, and the existence of an organizational culture that is at odds with Agile values.

However, despite these challenges, it is clear that the future will be even more Agile – and that those who fail to embrace business agility will be left behind.

13.1 It is all about organizational change

McKinsey stated in an article *The journey to an Agile organization* (2019) that moving to an Agile operating model is tough, especially for incumbents. The enemy of Agile and many other progressive practices is the old siloed operating model and antiquated management approach that goes along with it.

The shift to Agile is the toughest for management who must let go and trust their teams to do the right thing. Devolving decision-making to the lowest possible level is at the heart of the success of Agile, and for an industrial age company to get this right is the most challenging endeavor that the organization will face.

The experts are unanimous in their verdict. Enterprise-wide Agile transformation is needed, and it must be comprehensive and iterative to realize the benefits that Agile brings.

In their article, McKinsey highlights the key areas of change as:

- People
- Structure
- Processes
- Systems and tools

People
The key to success in the People category is getting the leadership of the organization to buy into their changed role. All management staff must be trained to shift their focus from supervising work and people to providing the vision for their team to latch on to and use to get things done in multi-disciplinary, self-managing teams. The job of management is also to inspire and coach, rather than to direct. Managers should help model work (WHAT), but not dictate HOW work will be done.

This implies that all team members need to contribute to getting things done and also collectively decide how things will get done. If you think about what happens in a sprint, this is exactly what Scrum proposes.

As part of coaching, managers need to focus much more on helping people to grow and acquire new skills and competencies. Effectively, Agile implies that a manager's job changes from managing activities, processes, and procedures, to empowering people and building teams. A good way of looking at work now is rather to recognize the value streams and how people and processes contribute to the creation of customer value.

Although this seems simple, it is infinitely hard for managers to shake off the Taylorist management behaviors that formed part of their development as managers and embrace their new role in the organization of servant leader, enabler, coach, and mentor.

It also means that influence is becoming much more important in organizations than status or position, and the natural outflow of this will be seen in the next section.

Structure
Flatter organizations and cross-skilled employees are the natural outflows of this different way of doing business.

Now a more mission-oriented or value-driven approach will determine what teams look like and eventually what the organization looks like. This, in turn, will impact the approach to workforce sizing and location.

Because teams work as multidisciplinary units and are often temporary structures, it means that organizations need to simplify and de-layer reporting structures greatly. The Agile organization is by default more decentralized, and large corporate head offices become a hindrance rather than a help.

This complicates governance and as a result, governance models need to be simplified and streamlined. We can no longer do things to protect ourselves because another organization found it meaningful. We have to understand risk intimately and build and continually refine governance structure, methods, and related controls, specifically the ones based on the risk of the organization.

Processes
Rigid process and conformance regimes often hinder organizations today more than they help them. Processes need to be defined at a high level and leave lots of freedom for teams to define their own way of conforming to the high-level guidance. This implies that each team may have its own way of doing (procedure) the same

thing (process) – and as long as output, outcomes, and essential controls to adhere to are clear, it should suffice.

Organizations often use processes and procedures as a means of measuring performance, but if the process or procedure is not the same, performance metrics should now become almost entirely outcome-based.

Systems and tools
Management should ensure the organization has the right tools and systems and that these are streamlined to support an Agile way of working.

This also implies that the architecture of the enterprise will only be defined centrally in principle, allowing the design and evolution of architecture based on requirements over time.

It is useless to deliver valuable increments if they cannot be used immediately. Automation, especially for products and services realization and delivery, should be a high priority.

In a software setting, this should mean an automated delivery pipeline that automates the testing and integration processes to enable fast and continuous delivery, using, for example, the three DevOps ways.

From an IT infrastructure perspective, it means building flexibility and scalability. The adoption of cloud methods and architectures is increasingly becoming unavoidable.

13.2 Facilitating the change

We have already described the ADAPT and ADKAR models earlier that serve as a primer to facilitating change.

Adopting Lean management principles is also a way of facilitating evolutionary organizational change that is inclusive and less prone to pushback.

An advantage of Lean management is that it starts with management and is driven by management in their new role as a servant leader.

The reason this is so important is that according to the Prosci *Best Practices in change management benchmarking report* (2021), the primary reason why managers resist change is a fear of losing control and authority. If managers own the new way of doing things better and have bought into the new way of working, change is much easier facilitated throughout the rest of the organization. With one proviso, managers need to be willing to change their behavior, showing employees it is safe to change theirs.

The report continues to cite the top reasons for both employees and managers as follows:

Employees	Managers
Lack of awareness	Fear of losing control and authority
Fear of unknown	Lack of time
Lack of job security	Comfort with the status quo
Lack of sponsorship	What's in it for me?
No involvement in the design	No involvement in the design

You can expect the biggest resistance from those who have the most to lose (perceived loss in their minds, not actual), and you can also expect some form of an alliance of parties against the change initiative.

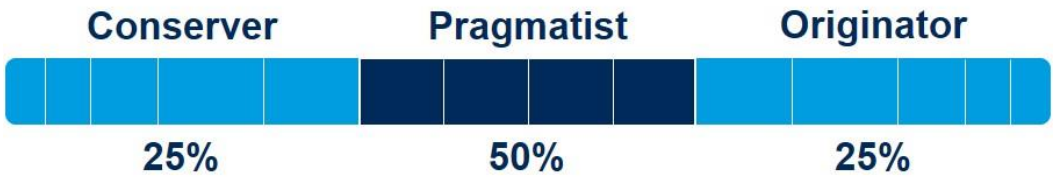
But there will be some in the organization that wholeheartedly embrace the change, mostly because they are dissatisfied with the status quo. A very small group will support it because they like to change and find it exciting.

Chris Musselwhite and Robyn Ingram refer to this range of individual response to change as the continuum of change, and they created the Change Style Indicator that characterizes individuals as falling into one of three key categories indicating an individual's perception of change.

The three categories on the continuum are:

- Conserver
- Pragmatist
- Originator

Figure 32 Scale representing three types of users during a change initiative



Picture created by EXIN based on: Underwood, J. (2013). *Musselwhite and Ingram's Change Style Indicator.* https://innovategov.org/2013/08/15/musslewhite-and-ingrams-change-style-indicator/

- The **conserver** resists change because of fear of the unknown and the uncertainty it brings.
- The **pragmatist** lies in the center of the continuum and changes when it is absolutely necessary.
- **Originators** are completely comfortable with implementing change and have a "let's try it out and see what happens" mindset.

Although varying significantly in their perceptions, each of these contributes valuable insights and traits that can benefit the change initiative.

The conserver's cautious nature should be utilized in the planning role. They prefer incremental (evolutionary) change and will implement change in a way that transitions the organization without causing significant disruption to the business.

The pragmatist facilitates, cooperates, and mediates, whereas the originator provides vision, energy, and novelty.

Team-centered, the pragmatist views both sides and often operates as the mediator to identify acceptable common ground. Proper training or a pilot project may help pragmatists to embrace the change.

The originators are the visionaries in change initiatives. The downside is that implementation realities and practicalities can be an afterthought for them as they are the big-picture people.

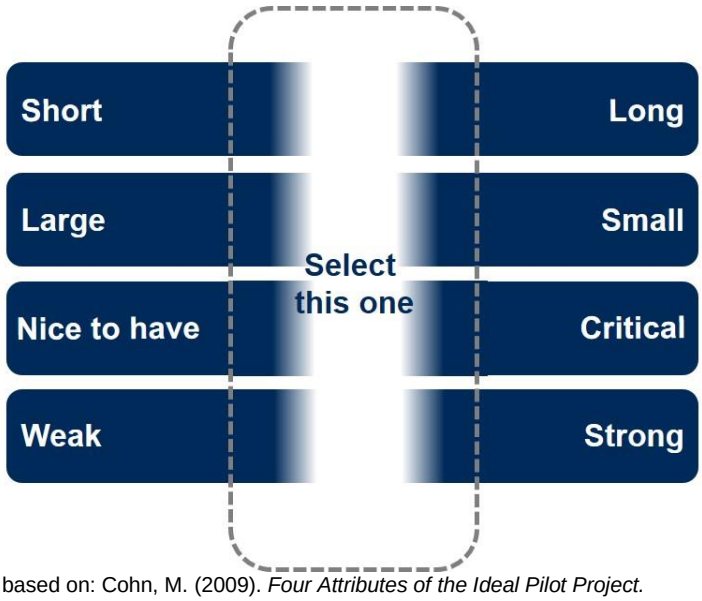
If organizations tap into the originators' willingness to apply new ideas, an organization can generate new ways of doing business, and streamline efficiencies and effectiveness, resulting in more profitable organizations and more value for customers.

13.3 Implementation approaches using pilots

Once it becomes known that the organization is adopting Agile Scrum, all eyes will be on the projects. Some will hope they will succeed, others will be wishing they will fail, but the majority will reserve judgment till later.

It is imperative that the first projects succeed, and as such, the pilot project should be carefully chosen. It is advised to follow the Goldilocks principle relating to four key metrics when choosing your first pilot.

Figure 33 Choosing a good pilot program



Picture created by EXIN based on: Cohn, M. (2009). *Four Attributes of the Ideal Pilot Project.* https://www.mountaingoatsoftware.com/blog/four-attributes-of-the-ideal-pilot-project

Duration
Selecting very short projects will result in opponents to Scrum claiming that Scrum works only on short projects.

However, if you select a project that is too long, you run the risk of not being able to claim success until the project is over.

Many traditional managed projects claim to be on track, yet in the end, are over budget and late, so a Scrum project proclaiming the same may not be very convincing.

It is best to select a project with a length that is near the middle of what is normal for your organization. If longer than six months, break it up into sub-projects with well-defined outcomes and do the first three months as the showcase or pilot.

This gives a team enough time to start getting good at working with sprints to enjoy it and to see both product and team benefits. A three to a four-month project is also usually good enough to demonstrate that Scrum will lead to success in even longer and larger projects.

Size
If possible, select a project where you can start with one team whose members are all collocated.

Always start with one team, even if the pilot project will grow to include more teams eventually, and also attempt to select a pilot that will not grow beyond five teams, even if such projects are common in your organization.

Initially, don’t choose projects where you need to coordinate work between many Scrum teams as that is setting yourself up for failure. You will probably not have time to grow from one team to more than five anyway.

Importance
It may be tempting to select a low-importance, low-risk project as a pilot. The risk is low if things go badly; people may not even notice a failure.

Your critics, however, will keep a close eye on what you are doing, and discard your efforts because it was too easy in any case.

Instead, pick an important project with relatively low risk. Also, remember that some of the aspects that teams need to get right in order to transition to Scrum are difficult; if the project isn’t important, people may not do all that is required of them.

Engage with the business sponsor(s)
Adopting Scrum requires changes throughout the business, not just technical project resources. Having a business sponsor with an interest in Agile and the time and desire to work with the pilot team is crucial.

A committed and engaged business sponsor can help the team to resolve conflicts and to challenge entrenched processes, practices, and controls that are rigorously enforced by departments or individuals. Product Owners, especially during pilots, play a critical role in facilitating communication with the business and between the

business and the Scrum team. It is not uncommon to see an increase in Product Owner utilization during these times.

Similarly, proper sponsorship shows management commitment and is a useful tool for promoting the success of the project, especially if they say it went better than expected.

13.4 Spreading Agile Scrum throughout the organization

What happens after a successful pilot? How do you scale the use of Agile Scrum in the organization?

Well, you need to get the rest of the organization to adopt Agile and you can either take a big-bang approach or do a phased implementation. Although many organizations say that big bang worked for them, it is recommended to use a phased approach.

Sometimes multiple pilots are also useful, after success in one area shows that it will work in another. You will often hear people saying: “it’s fine for the XYZ department, but it will not work in our area as it is not the same”.

You can use a scaled vertical and horizontal approach:

- **Scaling vertically** scaling within a business unit
- **Scaling horizontally** scaling across the organization

Scaling vertically is easier than scaling horizontally. It is much easier to teach other teams in the same business unit how to use Agile as the environment is similar and many of the lessons learned can be applied easily across the business.

A pilot helps to establish and prove the worth of different teams’ contribution to and use of Scrum. Since Scrum teams are cross-functional, members of several teams within the business units would have been involved in the pilot.

Split-and-seed approach
Using one or more team members of the pilot Scrum team as members of new Scrum teams means that immediately someone with some positive experience using Scrum is available to other Scrum team members who may have issues, doubts or just want to know or learn something about the new way of working. This approach is often called the split-and-seed approach.

Grow-and-split strategy
An interim strategy can also be applied: grow-and-split. In this scenario, the membership of the team involved in the pilot is increased, old members help new members for two or three sprints, and then the new, enlarged group is split to seed new teams. In this scenario, use both more and less experienced members when seeding a new team.

Coaching
The use of coaching is also an effective mechanism to expand the Agile program, and either internal or external coaches can be used.

Here coaches spend a few hours a week with their allotted teams and share with them how to best use Scrum and also lessons learned within the context of the organization. Internal coaches are therefore not permanent members of the Scrum team they coach.

Internal coaches have a better understanding of context; however external coaches may know Scrum better and have more experience to bring to the table.

It is recommended to use a combination of the techniques described here to spread the use of Scrum across the organization.

13.5 Dealing with people

People resist changing to Scrum for many reasons. Some may use well-reasoned logic and fierce arguments; another may resist by quietly sabotaging the change effort.

In his book *Succeeding with Agile* (2009), Mike Cohn shares some great insights in this regard; some of these insights will be explored here.

You will often hear people making misconstrued arguments or acting based on their mistranslated understanding because their resistance makes it impossible for them to hear the correct message.

For instance, someone may say or think: *You think no documentation is a good idea? I'll show you no documentation.* When the agile principles were defined, it was never the intention to imply that documentation is not important. But people resisting change may latch onto something like this and proceed to write nothing down, even if the team has agreed that something should be done.

Others may resist by quietly ignoring the change, working in the old way as much as possible, and waiting for the next change to come along and sweep Scrum away.

So how do people resist changing their behavior and how do you deal with it? The first thing is to identify how people are resisting, actively or passively.

- **Active resistance** means that actions are taken to impede or derail the implementation of Agile Scrum.
- **Passive resistance**, on the other hand, is evident when someone fails or neglects to act. They feel that if you ignore it long enough, it will go away.

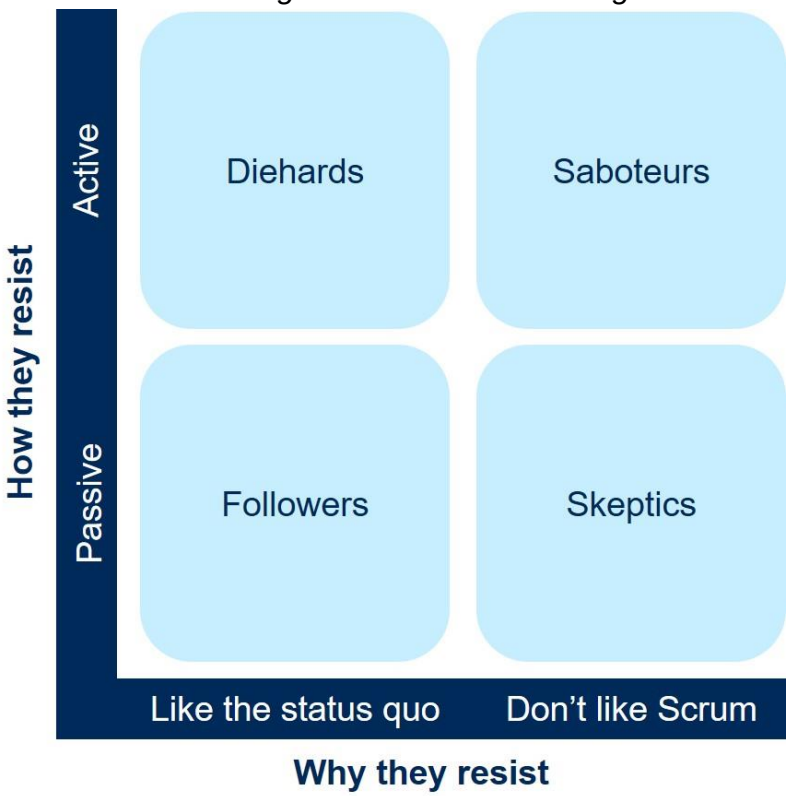
Secondly, you need to determine why people are resisting change. Here two main reasons are usually evident:

- **Like the status quo**: a fear of the unknown
- **They don't like Scrum**: A genuine dislike for Agile and Scrum

They may have a bad experience as a reference (it does not have to be their own experience) or alternatively, the ideas propagated in Agile are just too foreign to them and therefore objectionable.

So, there are two ways in *how* people resist, and two reasons *why* they resist. Based on that a 2x2 matrix can be created, as shown here.

Figure 34 Understanding how to deal with change resistance



Picture created by EXIN based on: Cohn, M. (2010). *Succeeding with Agile.* Addison-Wesley.

Each quadrant is given a name descriptive of the person who resists in the way indicated by the labels on the axes.

13.5.1 Skeptics

Skeptics are those who do not agree with the principles or practices of Scrum but who only passively resist the transition. Skeptics politely argue against Scrum and forget to attend the daily scrum far too often. They are individuals who are trying to stop the transition, as opposed to those who find it a bit foreign but are willing to give it a try.

Some useful means to overcome resistance from skeptics include:

- Let time run its course
- Provide training
- Solicit peer anecdotes
- Appoint a champion skeptic
- Push the issue
- Build awareness

13.5.2 Saboteurs

Like skeptics, saboteurs resist the transition more from a dislike of Scrum; however, a saboteur provides active resistance by trying to undermine the transition effort, perhaps by continuing to write lengthy up-front design documents, and so on.

Here are some tools that have proven useful when dealing with saboteurs:

- Success
- Reiterate and reinforce the commitment
- Move them
- Fire them (yes you read correctly, sometimes it is the only solution because they “poison the well!”)
- Be sure the right people are talking

13.5.3 Diehards

On the left side are those who are comfortable with the status quo. They are used to doing things in a current manner, and their work often creates prestige value with co-workers. In principle, they may not oppose Scrum as such, but they are opposed to any changes that may put their current position at risk.

Diehards like the status quo and will actively resist changing. They will even try to prevent the transition by rallying others to their cause.

Here is how you can deal with diehards:

- Align incentives
- Create dissatisfaction with the status quo
- Acknowledge and confront fear

13.5.4 Followers

Followers like the status quo and passively resist changing. Followers are not enraged by the prospect of change, but they will do as little as possible, hoping the change is just a passing fad. The way to win them over is to show them that Scrum has become the new status quo.

Lastly, the tools you may want to employ when dealing with followers:

- Change the composition of the team
- Praise the right behavior
- Involve them
- Model the right behaviors yourself
- Identify the true barrier

13.5.5 How to deal with each type of resistance

Cohn’s advice is pragmatic and simple to use. However, you must consider the culture of the organization, the society, and the legislative environment you operate in before deciding how you will deal with those who resist change.

13.6 Building the right environment

One of the first challenges you will face as you start transitioning to Agile Scrum is that of working as self-managed, cross-functional teams. Considering the value statements, we can surmise that Agile teams primarily work in the following way.

Members must…

- work as one Scrum team.
- do one job at a time.
- do work in short iterations.
- deliver something valuable each iteration.
- focus on business priorities and outcomes.
- inspect and adapt everything.

Looking at the above one can immediately spot some challenges for traditional organizational structures, facilities, and roles.

If Scrum teams are cross-functional, but they will have to work closely and regularly collaborate, it would be best to create a physical space for them to operate.

Visual management is key, and the team should all be able to see key information radiators like the Scrum, Kanban, or Nexus boards. As such, open space, re-configurable workstations are ideal. That being said, sometimes someone may need to concentrate or meet with someone, and the humdrum of common spaces becomes counterproductive.

Make sure break-out facilities are available for this purpose but also make sure that there are few enough so that everyone cannot sit and work in these areas permanently. Day-to-day activities must take place in the shared common space. This is sometimes referred to as caves-and-commons.

It is especially important to have space where the daily scrum can take place. Team artifacts must be able to remain visible even after the daily meeting is done.

If a software is used for Scrum/Kanban boards, make sure that they are permanently projected against a wall in the common area where the daily meetings take place.

13.7 Working as virtual and remote teams

If there is one thing we learned from the 2020 pandemic, it is that virtual teams are here to stay. Even before 2020, many Scrum teams were not co-located and were often spread across the globe, and as anyone who is part of a virtual team knows, this presents some challenges.

But what is a virtual team?

Virtual teams are teams comprised of people pursuing the same objective even though they are not in the same physical location. The various team members can work from home or out of offices in different cities or countries. For virtual teams to work, it is important to be aware of the formula for virtual team success.

There are currently many different views on the issue of virtual teams and how they should work, but the good news is that part of working as a successful virtual team is already built into the DNA of Agile Scrum. The same principles and practices that help multi-disciplinary teams work together as a cohesive Scrum team, also help virtual teams to work together as a cohesive Scrum team.

So, let's have a look at virtual Scrum teams and how they can work.

- **Create a clear process for all Scrum activities**. This does not need to be detailed, but everyone should know exactly when and how what happens. Specifically, look at:
    - Sprint planning
    - Daily scrum
    - Sprint reviews
    - Sprint retrospectives
    - Product backlog refinement
    - Ad-hoc problem-solving meetings
- **Use a common set of collaboration and management tools**. Whichever tool you use, make sure everyone has the same view of tasks, progress, and issues, and have this view always available.
    - Also, make sure that collaboration tools work for the team and be aware that **face-to-face contact is very important.** Seeing one another is extremely important as seventy percent of communication is non-verbal.

  Although the best form of communication is people in one room, the reality is that this often cannot happen. Therefore, all virtual meetings must be **video-based team meetings** and not only voice. Being on video also means that people cannot multitask.
    - When participants connect it should be from **a standard and predefined venue** that is conducive for the productive contribution: not from your bed or from the train for instance
    - **Use interactive questions and voting software** where people need to answer questions or vote about issues during the meeting, to increase interaction.
    - You may also create team spaces for **casual and normal daily interaction** for the team, such as creating a WhatsApp group for the team.
- Make sure **expectations for all activities are set up-front**, even if you have repeated them for the umpteenth time.
    - What does the team expect from participants for this activity?
    - What can participants expect from the team?
    - What are the output and outcomes expected from the event?
    - Give everyone the opportunity to highlight if they have an expectation that does not form part of the norm for the event.
    - If no one follows the meeting rules, the meeting ends, and everyone must reschedule. Don't let the rules slip once.

- **A good facilitator is key**. Here, Scrum Masters can make a huge contribution but remember whose meeting it is. If they don't appoint you as a facilitator, you are not the facilitator. Daily scrum is the Developers' meeting, product backlog refinement is the Product Owner's meeting, etc.
    - Avoid tools like PowerPoint; it is just a one-way communication medium. Conversation and participation of everyone become even more important in virtual meetings!
    - A great tip comes from Mike Dwyer – he calls it the NOSTUESO rule – **No One Speaks Twice Until Everyone Speaks Once**.
    - This also creates space for people to speak up, and research has shown that if a participant speaks up in the first five minutes, they are three times as likely to speak again.
    - Also, when a woman speaks first, the participation of other women has been shown to be much higher.
    - Ensure the establishment of good relationships and positive but frank communications in the Scrum team.

14 Accountabilities – Scrum & Nexus events and practices

Throughout the book, events or practices are referenced, together with an indication of who should do it (for example the Scrum Master, Developer, Product Owner). However, nowhere will you find a definitive list of what each role is accountable and responsible for.

However, it is of course important to understand what is expected of each of the key roles in Scrum. This chapter has three sections, one for each role. In each of these sections, the accountabilities and responsibilities of each role are plotted and described against Scrum events and practice.

14.1 The Product Owner

14.1.1 Sprint planning

	Accountabilities and responsibilities
Attendees	Scrum team but owned by Developers.
Preparing for sprint planning	Ensure that the product backlog is refined and in line with current business needs.
Defining a sprint goal	Re-iterate the product goal and ensure that all know current business needs. Agree to the sprint goal.
Defining the sprint backlog	Give input where required and ensure business needs are reflected in backlog items. If discrepancies exist, highlight them and negotiate change with Developers if possible.
Doing estimation and task breakdown	Validate assumptions and provide greater clarity on requirements and business needs.
Creating the sprint backlog	Provide input if consulted.
Refine product backlog	Collect input from team members, update the product backlog, and re-order items based on new insights.

14.1.2 Daily scrum

	Accountabilities and responsibilities
Attendees	Developers and Scrum Master – sometimes other stakeholders including the Product Owner.
Inspect & adapt	Give input if required and ask probing questions.
Refine product backlog	Collect input from team members, update the product backlog, and re-order items based on new insights. Consider capabilities, imperatives, time, and budget.

14.1.3 Sprint review

	Accountabilities and responsibilities
Attendees	Scrum team and critical stakeholders identified by the Product Owner. Sometimes chaired by the Product Owner. It is, however, the Team's meeting.
Kick-off	Ensure attendees understand why they are there. As this event deals with stakeholders outside of the Scrum team, the Product Owner takes the lead with the sprint review.
Demonstrate and inspect	Help demonstrate the delivered increment.
Get feedback on what you do	Ask probing questions.
Get feedback on changes in the environment	Record any changes in the environment.
Issues and problems	Specifically, note dependencies that may impact the order of delivery.
Refine product backlog	Collect input from team members, update the product backlog, and re-order items based on new insights. Consider capabilities, imperatives, time, and budget.

14.1.4 Creating & maintaining the product backlog

	Accountabilities and responsibilities
Requirements gathering	Accept gathered requirements. Sometimes Product Owners collect requirements themselves; however, this may be done by customers and users, analysts, or other team members. Product Owners must vet requirements and often go back and refine requirements before adding them to the backlog.
Initial backlog refinement	Consider initial requirements, collect input from team members, and populate the product backlog based on business requirements and other insights. Consider capabilities, imperatives, time, and budget.
Continual backlog refinement	Collect input from team members, update the product backlog, and re-order items based on new insights and changing business requirements. Consider capabilities, imperatives, time, and budget.

14.1.5 Sprint retrospective

	Accountabilities and responsibilities
Attendees	Scrum team; this event is driven by Developers with input from other roles.
Inspect	Contribute but don’t lead.
Adapt	Contribute but don’t lead.
Sharing learning	Although learning is primarily team-focused, the Product Owner may choose to share learnings with other Scrum teams working against the same backlog.

14.1.6 Nexus sprint planning

	Accountabilities and responsibilities
Attendees	Members from participating Nexus Teams & one Product Owner.
Preparing for sprint planning	A Nexus is never executed cross-product backlog, the Product Owner plays a more dominant but still guiding role helping teams involved to plan the Nexus. In reality, the Product Owner is the convener of the Nexus so plays an important role in constituting the Nexus and involving teams in the Nexus. Their role otherwise is similar to normal sprint planning.
Kick-off	The Product Owner is also the Nexus owner and convenor; planning and executing kick-off meetings is, therefore, the role of the Product Owner.

14.1.7 Nexus sprint review

	Accountabilities and responsibilities
Attendees	Members from participating Nexus Teams & one Product Owner.
Execution	Although the Nexus sprint review's content and format are similar to sprint reviews, it requires coordination and planning. There are no separate sprint reviews, only the Nexus sprint review in a Nexus. This meeting is owned and run by the Product Owner.

14.1.8 Nexus sprint retrospective

Same as in sprint retrospectives.

	Accountabilities and responsibilities
Attendees	Scrum team; this event is driven by Developers with input from other roles.
Inspect	Contribute but don't lead.
Adapt	Contribute but don't lead.
Sharing learning	Although learning is primarily team-focused, the Product Owner may choose to share learnings with other Scrum teams working against the same backlog.

14.1.9 Nexus product backlog refinement

Nexus product backlog refinement is not only a continual activity but also defined as a Nexus Event. The event is run and coordinated by the Product Owner.

14.2 The Scrum Master

Important note: Scrum Masters enable Scrum team members and *never* drive any event or action. They facilitate, coach, and assist.

14.2.1 Sprint planning

	Accountabilities and responsibilities
Attendees	Scrum team but owned by Developers.
Preparing for sprint planning	Help with coordinating and setting up the meeting.
Defining a sprint goal	Act as facilitator, mediator and help resolve conflicts.
Defining the sprint backlog	Act as facilitator, mediator and help resolve conflicts.
Doing estimation and task breakdown	Act as facilitator, mediator and help resolve conflicts.
Creating the sprint backlog	Although the sprint backlog belongs to the Developers, they often ask the Scrum Master to help with the creation, documentation, and upkeep of the sprint backlog.
Refine product backlog	Act as facilitator, mediator and help resolve conflicts.

14.2.2 The daily scrum

	Accountabilities and responsibilities
Attendees	Developers and Scrum Master – sometimes other stakeholders including the Product Owner.
Inspect & adapt	Facilitate the conversation, provide coaching when required, mediate if necessary.
Dealing with impediments	A significant function of a Scrum Master is helping Developers to deal with impediments; usually, this involves facilitating meetings, acquiring assistance, resources or skills, and frequent communication and follow-up with affected stakeholders.
Tracking progress	Keeping information radiators updated is usually done by the Scrum Master. It is not their responsibility, but the task is more often than not delegated to the Scrum Master by Developers.
Coaching the team	A significant function of a Scrum Master is coaching and teaching the team to ensure they are proficient with Scrum practices and aware of the latest developments in Scrum.
Refine product backlog	Facilitate the conversation, provide coaching when required, mediate if necessary.

14.2.3 Sprint review & retrospectives

	Accountabilities and responsibilities
Facilitating	Facilitate the conversation, provide coaching when required, mediate if necessary. The Scrum Master prepares the sprint retrospective extensively.

14.2.4 Creating & maintaining the product backlog

	Accountabilities and responsibilities
Requirements gathering	In this activity, the Scrum Master offers support to the Product Owner.
Initial backlog refinement	Facilitate the conversation, provide coaching when required, mediate if necessary.
Continual backlog refinement	Facilitate the conversation, provide coaching when required, mediate if necessary.

14.2.5 Nexus

Not all Scrum Masters from the participating Scrum teams will participate in Nexus-specific events like Nexus planning, the Nexus review, and retrospective. However, at least one Scrum Master must be part of the Nexus Integration Team as the Scrum Master. In this role, much time is spent coordinating and refining activities between teams. Although this may not be a full-time job, it is better if it is a permanent and focused role in a complex Nexus. The work of the Nexus will ALWAYS take precedence over Scrum teamwork.

14.3 Developers

14.3.1 Sprint planning

	Accountabilities and responsibilities
Attendees	Scrum team but owned by Developers.
Preparing for sprint planning	Sprint planning is the job of Developers; Developers often delegate the task to prepare meetings to the Scrum Master.
Defining a sprint goal	The sprint backlog is defined by Developers in consultation with the Product Owner.
Preparing the sprint backlog	The sprint backlog items are selected by Developers with the input of the Product Owner and sometimes with the help of the Scrum Master.
Doing estimation and task breakdown	Developers do estimation and task breakdown with assistance and input from the Product Owner and help from the Scrum Master.
Creating the sprint backlog	Developers create the sprint backlog; they often ask the Scrum Master to be the custodian of the sprint backlog which makes it easier because then only one person is tasked with the upkeep of the sprint backlog.
Refine product backlog	Developers should help the Product Owner with product backlog refinement. Although this is an ongoing activity, during sprint planning it is common to do a quick refinement session based on what was learned during planning.

14.3.2 The daily scrum

	Accountabilities and responsibilities
Attendees	Developers and Scrum Master – sometimes other stakeholders including the Product Owner.
Inspect	Developers report on progress, discover dependencies and impediments.
Adapt	Developers decide how to handle impediments and dependencies, and if external help is needed, they may ask the Scrum Master and Product Owner to facilitate a positive outcome.
Tracking progress	Tracking progress is the responsibility of Developers; this is often delegated to the Scrum Master.
Refine product backlog	Any new information that may require refinement of the product backlog should be done as Developers expose issues. The Product Owner owns refinement.

14.3.3 Sprint review

	Accountabilities and responsibilities
Attendees	Scrum team and critical stakeholders identified by the Product Owner. Sometimes chaired by the Product Owner. It is, however, the Team's meeting.
Kick-off	As this event deals with stakeholders outside of the Scrum team, the Product Owner takes the lead with the sprint review.
Demonstrate and inspect	Developers will show other stakeholders what they have done and answer questions.
Get feedback on what you do and the environment	Note that the sprint review is not a sign-off meeting, but rather an opportunity to get feedback from stakeholders, specifically customer and users. Stakeholders often share new requirements or changes to the environment that the Scrum team was unaware of! Feedback sessions may also from time to time highlight issues or problems; this practice is however discouraged as it can quickly get out of hand.
Refine product backlog	With new information, insight, and feedback, it is a great time to do some product backlog refinement, while this information is still fresh in the team's mind.

14.3.4 Creating & maintaining the product backlog

	Accountabilities and responsibilities
Attendees	Product Owner with the help of Developers.
Requirements gathering and product backlog refinement	Developers are not often involved in initial requirements gathering but often unearth new requirements as part of their interactions with users during a sprint or when planning and executing a sprint. Requirements need to be given to the Product Owner, normally as part of product backlog refinement.

14.3.5 Sprint retrospective

	Accountabilities and responsibilities
Attendees	The Scrum team.
Inspection	Various techniques are available; one of these is to ask what went right, what went wrong, and what can be improved. Here analysis of issues and blocker tickets collected during the sprint can be helpful.
Adaption	Similarly, the team can create means to improve by asking themselves what they should change, improve and avoid – and turning this into actionable plans.
Sharing learning	Especially in scaled implementations – sharing lessons learned with other Developers may be helpful.

14.3.6 Nexus sprint planning

	Accountabilities and responsibilities
Attendees	Members from participating Nexus Teams & one Product Owner.
Coordination	Nexus sprint planning will be done by representatives of all the participating Scrum teams, called the Nexus Integration Team. Often this involves experienced Developers seconded from the participating Scrum teams.

14.3.7 Nexus sprint review

During a Nexus, a combined sprint review is done where the work of the combined Nexus is reviewed including all increments that formed part of the Nexus. As a minimum, representatives of all the Scrum teams in the Nexus will show their work and get feedback. The Nexus sprint review otherwise works the same as a sprint review.

	Accountabilities and responsibilities
Attendees	Representatives of Scrum teams, the Product Owner, and selected stakeholders, including customers and key users, attend the Nexus Scrum review.

14.3.8 Nexus sprint retrospective

All Scrum teams involved in a Nexus must first perform a normal Scrum retrospective, and then all members of teams should repeat the exercise for the Nexus as a whole. This is where lessons shared can lead to significant gains if applied across all teams.

	Accountabilities and responsibilities
Attendees	All members of constituent Scrum teams.

14.3.9 Nexus product backlog refinement

Shared backlog refinement is focused on decomposing product backlog items (PBIs) sufficiently for teams to understand the work they could deliver, and the sequence of delivering the work (over upcoming sprints). The event also allows the teams to focus on minimizing and removing dependencies between teams.

	Accountabilities and responsibilities
Attendees	At a minimum this activity should be attended by members of the Nexus Integration Team; however, broader representation from constituent Scrum teams is advised.

Appendix A – Other Agile methods

This is a compilation of content derived from several public sources which are indicated. The only purpose of this document is to be a reader in preparation for the EXIN Agile Scrum Product Owner and EXIN Agile Scrum Master certifications.

Since the publication of the Agile Manifesto back in 2001, many organizations have adopted different Agile methodologies to enable their teams to achieve their goals and deliverables. The most famous methodology is Scrum, which is widely used in all kinds of companies. In the early days, these Agile practices were only applied by IT departments. In contrast, Agile methods are now broadly used in a wide range of different non-IT business areas.

This appendix will provide an overview and brief explanation of the most practiced Agile methodologies aside from Scrum:

- Crystal
- Extreme Programming (XP)
- DSDM (now called ABC)
- LeSS
- SAFe
- Kanban

It is important to keep in mind that although these methods are all different, they have three things in common: open communication, togetherness, and flexibility. These elements are shared by all of these methods because they are at the core of the Agile Manifesto.

Crystal methodologies[14]

The Crystal family is a group of lightweight Agile methodologies. They are considered lightweight because these methodologies consider the process of secondary importance and instead place the emphasis on other elements.

These elements are:

- People
- Interaction
- Community
- Skills
- Talents
- Communications

Crystal methodologies were developed by Alistair Cockburn in the 1990s. They were created as a result of research conducted by Cockburn that showed that Developers weren't using formal methodologies as they were intended to be used, but still they were delivering successful projects. Cockburn differentiates between some important aspects:

- **Methodology**: a set of elements (e.g. practices, tools).
- **Techniques**: the skill areas (e.g. developing use cases).
- **Policies**: definitions of how the organization must behave.

As a result of his research, Cockburn defined the behavior of people in teams as follows:

- People are communicating beings, doing best face-to-face, in person, with real-time question and answer.
- People have trouble acting consistently over time.
- People are highly variable, varying from day to day and place to place.
- People generally want to be good citizens, are good at looking around, taking initiative, and doing 'whatever is needed' to get the project to work.

Crystal methodologies are divided into 8 different colors. The colors are used to indicate the 'weight' of the methodology. The list starts with Crystal Clear that is suitable for projects with up to 6 people and carries on through to Crystal Diamond or Crystal Sapphire that is suitable for mission-critical projects that involve a potential risk to human life.

[14] Free after: Crystal Methods. (2021). Retrieved on September 23, 2021 from https://en.wikiversity.org/wiki/Crystal_Methods.

An important point that requires attention as well is that there are seven common properties shared by all of the Crystal family. These properties are:

- Frequent delivery
- Reflective improvement
- Close or osmotic communication
- Personal safety
- Focus
- Easy access to expert users

Frequent delivery
Regular releasing of iterations of the software/product that is in development.

Reflective improvement
The project team is invited to think about how to improve their processes from time-to-time. This not only provides a break from development tasks but also helps to improve their efficiency.

Close or osmotic communication
Communication must flow through the project team. To ensure this happens, the project team should be together in the same room so that communication that supports the progress of the project is encouraged.

Personal safety
Everyone in the team must feel they are in an open and safe environment where they are free to speak up. Negative responses, such as being ridiculed when they ask a question or suggest an idea, must be completely discouraged. This is because people must be able to trust each other and anyone who is the target of negativity is likely to not participate proactively anymore.

Focus
Keeping focus is crucial for success. When the focus is referred to in Crystal methodologies, it refers to two things: progress and direction. Progress means dedicating enough time to a task in a project to ensure that progress will be made. Whilst direction refers to the direction the project is heading in.

Easy access to expert users
An actual/real-life user of the new feature that is being developed must be reachable by the project team to answer any questions and help with solutions to problems. This expert can help the project with their hands-on experience, and ideally, they should be available often, also during meetings or through phone calls.

A technical environment should be available with automated tests, configuration management, and frequent integration.

Frequent integration and testing are essential so that any issues (errors, bugs, etc.) can be spotted early. If integration is continuous, it prevents problems from growing as they are resolved early on.

Extreme programming (XP)[15]

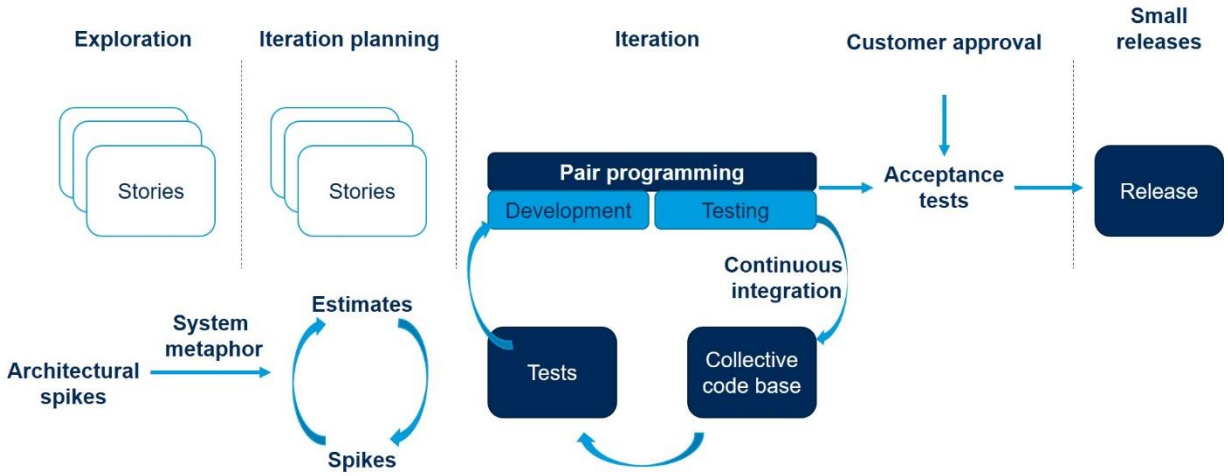
XP is a software development methodology that puts the focus on customer satisfaction. This guarantees that what is being produced is based on the needs of the customer. One of the ways it does this is by transforming user stories into valuable business assets.

One of the defining characteristics of the XP methodology is continual and open communication. Without this important aspect, the practice would not work in line with expectations. An important benefit of this style of communication is an increase in confidence for both the customer and the project team.

XP was created to answer a need in the market to accommodate the continuous changes that service providers were facing while developing a new product or service. Its success can be attributed to the main communication and collaboration aspects between all parties that are involved in the project. This means that customers, managers, project teams and anyone else involved will work together from the moment the scope is set and the stories defined, right up to testing and the final release.

The methodology was designed for small project teams, although large teams are using XP and have reported success as well.

Figure 35 Extreme Programming (XP) at a glance



Picture created by EXIN based on: Meier, J. D. (2019). *Extreme Programming at a Glance.* https://jdmeier.com/extreme-programming-at-a-glance/

How does Extreme Programming (XP) work?

The first step in XP is the collecting of user stories as well as conducting spike solutions for those that could represent a risk.

[15] Free after: Wells, D. (2013). *Extreme Programming: A gentle introduction.* http://www.extremeprogramming.org/ and associated pages.

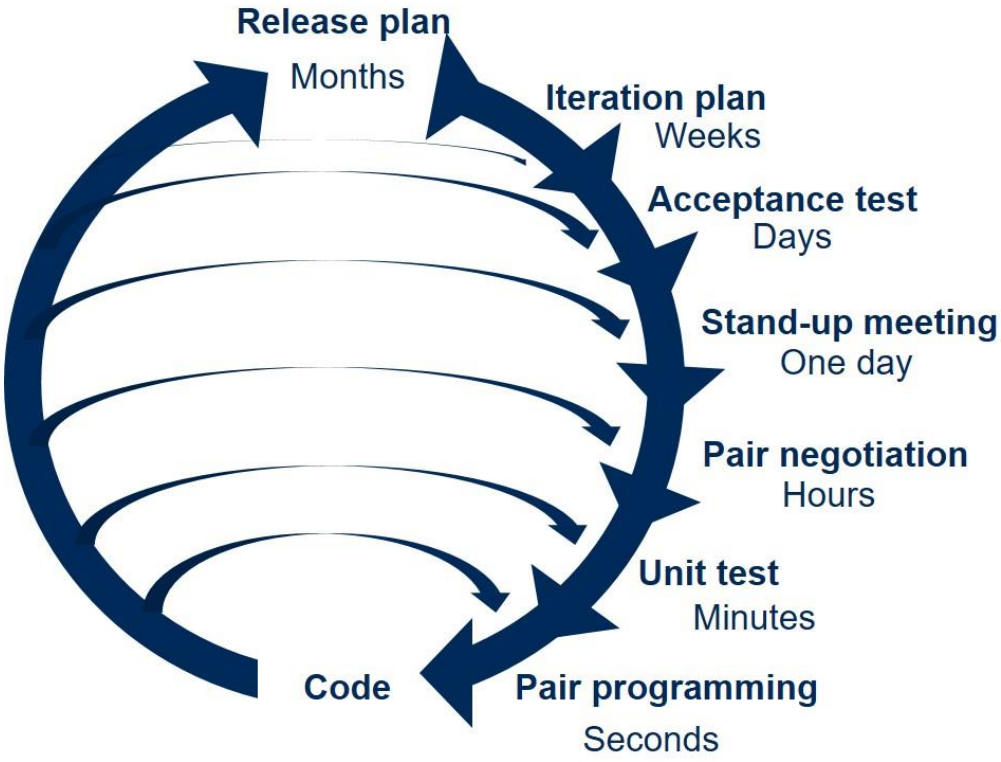
After the initial steps, a release planning meeting must be scheduled. At this stage, it is important to invite all of the required people. This can include customers, project teams, and managers. The main goal at this stage is to define a common goal that is acceptable to everybody. This meeting and setting of the goal are followed by the iterative planning meeting in order to agree on the next releases.

With just these few steps, the project team is already moving in the direction of developing the required features.

The rules of Extreme Programming

XP depends on frequent feedback loops as illustrated in the graphic below.

Figure 36 Planning loops in XP



Picture created by EXIN based on: Wells, D. (2000). *Introducing Extreme Programming.* http://www.extremeprogramming.org/introduction.html

Although some of these feedback loops may seem very long from today's perspective, they were quite revolutionary at the end of the 1990s.

Core to XP is the idea of rules. In short, the rules of XP are about:

- Planning
- Managing
- Designing
- Coding
- Testing

Planning

- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Sprint planning starts with each iteration.

Managing

- Give the team a dedicated open workspace.
- Set a sustainable pace.
- A stand-up meeting starts each day.
- The project velocity is measured.
- Move people around.
- Fix XP when it breaks.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC (class responsibility collaborator) cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

Testing

- All codes must have unit tests.
- All codes must pass all unit tests before they can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

Pair programming

Pair programming is an Agile technique originating from Extreme Programming (XP) in which two Developers team together and work on one computer. The two people work together to design, code, and test user stories. Ideally, the two people would be equally skilled and would each have equal time at the keyboard, but this can also be an effective way of transferring skills from a senior to a junior programmer.

DSDM[16]

Dynamic Systems Development Method (DSDM) is a framework that focuses on the full project lifecycle. Although many still refer to DSDM, the new name for the organization is called the Agile Business Consortium.

This approach dictates that only the minimum work will be done in every step, in order to move forward for the next item. The mindset behind this is that continuous change is a natural part of projects. It is a well-known fact that business requirements can change at a moment's notice, so doing only the necessary work for the required step to be considered complete, saves the project team's effort, resources, and time.

The DSDM framework can be used for different applications in organizations, from the development of a new product or service to finance departments. Any part of the business can benefit from it.

Principles of DSDM

1. Active user involvement is imperative
2. DSDM teams must be empowered to make decisions
3. The focus is on frequent delivery of products
4. Fitness for business purpose is the essential criterion for acceptance of deliverables
5. Iterative and incremental development is necessary to converge on an accurate business solution
6. All changes during development are reversible
7. Requirements are baselined at a high level
8. Testing is integrated throughout the lifecycle
9. A collaborative and cooperative approach between all stakeholders is essential

Source: Agile Business Consortium

The DSDM Framework Phases

The DSDM Framework comprises 6 phases. This consists of a pre-project and post-project phase, and the project phase has four main phases: Feasibility, Foundations, Evolutionary Development and Deployment.

Phase 1. Pre-Project
Phase 2. Feasibility
Phase 3. Foundations
Phase 4. Evolutionary development
Phase 5. Deployment
Phase 6. Post-Project

[16] Free after: Agile Business Consortium (2014). *The DSDM Agile Project Framework (2014 Onwards).* https://www.agilebusiness.org/page/TheDSDMAgileProjectFramework

Phase 1 – Pre-Project

As part of the preparation for the project, a clear objective is defined. The pre-project phase helps to ensure that projects are set up correctly and that only the right projects are started.

Phase 2 – Feasibility

As suggested by the name of this phase, the focus of this part is whether the project in question is feasible from both a technical and cost-effectiveness perspective. This phase should not take up too much time – only enough to be able to decide whether looking into the project is worthwhile or that it is unlikely to be viable.

Phase 3 – Foundations

During this phase, the goal is to understand the scope of the work. This includes the how, who, when, and where of carrying out the actual work. The project lifecycle is also determined by looking at how the DSDM process will be applied.

Phase 4 – Evolutionary development

The Solution Development Team will apply practices including Iterative Development, MoSCoW, or timeboxing to evolve the solution. The aim is to further evolve the solution so that it is accurate, is built-in ‘the right way’ from a technical perspective, and that meets business needs. The team will iteratively develop and test continuously as the project progresses.

Phase 5 – Deployment

This phase focuses on three main activities: Assemble, Review and Deploy. This phase aims to release a final solution or a subset of it. The project is formally closed after the last release.

Phase 6 – Post-Project

During the post-project phase, checks are conducted to see how well business benefit expectations have been met in the form of a Benefits Assessment.

LeSS

LeSS stands for Large-Scale Scrum. This is a method that can be applied when multiple teams are working together on one product or service that is being developed.

LeSS can be applied in cases when:

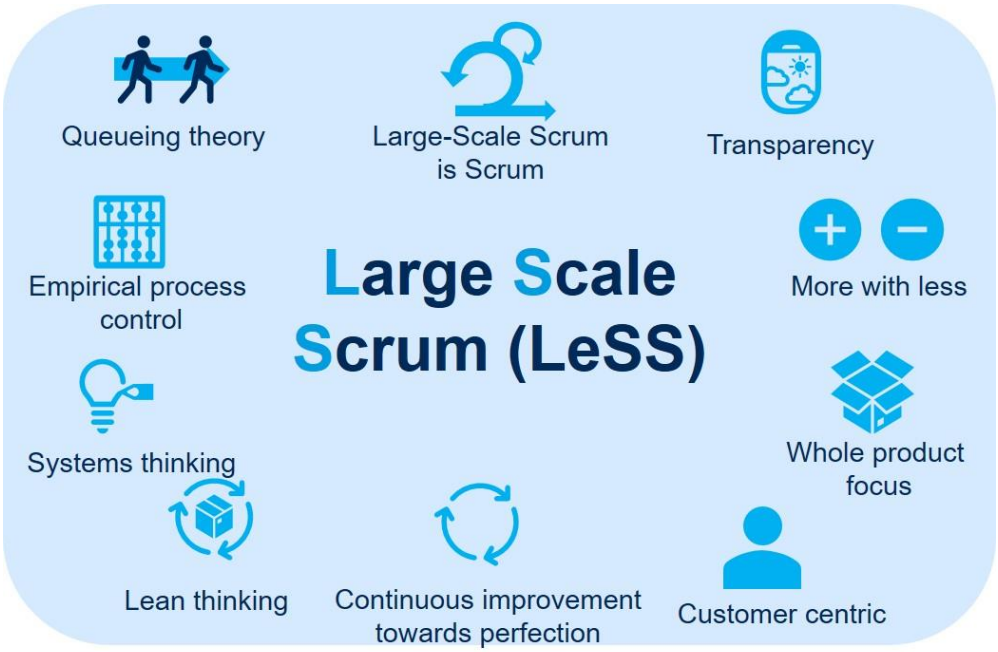
- Multiple teams are required for a specific development
- Multiple teams work together towards one common goal
- Multiple teams work at the same time on a product or service

If your project does not fit these requirements, then it is better to use standard Scrum.

The 10 Principles of LeSS

The 10 Principles of LeSS are summarized in the next Figure.

Figure 37 Concepts used in LeSS



Picture created by EXIN based on: The LeSS Company B.V. (2014). *Large Scale Scrum is Scrum [Visual].* https://less.works/img/principles/principles.pdf

Large-scale Scrum is Scrum

LeSS is still a form of Scrum; however, it is scaled for large products or services development. LeSS provides a set of rules combined with guides to apply Scrum in a multi-team context. It is not a new Scrum framework.

Transparency
The main outcomes must be visible to all parties. Creating short feedback loops increases transparency. Transparency is essential to ensure adaptive control and improvement is possible.

More with LeSS
This principle is central to LeSS and acknowledges that complex organizational solutions are disadvantageous and can cause more problems than they solve. It aims to remove complexity by adopting simpler and different solutions to problems.

Whole product focus
In LeSS, the focus is on the complete product. This is because individual contributing parts of the product have no actual value until they are put together to create the final product. After all, customers buy a whole product, not its parts. Retaining a focus on the ‘whole’ is one of the biggest challenges in large development groups.

Customer-centric
As with product focus, the large scale of LeSS poses a risk to customer satisfaction. In LeSS the Product Owner functions as a connector of customers/users and project teams. To ensure the customer remains the focus, teams are organized based on end-to-end features and components.

Continuous Improvement towards perfection
As one of the two pillars of Lean thinking, continuous improvement towards perfection is also a part of LeSS. In LeSS change is expected, continuous, and embraced.

Lean thinking
Respect for people and continuous improvement are central concepts from Lean adopted by LeSS.

Systems thinking
This is the principle that long-lasting systemic improvements are preferable over quick-fix-solutions.

Empirical process control
This Scrum concept is applied in LeSS to ensure that teams have “just enough process”. This way, they do their work in a cycle of transparency, inspection, and adaptation.

Queueing theory

This theory of how items move through a system queue is a thinking tool that can be used to improve the ability to handle big batches of work. This is especially relevant to Large-Scale Scrum.

Scaled Agile Framework® (SAFe®)[17]

The Scaled Agile Framework is a freely available body of knowledge that was created to address issues that are encountered when scaling Agile beyond a team. This enterprise-scale development methodology combines principles from Lean and Agile. The organization and workflow patterns in the framework have been designed to support organizations in scaling their Lean and Agile practices.

The nine SAFe Lean-Agile Principles

1. Take an economic view
2. Apply systems thinking
3. Assume variability; preserve options
4. Build incrementally with fast, integrated learning cycles
5. Base milestones on an objective evaluation of working systems
6. Visualize and limit work-in-progress, reduce batch sizes and manage queue lengths
7. Apply cadence (timing), synchronize with cross-domain planning
8. Unlock the intrinsic motivation of knowledge workers
9. Decentralize decision-making

The SAFe framework

The Scaled Agile Framework website offers this overview of SAFe. In SAFe, there are four configurations:

- Essential
- Portfolio
- Large solution
- Full

Essential
Essential SAFe is the most basic configuration. It describes the most critical elements needed and is intended to provide the majority of the framework's benefits. It includes the team and program level, which it calls Agile Release Trains or ARTs.

Portfolio
Portfolio SAFe includes concerns for strategic direction, investment funding, and Lean governance.

Large solution
Large Solution SAFe allows for coordination and synchronization across multiple programs, but without the portfolio considerations. In earlier versions of SAFe, this level was referred to as value stream.

Full
Full SAFe combines the other three levels.

---

[17] Free after: Scaled Agile (2021). *Scaled Agile Framework.* https://www.scaledagileframework.com/

Disciplined Agile Delivery (DAD)

Disciplined Agile Delivery (DAD) is a framework that provides context-specific guidance, that suits your enterprise needs, to produce high-quality products quicker. It is a hybrid model, which is formed by a collection of the world's proven Lean-Agile methods such as Scrum, Kanban, XP, Agile Modelling, Unified Process, and many more.

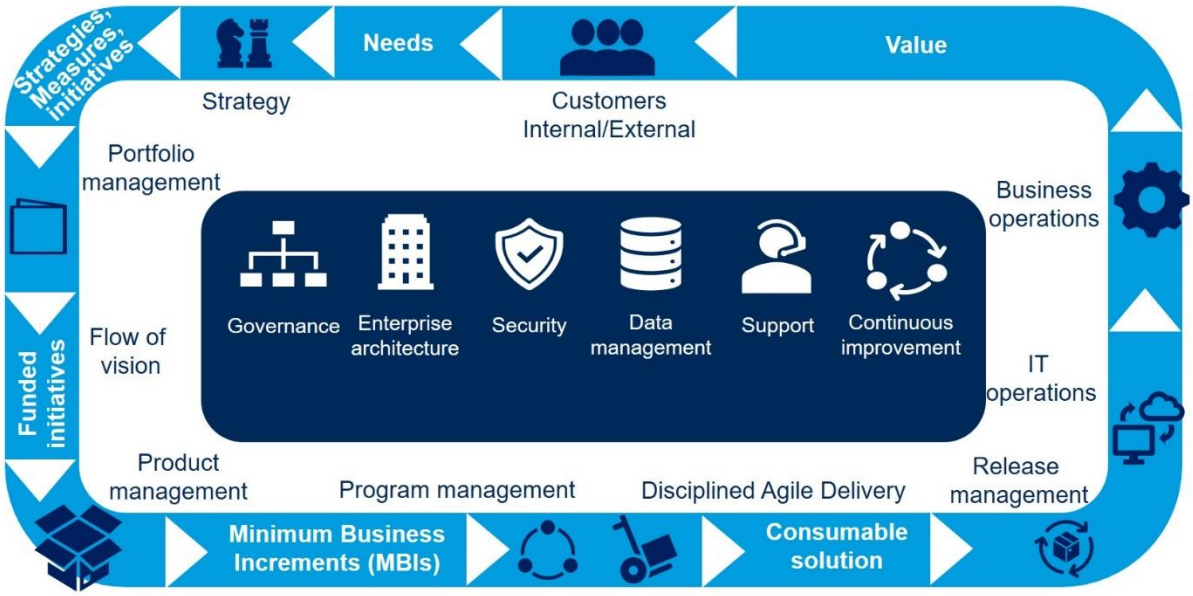
The framework was originally developed at IBM Rational from early 2009 to June 2012. The IBM team worked closely with business partners, including Mark Lines, and was led by Scott Ambler.

Ownership of DAD intellectual property effectively passed over to the Disciplined Agile Consortium in October 2012, a fact which was legally recognized by IBM in June 2014.

DAD later developed into a comprehensive toolkit and the now called Disciplined Agile 5.x was released in May 2020 and published in a book called *Choose Your WoW! A Disciplined Agile Delivery Handbook for Optimizing Your Way of Working (WoW)* (Ambler & Lines, 2020).

DAD is now just a subset of Disciplined Agile which covers a much broader scope.

Figure 38 How role-players work together in DA Flex Workflow



Picture created by EXIN based on: Project Management Institute, Inc. (2021). *Introduction to Disciplined Agile® (DA™).* https://www.pmi.org/disciplined-Agile/introduction-to-disciplined-Agile

Kanban

Kanban is a Lean technique popular in manufacturing and has recently seen increased use in specifically IT projects and Operations. What people mean by “Kanban” in IT environments is usually a development method that is a combination of various Scrum elements with the Kanban technique.

The use of Kanban with Scrum has already been covered in the main text in the section Using the Kanban method.

Appendix B – More about releases and often used release management tools in Agile environments

Release or not-release, that is the question

The criticism of using the release concept in Scrum is growing amongst practitioners and for good reason. The concept of a release makes a mockery of not planning far ahead, whereas at the same time proponents of releases in Scrum will tell you it is not detail planning! Well, it is, and it isn't.

The concept of releases is now officially not part of the Scrum guide since 2011. In fact, why has release planning been removed from the Scrum guide? The answer is short and simple: It is possible to successfully use Scrum without using release planning.

Not needing release planning may be a positive indicator of a product's health. Scrum is intended to deliver value to customers continuously through short iterations. Using releases may delay the delivery of value, customer feedback, and return on investment (RoI). With mature development practices and excellent rollout strategies, the delivery of value can take place every sprint."[18]

Let's explore why a traditional release management approach is not a good idea in Scrum, by dispelling some of the myths that proponents of release management will put forward, namely:

Myth 1

In complex and scaled environments, pre-planning is necessary.

Reality 1

Yes – but all the pre-planning that is needed can be done by ensuring that the product backlog is properly refined.

Myth 2

Product backlog refinement is insufficient planning to consider coordination, dependencies, and sequence.

Reality 2

Yes – but using a scaled mechanism like Nexus suffices to ensure proper coordination between Scrum teams that are part of the Nexus, and to properly handle dependencies, even if these are across iterations.

---

[18] Scrum.org (2021). *Gone Are Release Planning and the Release Turndown.* https://www.scrum.org/resources/gone-are-Release-planning-and-Release-turndown

Myth 3

Methods like Nexus do not cater for successive sprints and therefore do not properly cater for dependencies over time.

Reality 3A
Planning what will happen in successive sprints in a fair amount of detail, as people do in release planning, is turning Scrum back into Waterfall. Understanding sequence, dependencies, and what would most probably happen in the next sprint can quite sufficiently be dealt with by just properly managing your product backlog. In addition to this, a scaled approach like Nexus specifically notes and manages sequence for dependencies across sprints and only that, as the rest should happen when managing the product backlog.

Reality 3B
One of the most disadvantageous consequences of the release approach is that teams fall into the trap of saying: *Well in sprint 2 we will be doing x, which was already decided.* Thereby violating Agile principle 2 and possibly as a consequence also 1, 4, 5, 10, and 11.

Myth 4

Releases create predictability and help us to commit to business targets, as the business wants end-dates.

Reality 4
What? That is WaterScrumFall. You are not Agile.

Release guidance

However, if you believe that using release management is something that your organization should continue to do, here is some guidance.

- Do not stretch release cycles beyond three sprints.
- Make sure that everyone knows that the release plan could and should change after every iteration unless everything in the business remains the same, which is highly unlikely.
- Creating and using a release backlog (like a sprint backlog but for all the sprints that form part of the release) must be high-level and very flexible. That means that new items will continually find their way into the release backlog and as a consequence items will have to be dropped unless more teams join the release.
- Use release planning as a guide and a means to focus on the product goal; you will discover things you did not know and have to deal with quickly.
- Be flexible and focus on what is the right thing to do for the customer and not just sticking to the plan.
- Note dependencies and sequences in the product backlog.
- Use a scaling mechanism like Nexus. If you do, your release planning and management will be much simpler and high-level. Note: methods like

Scrum@Scale include the release management concept so it may be useful to explore the guidance there.

- Don’t do task breakdown in release planning; leave that for every sprint cycle. In essence, this means that if you don’t use a scaling mechanism like Nexus, you do a bit of release planning every sprint cycle.
- Don’t end up doing WaterScrumFall.

Burn-down and estimation

Burn-down charts are an especially helpful tool to track releases. The idea is not dissimilar to the normal burn-down chart, but with a few tweaks.

Progress in Scrum projects can be tracked by means of a release burn-down chart. The Scrum Master managing the release should update the release burn-down at the end of each sprint.

Like a sprint burn-down chart, the vertical axis shows the amount of work remaining at the start of each sprint, and to express this you can use story points, ideal days, team days, or whatever you choose in your organization. The horizontal axis shows time but, in this instance, the number of sprints before the release (project) is done. Why bother if a release should be no longer than three sprints?

This is where the kicker comes in.

If you use releases the likelihood is that you do so because you believe Myth 4.

If work is constantly being added, and your customers don’t want to de-prioritize product backlog items in the release backlog, in fact by allowing you to remove items and place them back into the product backlog, the only logical consequence anyone can reach is that it will take longer!

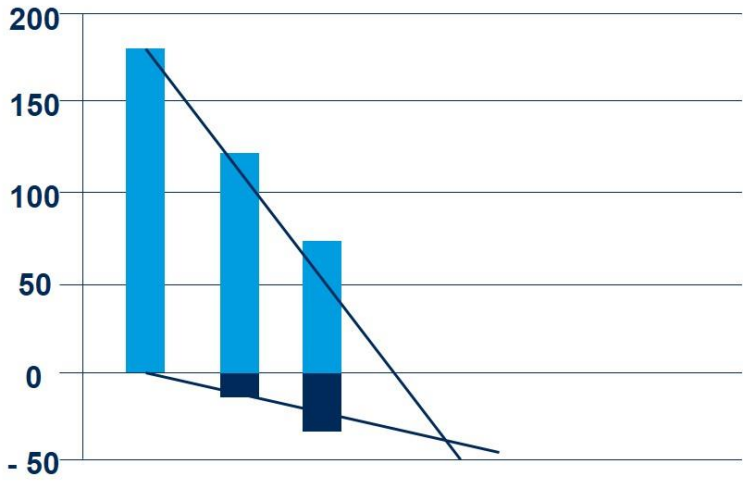
But how much longer?

The release burn-down bar chart will help you find the answer.

Release burn-down bar charts don’t only go to zero but allow you to go negative on the vertical bar. As additional work is added during the release, it is added below the zero line as it is added every sprint cycle.

So why is this helpful? Because by doing this you can use two trend lines to predict how many sprints it will take to complete the work in the release backlog.

Figure 39 A Release burn-down bar chart



Picture created by EXIN based on: Botha, J. (2018). *Scrum Masters and Product Owners [Courseware].* GetITright.

The release burn-down now features two trend lines: one for the data above the Zero line; and one for the data below the Zero line. The point of intersection now tells the Scrum Master who is guiding the release that an additional sprint cycle is needed to complete the release with the newly added work in sprints 2 and 3 – not ideal according to the above guidance of three sprints.

The reality is that in practice this often looks much worse with projections of 10 or more sprints not being uncommon. There you have it – a Waterfall project.

Using Gantt charts for release planning

It is a common practice to use Gantt charts to visually remind teams in the release about what works in the release backlog will happen in which sprint. It is also used to show the interdependencies of product backlog items and their related tasks.

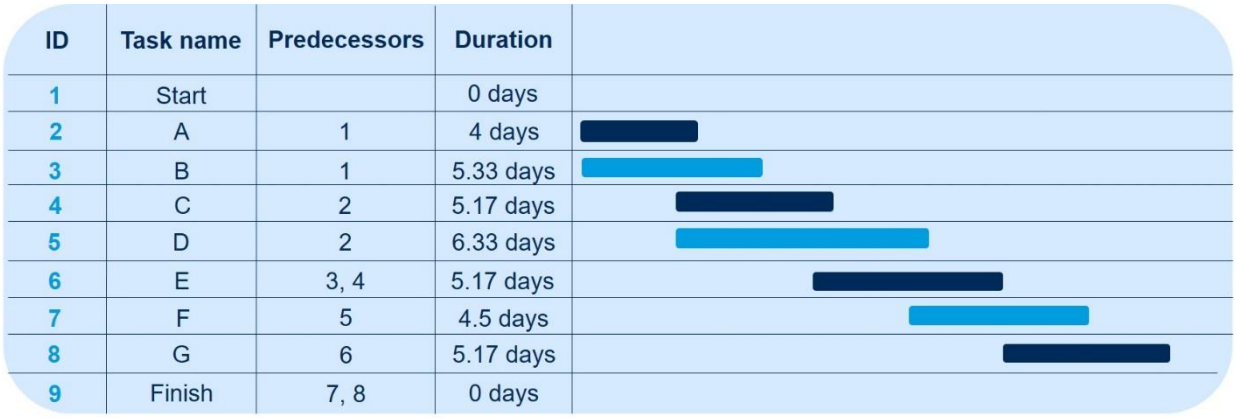
But what is a Gantt chart?

A Gantt chart is a visual view of tasks scheduled over time, their relationships, and their dependencies.

The concept was developed by Henry L. Gantt around 1910, who was working with Frederick Taylor as a method of describing production planning and resource loading for factories and workshops. One of the first major applications of Gantt charts was by the United States during World War I, to plan logistics projects for the war.

You have all seen these charts during project meetings and they look something like this:

Figure 40 Release planning using Gantt charts



ID	Task name	Predecessors	Duration
1	Start		0 days
2	A	1	4 days
3	B	1	5.33 days
4	C	2	5.17 days
5	D	2	6.33 days
6	E	3, 4	5.17 days
7	F	5	4.5 days
8	G	6	5.17 days
9	Finish	7, 8	0 days

Picture created by EXIN based on: Gantt Chart. (2021). Retrieved on February 14th, 2021 from https://en.wikipedia.org/wiki/Gantt_chart.

In principle, there is nothing wrong with using Gantt charts as a means of visualizing data; the important thing is the context in which it is used and what it implies. If used for detailed pre-planning, it makes a mockery of Agile and Scrum.

Bibliography & references

Agile Business Consortium (2014). *The DSDM Agile Project Framework (2014 Onwards).* https://www.agilebusiness.org/page/TheDSDMAgileProjectFramework

Ambler, S. & Lines, M. (2020). *Choose your WoW!: A Disciplined gile Delivery Handbook for Optimizing Your Way of Working (WoW).* Project Management Institute.

Bigelow, S. J. (2020). *7 techniques for better Agile requirements gathering.* https://searchsoftwarequality.techtarget.com/tip/7-techniques-for-better-Agile-requirements-gathering

Botha, J. (2018). *Scrum Masters and Product Owners [Courseware].* GetITright.

Botha, J. (2019). *Agile: A Manager's Guide to Unlocking Business Value.* Amazon Digital Services LLC - Kdp Print Us.

Botha, J. (2020). *Scrum Lego-game workbook [Courseware].* GetITright.

Botha, J. (2021). *[Courseware].* GetITright. Retrieved in personal communication from the author of the book.

Botha, J. (2021). *Lean fundamentals for IT [Courseware].* GetITright.

Cohn, M. (2005). *Agile Estimating and Planning.* Pearson Education.

Cohn, M. (2009). *Four Attributes of the Ideal Pilot Project.* https://www.mountaingoatsoftware.com/blog/four-attributes-of-the-ideal-pilot-project

Cohn, M. (2010). *Succeeding with Agile.* Addison-Wesley.

Crystal Methods. (2021). Retrieved on September 23, 2021 from

EXIN Holding B.V. (2019). *Agile Methodologies.* EXIN Holding B.V. https://dam.exin.com/api/&request=asset.permadownload&id=3144&type=this&token=8659eabedff466d86ac2eba5ed72b33d

Exoskeleton. (2021). Retrieved on September 23, 2021 from https://en.wikipedia.org/wiki/Exoskeleton

Gantt Chart. (2021). Retrieved on September 23, 2021 from https://en.wikipedia.org/wiki/Gantt_chart

Goldratt, E. M. & Cox, J. (2012). *The Goal.* (3rd Edition) North River Press.

Hiatt, J. M. & Creasey, T. J. (2012). *Change Management: The People Side of Change.* Prosci Learning Center Publications.

https://en.wikiversity.org/wiki/Crystal_Methods

Kano+ (2020). *The Kano model – Assessing Product Features based on Customer Satisfaction.* Kano+ https://kano.plus/about-kano#analyze-a-study

Maher, R., & Kong, P. (2016). *Cross-Team Refinement in Nexus™.* Scrum.org. https://www.scrum.org/resources/cross-team-refinement-nexus

Maher, R., & Kong, P. (2016). *The Nexus Integration Team.* Scrum.org. https://www.scrum.org/resources/nexus-integration-team

Maher, R., & Kong, P. (2016). *Visualizing the Nexus Sprint Backlog.* Scrum.org. https://www.scrum.org/resources/visualizing-nexus-sprint-backlog

Martin, B. A. (1980). *The goal: to improve credibility in the reporting of engineering progress.* Project Management Quarterly, 11(2), 14–22.

McKinsey & Company (2019). *The journey to an Agile organization*. https://www.mckinsey.com/business-functions/organization/our-insights/the-journey-to-an-agile-organization

Meier, J. D. (2019). *Extreme Programming at a Glance.* https://jdmeier.com/extreme-programming-at-a-glance/
Overeem, B. (2016). *The 11 Advantages of Using a Sprint Goal.* https://www.scrum.org/resources/blog/11-advantages-using-sprint-goal
Pichler, R. (2010). *Agile Product Management with Scrum.* Addison-Wesley.
Project Management Institute, Inc. (2021). Introduction to Disciplined Agile® (DA™). https://www.pmi.org/disciplined-Agile/introduction-to-disciplined-Agile
Prosci (2021). *Best Practices in change management benchmarking report.* https://www.prosci.com/resources/articles/change-management-best-practices
Rad, N. K. (2021). *Agile Scrum Handbook* (3de ed.). Van Haren Publishing.
Rother, M. (2018). *The Toyota Kata Practice Guide: Practicing Scientific Thinking Skills for Superior Results in 20 Minutes a Day.* Mcgraw-Hill Education.
Scaled Agile (2021). *Scaled Agile Framework.* https://www.scaledagileframework.com/
Schwaber, K., & Scrum.org (2021). *The Nexus™ Guide – The Definitive Guide to Scaling Scrum with Nexus.* Scrum.org. https://www.scrum.org/resources/nexus-guide
Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game*. Scrumguides.org. https://scrumguides.org/scrum-guide.html
Scrum.org (2021). *Gone Are Release Planning and the Release Turndown.* https://www.scrum.org/resources/gone-are-Release-planning-and-Release-turndown
Scrum.org., Vacanti, D., & Yeret, Y. (2021). *The Kanban Guide for Scrum Teams.* Scrum.org. https://www.scrum.org/resources/kanban-guide-scrum-teams
Senge, P. (2006). *The Fifth Discipline.* Bantam Doubleday Dell Publishing Group Inc.
Sutherland, J., & Scrum Inc. (2021). *The Scrum@Scale® Guide – The Definitive Guide to Scrum@Scale: Scaling that Works.* Scrum.org. https://www.scrumatscale.com/wp-content/uploads/2020/12/official-scrum-at-scale-guide.pdf
Technical Debt. (2021). Retrieved on September 23, 2021 from https://en.wikipedia.org/wiki/Technical_debt
The LeSS Company B.V. (2014). *Large Scale Scrum is Scrum [Visual].* https://less.works/img/principles/principles.pdf
Underwood, J. (2013). *Musselwhite and Ingram's Change Style Indicator.* https://innovategov.org/2013/08/15/musslewhite-and-ingrams-change-style-indicator/
Wells, D. (2000). *Introducing Extreme Programming.* http://www.extremeprogramming.org/introduction.html
Wells, D. (2013). *Extreme Programming: A gentle introduction.* http://www.extremeprogramming.org/

Contact EXIN

www.exin.com