# Section 1

## Overview of the SCRUM framework

-: Scrum is a framework for dealing with complex work often encountered when developing new products. Nowadays, with constantly changing market condition, and technology improvements, giving a high level of uncertainty, it is impossible to predict from the beginning, how a product should be developed. Let me put this in simpler terms. We have absolutely no clue what customers will want six months from now, and we must accept the fact that we cannot predict the future.

Spending six months to create a detailed plan with requirements and following that plan for another one or two years until the full product is ready to be released does not work anymore. So in these conditions, working with time boxes and quick adaptation is mandatory to ensure that a product does not fail. Scrum is a framework that imposes time boxes feedback loops, and encourages small increments while overall trying to deal with uncertainty. What Scrum does is a bit of all the steps required to develop a product.

This may include gathering requirements and understanding what customers want, planning, design and architecture work, development, testing, documentation and to put them in a fixed-length iteration called a sprint. So a sprint combines all aspects of the work required to build a product. Right from the first sprint, a Scrum team will try to create a working, tested and potentially shippable product, even if it is not released to the customers or end users.

After each sprint, the Scrum team demonstrates what they have accomplished and discuss what to do next. Customers often need to see the wrong product before they can say what they really want. The short iterations allow for constant feedback and improvement. This considerably increases the probability of creating the right product: a product that customers use and love. In order to accomplish that, the team needs to have all the necessary skills to understand the business requirements to do any design development and testing work, so that by the end of the sprint, a potentially shippable product is created.

A Scrum team consists of a Scrum master, a product owner, and the developers. The product owner will create a list of features, called the product backlog and organize that list to ensure that customers get a maximum amount of value. During sprint planning, the developers will select a list of items from the top of the product backlog work on them during the sprint, and turn them into potentially reasonable product increment. Product increment is simply a new version of the same product. The Scrum team has a fixed timeframe to complete a work which cannot be longer than one month and the developers meet in a daily Scrum to synchronize, identify problems, and keep the work moving forward. Along the way, the Scrum master keeps the team focused on the sprint goal and helps remove any impediments that affect them.

At the end of the sprint, the product increment should be potentially shippable and the Scrum team, together with the stakeholders, conduct a review of the work completed and discuss what to do next. The last event of the sprint is a retrospective of the process. The Scrum team looks back at how the sprint went and figures out a way to improve their development process. Then they start from the beginning with the next sprint planning and a cycle repeats.

## What is Scrum?

Instructor: By now you already have an overview of Scrum but some concepts are a bit harder to grasp. In this lecture, we'll go a bit deeper and fill any gaps in your understanding of what Scrum really is. The Scrum Guide defines Scrum as a lightweight framework that helps people, teams, and organizations generate value through adaptive solutions for complex problems. There are three keywords I want to point out, framework, value, and complex problems. Let's begin with a term framework.

To make an analogy to support, the Scrum Guide is like the football rules. Does knowing the football rules make you a good player? Most likely not, but it definitely helps to know the rules. You don't want any player on the field who doesn't know the rules. The Scrum Guide presents the rules of the game. It is on the cover of the Scrum Guide, but you have probably missed that, even if you have read the Scrum Guide a couple of times. Previous Scrum Guide versions used to say that Scrum is simple to understand, but difficult to master. I think that really sums it up. Many people incorrectly say that Scrum is a methodology or use the terms framework and methodology interchangeably.

They are not the same. Without getting too technical, a methodology is much more than a framework and provides exact steps or methods for solving a problem. As you have seen, the Scrum Guide is not a step by step guide for implementing Scrum. Even the Scrum Guide says it is incomplete and does not provide detailed instructions. The idea is that people using Scrum are smart enough to figure things out on their own and don't need to be told what to do. Scrum is not a methodology containing answers to all situations you may encounter.

With Scrum, you have to make decisions on your own within the boundaries explained by the Scrum Guide. So the Scrum framework consists of the Scrum Team or teams working on a product, their accountabilities, events, artifacts, and the rules that bind them all together. Let's move on to the next keyword, value. We are using Scrum because we want to generate value for the organization, stakeholders, clients, and users. So we shouldn't use Scrum because this is what everyone else is doing. I've encountered so many Scrum Teams wandering in the dark, building new features, Sprint after Sprint, without any goal or end in sight. Without even knowing if the features they built have any real impact.

More output is not more value. Make sure you don't end up in a similar situation, and if you happen to be in one, have conversations around goals and value. Finally, you want to be using Scrum for solving complex problems, but what is a complex problem? Without getting too deep into the Cynefin framework, which can help you understand the type of problem you are dealing with, here is a very simple example. Is manufacturing nuts and bolts on a production line, a complex problem? Most likely not, as all the steps of the process are known in advance. It has been done times and times again in the past. It is essentially like following a recipe.

As long as you do everything in the right order, you get the expected results. How about building a space rocket and flying to Mars? I would say that clearly falls into the category of complex problems. However, most of you are not building space rockets. Generally, software development is considered a complex problem because quite often we are dealing with complex systems. In some of the software projects I have worked as a developer, I sometimes needed three to six months to be confident enough to make changes on my own.

However, no matter how experienced you are, you never have a hundred percent certainty. Adding a single character to a line of code can make the entire application crash. Regardless of whether you work in software development, we all are knowledgeable workers and often what we do falls into the category of complex work. Since the environment we are working in is complex, we often need to adapt our solutions to the current circumstances.

What worked last week may not work today, so we don't know the solution to a problem in advance, and we don't know which path to take. We need to try out different things. We develop experiments. This is why we need some flexibility and constant feedback. So to recap, Scrum is a framework that helps motivated individuals build valuable products in complex environments by establishing the structure to do so.

## Scrum Theory

Instructor: In this lecture, we look at a Scrum Theory. I know what you're thinking. The word theory doesn't sound very exciting, does it? This is an important part of Scrum that you need to understand. And I will try to make it as digestible as possible. Scrum is founded on the empirical process of control theory also called empiricism and lean thinking. Let's begin with empiricism.

Empiricism reasons that knowledge comes from experience and you need to make decisions based on what is known right now. You may have heard the terms empirical data or empirical evidence before. This is all about facts and data. For example, if our product is being used by 10,000 customers each month, that is empirical data. However, if we look at this usage data and conclude that customers love our product, this is no longer empirical data. This is just speculation. And the conclusion may be incorrect as we have no direct data to back this up. If we had sent a survey and asked our customers if they liked the product or not, the data gathered would have been empirical data.

So in Scrum, we need to do everything based on our past experiences and what we know has happened so far. Scrum is built on the empirical pillars of transparency, inspection, and adaptation. So to recap, Scrum has its foundation in empiricism and lean thinking and is built on transparency, inspection, and adaptation. And these are the pillars you need to remember: transparency, inspection, and adaptation. Let's begin with transparency. So what does transparency mean? Transparency refers to the fact that the process and the work need to be visible for everyone involved in Scrum. We make decisions based on this information so our decisions won't be right if we have no transparency. For example, the three Scrum artifacts meaning the product backlog, the sprint backlog, and the increment need to be transparent.

Transparent does not only mean that you need to be visible or accessible to everyone involved. It is actually more than that. Everyone involved in the process needs to have a shared understanding of the work. Transparency is also required for inspection which is the next step. In Scrum, we constantly inspect our work, the process and the artifacts. We try to identify any issues or deviations from what we expected. As soon as the inspection is over and we have determined that some parts of the process or anything else deviate from what we actually expected, we need to do something about it.

Adaptation needs to be made as soon as possible. Scrum has four dedicated events for inspection and adaptation. The sprint planning, the daily scrum, the sprint review and the sprint retrospective. They are all used for inspection and adaptation. The main idea is that we need to regularly inspect what we are doing to understand if what we are working on is indeed what is needed. As soon as we notice that what we're building is not the right thing, we need to make some changes. We need to adapt. In Scrum, we have sprints and we build increments. Through this iterative and incremental approach, we optimize predictability and control risk.

All Scrum events are an integral part of Scrum and implement the empirical scrum pillars of transparency, inspection and adaptation. However, the Scrum team does not need a Scrum event to do inspection and adaptation. Transparency needs to be insured at all times and inspection and adaptation can and should be done anytime during the sprint. If someone on the Scrum team notices something, for example discovers a defect, they need to share this information with everyone else on the Scrum team as soon as possible.

This helps to restore transparency. Withholding this information would be the opposite of that. So this is why it's crucial to have this inspection and adaptation done regularly and it can be very, very valuable. Now let's discuss lean thinking. Lean thinking is a framework that focuses on creating something beneficial while at the same time eliminating waste. Lean thinking has its origins in lean manufacturing also known as lean production which is the right from Toyota's operating model,

The Toyota Way. Scrum adheres to this framework as it focuses on building valuable quality products that customers love while at the same time striving to reduce or eliminate waste. Waste is the opposite of something valuable.

For example, if you build a feature that nobody wants or uses, that is waste as it has generated no value. Scrum does not make teams work faster. But if they focus on what is valuable and eliminate wastes, they produce better results. This is one of the reasons why Scrum works. So to recap, Scrum is founded on empiricism and lean thinking and implements the empirical pillars of transparency, inspection and adaptation.

## The most important ideas regarding the Scrum framework

Instructor: Let's take a moment and briefly go over the most important concepts discussed in this section. Probably the most important word that needs to enter your vocabulary is the word framework. Scrum is a framework, not a methodology or anything else. Agile is also not a methodology. Scrum can be used to develop many kinds of products and services and is not limited to software development.

Scrum is based on empiricism and lean thinking. Any scrum implementation must uphold the empirical scrum pillars of transparency, inspection, and adaptation. For Scrum to be successfully used, the people involved need to have the five Scrum values embedded into their day to day activities.

The five Scrum values are:

* Commitment,
* Focus,
* Openness,
* Respect, And
* Courage.

All these values help build trust. When we implement Scrum, we should not adjust the Scrum rules meetings or any other aspects. Scrum is already flexible enough and can accommodate many types of products. Scrum has been purposefully made lightweight and it is relatively easy to add additional practices on top of Scrum. However, removing anything or somehow modifying Scrum, can lead to little change, if any, and any benefits of implementing Scrum will most likely be lost.

## Uses of Scrum

Instructor: Let's talk a minute about where we can use Scrum. There is no secret that the Agile movement and the Scrum framework have their roots in software development. However, many people realize that the principles behind the Agile manifesto and the process introduced by the Scrum framework could be applied for building other types of products.

Yes, Scrum is still hugely popular for developing or maintaining software products, but many are using it successfully in other areas outside of software development. The Scrum guide does not say that Scrum can only be used while building software products. Actually, the Scrum guide does not even mention software development and the term developer is not a reference to software developers.

Scrum has been successfully used in developing hardware products while researching self-driving cars, in marketing, and in the personal lives of the people familiar with Scrum. And these are just some examples. I know of people who have used Scrum while building their homes. Scrum can be used anywhere where a complex problem needs to be solved. A product is not only something physical that is being shipped and sold to customers.

The Scrum guide defines a product as a vehicle to deliver value. It has a clear boundary, known stakeholders, well defined users or customers. A product could be a service, a physical product, or something more abstract. So if you want to think outside of the box and apply Scrum to areas outside of software development, you need to think about defining your product and being able to explain how it will deliver value.

Scrum helps when dealing with complex problems where you don't know the solution in advance. I've added to the resources folder many examples of how Scrum is being used in different areas, from building insect hotels to building warehouses. I invite you to check them out.

## How to understand empiricism

Instructor: Maybe you're still confused about empiricism. So to recap, empiricism means transparency, inspection and adaptation. All are equally important and they cannot work one without the other. If one of them is weak or neglected, the entire chain is broken. Let me share with you an analogy to understand empiricism better. Say you are riding your bike and it is getting dark outside. You can hardly see anything. You turn on your headlight.

Now you can better see the road ahead and where you are going. This is transparency. You see the road and the proximity, but only as much as your headlight allows you. All of a sudden you notice a large pothole right on your path. This is inspection. You quickly react and steer to the left, avoiding the pothole. This is adaptation. You have avoided a serious accident only because your headlight was working, you were paying attention to the road and spotted a pothole, and you had good reflexes and changed your direction in time.

If you did not have your headlights on, you would not have spotted a pothole and caused an accident. If you had your headlights on, but you were distracted and not looking at the road, again you would have caused an accident. If you had your headlights on and were not distracted but still failed to take any avoiding action, you would still have caused an accident. This is why transparency, inspection, and adaptation go hand in hand and are equally important.

## Scrum Values

Narrator: The culture of the organization where Scrum is being used is quite important. Scrum requires a supporting organization and culture to be successful. The values of Scrum are tightly connected to this culture. They are courage, focus, commitment, respect, and openness. People are inherently reluctant to change and quite often take the easiest path when adopting Scrum. Scrum is not about renaming existing meetings and using different titles, but preserving the old habits.

This approach will have little or no impact and the benefits of Scrum are lost. The Scrum guide clearly states that the Scrum framework is immutable. While implementing all the parts of Scrum is possible, the result is not Scrum. Effective Scrum requires all these values to be embodied by the organization, and the people using Scrum.

The five Scrum values of courage, focus, commitment, respect, and openness, together with the Scrum pillars of transparency inspection, and adaptation, build trust. A Scrum team needs trust to work properly, otherwise it is dysfunctional. The five Scrum values are interconnected, and if one value is diminished or undermined, the other values are also affected. For example, it does not matter how committed you are if you are not open about the challenges you face.

Scrum team members will respect one another as capable independent people. With every sprint, they are focused on the product goal and the sprint goal. The culture in the Scrum teams should ensure that everyone is treated equally. There are no hierarchies and nobody is managing anyone. Everyone is respectful and helps each other reach commonly defined goals.

Commitment is all about doing your best while at the same time helping the ones surrounding you, is successful as well. The Scrum team is committed to reaching the sprint goal. Many misunderstand commitment as putting in many hours, working overtime, or doing a fixed amount of work every sprint. Remember that Scrum is all about working at a sustainable pace.

Everyone involved in the Scrum process, that being the Scrum team, the stakeholders or people from within or outside the organization, need to be open about the work and the challenges that appear. When doing complex work, not everything will pan out as expected. Even if it is hard, it is best to be open about the current situation and what needs to be done instead of concealing information. So to recap, the five values of Scrum are courage, focus, commitment, respect, and openness. They are all interconnected and together help build trust.

## Meet the team

Narrator: Jack & Mary Organic Farm has a tradition of over 50 years in the market. It specializes in the production of jams and jellies, and, as it often happens with traditional companies, they have not really managed to keep up with the latest technology changes. Until now, they only sell their products to wholesalers, or at a farmer's market. But their customers, time and time again, ask if they could order online.

Now they have brought onboard a team of six professionals to create a new eCommerce platform for the company so that they can sell their products over the internet. So let's meet them. Ali has experience with Scrum, and has proposed to give Scrum a try for this project. All others, including the company CEO, are excited to learn what Scrum is, and how they can all benefit. Jessica has a good understanding of eCommerce, and has been involved in similar projects in the past.

Aki, Michelle, Tom, and Max, bring a mixed set of skills on design, programming, and testing, all crucial to the building of the eCommerce side. The company CEO and the board of directors are the ones who wanted to have the eCommerce solution, and they all know that the future of this company depends on the outcome of this project. So they are the stakeholders in this project. Together, they will learn what Scrum is, and how it can apply to real world problems.

# Section 2

## What are Scrum Artifacts?

Instructor: In this section, we will discuss the scrum artifacts. I assume you are already a bit confused about the term artifact, so let me explain a bit. In archeology, an artifact is something made or given shaped by humans, usually taking the form of a tool or a work of art. In scrum, an artifact is something that is created by the ones using scrum. The scrum guide defines three artifacts: the product backlog, the sprint backlog, and the increment. We often also call it the product increment.

Artifacts in scrum represent work or value. They provide transparency and opportunities for inspection and adaptation. As you'll notice, scrum uses many terms that you would not necessarily use in your day to day conversations. If you create a shopping list you won't call it an artifact.

However, this is a scrum language you need to get familiar with. Each of the scrum artifacts contains a commitment toward something. The product backlog exists to reach the product goal, which is a long term objective. The sprint backlog exists to reach the sprint goal that the scrum team defines for every sprint. Each sprint goal is a step toward achieving the product goal. The product increment is committed to fulfilling the Definition of Done, which usually describes the desired quality that a product should have.

Let's understand the word commitment for a second. In this case, a commitment means to be dedicated to achieving something specific. Having a commitment ensures that everyone knows why the work is important and what is the desired outcome. This is an essential aspect because we are not using scrum just to have some new meetings, fancy new roles or titles, and to say that we are agile. We are using scrum because we want to have a specific result or a specific outcome from doing this. If you want to achieve something, you better have some clear goals and to be dedicated to reaching them.

Now, don't worry if this is too much to digest at this point. In this section, we'll take these concepts one by one.

## Scrum Artifacts summary

Instructor: This is the end of the section and it is time to recap some of the most important aspects in regards to the scrum artifacts. The product backlog is managed by the product owner and contains the product goal and the product backlog items. The product goal is the commitment for the product backlog. A product backlog item is not a requirements document or a contract between the product owner and the developers.

The sprint backlog is created during each sprint planning meeting and contains the sprint goal, the selected product backlog items from the product backlog, and the plan for the sprint. The sprint call is created by the entire scrum team and does not change during the sprint. The developers manage the sprint backlog. The sprint call is the commitment for the sprint backlog and the scrum team must do everything that they can do to reach it. The selected items are a forecast, and the developers in collaboration with the product owner, may add new items, remove some, or even modify the existing ones.

In this context, understanding the difference between a forecast and a commitment is essential. Both the product backlog and the sprint backlog are never final or complete. They are updated throughout the sprint as needed. Every time the developers complete one or more items from the sprint backlog, a new increment is created. The increment is a new version of the product. Any completed work as well as the increment must follow the quality criteria defined in the definition of done which represents a shared understanding of what complete work is. The definition of done is a commitment for the increment. The developers must create at least one increment by the end of the sprint.

The product owner can decide if and when to release the increment without waiting for the sprint review meeting. Scrum is founded on empiricism, and all artifacts support the three pillars of transparency, inspection, and adaptation. Decisions that are made to improve the value and control the risk are based on the state of the artifacts.

If the artifacts are not transparent enough, then also the decisions that were made are flawed. This also impacts the delivery of value. The scrum master's job is to work with everybody within and outside of the scrum team to ensure that the artifacts are transparent.

## Organizational goals

-: Most products are developed by an organization that typically has it's own goals, and a larger vision. Apple wants to make the best products on Earth. McDonald's wants to be the best quick service restaurant experience. And Action Against Hunger is committed to ending world hunger. We can safely say that any product that an organization decides to develop, should sustain the objective and the larger vision that organization has defined for itself.

But what is a vision anyway? For we have given this program a high national priority Even though I realize that this is in sumnasia An act of faith, and vision, for we do not now know what benefits await us. But if I were to say my fellow citizens, that we shall send to the Moon, 240,000 miles away from the control station in Houston, a giant rocket, more than 300 feet tall, the length of this football field. Made of new metal alloys Some of which have not yet been invented.

Capable of standing heat and stresses, several times more than have ever been experienced. Fitted together with a precision better than the finest watch. Carrying all the equipment needed for propulsion, guidance, control, communications, food and survival. On an untried mission to an unknown celestial body.

And then return it safely to Earth. Re-entering the atmospheres at speeds of over 25,000 miles per hour. Causing heat about half that of the temperature of the sun. Almost as hot as it is here today, and do all this, and do all this, and do it right, and do it first, before this dictate is out, then we must be bold.

Many years ago, the great British explorer,

George Mallory, who was to die on Mount Everest was asked why did he want to climb it. He said because it is there. Well space is there, And we're going to climb it. And the moon and the planets are there. And new hopes for knowledge and peace are there. And therefore as we set sail, we ask God's blessing, on the most hazardous and dangerous and greatest adventure on which man has ever embarked. Thank you. You can call this a vision, or an ambitious goal, but the plan of this vision was to land a man on the moon, and return him safely to Earth. This vision was created without knowing how it will be done, and without having the technology and experience for doing that

However, this was the vision that guided the US Space Program in the 1960's. To reach this vision and the ultimate goal, NASA had to go through multiple milestones, or call them smaller goals if you wish. They did not just plan one primary mission, build a rocket, push astronauts on it, start the countdown, and hope for the best. Sometimes you need to break down large goals into clear and achievable steps. They first started with rockets that had no astronauts on them. They learned from that, some rockets exploded and created setbacks.

Other missions were successful, and accelerated progress. Finally, they did it.

Astronaut: Ah gee, that's great. Is the lighting halfway decent?

-: Yes, indeed. They've got the flag up now, and you can see the stars and stripes.

-: Now, I'm obviously oversimplifying, but each milestone was a step toward the larger goal.

Let's now come back to Earth. When talking about Sprint goals, product goals, and product visions in Scrum, things are surprisingly similar to the Apollo Space Program. When building a product, it makes sense to have a product vision aligned with organization's overall vision. Maybe that vision is 10, or 20 years away from being realized. While this Scrum guide does not mention or require a product vision, it often makes sense to define one. A product goal represents a long-term objective, and can be viewed as an important step toward that vision.

At one point in time, there is only one product goal. This helps direct the energy in a single direction, instead of running after multiple things at the same time. There are no top 5 priorities that compete with each other. Every sprint goal that is reached, and every increment delivered, is a step toward achieving the product goal. Anyone involved with Scrum can understand how the work that is happening today, right now, is relevant towards reaching the sprint goal, the product goal, and achieving the larger product vision. A goal gives direction, helps focus on one thing, and most importantly, creates transparency, and opportunities to check if the product development is on track. We call this inspection in Scrum, and to change the course if needed. Also known as adaptation.

## What is a Backlog?

-: One fundamental artifact in Scrum is the product backlog. So let's break down this term. A product can be something that you might think a product is, a laptop, a car an app on your phone, or a computer game. But it can also be something less intuitive, like a vaccine, your car insurance, a spaceship, or even a marketing campaign. Having your taxes done by a CPA can also be a product.

Essentially, a product is something that users or customers use and get some benefit. You play a game on your phone because it relaxes you and it is fun. This is the benefit. In Scrum, we refer to this as the value you get from using the product. Let's move on to the term backlog. Generally speaking, a backlog represents an accumulation or a buildup of unfinished work. So a product backlog contains undone things that are related to the product. You can see it as a to-do list.

Here's a simple example.

Let's say you want to go on vacation. That is your goal. You create a list of things that need to happen to reach that goal. Here are some examples. Find a destination. Book flights. Request an annual leave from your employer. Wash the car. Buy sun cream. Book a hotel. Book a rental car. This is your backlog or your to-do list for organizing your vacation. As you can observe, none of the items have been done yet so they are in the backlog. The list of undone work. Now let's observe a few additional things. The items in your list are in random order. What would be the first thing that you should do? Would it make sense to buy plane tickets before the company approves your annual leave? Probably not.

This is risk. What if you buy plane tickets and the company says you can't be absent during that period because they need you for a very important project. So to minimize risk, it would make sense to ask for your annual leave first. How about getting a hotel before knowing when and where you fly? That is a dependency. You first need to know where you fly before making other arrangements. Will buying sun cream before getting a hotel will help you achieve the goal? I would say not. This is why the backlog needs to be ordered in a way that moves things forward and brings you closer to your goal. You start working on the items from the top of the list.

How about washing your car? Yes, that needs to be done. But how does it help with the vacation goal? It's better remove it to get clarity over what is really important. A few days pass by and your leave has been approved. The next thing on the list is finding a destination. But you know what? Ah, you're a busy person. This vacation planning is really taking a lot of time. You decide to let Olivia, your trusted travel agent, do the holiday planning for you. Olivia, is any way better than you because she knows where to look for flights and how to get a good deal on a hotel.

Now, Olivia is confused about what you expect. For you, it can be relatively clear how your destination should look like, but not so much for her. This is when this thing called transparency kicks in. It would help if you refine your destination item to something like, I want a quiet location right on the beach so that I can swim anytime I'm in the mood. Now, this item on the list is much more precise and helps Olivia make decisions on her own instead of bugging you with minor questions. The process of looking at a backlog and observing its current condition is called inspection in Scrum.

When you notice something is wrong, missing or unclear, you go ahead and change that. We call this adaptation. The backlog needs to be continuously improved. In Scrum, we call this activity refinement. By clarifying details, answering questions, we're adding things that need to be done. This would be a short explanation of the term product backlog.

## Defining the Product Goal and Product Backlog

Narrator: One of the first things Jessica needs to do is to define a Product Goal, and get started creating the product backlog before the first Sprint can begin. Remember, the Product Owner is accountable for defining a Product Goal and managing the product backlog. Therefore, she will start working with the stakeholders to understand what is needed from a business standpoint, also with the rest of the Scrum Team.

Of course, Jessica already knows that Jack and Mary Organic Farm wants an online store, but how exactly everything should work still needs to be figured out on the way, and this is fine. In Scrum, there is no detailed planning that happens in the beginning. The Product Goal for the online platform that Jessica has defined together with the stakeholders and the Scrum Team is as follows, reach 10,000 online orders in the next six months. The next step is the product backlog. Initially, the backlog can be just a list of a few ideas, nothing too detailed, so she starts creating a list of features, trying to keep them as small as possible.

The first product backlog item would be to create a start page for the online shop. The next item would be to create a page to display one product, build an order form, display multiple products, add to card functionality, credit card payments, and so on, so any features or ideas regarding the product will land in the product backlog.

## User Stories

Narrator: As this stands, these items in the product backlog don't really mean a lot. So next, the product owner will work together with the developers to add more details to them. How a product backlog item is described is decided by the Scrum team. Product backlog items can have characteristics like description, order, size, value, and acceptance criteria, and generally anything that would make sense to add.

But how exactly will this look like in practice? The Scrum guide does not offer any templates or make any recommendations regarding the format itself. In practice, it is quite common for Scrum teams to use user stories. It is actually so common that product backlog items are commonly referred to as stories. Ellie suggest experimenting with the user stories format and the rest of the Scrum team agrees to give them a try.

But what our user stories anyway? User stories are relatively short descriptions of a feature explained from the perspective of the person who desires the functionality. Usually a user of the product. Commonly when writing user stories, you go through a three step process: card, conversation, confirmation, also known as the three C's. The card refers to the paper card size used to write the user story, usually a post it or similar in size. While less often used today, the main idea of the card remains the same because the card can only hold so much information, you need to be brief.

A template for a user story typically looks like this, but feel free to adapt. As though, you say here the user type. I want to reach some objective, so that I get some outcome. So here's how the first product backlog item can be formulated as a user story. Create the start page. As a customer interested in Jack and Mary Organic Farm, I want to open a browser and use some basic contact information that can help me get in touch should I have any questions. Another way of looking at the user story format, is identifying the who, the what, and the why.

Often what brings value to the conversation is understanding the why, the benefit of implementing the user story. We don't want to build things that have no value and nobody wants. I mentioned an important aspect conversation. This is the second C. The information contained by the product backlog item should be brief and to the point. Think of the items more like a reminder of what needs to be done. A conversation starter. Not necessarily detail specification. This activity helps to have the same understanding of the work to be done.

The last C is confirmation. We commonly refer to this as acceptance criteria. It is a way to test if the story has been completed. Let's add some acceptance criteria. I should see the logo of Jack and Mary Organic Farm. I should see the postal address, the email address, and the phone number. So in this case, the product backlog item is written as a user story, but this does not mean that every item in the product backlog needs to be written using this format.

There are also other ways to do so. The Scrum team decides which format works best, and yes it is okay to have different formats in the same backlog. Use which format makes sense. Remember that the Scrum guide makes no references to use the stories, and they are definitely not mandatory in SCRUM. Let's go ahead and see how another story looks like.

Display one product, as a customer interested in products from Jack and Mary Organic Farm, I want to open a browser and find some basic information about the strawberry jam product, to be better informed about the ingredients and available sizes. Here's some background information. The strawberry jam is the best selling product at Jack and Mary Organic Farm and a lot of customers ask questions about the product. Acceptance criteria.

The start page should include a button called "our products." when clicked, it should open a new page, the products page. The product page should display one or more images of the product. The product page should include a short description. The product page should include an FAQ section with questions and answers. So Jessica has written a few product backlog items using the user story format and is done for now. But the product backlog is by no means complete nor it'll ever be. More product backlog items will be created and refined in the following spreads as more is learned about the product.

While the product owner is accountable for the product backlog's content and order, it does not mean that the rest of the Scrum team cannot write product backlog items, or user stories and discuss them with the product owner. However, the product owner still remains accountable.

## Managing the Product Backlog using Jira

Instructor: Scrum does not have a section dedicated to how the first sprint should start. Initially, just a simple product backlog with some ideas is enough to get started. Product backlog does not need to be complete to start the first sprint. Essentially, we work with what we know right now. Remember that a product owner is the only person accountable for the product backlog. Product owner will typically talk to the stakeholders to understand what is needed from a business standpoint.

To store the product backlog, the product owner can use a simple whiteboard with Post-its. But nowadays in this digital age with so many of us working remotely, most teams prefer to work with a software tool. It is important to remember that the scrum guide does not impose any specific tool or way of handling the product backlog. There are many options available on the market but let's just assume that Jack and Mary Organic Farm already uses JIRA, a software product created by Atlassian.

So Jessica, the product owner, will create a new project in JIRA and will start adding ideas to the product backlog.

Currently, the product backlog is just a simple ordered list. Each item in the list is a product backlog item. The order is given by the position of each item in the list from top to bottom. This means that the developers will work on the items in the order decided by the product owner. Anything that needs to be done on the product must go into the product backlog. It does not matter if it's a new feature, a bug, or anything else that is needed. Everything goes in there. The product owner can go and edit each individual item in the list to add more details to it.

This product backlog item uses the user story format and will have a description. Jessica will also add acceptance criteria to the item to make it even more clear, and the same goes for all the other items as well. You can observe right from the start that anyone who's involved with building the website can view the product backlog in JIRA in its current state. So the product backlog is visible, ordered, and up to date. Anyone who needs access to it should have access. This helps with transparency.

## Product Backlog Refinement in Practice

Instructor: Now the stories have a description, acceptance criteria, and order, as they are the first and second items in the backlog. As you can notice, the story is still missing something. The next step would be to take this product backlog item, written as a user story, and discuss it with the Scrum team. Clarify details, see if the order makes sense, and get an estimate.

Please note that we commonly use the words estimation or estimate, but the current Scrum Guide uses the terms size or sizing. They are referring to the same thing. This collaboration between Jessica, the product owner, and the developers happens during the product backlog refinement activity. This time, Jessica will take the first story from the product backlog and present it to the team.

Right away, Tom has a question. "What if the phone number changes? Do we need an admin panel to be able to change that?" "Actually, that would be nice to have." "Maybe we should update the story to reflect this new information," says Ali. "Any other questions? If not, can you give an estimate for this product backlog item?" "For me, it's still unclear what an estimate is and how to do it," says Michelle. Ali agrees with Michelle. "Indeed, we haven't discussed yet how it will estimate product backlog items.

Let me explain a bit. An estimate in the usual sense is a guess of the effort necessary to carry out a given task, in our case, one product backlog item. But as I said, it is a guess, not a commitment or a promise. There is always some uncertainty, and that is fine." "So it means we should say how long it will take us in days?" asks Michelle. "That would be an option, but it's not the best way to think about this," explains Ali.

As in a lot of other aspects, Scrum is different from classic project management methods. You see, humans are good at comparing things, determining, for example, which building is higher than the one next to it. We have a good intuition at comparing things without actually needing to know the exact height or the number of floors each building has. We call this relative sizing. At the same time, studies have shown that people are terrible at guessing the building's actual height.

This would be absolute sizing. For that reason, let me introduce you to an approach that can help with relative sizing. Compare it to climbing a mountain. If it is just a small hill, it is kind of easy. And if you want to climb Mount Everest, that would be more complex. So when building a feature, we want to know the size first. Is it just a small hill or a mountain?

This is why we call this process sizing or estimating. "Make sense?" "Yeah, kind of, but I'm still unsure how to proceed," says Michelle.

"Let me try to put this on a scale. Imagine that zero is a walk in the park, almost no effort, and 100 is climbing Mount Everest. This would be the start and the end of the scale. Could you now pick a number between 0 and 100 that you think represents the effort of building this functionality?

Look, there's no wrong or right here." "I guess so," says Michelle. Then each of you should write your guess on a Post-it and keep it secret just for a moment. Okay, is everyone ready? Please show your Post-its. 7, 14, 19, 30. Now let's imagine that you can only pick one of the following numbers: 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100. This is an estimation technique called planning poker.

You can also do this estimation by having cards with these numbers. Now, the interval between the numbers is much smaller in the beginning and much larger toward the end. Just try it. Now we only have 8, 13, 20, and 20. "Looks much better than what we have started with," explains the Scrum master. "Let's now discuss your estimates and try to reach a consensus. Okay, you had an eight. What were your reasons?"

"Well, this is a simple page with a logo and some text. No big deal." "Have you considered that we need to have an admin panel to edit that information?" asks Max. "Oh, I'm afraid I've only estimated the design work." All right, everyone, let's try again. 20, 20, 20, and 20.

"Is there anything that can change regarding the product backlog item in order to reduce the complexity?" asks Jessica. "I guess the admin panel is something that is driving complexity up." "Okay, I'm more than happy to leave that out, at least for the moment. Can you estimate the product backlog item again, this time without the admin panel?" Eight, eight, eight, eight. "It seems that we agree," concludes the Scrum master. "Remember, in this case, eight has no meaning.

We are not talking about eight hours, eight days, or anything else. This is relative sizing. However, we will use this story as a reference when estimating other stories. We will ask ourselves if another story is smaller or bigger than this one. Later on as we learn more, we may even agree to go back to the story and estimate it again." "Great, thank you all," says Jessica. "I'll write down your estimation."

## Sprint Backlog

Narrator: During the sprint planning meeting, the product owner explains what needs to be done next to increase the value of the product. And the entire scrum team formulates a sprint goal. By collaborating with the product owner, the developers will decide which items starting from the top of the product backlog will be added to the sprint backlog. The sprint backlog will also contain a plan to deliver the product increment and realize the sprint goal.

Let's take a minute to understand the difference between the product backlog and the sprint backlog. The product backlog is an ordered list of ideas or features, the product should or could have, basically everything that could be done. We call these items in the list product backlog items. The sprint backlog is created from scratch at the beginning of every sprint, and contains the product backlog items that will be done in the current sprint.

A plan on how to deliver the functionality and the sprint goal. All items in the sprint backlog come from the product backlog. A plan is essentially a decomposition of each product backlog item in smaller work units that allow the developers to build an increment.

In practice, we often call the smaller units of work tasks. Many people learning Scrum quite often forget that a sprint backlog also contains a plan and the sprint goal, not just the selected items. Don't make the same mistake. The product backlog items represent what will be delivered. The plan handles how this will happen, and the sprint goal gives guidance and flexibility and explains why the sprint is valuable.

All are part of the sprint backlog. When the developers select the product backlog items they think that can be completed in a sprint, they create a forecast for what will be delivered. Like the weather forecast, the sprint forecast is not a guarantee, a promise or a commitment. Unexpected things may happen during the sprint.

The sprint backlog makes transparent all the work that the developers deem as necessary to reach the sprint goal. Which represents the commitment that the sprint backlog has. You can view the sprint backlog as a temporary artifact that exists only during the sprint. Every sprint will have a new sprint backlog. Any unfinished work that remains in the sprint backlog at the end of the sprint will be put back into the product backlog. The product owner will decide what should happen next. While the product backlog is the product owner's accountability, for the sprint backlog the developers are accountable.

We can say that the developers own the sprint backlog. Nobody, not even the product owner can make changes to it without collaborating with them first. The developers will modify the sprint backlog throughout the sprint as they think it is necessary to reach the sprint call. When they identify new work that needs to be done, they immediately added to the sprint backlog. The total work remaining in the sprint backlog will be tracked, at least once with every daily scrum. The developers are responsible for monitoring the progress toward the spring goal.

## Sprint Scope vs. Sprint Goal

Instructor: One of the most confusing parts of Scrum is understanding the difference between the sprint scope and the sprint goal. The Scrum Guide refers to the scope of the sprint backlog. What is that, and why can the scope be renegotiated, but the sprint goal remains the same? We'll clarify all these aspects in this lecture. Let's begin by defining them in a way that is easy to understand. Think about your upcoming weekend.

I know this does not sound like fun, but say you want to clean your place on Saturday. This is your goal; to have a clean place to live in for the next week. The scope is the work that you want to do to reach this goal.

For example, you want to clean the bathroom, wash all the dishes, vacuum the floors, clean the windows, and so on. So essentially, the scope is what you want to do and the extent of the work. By extent of the work, I mean vacuuming, for example. Do you need to move the furniture, or can you keep it simple and leave the furniture as it is and vacuum in that way?

You have Saturday at your disposal. The scope can change as long as the goal remains the same. If during the day, you notice that you are running behind your plan, you may reduce the scope.

For example, you decide to skip cleaning the windows and the garage. So your scope has changed a bit, but you are still committed to having a clean place by the end of the day. Having a goal is important as it keeps you focused. Being focused prevents you from starting other unrelated work, like baking or, I don't know, washing your car. In Scrum, the sprint scope represents the functionality that will be developed during the sprint. The scope of the sprint backlog is the amount of work selected by the developers. In other words, the selected product backlog items. The scope is flexible and can be modified during the sprint. The terms scope, sprint scope, or scope of the sprint backlog are referring to the same concept. If work is not progressing as expected, Scrum will not do miracles. However, with the time available, the developers will try to stay focused and do their best to reach the sprint goal. The scope renegotiation with the product owner can happen anytime. The idea is to meet the sprint goal while making some compromises, usually in terms of features, never on quality. In this process, the product owner and the developers decide which is the best course of action. Some items in the sprint backlog may not directly contribute to reaching the sprint goal.

Others can be simplified. Items can even be replaced with other items from the product backlog. Anything is possible as long as the sprint goal is not endangered. In practical terms, it is rarely the case that the entire sprint backlog will contribute to reaching the sprint goal. While most of the items in the sprint backlog must be related and must help reach the sprint goal, it is not always possible to have a sprint backlog that only focuses on reaching the sprint goal. And this is also the part that allows for some flexibility to happen.

## Increment & Definition of Done

"Instructor: The increment is a new version of the product and is additive to all previous increments from all previous sprints. You can view it like a building block placed on top of all previous work completed. The developers work to deliver at least one new product increment with each sprint. Each new increment is an improved and usable version of the product. It is solely the discretion of the product owner if and when to release the increment. Still, the increment needs to be usable without needing any additional work, like testing, documentation, or even integrating it with the work other scrum teams did.

When a product backlog item or increment is considered complete or done, everybody must understand what "done" means. Sometimes the organization may define quality standards, and those need to be followed at a minimum to make it easier for everybody to understand what complete work looks like. The scrum team should create a definition of done which can be used to assess if the work performed is following the agreed quality criteria. I know it is hard to grasp this concept of complete work described in the definition of done. Try to remember that the definition of done is mostly about quality.

Allow me to simplify a bit. Let's assume Jack and Mary would define some minimum requirements for any of their products. For example, it should contain only organic ingredients, raw materials should come from small independent farms, the logo of Jack and Mary must be visible on the product, and the product label must indicate the number of calories, sugar contents, and so on. It should also conform with health and safety standards. So if you create a new and improved strawberry jam with less sugar but you don't label it properly, it is not done. It is incomplete as you cannot sell it to the customers.

In scrum, each sprint will create at least one product increment that needs to adhere to the definition of done. If a product backlog item in the sprint does not follow the definition of done, it will not be included in the product increment. As the Scrum team gains more experience, it is expected that the definition of done will contain more strict criteria to ensure higher quality. During the Sprint retrospective, the Scrum team plans on how to increase product quality, and one way to ensure this is by adapting the definition of done.

There is no standard on what a definition of done should include as this can vary from team to team and from product to product. By using a definition of done, it becomes transparent for everyone what it means for the product increment to be done. The definition of done will also guide the developers on how many product backlog items to choose during the sprint planning meeting. The definition of done represents a commitment for the increment. As soon as one product backlog item is completed and meets the definition of done, a new increment is created.

## The Definition of Done for Jack & Mary Organic Farm

Instructor: In this lecture, we'll take a look at how the definition of done looks in practice. The Scrum Guide does not say which format the definition of done should have or where and how to store it. You can have it on paper, but quite often teams use shared collaboration spaces to work together on documents. One very popular tool is Confluence from Atlassian, and this is precisely what you see right now. Just as a reminder, Atlassian is the same company that created Jira.

What you see here is the definition of done for the eCommerce platform that the Scrum team is building for Jack and Mary. This is a software product, and the definition of done is rather technical. Don't worry if there are aspects you don't understand, that is totally fine.

While the definition of done may look like a checklist, we don't refer to it as a checklist, and definitely the Scrum Guide does not call it a checklist. So be very careful when you are referring to the definition of done as a checklist. Try to remember, as this aspect is important.

Let me take you through this example. And you can see here, right in the beginning it says something like, before a product backlog item is considered done or complete, it must respect the following criteria. And one of the first things that I mentioned here is that any work that is completed, or the increment itself, must first meet the acceptance criteria for that item. So, say developers have completed one item, and they also need to check that all acceptance criteria that were mentioned in that item are met, as well, as part of the definition of done. So sort of, we can say that the definition of done also checks again or makes transparent to everyone that, hey, if the acceptance criteria is not met in an item, it cannot be done.

In this case, the definition of done is separated into different blocks, and this is simply an example. In terms of quality, for software products, we often talk about testing and just generally making sure that the code that has been written is of the highest quality. And one of the ways this happens is by making sure that any code that was written is reviewed by other developers. So developer A is writing the code and two other developers look over it and give their point of view, maybe make suggestions. Generally just making sure that whatever was done is really, indeed, of the highest quality possible.

Quite often, the definition of done will also include documentation. This can be technical documentation or user documentation. For a website, the user documentation may be, for example, like the help pages explaining how to make an order or something like that. It's not the case here because the product is still very early in the development, but it may include technical documentation explaining how different things, for example, how to deploy this application and so on.

In terms of coding, there are a lot of automated tests that can be executed. The smallest unit of tests is a unit test. This is essentially testing a small piece of the code. On the other side of the spectrum, we have acceptance tests which are practically testing if the entire application as a whole is working as expected. There are also various levels of testing in terms of cross-browser tests. So you can either do it manually, you can do it automatically, essentially you have an application and you're testing it on different browsers. You want to make sure it works in Google Chrome, works in Firefox, works on a Mac system, and so on. The same goes with mobile devices. So we want to make sure it works on iPhones, works on Samsung Galaxy, or on Google Pixel, and so on. So various levels of testing, just to ensure that everything looks and works the same regardless of where the users are coming from.

Accessibility testing is also something that's very important to allow a software product to be used by various types of people. Some of them may be visually impaired or may have other disabilities that are preventing them from using the respective applications. So being inclusive when doing software development is very, very important.

Quite often, teams use automation tools. One of these is continuous integration, an approach that ensures that everything happens automatically. So for example, all the tests that we mentioned here, it's not about having someone manually running them but essentially any change that is being made on the product will go through this pipeline of testing and just to ensure that everything is working as expected.

Another aspect of the definition of done can be the so-called non-functional requirements. So essentially, a functional requirement is describing, for example, hey, you have a green button and when you click it, it should do this, or I have a form and it should include first name, phone, and address, and so on. That would be a functional requirement. A non-functional requirement can be something that you cannot directly see or feel, but you know it's there when something doesn't work. So examples of non-functional requirements will be, for example, performance. So when you submit an order, you want it to be as fast as possible. For example, here, it shouldn't take more than two seconds to do any action, to maybe log into your account, to view a product, to view a page, to submit an order. So this is a non-functional requirement specified here, so it means that the developers must ensure that these requirements are met through various kinds of testing that they do.

The same thing goes for availability. For example, if you have the Christmas period coming and you have a lot of people on the website, maybe making a lot of orders, you don't want your application to crash. So in this case, you also do testing, and this is like a very clear way of specifying, hey, this platform should support, essentially, 5,000 concurrent users that are coming to the platform. It's not a big website, but still, it can make transparent what is the load that this system should be taking.

The same goes with security. Security is something that you don't necessarily feel or see, but it's very, very important. So in this case, the security requirement here in the definition of done is to ensure that any code libraries that are included with the product, code libraries are essentially external modules that are added to the product itself to simplify the development work, that they are secure and there are automated ways of ensuring that everything is, indeed, secure.

So this would be the non-functional requirements. And of course, the definition of done can include any other requirements that are necessary to ensure that the product itself, the product increment, is releasable, and that everyone has the same understanding of what it means for work to be complete or what it means for work to be done.

## When is the Definition of Done created?

"Instructor: One common misunderstanding is about a point in time when the Scrum Team should create a Definition of Done. If you have been reading the Scrum Guide closely, you may have noticed that this aspect is not mentioned at all. The Scrum Guide only explains the need for the Definition of Done. This is because Scrum is a framework, not a methodology or anything else that explains what you should do, like step-by-step.

So when should the Definition of Done be created? Essentially, whenever you wish, as long as you don't break any Scrum rules. The main requirement from the Scrum Guide is that there must be a Definition of Done during the Sprint Planning event.

Often, people think that everything needs to happen in one of the Scrum events, and this is not accurate. The Scrum Team can also have other meetings. This is not prohibited by Scrum. Nothing prevents the Scrum Team from defining the first version of the Definition of Done before the First Sprint begins. The initial Definition of Done is not perfect or final. The Scrum Team can adopt it in the upcoming Sprint. The best opportunity to do so is at the end of the Sprint, during the Sprint Retrospective meeting.

Adapting the Definition of Done, say, in the middle of the Sprint, is not a good idea as it may impact the forecast and any existing work that has been completed already."

## Definition of Done vs. Acceptance criteria

"The coming topic of confusion is regarding the definition of done versus acceptance criteria. Let's look again at an example. You can notice the definition of done is very general and makes no reference to a particular functionality. If you go ahead and remove the title of the document, you may not even know what kind of product this is referring to. The definition of done tends to be more technical, talking about code quality, code coverage, security best practices, how many code reviews are needed, and so on.

On the other hand, acceptance criteria are specific to a particular item. The acceptance criteria try to describe when an item is complete, mostly from a functional perspective. This is a way to complement the user story if this format is used. Acceptance criteria apply to a single PBI (Product Backlog Item), while the definition of done applies to that PBI but also to the entire product increment.

Let's recap a bit. The definition of done applies to every PBI and the increment, while the acceptance criteria apply to a single PBI. The definition of done is created by the Scrum Team, while acceptance criteria are typically created by the product owner unless the product owner has delegated this responsibility to the developers.

The definition of done is the commitment for the increment, so the increment must always fulfill the definition of done. Acceptance criteria are part of a product backlog item. The definition of done is mandatory in Scrum; the Scrum team must always have a definition of done, while acceptance criteria are an optional attribute of a product backlog item. You can use it, but Scrum doesn't force you to have it. It often helps to bring more clarity. Some items can have acceptance criteria, while others may not have it.

For one product backlog item to be released to the users, it needs to meet the acceptance criteria if they exist and the definition of done. You may even have a definition of done that explicitly mentions that the acceptance criteria must be fulfilled. The increment created from one or more PBIs also needs to meet the definition of done.

That's a short overview of the differences and similarities between the definition of done and acceptance criteria."

## What is an epic?

"In real-world projects, you may encounter the term epic. Here is a brief introduction to this concept, which, by the way, is not a Scrum concept and is not mentioned in 'The Scrum Guide.'

An epic is a large body of work that we want to do. As the name suggests, it is significant in size and does not fit in a single sprint. Let's imagine that Amazon.com wants to do a UI redesign. That's an epic. It needs to touch different parts of the website and requires a lot of work.

By having conversations with the developers, a product owner can get early feedback and understand if a particular feature is an epic, meaning it is really big and we need a few sprints to do it, or just a regular story that can be done in a sprint without much effort.

Since we can't pull an epic into a sprint, what we want to do is to break it down into multiple user stories. Let's see how this looks in practice.

In Jira, there are multiple ways to create an epic. From the backlog view, I can simply enable the epics view and, by clicking the plus sign, create an epic. Let's call the epic 'Website Redesign' and provide a short description.

When we break down an epic into user stories, we typically mark those stories as belonging to an epic. In Jira, we do that by linking a story to an epic. This makes it easier to visualize the work and get instant context about the stories in the product backlog. In tools like Jira, you can easily filter by epics or track how many stories need to be done until the epic is also done.

As you can notice, the epic is not visible in the product backlog. It is not something that we can pull into a sprint. By breaking down an epic into smaller pieces, we can use Scrum and follow all rules. And as a side benefit, we can deliver value with every increment, even if the work on the epic still continues.

So to recap, an epic is just a large amount of work that spans multiple sprints. To better manage the work, we break down the epic into multiple user stories that we can then pull into sprints. Remember that the concept of epics and user stories are additional practices on top of Scrum and are not mandated by 'The Scrum Guide'."

# Section 3

## Scrum Events

Scrum uses prescribed events or coding meetings or ceremonies if you prefer to create a routine and to reduce the need for other meetings that are not defined in Scrum. This does not mean that the Scrum team cannot have other meetings but it is mandatory to have the Scrum events.

At the heart of Scrum is a Sprint which acts as a container for all Scrum events. Remember that there are no pauses or gaps between Sprint and everything happens within the Sprint container including the Scrum event. For this reason, the Sprint is a special kind of an event. All events within Scrum have a maximum duration and are therefore time boxed.

Events are designed to enable transparency and inspection. The Scrum guide recommends that the events are held at the same time and place to create a routine and to reduce complexity.

Having one meeting last the schedule is always a win. Sprint planning, where the work to be performed in the Sprint is planned. The daily Scrum, which is held every day of the Sprint. Sprint review, which is held at the end of the Sprint to review the increment and Sprint retrospective, which is an opportunity to discuss ways to improve our old Scrum events which are a former opportunity to inspect and adapt.

In other words, old scrum meetings are an excellent opportunity to get feedback and to take action based on the feedback received to achieve the desired results.

These events are mandatory and the Scrum team cannot simply decide to skip or postpone them. While Scrum makes this events mandatory, it does not mean that the Scrum team cannot have other meetings.

It is a common misconception to assume that any discussions within the Scrum team or between the Scrum team and the stakeholders must happen in a Scrum event.

One last thing. The product backlog refinement activity is not a mandatory Scrum meeting and it is not considered a Scrum event.

## What is a Time Box?

What is a Time-Box, we will clarify this in this lecture.

In Scrum, the Time-Box indicates the maximum duration of an event. Let's take the Sprint planning meeting, for example. This event is Time-Box to eight hours for a one-month Sprint. The Scrum team can use exactly eight hours to plan the Sprint, and this is totally fine. The Scrum team can reach the purpose of the meeting in only six hours. That is two hours less than the Time-Box and that is fine as well.

As soon as the event has reached its intended purpose, the meeting ends. There is no reason to sit around and waste time. Scrum is trying to reduce waste not to create it. What is not allowed is for a meeting to go over the Time-Box and the Scrum master should coach the Scrum team why Time-Boxes are important and how to manage their time better. Why are Time-Boxes important? Well, think about the following situation in your life. Did you ever have to catch a flight and manage to do your luggage in one hour when you typically need half a day? Isn't this amazing?

Okay, maybe you have forgotten your toothbrush, but you have saved quite a few hours and you can still buy a toothbrush at any store in the airport or at your destination. This is how Time-Boxes work. They force you to focus on the most important thing. In this case, catching your flight. The Scrum team must make the best decisions based on the information available.

Things are never perfect. Time-Boxes are also useful for making any issues visible and having the Scrum team discuss them in the following retrospective. So to recap, a Time-Box indicates the maximum duration of an event, not the minimum duration.

Any scrum event can end before the Time-Box expires as long as the purpose of the event has been reached. Under no circumstances is a Scrum event allowed to go over the Time-Box.

Time-Boxes apply to all Scrum events apart from the Sprint itself, which has a fixed duration. The Sprint is a special case.

Even if you finish the work planned before the Sprint ends, the Sprint does not end earlier. Doing so will break the routine that Scrum is trying to create. If you have extra time, you can just go ahead and pull more work from the product backlog.

But I can assure you this is rarely an issue in practice.

## The Sprint

A sprint has a duration of one month or less in which a potentially shippable product increment is created. A sprint in Scrum is not like a sprint in sports. It's not about using your entire energy in a short time. Any development work must happen at a sustainable pace. Sprint after sprint, working overtime, using up your entire energy and ending up burned out. It's not something that aligns with the scrum values and has also been proven to be ineffective over time.

If the duration of the sprint is too long, the complexity and risk may increase. Think about it like this. What is easier?

A, to plan your next week.

or B, to plan your entire next year?

Having these relatively short horizons, it is easier to plan what is being built and to get early feedback. This allows the Scrum team to adapt the plan as they learn more. The idea of the sprint is to have the Scrum team focused on a specific goal. Sprints are used to accomplish something even if it is very small. Each sprint has a sprint goal, an objective that will be met within the sprint, which helps the Scrum team better understand why it is building the increment. No distractions, no interruptions. Nothing is more important than reaching the sprint core.

This is challenging for organizations where everything is important, and everyone works on four different projects at the same time. Everyone is busy with work, but nothing is completed.

Scrum is different. The sprint is like a small project that should produce a working product by the end of it, at least after one sprint you know if your idea was good or bad. Scrum is never about getting everything right from the start. This is impossible in complex environments. Sprints contain all prescribed scrum events of flexible plan on how to build the increment, and of course the development work needed, which to be honest occupies most of the sprint.

Apart from this, the sprint will contain the product backlog refinement activity, and any other meetings that the scrum team deems relevant. During the sprint, no changes should be made that endanger the sprint goal. It is also essential that the quality standards do not decrease.

Especially, if the sprint is coming to an end. As the product increment is being built, new things are learned. When necessary, the scope of the sprint may be clarified and renegotiated between the developers and the product owner. We can reduce the scope, but we never compromise on quality. The duration of the sprint will remain the same over multiple sprints. We don't extend the sprint if we feel like we need more time to finish the work. Let me give you a few examples. If you're working on accounting software, the legislation will typically only change once per year. So you can see that the marketplace is rather stable and requirements are more predictable.

So having a sprint duration of one month may be a good choice. The technology used can also play a factor in regard to the length of the sprint. Building a physical product is much harder than with software, so again, you may need longer sprints. If you're building the next social media app, you may need to react fast when Instagram or TikTok introduce new features. So, waiting one month until you can adapt your plan is way long. Probably one- or two-week sprints work best.

Most commonly, sprints tend to be two or three weeks long, especially if you're working on software products. The duration of the sprint should not be dictated by the organization. The Scrum guy does not explicitly mention who gets to decide the duration of the sprint. As the scrum team is self-managing, they need to figure this out depending on various factors.

The Scrum team in collaboration with the stakeholders and the organization, may decide a sprint duration and align it with internal events, current market conditions, or any other factors deemed relevant. If the Scrum team gathers empirical data and thinks they can benefit from shorter or longer sprints, they should be able to change the duration of the sprint.

Like many things in Scrum, you can't get everything right from the start, and you need to adapt along the way. You do something, measure the results, and adapt as needed. The sprint duration is not a topic of discussion during sprint planning and has nothing to do with the scrum team's capacity, the amount of work or the sprint goal. The duration of the sprint is not connected to the release of the product, as this can happen during the sprint. The sprint review meeting is not a release gate. The work doesn't have to be reviewed in the sprint review before it is released.

## Sprint Planning

Let's begin with the first event of the sprint.

Sprint planning. You need to pay close attention to this event as this is one of the main topics in the exam. The sprint planning event is all about planning the sprint and it's the first event of the sprint. As with any other scrum events, it makes sense to hold it at the same time and place to reduce complexity.

The sprint planning meeting is time boxed to a maximum of eight hours for a one-month sprint. For shorter sprints, the event is usually shorter. Strictly following the scrum guide. The time box is still eight hours even if the sprint is only two weeks long, but it is expected for the meeting to take less in this case, however, we do not say that it will be proportionally less.

Each sprint is like a small project. Instead of planning over a long period, in the sprint planning event the scrum team will only plan a very short period, no more than one month, whatever the sprint duration. As there are no gaps between sprints, sprint planning happens right after the conclusion of the previous sprint, which is marked by the sprint retrospective.

Now, don't get me wrong this does not mean that sprint review, sprint retrospective, and sprint planning needs to happen on the same day. The main idea that you need to take from here is that there is no pause or preparation period for the upcoming sprint between the sprints.

No other work like product like local refinement, testing, fixing bugs, or releasing the increment happens between the sprints. There is no gap between sprint. Everything must happen within a sprint. Starting a new sprint is also never delayed for any reason. Even if some items in the product backlog are not clear enough or unrefined. The scrum team can use the sprint planning meeting to refine the product backlog.

If there are, for example, new insights from the sprint review meeting. The sprint planning event tries to answer the questions of why, what, and how. Why is the sprint valuable? What can be done with this sprint? How will the chosen work get done?

The outcome of this meeting is the sprint backlog, which contains the sprint goal, the selected product backlog items that would help achieve the goal and the plan for the sprint.

## Sprint Planning - Why is this Sprint valuable?

-: Let’s look at the questions of the sprint planning event one by one.

Why is this sprint valuable? This is a question that the scrum team should try to answer. As each sprint can be seen as an investment and it is expected that the sprint will generate value in some form. No organization should invest resources to build things just for the sake of it. So, each sprint should create something valuable and be a step toward reaching the product goal. The primary tool that the product owner has is the product backlog. Through the ordering of the PBIs the product owner will decide what will be done next. So what is at the top of the product backlog will be implemented in the sprint.

It is that simple. The product owner could suggest which items from the product backlog can make the product more useful and generate more value. To make this clear and actionable The entire scrum team will define a sprint goal which is the objective that the sprint should achieve. So the entire scrum team defines the sprint goal, not just the product owner. It helps if the sprint goal is specific and measurable but leaves some room for flexibility in the implementation.

Lets take a look at some examples. Customers can order a single product. This sprint goal does not get into the technical aspects of how they should look and work, which product can be ordered, how customers will pay, and so on. It creates some flexibility during the sprint. It allows the scrum team to adjust their plan so that they will reach this goal. At the end of the sprint it is easy to understand if the sprint goal has been reached or not. Lets have a look at some other examples.

Improve the average order size by five percent by suggesting customers other products on checkout. And you have probably seen this often when shopping online. You will have your shopping cart and below it you will have some suggested products and maybe you'll click on some of them. Decrease the number of customer complaints by ten percent by allowing them to select a delivery date when placing the order. These goals are a bit more specific and have a strategy on how to reach a specific metric. With the last two goals it may seem harder to know if they have been reached at the end of the sprint.

Here is how to look at this. The scrum team may have different ideas on how to solve a problem but no one can know for sure. They can try out different approaches during the sprint, release the product to actual customers and measure the results. All within the sprint. We implement the specific feature. You release it and use empirical data to see if you have reached the goal. For digital products we have the technology to measure the impact of the changes we make in real time. It is a misconception that you need to wait for the end of the sprint to release.

The scrum team does not need to wait for the sprint to end to release the increment and test their assumptions. They can release their work even multiple times per sprint. The sprint goal will also bring transparency towards the stakeholders and help them understand why the sprint is valuable and get a rough idea of what to expect at the end of it. By the end of the meeting the sprint goal must be defined and everyone should have a clear understanding of what the sprint is trying to achieve. It is not possible to change the sprint goal after this meeting.

## Sprint Planning - What can be Done this Sprint?

Now, let's move to the second question. What can be Done this Sprint?

Done, written with capital D, is a hint to the definition of Done. This is a reminder that only Done work is useful and releaseable. By understanding the Sprint goal and collaborating with a product owner, the developers will forecast what can be Done in the Sprint. Allow me to explain the term forecast. When you hear the word forecast, think about the weather forecast.

You are probably looking at your phone's Weather app to see how hot or cold it is outside and if you should carry an umbrella. Most of the time, the forecast will roughly match the reality. But did the Weather app on your phone ever announce a sunny day but you got clouds and maybe some rain? This is what a forecast is. You are never 100% sure, nobody guarantees it. The weather people make some predictions based on some complex calculations and historical data that they have. Nobody knows for sure, but most of the time, they are just right or just about right.

A forecasting Scrum is very similar to this and should never be confused with a guarantee or a commitment. The forecast represents the PBIs selected by the developers that they think they can finish by the end of the Sprint. The order of the items stays the same as they were in the product backlog.

The developers cannot cherry-pick items by selecting only what they wish. So the product owner decides the order in which the work will be done, but they cannot decide the amount of work the developers will do. This does not mean that they don't collaborate in this process.

They are all on the same team and share the same goal. Of course, they collaborate. If the product owner feels that the developers have not selected as much work as they have expected, nothing stops them from talking about this and maybe reducing the complexity of some of the work by making trade-offs.

It is not uncommon for the Scrum Team to refine items in the product backlog during this event. So product backlog refinement can happen anytime during the Sprint, but also during the Sprint planning meeting. Remember that the number of product backlog items selected is solely up to the developers. The product owner cannot force them to select more work than they think is possible. This is important to remember. Pull, not push. The developers pull work. Work is not pushed or assigned.

You should expect that most of the PBIs in the Sprint are somehow related to reaching the Sprint goal. Now, there is no requirement in the Scrum Guide saying that absolutely all PBIs must contribute to the Sprint goal. This would be impractical. The Scrum Team may also need to do other work like implementing improvements that they have discussed in the last Sprint retrospective, fixing defects, or implementing smaller features.

There is always some less important work than the Sprint goal, but that still needs to be Done. You can't and probably should not try to formulate a Sprint goal that covers all selected items. Teams often struggle to formulate a Sprint goal because their Sprint is handling too many unrelated PBIs. As a side note, Scrum does not use the term hand-off when referring to the selected product backlog items. Pay attention to the Scrum vocabulary here. Scrum is not a process where work happens in stages. There are no hand-offs from one person to the other.

The product owner will not hand off a list of items to the developers for implementation. A developer will not hand off a task to another developer. In order to make the previously mentioned forecast, the developers have a few aspects to consider, so they will take into account their past performance, we also call this velocity; the capacity for the Sprint; and the definition of Done.

Let's briefly go over them. The past performance refers to the amount of work completed in previous Sprints. This is what we call empirical data. If you managed about 20 story points of work per Sprint in the past three Sprints, it is highly unlikely that exactly in the Sprint you will manage 40.

The capacity of the Scrum Team refers to the time in hours or days the developers are available to work on the Sprint. Let's say that in the last Sprint, they have completed 20 story points with all developers available. If, in this Sprint, half of the developers are on vacation, it is unlikely that you will manage just as much work.

Not impossible, but unlikely as now the capacity is only 50%. Finally, there's the definition of Done. This is probably one of the things you would not think is that important. I suppose you may be wrong. If you adapted the definition of Done in the Sprint retrospective to include higher criteria for quality, you need to spend more time on quality.

Say your test coverage was 40%, but you want to bump that to 60%. That is a lot of work that goes on quality, but this also means you will complete fewer features, so you need to take this into account as well. All these aspects don't form an exact mathematical formula that predicts how much work to select.

Many things can happen during the Sprint that are totally unexpected, just like the weather, but it still gives a very good indication of what is possible. Their forecast may be less reliable in the first Sprints, but over time, the developers tend to develop a very good sense for this. Just remember that developers don't commit to doing all the work selected. They're only committed to reaching the Sprint goal.

## Sprint Planning - How will the chosen work get done?

-: Let's move on to the third and final question.

How will the chosen work get done? Once the Sprint goal has been defined, and a product backlog items for the Sprint selected, the developers discuss how the functionality will be built into a product increment. Only The developers decide how to plan and do their work. They are responsible for the development work, and nobody tells them how to do that.

Even if we find a single PBI is often too large to be directly implemented without any further discussion. The developers break down the work into units of one day or less. While the scrum guide does not use this language, this is essentially when you break down a PBI into multiple tasks.

The developers will discuss which are the steps they need to take to complete the item. Since they're the professionals doing the work, they have a very good idea of what needs to be done. They also use the definition of done to identify work, and they keep in mind that they need to create at least one working increment by the end of the Sprint.

This breakdown of work or call them tasks, if you wish is not part of the PBI, but part of the Sprint backlog. This is the plan for implementing the PBIs. The scrum guide does not require each task to be estimated, but don't be surprised if some teams choose to do so. As they plan the work the developers may notice that their forecast is no longer accurate. In case the work is too much or too little, the developers may renegotiate the selected product backlog items with a product owner during the meeting or even during the Sprint.

Because of work emerges during the Sprint, this meeting cannot identify all the work that needs to be done in advance. So the Sprint backlog is never complete after this meeting. It is just a plan with enough details so that the development work can begin. The Sprint backlog will be adapted throughout the Sprint. The only thing that cannot change after the meeting, is the Sprint goal.

As long as the meeting has not ended, it is also possible to change the Sprint goal, or the forecast for the Sprint. Even if the developers distribute work between them, a single developer will never own an item. Again, pay attention to the scrum language.

All developers are responsible for the development work and the outcome, even if a single developer implements an item. The desired outcome of the Sprint planning meeting is the Sprint backlog, which will contain the Sprint goal, the selected PBIs, which is the forecast for the Sprint, or sometimes called the scope of the Sprint, and the plan which is typically the breakdown of work into smaller units.

## Sprint Planning - Who attends the meeting?

-: Let's look for a moment at who attends the Spring Planning meeting. The Scrum guide describes the outcome of this event as, the collaborative work of the entire Scrum Team. Spring planning is a mandatory scrum event and the entire Scrum Team must be present even if the product backlog is refined and clear the product owner must still attend this meeting as well. This is not something they can delegate. While the product owner and the developers have a clear role in this event

Scrum Master still plays an important function. Scrum Master needs to ensure transparencyover the Scrum Artifacts. In other words, to ensure that everyone has a shared understanding. Scrum Master should explain the purpose of the meeting if they think it is not clear. Make sure it is positive and productive and help the Scrum Team keep the event within the time box. It is easy to get lost in details and the Scrum Master guides everyone toward reaching the desired outcome of the meeting before the time box expires. The Product Owner is knowledgeable about the product and can answer most questions that the developers have.

However, the Product Owner is not always the expert in every topic concerning the product. When needed, the Scrum Team may invite people outside of the Scrum Team to attend and provide advice, often technical advice or related to the product domain. However, this is not the same as saying that anyone can attend this meeting or that the Stakeholders attend this meeting. So nobody outside of the Scrum Team attends this meeting unless invited.

There can be security or regulatory requirements that the Scrum Team needs to handle and experts from these domains can give advice. For example, it makes sense to invite maybe the Chief Security Officer to explain some security aspects that the product must fulfill like collecting and storing personal information.

Needless to say, nothing is preventing the developers from collaborating with these experts throughout the sprint. Here's a quick remark in terms of the language used by the Scrum guide. Providing advice is not the same as telling the Scrum Team how to do their work. So to recap, the Sprint Planning event is for the entire Scrum Team. Everyone must attend and they can also invite others to join to provide advice.

## Coherence among the Sprint Backlog Items

Let's spend a bit more time to understand the Sprint goal. The Scrum guide says that the sprint goal creates coherence. Now, what does it mean? First of all, the word coherence means that the team is working on a joint initiative, instead of operating in different directions. Coherence comes from coherent, which refers to something logical and consistent. I'll give you an example that you can easily understand. I'm sure you'll be able to translate this into whatever work you're doing. Say you want to build a house, this is your product goal. Every Sprint is a step toward reaching that goal.

Let's go into the planning of one sprint. In this sprint, you want to get all the permits for building the house. In this case, you may have multiple items in the sprint backlog, each item representing one individual permit that you need. So, if the Sprint goal is get approval to build the house, most of the items in the sprint backlog should be related to this. This is the concept of coherence among the items in the Sprint backlog. In this Sprint, it does not make sense to plan for the kitchen and decide which appliances you will buy.

Yes, it needs to be done, but that does not help you with the current Sprint goal. So, completing the items in the sprint backlog should help reach the sprint goal. In other words, there is coherence among the selected product backlog items. You can logically see a connection between them, and understand how completing them helps reach the Sprint goal.

This also allows the Scrum team to focus on a single direction. Now, don't get me wrong, this does not mean that absolutely all items need to be related to the Sprint goal. This is often impossible. But still, if you look at the Sprint goal, you should clearly see the relationship between the items and the Sprint goal.

Most teams are struggling to formulate a meaningful Sprint goal, precisely because of this. Their Sprint backlog is just a list of items that need to be done. They have almost no relation between them. The focus here is not on reaching the Sprint goal, but on completing all the work selected, which is not what's Scrum is all about.

Without a shared goal, the Scrum team will collaborate less, as each developer is working on a separate initiative. You need to remember that in Scrum, we only have one Sprint goal per Sprint, not multiple, and having a sprint goal is not optional. This helps us focus on a single goal, and work together towards reaching that goal.

## Daily Scrum

The daily scrum is a time box event held at the same time and place, each day, to reduce complexity. In practice, we often call this meeting daily or daily stand-up, even though there is no requirement in scrum to actually stand up during this event. Since these names are informal, make sure you don't select these answers during the exam. The official name of the event is daily scrum.

The daily scrum is held every working day during the sprint, and it is an event intended for the developers. And I'm stressing this a bit because it is important. During this event, the developers plan, what work will be performed in the next 24 hours. This is a key inspect and adapt meeting in scrum, or you can call it a feedback loop, if you wish. This means that the daily scrum helps the developers inspect progress towards completing the work in the sprint backlog and reaching the sprint goal. If needed, they will adapt their plan.

As this is a mandatory event, every developer must attend. By the end of the meeting, every one should have a shared understanding of how work is progressing, and what are the next steps?

The developers are responsible for monitoring the progress during the sprint. The structure of the meeting is decided by the developers, and can involve questions or be more discussion-based. Any format works as long as it focuses on the sprint goal and planning the work. Just remember that the developers are self-organizing and they decide how to conduct the meeting.

The scrum guide does not mention using a scrum board or updating the burndown chart during this event. There're optional practices that can be used, but are not mandatory. The interactions that the developers have, improve communication, identify impediments, and promote quick decision making. Nobody tells the developers what to do, they're self managing their work. Regardless of the size of the team or the length of the sprint, the daily scrum is a 15 minute time box event.

Because this event is so short, we can say it has a duration of 15 minutes, but of course, if the meeting has reached it's intended purpose, there's no reason to stick around and waste time. Sometimes the developers don't see the value in having the daily scrum. Even if the developers constantly interact during the day, it's still mandatory to have this event. This is where the scrum master steps in as a scrum coach. The role of the scrum master is to make sure the developers have the meeting, understand a purpose, and keep it within the time-box.

The scrum master does not need to be present to do this. The scrum master is not responsible for scheduling the meeting or deciding which format should be used. The developers are self managing and can decide this on their own. If they need suggestions, they can ask for help. When needed, the developers can meet during the day to further discuss and decide on how the rest of the sprint work will be completed. A common misconception is that the scrum master needs to be present to learn about new impediments.

This is not true. The developers can communicate any impediments to the scrum master right after the meeting. Remember, the developers, the product owner, and the scrum master, are part of the same team, and they do communicate during the day. Their interactions are not dictated by the scrum guide.

The product owner and the scrum master should attend this meeting only if they do work in the sprint backlog. So they will attend as developers. Keep in mind that this is a meeting for the developers and not for reporting progress to the product owner or the stakeholders.

There is no reporting in scrum. People collaborate to inspect the scrum artifacts, for example. Any other discussion within the scrum team, can happen outside of this prescribed scrum event. Any urgent news or impediments can be shared with the rest of the team immediately after this meeting. There's no reason to add people and topics of discussion to the daily scrum, just because the meeting takes place.

Overall, daily scrum improve the communication, and reduce the need for additional meetings,identify impediments, and help the developers make quick decisions. The daily scrum is how the developers plan the next 24 hours.

## Scrum Board

During the Sprint, the developers will collaborate and distribute the work between them. They decide who should do which task but they all remain accountable for the work. A Scrum board is a tool that helps developers make the work in the Sprint backlog visible. When planning their work, the developers break down the product backlog items, or user stories, into small work units that we often call tasks. Here is how a Scrum board looks like in Jira.

At a minimum, it typically has the columns TO DO, IN PROGRESS and DONE. Each developer will let others know what they are doing by pulling a task from the TO DO column to the IN PROGRESS column. In Jira, when you move a task, it will automatically be assigned to your name. In this, way everybody knows which tasks are still to be done, which are in progress, and finally, which of the tasks are completed. Tasks typically move from left to right and create a workflow.

The developers can also add other columns which match their workflow like testing or review. The developers will use a Scrum board most of the time and this is an indirect way on how they communicate regarding what needs to be done. The Scrum board can also be useful during the daily Scrum to visualize what has happened yesterday and plan the next 24 hours.

While most Scrum boards are digital you can also have a physical board with Post-Its or cards. Even if using a Scrum board is a very popular tool this is not explicitly mentioned in the Scrum guide and is not required in Scrum. The developers are self-managing and decide which tools to use. For this reason, exam answers referencing a Scrum board are rarely the best choice. So remember that a Scrum board is an optional tool in Scrum, used to visualize the Sprint backlog.

## What happens during the Sprint?

Instructor: When the sprint planning meeting is over, this is when the development work begins. Here's a brief overview of what typically happens. The developers will have every day the Daily Scrum and will mostly focus on turning the selected PPIs into working product increment that generates value. Throughout this process, the developers will collaborate with a Product Owner to clarify any items, ask questions, and show any progress.

Together with the rest of the Scrum Team, the developers will also refine the product backlog or the upcoming sprint. This is not the rule, but most teams tend to do at least one refinement session per week. Remember, product backlog refinement is not a mandatory Scrum event. The Scrum Team decides if and when to have it.

Apart from the Daily Scrum, there are no other pre-prescribed Scrum events during the development period. The developers and the rest of the team will also attend other meetings that are not defined in Scrum, and this is totally fine.

The Scrum Team may also decide to release any available increments during the sprint. There is no need to wait for the sprint review meeting. Apart from collaborating with the developers or the Scrum Master, the Product Owner will closely work with the stakeholders to better understand their needs, communicate the most likely release of the increment and add new items to the product backlog.

Essentially, it all revolves around managing the product backlog and collaborating with the Scrum Team and the stakeholders. The Scrum Master will help everyone understand how to apply Scrum to their day-to-day work and help handle impediments. At the end of the sprint, the sprint review and sprint retrospective meetings take place as planned.

## Sprint Review

The sprint review is held at the end of the sprint, and the Scrum Team, together with the stakeholders, inspect the outcome. This is another important inspect and adapt meeting. The sprint review is time boxed to four hours for a one-month sprint. And for shorter sprints, the event is usually shorter. The Scrum Guide refrains from giving too many guidelines regarding how to structure and conduct this meeting, but here is what can happen. Typically, the product owner is the one who will know which stakeholders to advise to this event. Only the key stakeholders or the relevant stakeholders will attend. The term key stakeholder refers to the most important stakeholders that the product has. The product has many stakeholders, and not all of them may be interested in the outcome of a particular sprint, so they don't need to attend.

In other words, not every change is relevant for everyone, and attending just for the sake of having another meeting makes little sense. Also taking part are the developers and the Scrum Master. The Scrum Master's role is to facilitate this meeting, ensure everyone understands the purpose, and keep it within the time box. The entire Scrum Team must be present. For example, the product owner cannot delegate their participation to someone else. This is a very exciting event for the Scrum Team as they get to show the results of the work.

The outcome of the sprint is essentially what a Scrum Team has accomplished in the sprint. This can be the product increment, reaching the sprint call, or any other results the sprint has produced. It is not only about looking at the bright side. Talking about what didn't work is just as important. The outcome of the sprint should not be a surprise for the product owner unless they failed to collaborate with the rest of the team during the sprint. The Scrum Team will demonstrate the product increment to the stakeholders to gather feedback.

This is the highlight of the meeting. Only work completed according to the definition of done is part of the increment and is demonstrated. It really makes little sense to present incomplete work as this is not releasable. If the Scrum Team has created multiple increments, only the last one is inspected. Remember that an increment is a new version of the product and is additive to all previous increments.

So the last increment will always contain all the features the previous increments had. The demonstration of the increment is done to obtain feedback and encourage collaboration on what needs to be done next. The Scrum Team should avoid limiting this event to showing a presentation. It is particularly valuable to have the stakeholders try out a product with their own hands to understand better what has been built.

It is essential to remember that the sprint review is not a release kit. The Scrum Team does not need a stamp of approval from the stakeholders to release the increment. They can do this anytime during the sprint. If the stakeholders notice something important that needs to be fixed, the Scrum Team can do that in the next sprint and immediately release the new increment.

Apart from inspecting the increment, the participants will try to understand what the Scrum Team should do next. The entire group will look at how the progress toward the product goal is advancing as each sprint should bring them closer to reaching that goal. This meeting is a good opportunity for the stakeholders to ask questions or just changes or new features to the product, or they'll also try to understand what has changed in their environment since the last sprint. The term environment is very generic, but it essentially means anything that can impact the product.

This can be marketplace changes, like low customer demand, new opportunities, the competition releasing a new product, a new regulation that impacts the product, pandemics, anything that is in the environment where the product operates. Often, this meeting can result in a revised product backlog based on feedback or environment changes. This is why the key stakeholders need to be present to inspect the progress made and provide feedback. This is their chance to do this and they need to take the time to be present.

If the stakeholders skip too many sprint reviews, they get disconnected from the development work and may end up with a product that does not meet their expectations. This is why Scrum makes this meeting mandatory. We don't want this disconnect between the Scrum Team and the stakeholders to happen. By being present, they provide feedback and close the feedback loop with each sprint. So what happens with product backlog items that have not been done yet or that are not fully done? For example, some functionality has been built, but more is needed. Maybe testing or documentation is not completed yet.

First of all, unfinished work will not be demonstrated during this meeting and should not be part of the product increment. Any incomplete work will be put back in the product backlog. The product owner will decide what to do next.

Let's recap. The sprint review is another mandatory inspect and adapt meeting in Scrum. It is time boxed to four hours for a one-month sprint. The entire Scrum Team and the key stakeholders attend, and most importantly, collaborate. The sprint review is not just the presentation. The purpose of the meeting is to inspect the outcome of the sprint and to adapt the product backlog as needed.

In a nutshell, this is the sprint review meeting.

## Sprint Retrospective

The Sprint Retrospective is the very last event in the Sprint, right after the Sprint Review. Informally, it's meeting is called the "Retro". While in the Sprint Review meeting, the focus was on inspecting and adapting the product, the goal of the Sprint Retrospective is to inspect and adapt the development process in a way that increases quality and effectiveness, so, make sure you don't confuse the two meetings.

Only teams that reflect on what has happened and identify what can be improved in the future can become more effective over time. The Sprint Retrospective is an internal Scrum Team event where no external parties are involved. Having stakeholders or management present would most likely inhibit a team from openly discussing the problems they see, and would reduce the effectiveness of the meeting.

Any discussion with external parties regarding possible improvements should be done outside of this event. The Scrum Master acts as a facilitator for this meeting, ensuring the meeting is positive and productive. As with any Scrum Event, the Sprint Retrospective has a time box that needs to be respected. For a one month Sprint, the maximum duration is 3 hours. For a shorter Sprint, the event should be shorter. This event is not just for the developers. The Scrum Master and the Product Owner must be present and participate as team members.

The Scrum Guide does not make any explicit suggestions on how this meeting should be structured and conducted. It just points out it's purpose and the desired outcome. The purpose is to look into how the last Sprint went in regards to individuals, interactions, processes, tools, and the Definition of Done. The Sprint Retrospective is a positive space. It is not about finger pointing, blaming others, and expressing frustration. The Scrum Master helps the Scrum Team get value from having this meeting.

Usually, as a first step, the team will gather reflections on what went well, and what can be improved. If something went wrong during the Sprint, it is important to identify the root cause, and, trust me, there is always something to discuss. Why did the team fail to reach the Sprint Goal? How some impediments effected the forecast, and so on.

For this reason, this event is mandatory. No matter how mature a Scrum Team is, there is always a need for the retrospective. The Scrum Team will discuss the findings, attempt to understand different perspectives, identify the parts that went well, and potential improvements.

It is just as important to understand what went well, so that the team continues doing that. Once the most significant improvements are identified, it is time to create a plan for implementing those improvements as soon as possible. The Scrum Team can add these improvements to the Product Backlog, but ideally, it will add one or more improvements directly to the Sprint Backlog for the upcoming Sprint.

If improvements are only identified but not implemented, nothing changes. This is why it is crucial to take action after this meeting. To improve the quality of the work they deliver, the Scrum Team may want to plan changes to the Definition of Done during the retrospective.

The Scrum Team can change the Definition of Done to include new conventions, standards, or guidelines, as seen appropriately, or make the existing ones more strict. Since this is the end of the Sprint, it is the best time to make changes to the Definition of Done. Modifying the Definition of Done after a new Sprint has started directly impacts the forecast, and the work already completed, so, it is not a good idea to do so.

The Definition of Done is not something that needs to be created or updated with every Sprint, so this will not happen very often. While issues and potential improvements can be identified by the team any time during the Sprint, the Sprint Retrospective is a dedicated space to inspection and adaptation.

In a nutshell, this is the Sprint Retrospective.

## Special Sprints

Instructor: Any sprinting Scrum should have a clear sprint goal, reach that goal and produce at least one releasable increment. When starting to work on a new product, some Scrum teams feel the need for a special sprint, called a kickoff sprint, or sprint zero to help them get started, get the requirements, create the product backlog, work on the architecture and so on.

Later on, some teams may decide to introduce another special sprint called release sprint or hardening sprint to test the increment and fix any defects before a release. Special sprints are not allowed in Scrum. Let me explain why not. As you've noticed, the Scrum guide does not say, "You are not allowed to have special sprints."

There's nothing like that in there. They are still not allowed because they break some fundamental Scrum rules. Each sprint must create a releasable increment. That increment must create value, typically by building a feature, no matter how small. A sprint zero where you do planning and preparation work does not create an increment with a feature that somebody can use, therefore, breaking a Scrum rule. Having a release sprint indicates that the previous sprint did not produce increments that follow the definition of done and are releasable, and this again, breaks another Scrum rule. The purpose of a sprint is to create something useful that the stakeholders can inspect. If you just get the requirements, create plans and documentation, work on some technical aspects, what can you show the stakeholders at the end of the sprint?

What can you release to the end users to get feedback? Well, not much probably. In other words, without having at least the chance to release something, you are disconnected from the users and you don't use the sprint as a feedback loop and this is a big issue. Any sprint is a way to validate an idea and get feedback. This is why any special sprints are an antipattern in Scrum. They don't help you do that. Even if this is the first sprint and it's not easy to create an increment, the Scrum team should put just enough effort into everything that is needed to create the first version of the product. It won't be perfect and it won't contain much functionality, but it is so important to have something reasonable. So, to recap, in Scrum, there is no sprint zero, release sprint, hardening sprint, integration sprint, sprints dedicated solely to planning or working on the architecture and infrastructure of the product, or any kind of special sprints devoted to anything else but reaching a clear and specific sprint goal and producing a valuable increment.

## Cancelling a Sprint

Instructor: Canceling a sprint before the time box expires is a very, very rare occurrence, but you need to be aware of this possibility as well. A sprint could be canceled if the sprint goal becomes obsolete.

This situation can be caused by some major and sudden changes in the market, if the company decides to change direction, or for any other reasons. But again, because sprints are usually very short, this is an extraordinary event and very uncommon.

Nevertheless, only the product owner has the authority to cancel the sprint. They may do so if advised by the stakeholders, the developers, the Scrum Master, but only the product owner can take this decision. The Scrum guide does not get into details about this, but if the sprint is canceled, any completed product backlog items will be reviewed during the sprint review meeting. Incomplete product backlog items will be re-estimated, and put back on the product backlog.

The Scrum team will also conclude the sprint with the sprint retrospective and start planning the next sprint. If the Scrum team thinks they can no longer reach the sprint goal, this is not a reason to cancel the sprint, since the goal is still valid. Nothing has changed, so the Scrum team should always try to reach the sprint goal and inspect their progress at the end of the sprint. So, a sprint can be canceled if the sprint goal itself becomes obsolete. There are no other situations mentioned by the Scrum guide where it would make sense to cancel the sprint.

# Section 4

## The Scrum Team

Instructor: Let's go back to our Scrum team and better understand who they are and what they actually do. A fundamental characteristic of Scrum is having a small group of professionals, which we call the Scrum team.

The Scrum team consists of a product owner, a Scrum master, and developers. The Scrum Guide does not use the term role on purpose, trying to make clear that the product owner, the Scrum master, and the developers are not job titles, even if many organizations use them as such.

The Scrum Guide calls them accountabilities. They are accountable for their work from the perspective of the Scrum process. What is important to remember is that Scrum teams are self-managing and cross-functional. Self-managing teams decide internally how to do their work, rather than being directed by others outside the team. It means that they determine how the product should be built, but also other aspects of their work.

Cross-functional teams have all the skills needed to accomplish their work without depending on others outside of the team, especially when it comes to the technical expertise required to build the product.

A Scrum team does not have sub-teams or hierarchies. For example, the product owner and the Scrum master are not managing the developers. All Scrum team members are equal. In the previous versions of the Scrum Guide, the developers formed a sub-team called the development team. This is no longer the case. The developers no longer form a sub-team. By being self-managing and cross-functional, Scrum teams often have more flexibility, demonstrate creative problem solving abilities, and achieve higher productivity. Scrum teams strive to deliver products iteratively and incrementally, and to maximize the opportunities for feedback. The Scrum team exists to build a product, generate value with each sprint, and to reach the product goal. A person can have different accountabilities in Scrum. The product owner or the Scrum master can also be developers if they're doing the work of the sprint backlog. While this is not recommended, it is definitely possible. Remember that a product owner, the Scrum master, or the developers are Scrum accountabilities. They are not job titles. Please note that the Scrum Guide always uses term developers in the plural, never in a singular form. The accountability is called developers, not developer.

## Scrum Team Size

Instructor: How big should a Scrum team be? The optimal Scrum team size is small enough to remain flexible and large enough to complete significant work within a sprint. The Scrum guide recommends a Scrum team size no larger than 10 people. This typically translates to having one product owner, one scrum master, and eight developers. The Scrum guide only makes a recommendation. This is not a hard rule.

Are just three or four members okay? This is possible, but it is not recommended. A very small team may encounter skill constraints during the sprint, causing it to be unable to build a product increment and reach the sprint goal. This can easily happen when somebody is on holiday, or absent for other reasons. If you have a team size of about seven members, so five developers, product owner, and a scrum master, you'll probably reach a sweet spot where communication thrives and work gets done fast. Having too many members is also not a good idea. Having more than 10 members requires too much coordination between them, making the entire process too complex.

You can have a Scrum team with two people or with 20 people. This is not breaking any Scrum rules. When a Scrum team becomes too large, it is often a good approach to let the people split into smaller teams. When multiple Scrum teams work on the same product, we refer to this as Scaled Scrum. For the exam, remember that the Scrum guide only makes a recommendation regarding the size of the Scrum team,

which should be 10 or fewer people.

## The Product Owner

Instructor: Next let's look at each accountability within the Scrum Team more closely. And we will start with the Product Owner. The Product Owner's main accountability is

to maximize the value of the product, resulting from the work of the Scrum Team.

This is one of the most important things you need to remember about the Product Owner. The main concern is always the value that is being generated. The Product Owner is one person, not a group of people. The Product Owner may represent the desires of others in the Product Backlog. Still, those who're wanting to change the Product Backlog, must take it through the Product Owner. This is a key aspect when it comes to this accountability. The fact that the Product Owner is a single person is not left to chance, but a decision made on purpose. And it all relates to accountability.

The Product Owner is, in the end, the sole person accountable for the outcome, so nobody should go over them when it comes to decisions, such as, the product backlog ordering, what to build, what not to build, and so on. For the Product Owner to succeed, the entire organization must respect their decisions.

For some organizations adapting scrum, this is a hard pill to swallow. But this is an essential part of scrum that needs to be understood, and respected. No one can force the Developers to work from a different set of requirements, not a Scrum Master, not the Stakeholders, not even the CEO. Nobody.

The Product Owner decides which features should be built, and in which order. Scrum is all about transparency. The Product Owner's decisions are visible in the content, and ordering of the Product Backlog. This makes it clear what the Scrum Team will be doing next. So the Product Owner is the sole person accountable for managing the Product Backlog. The Product Owner must define and communicate the Product Goal.

This gives a clear direction to everyone involved. The Product Owner also needs to make sure the Product Backlog items are clearly expressed and understood by the Developers. And here comes another key aspect to be remembered, The Product Owner is the decision maker. They decide what is built, or not. The Product Owner is maximizing the value by managing the Product Backlog.

The Product Backlog is the primary tool that a Product Owner has at their disposal. The Product Owner can also delegate the responsibility of managing the Product Backlog, however, they remain accountable. This means that the Product Owner can delegate some tasks to others, but they remain accountable for the outcome. Quite commonly, the Developers are allowed to add items to the Product Backlog, and discuss them during the next refinement meeting. The Product Owner can also ask other people within the organization to write some Product Backlog items, depending on the domain.

This delegation can happen only at the Product Owner's discretion. Nobody can force a Product Owner to do that. The most important rule to remember is this: for one product, there is only one Product Backlog, and one Product Owner. So this is the Product Owner accountability, as described in the Scrum Guide. And, just to be clear, there is no Project Manager role in Scrum. And in general, the Product Owner is by no means to be confused with the Project Manager. The Product Owner is not managing the Developers. Nobody's reporting, or giving status updates to the Product Owner. Everyone collaborates to get work done, reach the Sprint Goal, and ultimately the Product Goal.

## What is VALUE?

Instructor: The product owner's main accountability is to maximize the value of the product resulting from the work of the Scrum team. The product owner should be obsessed about value. So if this value is that important, what does it actually mean?

The Scrum guy is again, very vague about this aspect. Value comes in many different forms, so probably this is why the Scrum guy does not go into more details. Let me give you some examples. For a regular company looking to obtain profit, value can be revenue or profit. This is the most straightforward example of value.

The more money you make, the better it is. But it is not only this.

We need to take a more holistic view. Value can also be the current market share, customer satisfaction, employee satisfaction and many other aspects. It is rarely just a single aspect. If you are a nonprofit organization fighting to end world hunger, your concept of value is probably different. Your goal is not to make as much money as possible.

Your mission is to feed people, save lives, make the world a better place. This is why the Scrum guy does not get into value.

What value is needs to be defined by the organization and by the Scrum team as this largely depends on the context where Scrum is applied. But make sure you remember this. The main accountability of the product owner is to maximize the value, whatever that is. How the product owner does this, again depends on the context. Value does not need to be a concept foreign to the Scrum team.

This is something that should be a constant topic of discussion. If one day the product owner wishes to turn all buttons on the online shop from blue to green, the rest of the Scrum team must understand the value in doing this.

Will customers complain less? Will this lead to an increase in sales? Or are we just doing things for the sake of doing? The opposite of generating value is generating waste and we don't want that.

## How is the Product Backlog ordered?

Instructor: Let's talk for a minute about how the product backlog is ordered. The product owner is accountable for maximizing the value of the product.

So how is it done?

Mainly in two ways. First, by making sure that the work in the product backlog is clear and valuable, and secondly, by ordering that work. Ordering the product backlog is the primary tool that product owners have at their disposal. So we can say that the product owner orders the product backlog items to maximize value. Pay attention to the vocabulary used here. This is not the same as saying that the product owner orders the items by value starting with the most valuable items at the top.

Such an approach does not work in the real world. If this was the only rule, ordering a backlog would be very easy. Don't get me wrong, the top of the backlog will contain valuable items, but they are not always ordered by value. This is why we talk about maximizing the value. The product owner needs to squeeze as much value as possible from the work that the Scrum Team performs. Allow me to make an analogy so that you can better understand why this is not trivial.

Let's say you are renovating your house. You already have all the skills needed to do the work, and have made an improvement list currently in random order. You want an outside pool so that you can enjoy some time with your family during the summertime. You want a sauna, you want to fix some leaks in the roof, you want new plumbing throughout the house, a new kitchen with countertops, appliances, and so on. You have limited resources because you only have three weeks of vacation which you want to dedicate to improving the house.

So, how to proceed?

Let's try first to assign a size to the work in the list. Let's say that one hammer is the lowest effort, and then five hammers represent the highest effort. The outside pool will be five hammers. There's a lot of work that goes into that. The sauna will be only two hammers. Fixing the leaks in the roof, it's quite an easy job.

Only one hammer. New plumbing throughout the house, that's also something that's requiring three hammers. And also the kitchen just as well, three hammers. Now let's try to add the value that you expect to get from each of the improvements.

Let's say that one smiley face is the lowest value, and five smileys are the highest value you will get. So the outside pool will guarantee you four smiley faces. You'll be able to spend more time with your family and kids. Sauna will only give you two smileys. Fixing the leaks in the roof will give you five smileys, because that's something very important to protect your house, family, and investments.

New plumbing, two smileys. And for the kitchen, you will get three smileys. Now bear in mind that this is a very simple example. The idea I want you to get from this is that ordering a product backlog is not an easy task. Its many factors need to be considered.

We typically consider the size of the items and their value. For example, fixing the roof is easily done and we get a lot of value since it is not raining in the house and we don't get any water damage. So we could put this item first on the list. Now, what could we do next that is relatively low effort and high value? How about redoing the kitchen? Wait a minute, before you jump into it, let's think about it. Does redoing the kitchen have anything to do with the other items on the list? Actually, it does. You see, the kitchen also has plumbing. So it makes little sense to redo the kitchen before we have redone the plumbing. What we have here is a dependency.

Even if we see more value in the new kitchen and the effort is relatively low, we cannot do this item before doing the plumbing first. So dependencies between the items in the backlog play a role in ordering the backlog. We also don't want to create unnecessary dependencies on specific products or vendors. If a vendor cannot deliver the pool that we wanted, we search for another one.

We want an outside pool during summertime, not in winter. So if we have the pool ready in winter, I doubt we will get as much value as in summer. As you can see, ordering the backlog is getting more complicated. And this is exactly how it is in real life. And this is why we need a person to do that, a single person, the product owner. The product owner needs to take into account many factors. How would you order the list of improvements? Post a comment in the discussion below the video. I would love to hear your thoughts. And I have a question for you.

Does it matter who is doing the work?

Which developers or skills are available in the next sprint? In practical terms, you may say yes. However, from a Scrum perspective, things are a bit different. We shouldn't care which developer is available in a sprint, as we don't want to create hero developers or hero programmers. Essentially, having only someone doing the most complex work all the time does not help the team grow and is often seen as an antipattern. I have linked in the resources folder some materials around the hero antipattern.

This is probably one of the most widespread behaviors I have observed, and you should be aware of this as well. So to recap, we want to order the product backlog based on the value of the work, its size, the dependencies between the items, and whatever else the product owner deems as relevant. However, we don't want to order the product backlog based on who's available or not during the sprint or external products or vendors.

## The Developers

Instructor: Now, let's take a closer look at the developers. The developers are the specialists in the Scrum Team who have all the skills required to build the product. A common misconception is that the term developer refers to programming.

This is not the case. A developer is a person doing development work on a product. Any work required to build a product is considered development work. From a Scrum perspective, a person doing testing is also a developer.

Someone writing documentation? A developer as well. The developers aim to reach the Sprint goal and create a potentially releasable increment by the end of each Sprint. They do so by adapting the Sprint Backlog and monitoring the progress of work during the Sprint.

The developers don't work in isolation from the rest of the Scrum Team. It is expected that the entire Scrum Team will collaborate throughout the Sprint. It is worth mentioning that only the developers can work on the product increment. Nobody else external to the Scrum Team can do that. Since the developers are part of the Scrum Team, they are empowered by the organization to organize and manage their own work. Here are the most important aspects you need to know about them.

They are self-managing. Nobody, not even the Scrum Master, the product owner, the stakeholders, the CEO, tells them how to do their work, which tools and methods to use, or anything similar. Emphasis here is on self-managing. They decide how to turn the product backlog into potentially shippable increments. Self-management applies to other aspects of their day-to-day work when it comes to problem-solving or decision-making. They are cross-functional.

This means that between them, they have all the skills needed to create a product increment. This does not mean that every developer must be cross-functional. While a developer often has a clearly defined area of expertise, the accountability for their work belongs to all of them. All developers are accountable for the Sprint outcome. At Jack and Mary, Aki, Michelle, Tom, and Max bring a mixed set of design, programming, and testing skills, all crucial to building the eCommerce site.

While Michelle focuses on the design work, Aki and Max on programming, and Tom on testing, they don't work in silos. There's always an overlap between their work and they work together toward building the increment and reaching the Sprint goal.

Here's a simple example.

Let's say that Tom, who typically handles the testing work, is absent. That is not an excuse for not meeting the Sprint goal and failing to create an increment. In this case, the remaining developers are responsible for finding a solution to test their work. They also hold each other accountable for doing the work at a level of quality expected. The skills that the developer need depend on the domain of work. If you are developing a software product, you may need people who know programming, database design and maintenance, testing, operations, and so on.

Scrum does not recognize any official titles for the developers, regardless of the work they perform, such as programming, design, testing, documentation, marketing, or anything else. There are no hierarchies among the developers, no senior and junior developers, no team leads, testers, or anything similar.

All developers are equal. They all decide together. The developers are collaborating between them. They never pass work from one person to the other. They take ownership of the entire development process. The developers are committed to quality and always ensure that the definition of Done is respected. The developers should try to build the best product with the skills and tools that they have.

Conditions are not always ideal, but they should strive to improve their skills, tools, and quality over time.

The Scrum Master

Instructor: The Scrum master serves the developers and the product owner and is responsible for promoting and supporting Scrum within the organization. You can see the Scrum master as a Scrum coach. The Scrum master is doing their job by helping everyone understand the Scrum theory, practices, roles, and values.

The Scrum master is accountable for how Scrum is understood and lived both by the Scrum team and the organization. From a Scrum perspective, the Scrum master is considered a management position. The Scrum master is managing the Scrum implementation, but don't confuse that with a traditional management position. The Scrum master is also accountable for how effective the Scrum team works, and should help them identify practices that improve this aspect. The Scrum master helps those outside the Scrum team, for example, the stakeholders, understand the empirical nature of the work the Scrum team does, and how Scrum works.

For example, stakeholders providing business insights or support in removing organizational impediments is useful. Less useful is setting unrealistic deadlines, putting pressure on the team, or demanding overtime. When it comes to the Scrum team itself, the Scrum master is a leader who serves. The Scrum master puts the team before themselves, and in case you're not familiar with the term, let me explain a bit.

Traditional leadership generally involves power flowing from a leader towards lower levels of authority. By comparison, servant leadership turns the power pyramid upside-down, putting the needs of others first and helping people perform as highly as possible. So the Scrum master is there to serve others.

For more on this topic, I will share a few materials in the resources section. In case the term master creates any confusion, please remember that the Scrum master has no formal authority, like a manager or anything similar. The Scrum master is the master of the Scrum process. The Scrum master will not manage the Scrum team. They are not responsible for deciding who should join or leave the team. They don't have this authority. They are also not making any decisions for the Scrum team or controlling the developers in how they do their work. The main point here is that the Scrum master is a facilitator and a coach, and the goal is to serve the Scrum team in order to maximize the value created.

How is the Scrum Master serving the Scrum Team

Instructor: So how is the Scrum Master serving the Scrum Team? They do that in several ways. The Scrum Master has to defend the empowerment that the Scrum Team has and to coach it regarding self-organization and cross-functionality. The Scrum Master can also have the challenging task of coaching the Scrum Team in organizational environments in which Scrum is not yet fully adopted and understood.

One important focus is to help the Scrum Team create high-value products and to be effective at doing this. The Scrum Master is successful if the Scrum Team is successful. Quite often, the Scrum Team is facing impediments that endanger the sprint outcome. The Scrum Master is there to help resolve those impediments, regardless of their nature. They don't need to remove the impediments themselves.

They just need to ensure that the impediments are resolved, regardless of who actually resolves the impediment. The Scrum Master will not remove people from the Scrum Team since they don't have this authority. They need to help the Scrum Team work as a self-managing team and find a solution to any internal conflicts that may arise. The Scrum Master also deals with impediments at an organizational level by handling organizational impediments which are often a dysfunction of the organization.

The Scrum Master can also cause changes that increase the productivity of the Scrum Team. For example, by improving internal processes that affect the team and which cause delays, the Scrum Master is also facilitating Scrum events and other meetings, and I am emphasizing the word facilitating. Don't misunderstand that. Facilitating meetings means setting them up, booking and arranging rooms, sending invitations, starting the video call or anything else. The Scrum Master is not the secretary of the team. Facilitating simply means helping something run more smoothly and effectively. The role of the Scrum Master is to ensure that these meetings are useful, positive, and that everyone understands their purpose. The Scrum Master needs to ensure that all Scrum events happen and are kept within the time box.

It is not allowed to skip or postpone any Scrum events. The Scrum Master can't bend the Scrum rules. The Scrum Master should not use any of the prescribed Scrum events for training purposes. They can organize a Scrum training as a separate event. Apart from the dedicated events, the Scrum Master constantly reminds the Scrum Team about the Scrum rules, values, and the importance of empiricism.

What is an impediment in Scrum?

Instructor: We know that the scrum master needs to take care of impediments. But what is an impediment? Impediments are an everyday occurrence during the development process. Most of the time, an impediment is anything that stands in the way of reaching the sprint goal. But an impediment can also be something that prevents the team from delivering a high quality product, or from improving their performance.

From a scrum perspective, not every problem that the scrum team encounters is automatically an impediment. We consider an impediment something that a scrum team cannot solve on its own. Remember, the team is self-managing, and should try to solve most problems on its own. When the team cannot resolve the issue and the impact is significant, it becomes an impediment.

Let's take a look at a few examples. The coffee machine broke and the developers cannot work anymore.I also cannot work without a strong coffee in the morning.

Is this an impediment?

No, of course not.

Scrum master is not there to fix the coffee machine or to make sure the developers have coffee.

How about this one?

The scrum team uses a video conferencing software that crashes a lot and negatively impacts their meetings.

Is this an impediment?

It could be. If the scrum team has the possibility to stop using the software and to switch to something else, this is not an impediment, as they can solve the problem internally. If the IT department mandates everyone to use the software for security reasons or for other internal reasons, this can be an impediment as the Scrum team has no say in this. It is not something that totally prevents them from completing the sprint, but it is annoying and it is impacting their productivity.

In this case, the scrum master should consider this an impediment and act accordingly. Maybe have a chat with IT department, and make them aware of this issue. Make transparent to the managers how much time is being wasted with the software. Here's a more serious situation. The scrum team is working at integrating credit card payments in the checkout process. During the development work, they notice that they don't know how to configure some parameters for an external component that they need to use.

They ask the payment processor for support but didn't get an answer even after waiting for two days. The developers decided to notify the scrum master. The scrum master stepped in and escalated this issue with a senior manager who directly contacted the payment processor. The problem got resolved in just a few hours. It is not possible to resolve absolutely every impediment that occurs. Some impediments are more critical or more time sensitive than others.

The scrum master should cause the removal of the most important impediments that hinder the scrum team. To know which impediment is more important, the scrum master will collaborate with the rest of the scrum team to understand their impact. While the rest of the scrum team could also talk with people outside the team and handle such issues, this activity can be time consuming, and can also cause them to lose focus. Their primary focus should be the sprint and reaching the sprint goal.

This is why the scrum master is the best person to handle and cause the removal of impediments.

## How is the Scrum Master serving the Product Owner

Instructor: So how is the Scrum Master serving the Product Owner? The Scrum Master coaches the Product Owner to understand and practice agility and Scrum. The Product Owner needs to ensure that everyone on the Scrum team understands goals, scope, and product domain.

This is extremely important because if any of these are missing or unclear, the Scrum team will encounter issues while building the product. It is also essential for the Product Owner to understand the need for clear and concise product backlog items.

The Scrum Master is there to help the Product Owner find techniques to manage the product backlog effectively. The Scrum Master will mentor the Product Owner on ensuring the product backlog is ordered. And in a healthy state, nobody wants a cluttered product backlog. The most valuable work needs to be toward the top of the product backlog.

Items that don't provide much value can be removed. By looking at a product backlog, it should be clear what the Scrum team is doing next. The Scrum Master needs to make sure that the Product Owner understands how to do product planning in an empirical environment. In Scrum, we don't make a plan and never change it. Quite the opposite. The plan needs to be flexible and respond to many relevant factors and new insights that emerge.

The development direction is guided by past experience and new insights gained from experiments. Empiricism means that we constantly learn and improve our plan. Nothing is set in stone. The plan for the product is the product backlog. The Product Owner needs to identify the features that the product needs to reach the product goal and achieve the highest value possible.

A Scrum Master can also facilitate planning, product backlog refinement events, and collaboration with the stakeholders as needed or requested by the Product Owner.

## How is the Scrum Master serving the Organization

Instructor: The Scrum Master is not only serving the Scrum Team, but also the organization. The Scrum Master plans new Scrum implementations in the organization, explains why change is necessary, and leads the organization in that direction.

The Scrum Master can collaboratively work with other Scrum Masters within the organization to achieve these goals. The role of the Scrum Master is to lead and coach the organization in its Scrum adoption. This is a process, and it does not happen overnight. Agility cannot be learned from a book or by attending a course.

You learn it by doing it. This is why the Scrum Master plays such an important role. If the organization has been developing products using the traditional waterfall approach for a long time, understanding and enacting an empirical approach for complex work will not be easy. The Scrum Master is there to guide them through this process. By doing so, the Scrum Master brings the stakeholders and the Scrum Team together. If the organization is adopting Scrum, the Scrum Master should expect resistance and skepticism. The old way of doing business is not easy to change. There is pressure to adapt the Scrum framework to match the existing roles, structures, and procedures. The thinking is that less change is less disruptive to day-to-day operations. While this approach could make the management more open to adopting Scrum and be less resistant to change, some benefits will be lost.

The Scrum Master should not accept any compromises. Adapting Scrum results in something that can no longer be called Scrum, and your organization will most likely go back to old habits.

## The Scrum Master in real-life

Instructor: After going through the Scrum accountabilities, some of you may look at your existing organization and wonder, "Who's the business analyst, product owner, or the Scrum Master? How about project manager? Where is the tester and the other job titles?" Let's take them one by one. First of all, the Scrum accountabilities are not job titles or positions in an organization, even if many companies tend to use them as such.

Suppose you are a developer in a Scrum Team, but you do marketing, which is considered development work in Scrum. In that case, your business card will say something, like marketing specialist, marketing manager, or marketing coordinator, and not developer.

Scrum is adopted by organizations that already have a CEO, managers, HR, project managers, business analysts, hierarchies, and so on. They don't all disappear. Scrum happens within the existing structures. All involved need to understand Scrum and figure out their role, regardless of their official title. They also need to allow for agility to happen. Over time, job titles can change, but the essential difference is the change in thinking focusing on delivering value.

A business analyst is typically responsible for bridging the gap between development and the business side by handling requirements, asking the right questions, and carrying over that know-how into the development process. Of course, this is a very simplistic explanation.

Scrum does not have a business analyst role or accountability, but your organization may have one. A business analyst can take the product owner accountability, but just as well as Scrum Master accountability. There is no one-to-one match. The product owner is not a rebranding of the business analyst or the project manager.

A totally new person can take the product owner accountability and collaborate with the business analyst, maybe even delegate some product backlog management tasks. This is possible. Just because the Scrum Guide does not mention other roles, it does not mean that other roles cease to exist. The business analyst can continue doing their job, but in an Agile way. There are no more detailed requirement documents, months or even years of detailed upfront planning, passing requirements to developers.

What you need to get from this is that there is no list where you can look up existing roles and match them to a Scrum accountability or a representation that shows you the delta between a business analyst and a product owner. It really depends on the situation. It is absolutely normal to be unsure about which Scrum accountability suits you best.

Here's a quick overview. Regardless of your current job title, the product or accountability can suit you well if you have an entrepreneurial thinking, care about solving a problem, and having happy customers. The Scrum Master accountability can suit you well if you are a people person, have patience and excellent communication skills, put the needs of others before yours, like to help others grow, and often find yourself identifying problems and wanting to fix them.

If you like building something with your own hands, are fascinated by the technical aspects of the product, and don't like to sit all day in meetings, being a developer is probably a good choice. You may also find yourself between two accountabilities, and this is fine. These accountabilities try to balance different aspects of the work required to build the product, but there is no rule prohibiting you from being the product owner, also developer at the same time.

It's not always a good idea, but this does not break any Scrum rules. Over time, I've seen developers transitioning into a Scrum Master role, Scrum Masters becoming product owners, and so on. A question that I get a lot is regarding technical skills.

Does the Scrum Master or the product owner need to have technical know-how?

Maybe. It often helps, but this is not an explicit requirement. I know that many product owners are actively learning about the technical aspects of the product so that they can develop a better understanding. The more you know, the better it is. The focus should always be on doing your job well.

In the end, Scrum is just a framework trying to guide you. As long as you don't break any Scrum rules, you, your team, and your organization are in the best position to make a decision. Every Scrum accountability looks different depending on the organization, the product, and also other factors. The discussion is much longer. And if you're interested, I will add some additional content in the Resources section. Still, I hope this quick overview was helpful.

# Section 6: Scaling Scrum

## Introduction to Scaling Scrum

Instructor: When more than one Scrum Team works on the same product, we refer to the situation as Scaling Scrum. The intention is to turn the product backlog into increments at a much faster rate, so more output and hopefully more value. The Scrum Guide does not go into many details when it comes to what happens when more than one Scrum Team works on the same product, but some rules still exist.

The same rules that apply for a single team also continue to apply when you are scaling. Scaling Scrum is a very trendy topic, but it also poses many problems. Various scaling frameworks can be used with Scrum and provide more guidance on how to scale the development effort. Here are some of the most popular frameworks for Scaling Scrum. We have the Nexus Framework from scrum.org created by Ken Schwaber.

Scrum@Scale from scrum.inc created by Jeff Sutherland. A LeSS Framework, Large-Scale Scrum, from less.works created by Bas Vodde and Craig Larman. Another popular framework is SAFe from Scaled Agile, but SAFe is not scaled Scrum. It is an Agile scaling framework. Scrum is under the Agile umbrella, but don't confuse Scrum with Agile as they are not the same thing. You'll find many people suggesting you study the Nexus Framework by reading The Nexus Guide in your preparation for the exam.

You'll find a link to The Nexus Guide attached to this lecture, but I want you to remember that The Nexus Guide's content is not part of the scrum.org PSM I exam.

For the PSM I exam, The Nexus Guide does not come into play. Scaled approaches using a particular framework are not considered part of the exam. scrum.org has a separate exam dealing with Scaling Scrum called Scaled Professional Scrum, SPS. This is why I don't recommend reading The Nexus Guide at this point in your preparation. In this section, we'll go over the most important aspects that are relevant for the exam. I will explain all the rules you need to know without getting into a specific framework.

## Scaling Scrum recap

Instructor: This is the end of the Scaling Scrum section. I know that by now, you are not 100% sure how scaling Scrum is supposed to work in practice, and that you still have open questions. Since we are not discussing a particular scaling framework, it is also hard to make suggestions on conducting the events and organizing the sprint.

For the moment, I would worry less about this. The most important rules you need to remember are the following. For one product, there is a single product backlog, and a single product owner, which must be a single person. The Scrum teams must define a shared definition of done, and must produce integrated product increment. For the moment, it is more important to remember these rules and continue learning about Scrum without worrying too much about how to scale.

Let's first understand how Scrum is supposed to work with the single team, and worry about scaling later. Most organizations increase the complexity by focusing on scaling Scrum instead of doing Scrum as it is supposed to be done.

## What happens when multiple Scrum Teams work on the same Product?

Instructor: Jack and Mary's Organic Farm wants to speed up the development of new features and has hired more developers. Since the existing team became too large, the group discussed this and, as a self-managing team, they decided it is time to form another Scrum Team. It is best to let people self-organize into teams instead of having someone decide for them.

Read more about how developers can organize themselves into teams by visiting the Resources folder. The most important rule you need to remember when multiple teams work on the same product is the following.

For one product, there is a single product backlog, which is managed by a single person, the product owner. They decided to call the teams Team A and Team B. Because of this rule, both teams will share the same product owner and the same product backlog. So both teams must have the same product owner, which is Jessica. There are no exceptions to this rule, and if you remember it during the exam, you'll easily handle 80% of the questions involving multiple Scrum Teams. The used Scrum Team will also need a Scrum Master. Ali has agreed to be part of both teams. The Scrum Master accountability has no restrictions when having multiple teams. The Scrum Master can be one person who handles both teams or two different persons. It does not matter.

Let's revisit the rule. For one product, there is only one product backlog and only one product owner. It is important to remember that there is only one product backlog. There will not be separate product backlogs for each team or different product owners. During the sprint planning meeting, the developers will pull items from the product backlog in agreement with the other teams and with the product owner.

Each Scrum Team will have a separate sprint backlog during the sprint. In this section, we'll not get into the details of how each Scrum event is organized when scaling Scrum. It is a more advanced topic that is not relevant for the exam, and this is also the reason why scaling Scrum frameworks exist.

## Impact on velocity when scaling Scrum

Instructor: If a company wants more features in a shorter timeframe, the typical solution is adding more teams to scale the development effort. When multiple Scrum teams start working on the same product, different challenges arise. Say we have one Scrum team with an average velocity of 30 story points, and add a new Scrum team. What will be the velocity of the new Scrum team?

30 story points?

15?

We don't know. We need to start tracking it, and we'll have a better idea after a few sprints. What will happen to the velocity of the original Scrum team? Will it stay the same? Will it increase or decrease? In the short term, the velocity will most likely decrease. Any changes to a Scrum team, if people come or go, result in a decrease in velocity, but also having another team working on the same product can result in a decrease in velocity. The teams need to find ways to collaborate and integrate their work.

This takes a bit of time. The velocity of the first team will most likely come back to normal values after a few sprints, but the initial impact is typically a decrease. You're probably wondering, why is this happening? The team should be independent of each other, right? Well, not quite. The teams will work on the same product and need to collaborate to produce increments.

Dependencies between them slow down the work, increase the communication overhead, and the main concern should be to identify and reduce as much as possible any dependencies between the teams. Any dependencies need to be transparent to everyone involved. The fewer dependencies they have between them, the better the teams can work. When scaling Scrum, the key concern is to reduce as much as possible dependencies between the teams. We can't eliminate the dependencies, since the teams are working on the same product, but we can identify and reduce them.

At the same time, when scaling Scrum, we care less if there's enough work for everybody in the sprint.

The job of the product owner is to maximize the value and to make sure that everyone is busy with work. Scrum is not trying to achieve 100% utilization of the developers. Doing complex work is not working in a factory, producing goods. The results are measured in the value generated. More output is not always more value. This is also related to the velocity of the Scrum team.

Yes, nobody will say that the velocity is not important, but it should not be the primary focus. Delivering valuable increments is what the team should be concentrating on. We should not compare teams based on their velocity. Each team can estimate things differently, so velocity is not a good metric for that. If teams are judged or compared by a specific metric, such as velocity, they will pay attention to that metric, and you'll end up with the numbers you want to see, so be careful what you define as success.

## Integrated Product Increments

Instructor: Throughout the sprint, the different Scrum teams working on a product should collaborate to create an integrated product increment. If you look into the Scrum guide, you will not find this concept of an integrated increment. Still, it is essential to understand it.

Here's how to look at this. Scrum requires that each sprint produces an increment and as you know, an increment is a version of the product. Say there are two teams building a car. In this sprint, one team is making improvements to the engine while the other team changes to the chassis and a body of the car. Can the first team say they have a done increment when their engine is working?

No. The stakeholders need to see the final product, the car. The customers want to drive a car. They don't get value from the individual pieces. You cannot ride an engine, you need the entire car. What if the engine is too large and it doesn't fit anymore? This is an integration issue. You can't have one team building an engine, and the other team a chassis with some wheels, but no engine. None of them is a valid increment.

The increment is a working car that contains both the engine and the improved chassis and anything else that is required. The teams need to collaborate throughout the sprint and make sure their work is integrated. Separate work is not releasable. Nobody goes to a car dealership and walks off with a car that has no engine. Customers expect a fully working car, and Scrum expects that each sprint produces an increment which should be fully working and releaseable.

## The Definition of Done when multiple teams work on the same Product

Narrator: The definition of Done is also impacted when more than one Scrum Team works on the product. When multiple Scrum Teams work on the same product, they must coordinate to create an integrated Product Increment. This means that any work they create needs to be integrated by the end of the Sprint.

At one point in time, there is a single Increment. We will never have one Increment produced by one team, and another Increment produced by another team at the same time. Since there is a single Increment, everybody needs to understand what Done means. So the Scrum Teams must define a common definition of Done that all teams must respect. There is a single product, so there must be a single definition of Done.

The Product Increment is Done only when it is integrated, tested, usable, and potentially releasable. Individual Scrum Teams may choose to apply a more stringent definition of Done within their own teams, but for this to happen, they must have a shared definition of Done first, and any definition of Done must be compatible. Since the Scrum Guide does not go into such details, for the exam, remember that all teams will have the same definition of Done when working on the same product.

## Must the Sprints be aligned? (same length, same start/end)

Instructor: A common misconception when scaling Scrum is that all teams must align, or synchronize their sprints, sharing the same sprint length, start and end date. The Scrum guide does not set any restrictions regarding this aspect, so it would be acceptable for the Scrum teams to have different sprint lengths. One team can have four-week sprint, another team, three-week sprint. The sprints can also have different start dates, end dates, or any combination of those. The most important requirement remains.

Each sprint must produce an integrated and working increment. So if one team is about to end the sprint, they must collaborate with the other teams to ensure that at least one increment has been created. In some situation, this can drive the complexity up. Ideally, work is integrated as often as possible not only at the end of the sprint.

The earlier you get feedback about potential integration issues, the easier it is to solve them. In practical terms, it makes things easier if the sprints are aligned or partially aligned. For example, one Scrum team can have a four-week sprint and another Scrum team, two-week sprint. At least every four weeks, their sprints will end and start at the same time.

For the exam, remember that Scrum does not require that sprints start and end at the same time. There are no integration sprints, or hardening sprints in Scrum, regardless if the sprints are aligned or not. Any integration work is part of the sprint itself.

## How many Product Owners are there?

Instructor: When a single Scrum team works on a product, the following rule applies: For one product, there is one product backlog and one product owner, which must be a single person. When multiple Scrum teams work on the same product, the same rule still applies. It is essential to have a single person accountable for the results.

Having a committee or multiple persons can make the decision making process complicated as everyone must agree to move on with the decision. When compromises are made, the decision may not be the best one. This also makes it unclear who's responsible for what, and a shared ownership may lead to no ownership at all. Anyone can say "it's somebody else's problem, not mine." If one person is involved, it is clear which person is accountable for the results. This is why we have a single person for the product owner accountability.

Having multiple product owners can lead to micromanagement and coordination issues. They are now Chief Product Owners or Proxy Product Owners in Scrum. If a product owner has too much work to handle, they can delegate tasks to other, both within and outside of the Scrum team. Now don't get me wrong, just because the product owner can delegate tasks does not mean that the product owner can delegate anything.

For example, the product owner cannot delegate their participation in the Scrum events, nor can the product owner delegate work they are not responsible for in the first place. Can the product owner delegate estimation of the PBIs? Well, no. This is the responsibility of the developers. Can a product owner delegate the definition of done? No. This is created by the entire Scrum team. So when scaling Scrum, there is a single person who has the product owner accountability.

That person will be part of multiple Scrum teams at the same time.

## Feature teams vs Component teams

Instructor: Another term that you may encounter when learning about scaling Scrum is related to feature teams and component teams. Let me give you an example so that you can understand what we are talking about.

Currently, Jack and Mary has a single team developing the eCommerce platform. The architecture of the platform is presently divided into three separate layers. The user interface makes it easier for the customers to interact with the shop. This is what they see. There is an application running on the server within the company that handles the business logic and connects the user interface with the database.

The database contains all information regarding the products, the customers, and the orders. Most of the time, to deliver a new feature, the team will work through all the layers or components of the application. A feature is, for example, giving the customers the ability to order a product. The product must be visible on the website. The application must handle the order and the database must store the order for processing.

So feature team works through all the application layers to fulfill a customer need. A feature team is cross-functional and cross-component because it has all the skills needed to complete a feature and does that by working through all the layers or components of the application. A layer or a component is just one part of the product responsible for performing a specific function.

For example, the UI handles only the interaction with the customers and sends any inputs to the next layer. A component team, sometimes called a layer team, is focused on a single component or layer of the system. A component team alone will typically not be able to deliver new functionalities that fulfill a customer's need. Component teams will build components that integrate with the work other teams do, and their combined work provides value to the end user. If the user can't use it or somehow get value from a single component, that alone cannot be considered a product increment.

For example, building a backend service with no user interface, it's not something that a customer can use so no value is generated for the customer. Component teams tend to increase the dependencies between the teams and at the same time, reduce the chances of producing an integrated product increment by the end of the Sprint.

When scaling Scrum, we should focus on reducing dependencies between the teams. Feature teams are desirable as they are closer to the cross-functional aspect as described in the Scrum Guide. Nevertheless, feature teams are not mandatory in Scrum.

## The importance of creating an integrated Increment

Instructor: Let's talk for a minute about the importance of creating an integrated increment in Scrum. If multiple teams work on the same product, they are responsible for collaborating and integrating their work. At any point in time, there is a single integrated product increment containing the work that all the teams did.

Here is my favorite example of teams working on a product. As you may know, I like traveling a lot and I had the chance to see many places. This is a water jug I've used for camping while traveling around the United States. It is a great product, affordable, sturdy, and easy to use. The first aspect I want you to notice is that the product documentation is part of the product.

It is hard to sell this particular water jug without that label. Even if the water jug would work just fine without that label, it is still part of the product for various reasons. It offers product information, like how many gallons it can carry or the fact that is is BPA-free, which for some customers is very important. The label also contains some safety information. It clearly states that this has been designed for holding water and not something else.

The barcode helps retailers quickly identify and sell it. And the label also provides some brand recognition. Some product information is also engraved on the product itself. It is just less information and a bit harder to read, but it is there. To simplify, this product is done and releasable to customers when the plastic water jug holds water and the label has been applied.

So even for a simple product, documentation is part of the product. But I did not show you this example to talk about documentation. I just wanted to point out that at least two separate teams have worked on creating this product. One has worked on the plastic jug itself, including the engraved information, and another team has worked on the label. I will tell you how I know that two teams worked on this in a bit.

Since the company is selling only one product, which is the plastic jug with the label applied on it, they had to integrate their work, and this is the result. You can notice that the capacity in gallons is the same both on the label and on the plastic. But there is a notable difference in liters. This is why I'm sure that the team that designed the plastic water jug did not create the label as well.

Now, there is nothing wrong with having multiple teams working on the same product. But these teams need to communicate and integrate their work. They need to. Their output is a single product regardless of the work they do. In this case, it does not seem that these teams have collaborated or at least they could have done a better job.

Each team has worked on the product from a silo. If people from both teams had been inspecting their integrated work, they would have noticed a difference and found a way to resolve it. I mean, how much communication do you need to design a water jug? Each team can work independently without talking to the other team, right? Wrong.

This is a very simple example of a very simple product. Yes, the mistake is small, but you are smart people and you understand the point that I'm making. When teams work on the same product, they are never truly independent. Now imagine what can happen when teams work on complex products and don't collaborate to integrate their work.

Yes, they can probably glue their work together at the end, but the result may be less than ideal. This is why in Scrum, teams should try to reduce the dependencies between them and constantly integrate their work. In this example, the dependency has been related to the product information, which has been handled by two different teams, which did not collaborate enough.

# Section 7: Terms and Tools used in Agile and Scrum projects

## The Velocity of the Scrum Team

Narrator: Velocity measures the amount of work a Scrum team completed during a sprint. During sprint planning, the developers will select several product backlog items from the product backlog. Let's assume that the Scrum team employs user stories, and each story has an estimation in story points. While very common, using user stories and story points is optional in Scrum, and not something mentioned in "The Scrum Guide."

At the end of the sprint, it is easy to see how many user stories were completed and the total number of story points. The velocity of a sprint is the amount of work completed in a sprint. Only done work counts toward velocity. Even if a story is 90% complete, we consider it incomplete, and we don't add that work to the increment and don't count this toward the velocity of the sprint. Since each sprint is slightly different, we typically use the average velocity when doing forecasts. The average velocity is calculated by averaging the work completed in the previous sprints.

Typically, the more sprints are included, the more data we have, and the more accurate the value is. We also call this the past performance of the Scrum team. In Jira, there is a report that shows the last seven sprints. It does not calculate the average velocity automatically, but some tools may do this. You will be able to see the so-called commitment in gray, represented by how much the developers forecasted, and in green, the actual number of story points completed in a sprint.

Even if Jira shows this as commitment, there's no commitment in Scrum in terms of the amount of work completed in a sprint. Commitment goes only toward reaching the sprint goal.

For example, a new Scrum team has only completed three sprints.The average velocity will be calculated like this, eight, representing the story points completed in the first sprint, plus 13 for the second sprint, plus five for the third sprint, divided by three, the number of sprints, equals 8.66. So the team's average velocity is around eight story points per sprint. The developers may use this information to decide how much work to select for the sprint. Suppose the developers plan to select stories that total 20 story points.

In that case, they may realize that their initial guess may be too optimistic by comparing this with their average velocity, or even by comparing this to their best or worst sprint.However, the developers use this data only for orientation. They decide how much work to select. There is no given minimum or maximum. If this is their first sprint, they will have no past data to use, and that is also fine.

In this case, they will guess how much work to select. There is no right or wrong when dealing with complex work and estimates. It is in the spirit of Scrum to work with the information you have, even if very limited. There is no way for a new Scrum team to know how much work to do in a sprint without the experience of working together on the product.

It is a bit like reading a book. How many pages can you read in an hour? It depends on the font size, line spacing, and so on. If you read for an hour, you can count the total number of pages read at the end of it. This will give you an idea of what is possible in an hour. A similar idea applies to Scrum as well. After a sprint, the team will know what is possibleand use the insights from the past sprintto plan the upcoming sprint.

This is the essence of empiricism. The more sprints they complete, the more they learn and get better at what they do.This is why we have sprints in the first place, to learn and to do better in the next sprint. Many people confuse velocity with capacity. They are related but not the same. Think about capacity as follows. Say you have one-week sprints and four developers working full-time.

In a week, that is a capacity of 160 hours. If in a sprint, one developer is on holiday, your capacity is only 120 hours. The same can happen when a sprint contains holidays or other interruptions. Yes, this can impact the velocity, and this is why both should be taken into consideration when making a forecast. These capacity anomalies will average over time, keeping the average velocity as a good indicator of what is possible in an average sprint.

You need to remember that velocity is not a mandatory metric in Scrum. The Scrum team may track its velocity, but it is not required. The velocity is also intended only for the Scrum team. It is an internal Scrum team metric. Even if this information is available to others, it should not be used to compare different Scrum teams. It actually makes very little sense to compare teams based on velocity even if they work on the same product. Each team may have a different understanding of the work and give different estimates.

Additionally, velocity is not a metric that is directly connected to the value that a product delivers. While it may sound counterintuitive, more output from the Scrum team, in the form of completed items, does not necessarily equal more value being generated. So, more output is not more value. Some people say that velocity just shows that you can build a lot of crap real fast.

While this is a harsh statement, think about velocity as the speed of a team. It is good that you can go fast, but it is more important to go in the right direction. So the velocity alone cannot be used by the product owner to establish that value is being delivered or measure success. The forecasts that the developers make is just a forecast, not a commitment.

While the Scrum team continually works to optimize the development process, Scrum has no velocity targets. There is no such thing as a minimum velocity. It is nice to see a velocity that grows from sprint to sprint, but if the Scrum team focuses only on this metric, they may lose the bigger picture. Scrum teams should not be punished for having lower than expected output in the sprint.

Some sprints work great, others not so much. This is how things sometimes go. We learn from the past and do our best next time. To recap, velocity is an optional metric in Scrum that shows the amount of work completed in a sprint, nothing more, nothing less. The developers can use it to forecast the upcoming sprint.

It's just some data about the past print, but not a guarantee for future sprints and has no relation to the value delivered.

## Technical debt

Instructor: Let's talk for a minute about technical debt and allow me to start with an analogy. Say you need to create a presentation for a company and you only have one hour. You will begin with downloading images, charts, reports and put them on your desktop. Next, you'll put the presentation together and finish just in time. Your desktop folder is a mess, but the presentation is done.

Now you can go back to other work. Just one hour later, you receive a phone call asking you to create another presentation for another company. Again, you start downloading many images and documents and putting them again on your desktop. Now you have your desktop filled with lots of documents but you are having a hard time identifying which image is for which presentation. Is company that P and G for the first or the second company?

This is when you start losing momentum. To make things worse, you need to leave the office and a colleague offers to take over. Now your colleague has to split the files apart. He gets the job done. But between you two three hours were invested in creating this presentation. The fact that the files were not organized after the first presentation was finished made everything take much longer. This is technical debt also known as design debt or code debt.

Unresolved technical aspects from the past come back and haunt you now. The thing is that you never know how technical debt can affect you. If there is too much technical debt in the product it becomes very hard and time consuming to make further changes. Generally speaking, technical debt is something that should be continuously dealt with and not postponed. Dealing with it is part of the development approach and it is a continuous process similar to the architecture of the product which is continuously being worked on and improved.

Developing a product that shows some similarities to the example I've given you previously. Often changes need to be made on the product and due to time constraints, shortcuts are taken. Solving technical issues is rarely a trivial thing to do. While in the short term the product seems to work remember the first presentation, which only took one hour. Later when other changes are requested the scrum team will stumble upon the shortcuts they have taken in the past. This is the technical debt. Another analogy, if a pipe leaks in your house a fix would be to put a bucket under the pipe.

That is a quick fix, but now you need to remember to empty the bucket every day. If it spills over, it causes other issues. The right solution would have been to call a plumber, but that costs money. You wanted to save some money but now you are paying the price. You don't only have a pipe that needs fixing but you also need to redo your floors. If too many shortcuts are taken too often and the Scrum team does not get the chance to fix bad decisions from the past.

The software product is hard to maintain and adding new features takes a long time. So the velocity of the Scrum team is impacted by the technical debt that has accumulated. The technical debt is coupled with the quality of the product and the developers should continuously work together with the product owner to keep the technical debt manageable. If the product owner is not aware of the technical debt, they cannot order the product backlog to address these issues.

If you don't know about an issue, how can you solve it? A high level of technical debt impacts the value generated and often increases the total cost of ownership. This is why technical debt is also a concern for the product owner. So how to deal with technical debt as a Scrum team? First of all, transparency must be insured. As this is a technical aspect the developers need to communicate and remind constantly of any issues. The Scrum master plays a critical role in ensuring that transparency and facilitating discussions around technical debt. The developers need to collaborate with the product owner to reduce the technical debt with every sprint.

The Scrum team needs to make sure that any work tackling the technical debt is part of the product backlog. The technical debt is tightly connected to quality. The Scrum team can increase quality and reduce technical debt by adapting the definition of none to include stricter requirements. To sum up, the technical debt is undesirable, but also unavoidable and it is a concern for the entire Scrum team.

## Functional and non-functional requirements

Instructor: In this lecture, we will talk about functional and non-functional requirements. Let's begin with functional requirements. A functional requirement is a requirement that describes a function of the product. For Jack & Mary Organic Farm, a functional requirement can be the possibility for the customers to order a product.

As a customer of Jack & Mary Organic Farm, I want to use the website to order the desired product so that I don't have to order by phone or go to a grocery store. This is a functional requirement. It describes a function of the product, the possibility of ordering online. This requirement can be easily understood by the product owner, the developers, stakeholders, and users of the product. Actually, the product owner deals with functional requirements all the time, but we usually call them features.

You have probably recognized the format that we have used to write user stories in the past. The product backlog is where we record functional requirements. Apart from functional requirements, there can also be non-functional requirements. A non-functional requirement is maybe harder to describe, but if it is missing, you know that something is wrong.

Let me give you an example.

For the Jack & Mary Organic Farm e-commerce website, we focus on features, on what the customer can see and do. However, if the website is slow and takes a lot of time to do anything, customer may run out of patience and leave before ordering. Now, that is something we don't want. We sometimes forget about this aspect and become aware of them only when something goes wrong. We take for granted that applications should be responsive, load fast, be secure, and provide a great user experience. Customers often complain more about how slow a website is than to appreciate the time we have invested in designing the website logo. However, we can handle this requirement without making them transparent first. Transparency is at the core of Scrum.

How can the Scrum team handle these requirements if we don't know the expectations? So we need to make these requirements visible and clear. Let's take a look at a few examples of non-functional requirements related to the e-commerce platform. Performance. Each transaction, such as placing a new order or viewing an existing one, must complete in under two seconds. Availability. At peak times, the platform should support 5,000 concurrent transactions. Security. All external code libraries used in the application must go through a security testing tool to ensure no known vulnerabilities are included.

Usability. The platform must support visually impaired customers and follow the best practices in the industry. Regulatory compliance. The platform needs to be GDPR compliant. While the functional requirements should be part of the product backlog, it can be confusing what should happen with non-functional requirements. Where should they go? And how do we make these requirements visible? Allow me to point out that the Scrum Guide does not even mention functional and non-functional requirements and, as a result, does not give us a solution.

However, whatever solution we choose needs to be within the boundaries of the Scrum Guide. The Scrum Guide clearly states that the product backlog contains whatever is needed to improve the product. So obviously if we notice that the website's performance is not what we expect, we add this concern to the product backlog. So each requirement, functional or non-functional, should be added to the product backlog. Let's continue with performance as an example of a non-functional requirement. We can create a product backlog item written as a user story that looks like this.

As a customer of Jack & Mary Organic Farm, I want the website to load fast to avoid wasting a lot of time when ordering a product. Most of the time, we also add acceptance criteria to the product backlog item to bring more clarity over the expected outcome. For customers that use broadband internet on the desktop device, the initial load time should be below 1.5 seconds. For customers that use the website on a mobile device over a 4G network, the initial load time should be below two seconds. Each step of the checkout process must complete in less than 0.5 seconds.

Regardless of how we write the product backlog items, the requirements are in the product backlog. We often mix functional and non-functional requirements in the same product backlog item. We may want to implement a new payment method, but at the same time we want to ensure it is secure, fast, and reliable, so functional and non-functional requirements go hand-in-hand. Once a performance-related non-functional requirement has been implemented in the product, we want to ensure that any improvements stay long-term. However, this does not automatically happen. While working on a new feature, we can inadvertently negatively impact the product's performance.

At the same time, after we make a performance-related improvement, we don't want to add to every product backlog item a requirement that says, "Make sure that the website still loads in under two seconds. Thank you very much." Such an approach could soon be unmanageable. Essentially, once a non-functional requirement has been implemented, it becomes a quality of the product. Something like, "We have a fast e-commerce website, and we want it to remain fast."

So for non-functional requirements that apply to the entire product increment and span over the entire product backlog, the most common approach is to add these non-functional requirements to the definition of done. As you remember, the definition of done describes the qualities of the product. Since we add these non-functional requirements to the definition of done, every future increment will need to adhere to them. It just makes sense to add them to the definition of done.

So we could amend our definition of done with the following detail. A Scrum team will often use automated performance tests to ensure that the application performs as expected. Now let's go back to the Scrum rules and understand who does what. The product owner manages the product backlog and can add functional and non-functional requirements. Just as with functional requirements, non-functional requirements originate from the users of the product, stakeholders, the Scrum team, or anywhere else. What you need to remember is that there is no separate list or backlog only for the non-functional requirements.

The product backlog remains the single source of changes to the product, and it contain both functional and non-functional requirements. As non-functional requirements tend to contain many technical aspects, it is common for the product owner to collaborate with the developers or even delegate the management of such product backlog items that have non-functional requirements. Some people have asked if the definition of done also contains functional requirements.

While this is a valid question, so far I have never encountered a case where it would make sense to put a functional requirement in the definition of done. While this is not a rule, a functional requirement is usually related to a specific part or function of the product. However, the definition of done applies to the entire product. Let's just say that adding functional requirements to the definition of done would be an awkward way to add new features to the product. But can the product owner add a non-functional requirement to the definition of done? The answer is no, they cannot. Only the Scrum team can make changes to the definition of done, but of course the product owner, as a Scrum team member, can come up with ideas on how to improve the definition of done and discuss them with the team.

When should the Scrum team handle non-functional requirements? Are there any dedicated sprints? No, in Scrum, we never dedicate sprints to handling a single aspect of the product. Every sprint builds a new increment that contains both functional and non-functional requirements. Whatever is needed in the product is dealt with during the sprint. Every sprint should build some working functionality that the users can use. So to recap, the product backlog contains both functional and non-functional requirements.

Non-functional requirements can be expressed in the product backlog or the definition of done. While not a rule, most of the time non-functional requirements take the path of first being added to the product backlog, implemented during a sprint, and finally added to the definition of done. We don't dedicate sprints solely to addressing non-functional requirements.

## The process of emergence in Scrum

Instructor: An important concept in the HR world is the idea of emergence. Quite often you will hear about emergent architecture or emergent infrastructure, emergent requirements. Allow me to explain things in a very plain way. When we do complex work, we know very little in the beginning. For this reason, we never create final or extensive plans. We don't have a final product backlog in the first sprint.

We never have a final product backlog. By nature, the product backlog constantly changes. The idea of a sprint is to do just a bit of everything required to create a version of the product. This includes planning, design, development, testing, documentation, and anything else necessary. This also includes architecture or infrastructure work as well. For example, the architecture of a software product will define which components the system will have and the relationships between them. This is a very important aspect of the product. It is like the architecture of a house.

You can't build something without thinking about the architecture. In waterfall projects, the thinking has always been in stages. One of the initial stages is related to designing the architecture, the final architecture. By now, we know that this rarely works. In the beginning, we know the least about the product so it makes little sense to create something final when we have the least amount of information. Emergence is a process in which new information comes to light, and it usually happens sprint after sprint. In scrum, even if the architecture is very important, we never have dedicated sprints in which we focus only on the architecture and forget about everything else.

On the contrary, the verb to emerge simply means that something is revealed or discovered. The word emerge also appears in the principles behind agile manifesto. Make sure to revisit that. This is why we talk about emergent architecture. The architecture of the product emerges sprint after sprint In the first sprint, we do a bit of architecture, in the second, maybe a bit more or less, but the architecture is continuously being developed. The same goes for anything else that is needed in the product.

Every new product increment is a mix of the different aspects of the product. To produce an increment, every sprint requires some planning, design, building new features, fixing bugs, testing, documentation, architecture, infrastructure, dealing with technical debt, and anything else that is needed. Every sprint tends to contain a bit of everything. This means that this is a learning process. We try something out, see how that works, and make adjustments along the way. We design architecture of the product in a way that can help us in the current sprint.

We don't need to worry today about what architecture will look like in six months from now. This does not only apply to architecture also to any other work that is needed. People have a natural tendency to anticipate things and to look far ahead. For some doing just a bit of something feels wrong, feels incomplete. Why not build something once and do it well and never touch it again?

Since we deal with complex work and ever-changing requirements, we must accept that we cannot anticipate everything. We are truly agile if we embrace changing requirements and accept rework as part of the process. Maybe we need to completely rethink the architecture of the product to meet our customers needs. If you wish to get a bit deeper into this subject, I have added some interesting links to the resources folder.

## Burn-down charts

-: One of the charts that Agile projects can use is a burn down chart. Burn down chart is a graphical representation of progress showing the amount of remaining work against time. In this chart work is represented in story points. The remaining work is on the vertical axis and time on the horizontal axis. On this chart the time axis shows the duration of a sprint. If the burn down chart represents the sprint backlog it can be used to monitor the sprint's progress and to see if the work selected will be completed on time. The red line represents the remaining work and starts at the top. If you don't complete any work the line will stay flat indicating that no progress has been made. As soon as an item in the sprint backlog is completed the line goes down, hence the name burn down chart.

When we reach zero, all work has been completed. By the way, monitoring the sprint progress is the responsibility of the developers and they track the work remaining at least once per day during the daily Scrum. The developers are responsible for managing the sprint progress and can use a sprint burn down chart to do that. They track how much work from the sprint backlog has been completed. Smaller the items, the easier it is to track the progress and get an accurate picture. A burn down chart won't be very helpful if we have a sprint with only two 20 story point items.

As you can notice the burn down chart also includes a guideline marked in gray that shows linear progress. This is an ideal line that we can use to check how things are progressing. As long as the red line is below the guideline we are on track. If we are above the guideline, we are behind with work. If the developers notice that they are falling behind they can use this information to work with the product owner and adapt the sprint backlog while still keeping the sprint goal.

Sometimes new work is pulled into the sprint backlog. This is the case, the remaining work will go up and you'll notice a spike in the graph. While burn down charts can be used in Agile projects, they can also be applied to any project measuring progress over time. The product owner manages the product backlog and wants to track the progress of work. They're interested in the big picture. This means that the product owner can also use a burn down chart to monitor progress over multiple sprints. Remember that the developers are monitoring the progress of the sprint while the product owner is monitoring progress toward the product goal.

For a product owner, it makes more sense to look over a period that spans over multiple sprints. The product owner should track this at least once per sprint and evaluate the progress made towards reaching the product goal. Sometimes the burn down chart used to monitor the work in the product backlog is called a release burn down chart. Please remember that Scrum does not enforce any reporting tools or charts. It's totally up to the Scrum team to decide which tools to use. You may encounter exam questions that use the term burn down chart, as burn down charts are not mandatory in Scrum.

An answer containing a burn down chart is rarely the best answer. Burn down charts only show the remaining work and are not related to project costs, business value or the productivity of the Scrum team. If you want to dive deeper into burn down charts I suggest checking the resources folder where I have linked additional materials.

## Burn-up charts

Instructor: The burn-up chart is an alternative to the burn-down chart for tracking progress. It looks like a burn-down chart, but turned upside-down. Well, almost. When you use CS to track a sprint, the burn-up chart provides a visual representation of the completed work in relation to scope, which represents the total size of the selected product backlog items. Work done starts from zero, which makes sense since we haven't done any.

As soon as we complete an item, the green line goes up towards the scope, the total work that was forecasted. We also have a guideline. We are on track to complete the work selected, if we are above the guideline. We are below the guideline. We are falling behind. It is pretty similar to the burn-down chart, which the difference that the progress line is trending up, and that any added work will be visible in the change of scope. See, in this example, the red line going up. To better notice the difference between the charts, on the left, you have a burn-down chart, and on the right, a burn-up chart.

Both show the progress of the same sprint. Pause the video for a second to take a closer look. Please remember that Scrum does not enforce any of the reporting tools and charts mentioned. It's totally up to the scrum team, which tools to use. If you want to dive deeper into burn-out charts, I suggest checking the resources folder, where I have linked additional materials.

## The "cone of uncertainty"

Narrator: The cone of uncertainty is a visual representation of the uncertainty in a project. If you live in an area where you're experiencing hurricanes, you may have seen on the news something like this. This is a cone of uncertainty. So what does this graphical representation tell us? It tries to anticipate the path that the hurricane will take and shows which areas are most likely to be affected. So, where is the uncertainty? Well, hurricanes are very unpredictable and can change their path very rapidly.

So the uncertainty is that we don't know which direction the hurricane will take this time. As you can see, the uncertainty is initially lower, as we can better predict where the hurricane may be heading. However, the uncertainty increases over time. One change in the direction right now can completely change where the bad weather is heading in two days from now. So the longer the forecast, the higher the uncertainty. I'm going to repeat this because it is very important. If you create a forecast for tomorrow, the uncertainty is relatively low. But if you create a forecast for the next two weeks, the uncertainty is higher as much more can happen in this period. So the longer the forecast, the higher the uncertainty.

Short forecast, low uncertainty. Long forecast, high uncertainty. If you did not know this already, hurricanes and projects that go wrong have so many things in common. But leaving the weather and the hurricanes aside, the cone of uncertainty is used both in traditional project management and AGILE projects to represent uncertainty. Let's take a look at a practical example. Jessica, our product owner, is asked by one of the stakeholders when a specific feature will be ready.

Jessica can create a cone of uncertainty with the best, worst, and average estimates based on the past progress of the Scrum team. The forecast will indicate how many Sprints are needed for a feature to be done based on the empirical evidence from the previous Sprint. Here is the velocity, which represents the amount of work completed for the past five Sprints. This team uses story points. In the first Sprint, the team completed nine story points.

In the second one, 12. After this, 10, 14, and 11. Some simple math will indicate that the average velocity over the past five Sprints to be 11 story points. Also, we know the lowest velocity has been nine story points and the highest, 14. Looking at a product backlog, Jessica will identify the respective product backlog item. Regardless of which velocity is being used, the uncertainty is low. This is the second item in the backlog and the chances that it will be completed in the upcoming Sprint are very high. Now let's move further down in the product backlog and take a look at another item.

This time, the uncertainty increases. After crunching the numbers, we can see that we are unsure if this will be done in the next two or three Sprints but at least we know it'll not be available after the next Sprint. So with this information, Jessica tends to be a bit more conservative, adds a small buffer, and tells the stakeholders that the feature will most likely be available in three Sprints. Remember that we can never guarantee future velocity or the outcome of a forecast. The scope of work can change, unforeseen events can occur, and the product backlog may need to be reordered. With this in mind, the product owner will not make promises and forecast is not a commitment.

Just as with the weather forecast, as time passes by, we gather new data and we need to update the forecast we made. The further down we go into the product backlog, the less certain we can be when a feature will be completed. So to recap, uncertainty increases over time. The longer the forecast, the higher the uncertainty. Just keep in mind the image with the hurricane. A forecast is never a commitment and everyone needs to be aware of this. No matter how much historical data we have gathered, we'll never know the exact completion date for a project or a feature. This is simply the nature of complex work.