A POCKET GUIDE FOR SCRUM TEAMS

A collection of rules, principles and practises for everyone who wants to learn about Scrum

KATYA AGOSTI

A POCKET GUIDE FOR SCRUM TEAMS

I created this pocket guide for anyone who wants to learn about Scrum and how to get the basics up and running.

The book is divided into 4 parts:

- Part I is an introduction to Agile, comparing it with other approaches to product development.
- Part II focuses on explaining Scrum, covering all rules, roles and main principles.
- Part III is a collection of best practises that can be a helpful guide on the journey of discovery and fun, showing to the reader the playfulness of Scrum.
- Part IV explores some tactics and hints that allow Scrum to be played on a larger scale.

“I love how succinct and clear the writing is. Complex concepts are boiled down into the most essential pieces to make it really accessible to the readers. Very nice!”

(Kelly Cook, Agile Coach at AND Digital)

First published in 2022 by Katya Agosti

A pocket guide for Scrum Teams

A collection of rules, principles, and practises for everyone who wants to learn about Scrum

Subjects:
1. Introduction to Agile
2. Scrum roles and rules
3. Collection of practical tips and tricks
4.Scrum played at scale

Author: **Katya Agosti**

Cover Design: **Valeria Agosti**

ISBN: **9798420198308**

This book is intended to be an introduction to Scrum. The content of the book is a collection of rules and practises I experiment in my daily work. I summarised here what I think everyone should learn about Scrum to get the basics up and running, and I am sharing it for educational purposes.

The material in this publication does not represent professional advice and it should not be relied on as the basis for any decision to take action or not take action on any matter which it covers.
All rights reserved. You cannot copy and redistribute the material in any form or by any means without prior writer permission, and you cannot use the material for commercial purposes.
All enquiries should be made to **katya.agosti@gmail.com**

To Rosario,
who encourages me to explore the world

Table of Contents

Introduction

It is hard to start a career as Scrum Master with just a certification. Companies that will hire Scrum Masters with knowledge and no experience are difficult to find. One good way to start is by finding opportunities to be part of a Scrum Team and then slowly embark on a Scrum Master career. This was my career path. After getting my Scrum certification, I started my agile journey as a lead developer in a Scrum Team. Later I started helping the Scrum Master with some of his responsibilities, until eventually I took over from him in the Scrum Master role. Moreover, in order to keep improving my skills, I have been regularly attending Scrum/Agile meetups, reading Scrum Masters blogs and lots of books.

Now that I have learned the basics, and perhaps a little more, of the Scrum framework, I have decided to start sharing my passion regarding the Scrum Master role, and my thoughts and experience on Scrum with the world.

By writing this book, I have better hardened my own understanding of Scrum. Knowing that others can benefit from reading about my experience is something that brings me great joy and satisfaction.

Who is this book for?

This is a book for anyone who wants to learn about Scrum and how to get the basics up and running. If you are a beginner, you may want to read the whole book from cover to cover. If you are already familiar with Scrum, you can use this book as your reference guide.

How this book is organised

This book is divided into 4 parts:

The first part is an introduction to Agile, comparing it with other approaches to product development.

The second part focuses on explaining Scrum, covering all rules, roles and main principles.

The third part is a collection of good practises that can be a helpful guide on the journey of discovery and fun, showing to the reader the playfulness of Scrum.

The fourth part explores some tactics and hints that allow Scrum to be played on a larger scale.

Hope you enjoy reading this book as much as I enjoyed writing it.

Happy reading!

Part I

Introduction to Agile

Chapter 1

Origins of Agile

Agile is an umbrella term for several product development approaches sharing the principles and values described in the Manifesto for Agile Software Development (known more commonly as the Agile Manifesto), officially introduced in 2001.

The Agile Manifesto was written by seventeen software development leaders, who agreed there was an increasing need for an alternative to heavyweight software development processes and, despite having widely varying opinions on the right approach, did find consensus around four core values described below:

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.”

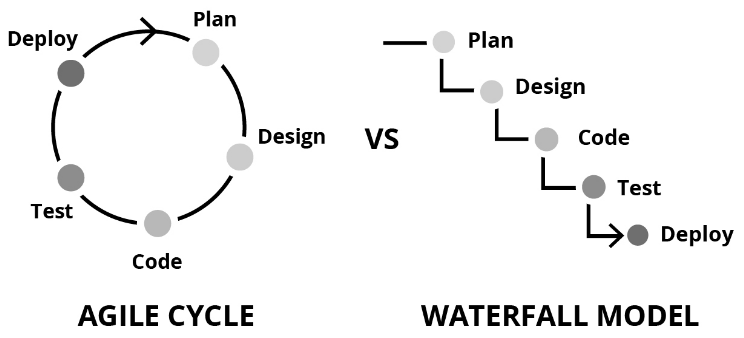
They also laid out 12 principles that stand behind these values. Those principles include:

- Satisfying customers through early and continuous delivery of valuable software
- Welcoming changing requirements at any point in the delivery cycle
- Delivering software frequently through shorter development timelines
- Using working software as the primary measure of progress
- Taking regular moments of self-reflection to identify opportunities for improvement

The 4 values and 12 principles continue to guide the Agile approach used by teams today.

Chapter 2

Agile vs Waterfall: which one is right for your project?



Before Agile, the Waterfall model, which is just one example of a broader class of plan-driven processes adapted from manufacturing, was considered the best practice for software development.

In this chapter I compare Agile, an approach based on flexibility and continuous improvement, with Waterfall, a linear and predictive paradigm. To avoid misunderstandings, I want to add that by predictive I mean that Waterfall greatly emphasises the importance

of anticipation versus adaptation, while Agile emphasises the opposite. But I am not saying that Agile teams don’t plan. Adaptive planning is an integral part of any agile project.
Moreover, my goal of making this comparison is not to convince you that Waterfall is bad and that Agile is good; both the approaches carry their own set of strengths and weaknesses to project delivery, and both have projects more suited to them.

The waterfall model, introduced in 1970, quickly gained popularity because it brought discipline in software development. It is a sequential design process, in which progress is seen as flowing steadily downwards (like a waterfall) through several distinct phases: Requirements, Design, Implementation, Test, Deploy, Maintenance.

The Waterfall model does have advantages, such as:

- It is easy to understand and use, it follows the same sequential pattern for all projects.
- Discipline is enforced, indeed every phase has a start and an end point. It is easy to share progress with customers.
- Puts great emphasis on documentation. The detailed documentation makes it easier to hand over to separate support teams. Moreover, in case of employee turnover, the strong documentation allows for minimal impact on plans.

- It is good to be used when the project is a familiar territory with a predictable path and the client knows what to expect.

It has also quite a few disadvantages, including:

- Clients usually appreciate what is needed when the application is delivered, therefore lots of educated guesses and assumptions are made upfront, but software is not delivered until late. The coding begins only at the third phase.
- Once a phase is completed, you are not supposed to go back. If the team discovers that a requirement is missing while being in the test phase, this could have an impact on the rest of the project.
- Changes cannot be easily accommodated: if requirements change, the plan and the budget will be adversely impacted.

Agile addresses the limitations of Waterfall, but it also has a few drawbacks, besides strengths.

Some of the Agile strengths are:

- Change is embraced: with shorter cycles, it is easy to accept changes in requirements at any time.
- It is very good when the end goal of the project is not well defined or when the end outcome is clear but there is flexibility in how it is met.

- Breaking down the project into iterations allows the team to focus on high-quality development, test, collaboration. Having tests during each iteration permits the team to find and fix bugs quickly.
- Working software is delivered at each iteration and customers have the opportunity to use it, share their input and have a real impact on the project.
- Feedback from users and team members is encouraged, so lessons learned are used to improve future iterations.

Some of the Agile drawbacks are:

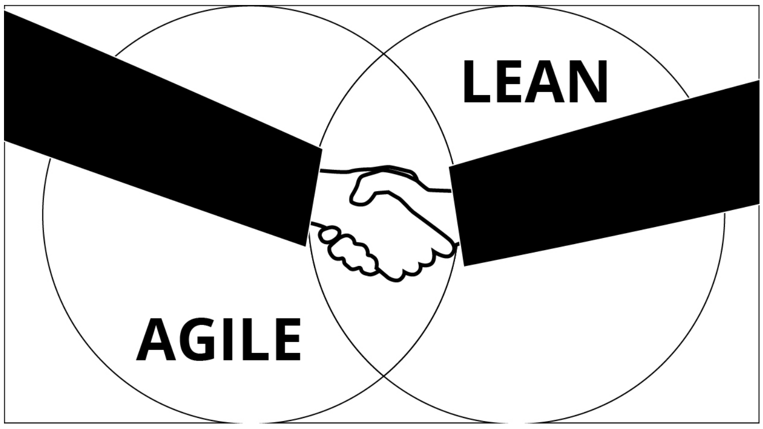
- Planning can be less concrete as tasks can be reprioritized at each iteration, it is possible that some work items initially scheduled for delivery may be not completed in time.
- The team must be knowledgeable. Agile teams are usually small, so team members must be highly skilled in a variety of areas.
- Documentation may be neglected. One of the Agile values puts emphasis on working software rather than extensive documentation, so team members could consider documentation less important. The team should find the right balance between documentation and code, and acknowledge that this can be challenging for them to balance sometimes.

As I said at the beginning of the chapter, the key to deciding which is right for you comes down to the context of the project. If it is going to be changing rapidly, eg. groundbreaking project with uncertain outcomes and high rates of change, complexity, and risk, choose Agile. In case you know exactly what you need, eg. produce a predetermined end result within a specific date and no flexibility in how to deliver it, then maybe Waterfall is the better option. A third option could be to consider taking aspects of both methodologies and combining them in order to make the best possible software development process for your project.

If you are interested in bringing agile development to your team or organisation, keep reading.

Chapter 3

Agile and Lean



We often consider Lean and Agile to be two different philosophies. However, if we take the principles of Lean to heart, we end up creating an Agile team as well. In fact, many advanced Agile coaches now use ideas and vocabulary from lean when teaching teams.
When we talk about Lean, the first name that strikes our mind is Toyota. However, the history of Lean started in 1913 with Henry Ford who was the first to truly integrate the concept of lean in the manufacturing system.

Ford created what he called a flow production (also known as mass production), which involves continuous movement of elements through the production process.

Ford used mass production to fabricate and assemble the components of his vehicles within a few minutes rather than hours or days. Unlike craft production, the mass production system delivered perfectly fitted and interchangeable components.

It was only in the 1930s that Toyota, inspired by Ford's flow of production concept, developed several novel ideas that became known as the Toyota Production System (TPS). After studying Ford's production system, they understood that the mass production system employed by Ford could not be used by Toyota. The Japanese market was too small and diverse for mass production. The customer's requirements ranged from compact cars to the most luxurious vehicles, but Ford's mass production system focused on standard products which could not meet all the customer demands.

Therefore, Toyota collaborated with the engineer Taiichi Ohno to build upon Ford's ideas and found a way to make high quality and low cost products that met the changing desires of the customer. They created a unique pull system, which then became the backbone of lean manufacturing, to avoid overproduction and meet the diversified customer demands.

Ideas from the Lean movement in manufacturing became internationally known and recognized during the 1990s and gradually began to be applied to software.

Lean is centred on defining value from the customer's viewpoint, continually improving the way in which value

is delivered, and eliminating every use of resources that does not contribute to the value goal.
The concept of continuous improvement, or Kaizen (which comes from two Japanese words, Kai and Zen which mean “improvement” and “good”), is at the heart of Lean philosophy. The goal of continuous improvement is to help identify opportunities for work process enhancements. This is done by empowering every individual worker to achieve his or her full potential, and so to make the greatest possible contribution, helping workers grow professionally and personally, allowing them to take pride in their work.

Lean and Agile are truly blending philosophies. Agile has distinct practises that match the main Lean principles and below are highlighted a few similarities between them.

Respect for people. People refers to every possible actor in the whole ecosystem of product development: customers, workers, teams, managers. All people contribute in their own way and collaborate across skills to build and deliver a product. Respect for people is essential for Lean, and Agile embraces this principle too, as emphasised in the Agile Manifesto (principle #5: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done).

Continuous flow of results. Agile mirrors the flow principles of Lean that have proven to be successful.

Agile is about working smarter, rather than harder. It's not about doing more work in less time, it's about generating more value with less work (principle #10: Simplicity - the art of maximising the amount of work not done - is essential).

Focus on customer value. Customer value is defined, in Lean thinking, as the perception of what products and services are worth to customers. In truly Agile organisations, everyone is passionately obsessed with delivering increasing value to the customer. Everyone has a clear line of sight to the end-user and can see how their work is adding value to that customer, or not.

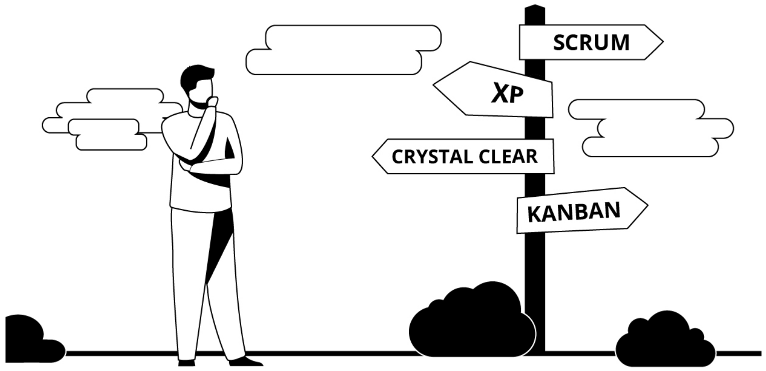
Eliminate waste. One of the key elements of Lean is to relentlessly identify and eliminate anything that doesn't create value. Agile supports this concept as well, by maximising the amount of work not done so that the team focuses on building the simplest solution, doing the least amount of work possible in order to deliver the right value.

Continuous improvement. This is where Kaizen, a Lean method for continuous improvement, comes into play. Kaizen focuses on improving and evolving processes to better support the evolution of the product and enable Agile teams and organisations to reach their goals of frequent, iterative value delivery. The practice of Kaizen ensures every iteration is better than the last in some way.

At the end of the day, we're all realising that it's not about picking one process or methodology and following it like it is gospel. It's about combining all our best practises and learnings together and then using what makes sense in our contexts, given what our organisation goals are.

Chapter 4

Agile frameworks



Now that we have fully equipped ourselves with the basics and abstract principles of Agile, let's have a look at some of the practice-oriented frameworks that turn them to life.

SCRUM

It is the most popular agile framework. The name is borrowed from rugby, where a scrum is a cluster of players trying to get the ball. The term scrum was chosen by its authors because it emphasises teamwork. Scrum is a lightweight, iterative and incremental framework for managing complex work. The framework

challenges assumptions of the traditional, sequential approach to product development, and enables teams to self-organise by encouraging close collaboration of all team members.
We will find out more about Scrum in Part II of this book.

XP

XP stands for eXtreme Programming. XP got its name because it takes cherry-picked programming practises to extreme levels. It places a strong emphasis on technical practises in addition to the more common teamwork and structural practises.
One of the key aspects is that XP teams perform nearly every software development activity simultaneously. Analysis, design, coding, testing, and even deployment occur with rapid frequency. That's a lot to do simultaneously. XP does it by working in iterations: week-long increments of work. Every week, the team does a bit of release planning, a bit of design, a bit of coding, a bit of testing, and so forth.
XP works towards a continuously improving and high quality product that will allow the team to respond quickly to customers' changing requirements. The framework aims to do that and reduce the cost of development, in the short term as well as in the long term, by improving the internal quality of its code and design.
Some of the XP core practises that keep the code clean include Test Driven Development, Refactoring,

Customer Testing, Simple Design, Continuous Integration, Small Releases and Pair Programming.
More and more often, we can see XP practises used by teams that utilise Scrum and other organisational Agile approaches to unlock the most out of the developers' potential.

CRYSTAL CLEAR

Crystal Clear is an agile framework focusing on individuals and their interactions, rather than processes.
The method has a few key properties needed to successfully run an agile project, such as:
Focus. Executives and leaders must set priorities and make it clear what developers should be spending their time on. Then the developers must be given time, without interruptions, to work on those priorities. Teams should be allowed to set up two hours of each working day where no interruptions are allowed including meetings and phone calls.
Personal safety. Team members must be able to speak up when something is bothering them without fear of reprisal. When able to speak freely a team can discover and fix its weaknesses. Three things software developers must be open about, in relation to the project, are being able to reveal their ignorance, errors made, and any incapacity they have in meeting the requirements of an assignment.
Easy access to expert users. Getting prompt response from users, and especially to questions, is key. When

problems occur because users do not want frequent releases, one solution is to find a single user willing to try out the new software on a trial basis. Not only with Crystal Clear, but with any agile framework, getting back user feedback is critical.

Technical environment with automated tests and frequent integration. Development teams, running Crystal Clear, integrate the system multiple times per day, or at a minimum, once daily. Moreover, doing continuous integration they will be able to detect integration level errors within minutes.

In general Crystal Clear method works best for small teams of six to eight developers, preferably being co-located as it facilitates osmotic communication.

Moreover, teams using other agile approaches can borrow useful techniques from Crystal Clear and vice versa.

KANBAN

Kanban has its origins in Just In Time (JIT) manufacturing processes at Toyota automotive. In the early 1940s, Toyota needed to increase the efficiency of its production to stay competitive with its American rivals, and Toyota engineer Taiichi Ohno looked at supermarkets for inspiration. Observing how supermarkets restock their goods based on what’s been picked off the shelves, he created a simple supply system where production plans would be driven by

actual consumption. He also wrote several books about the system, laying out the principles of Lean Manufacturing and Kanban. From here, both the approaches grow in popularity in the manufacturing sector in Japan and abroad.
The system's goal is to refrain from producing a surplus. It achieves that by using cards and a board to visualise the workflow (Kanban is a Japanese word that means a "visual sign or billboard"). This gives everyone involved maximum insight into the process and helps managers address surplus/shortage in real time.
In 2004, David J. Anderson, a Lean thinker, explored the Kanban system and decided to apply it to software development. The approach was refined over the next few years until he formulated the Kanban Method.
Soon, the Kanban Method started to catch the attention of Agile teams. The method was a natural fit for them with its focus on making incremental changes and increasing delivery speed.
The Kanban method also uses the notion of "pull" over "push," meaning that workers pull in work according to their capacity, as opposed to work being assigned to them from the top regardless of their capacity.
Some of the key practises of Kanban are: visualise the workflow, limit work in progress, manage the flow, make policies explicit, and implement feedback loops.
Kanban is often considered an Agile method, but it is not a management framework such as Scrum, it is a visual system for managing work as it moves through a process. Kanban is rather an alternative path to agility, it applies both Agile and Lean principles, and it can be

applied on top of any process or methodology to apply gradual improvements, whether you are already using Agile methods or more traditional ones.

Moreover, Kanban seems to be better suited for teams that have a lot of unplanned work coming up such as support issues, emergency fixes, and urgent feature requests because it is optimised for flexibility.

Part II

Scrum Roles and Rules

Chapter 5

Origins of Scrum

There was a time when “scrum” was only a rugby football term. A scrum (short for scrummage) is a method of restarting play in rugby football and involves eight players from each team tightly huddling as they compete to win the possession of the ball.
This form of collaboration inspired the Scrum method in business. Indeed, in 1986, two Japanese business experts introduced the term in the context of product development. Hirotaka Takeuchi and Ikujiro Nonaka published the article, "The New New Product Development Game" (the double “New” is indeed part of the title) in the Harvard Business Review. Their research showed that outstanding performance is achieved when

teams are small and self-organising units of people and when such teams are fed with (challenging) objectives, not with executable tasks. Teams can only achieve greatness when given room to devise their own tactics to best head towards shared objectives.
The applicability of those concepts was subsequently extended to the software development industry by Ken Schwaber and Jeff Sutherland who introduced their own adaptation of Scrum, during the object-oriented conference OOPSLA in 1995, to counter the established waterfall style project management processes.
They have been collaborating since that time to introduce ever greater clarity and maturity to the Scrum approach, and in 2010, they published their first version of the Scrum Guide, as an attempt to help practitioners understand the rules of the game.
In July 2011, the two authors added an 'end note' to the guide, offering a hint to what they are really striving for:

"Scrum is free and offered in this guide. Scrum's roles, artifacts, events, and rules are immutable and although implementing only parts of Scrum is possible, the result is not Scrum. Scrum exists only in its entirety and functions well as a container for other techniques, methodologies, and practises." (Source: Scrum Guide July 2011)

Ever since its first publication in 1995 up to now, Scrum has been adopted by a vast amount of software

development companies around the world. It is the most applied framework for agile software development.
The framework, however, has also been successfully applied in other domains, e.g. manufacturing, operations, education, marketing, finance, etc.

Chapter 6

Scrum Pillars and Values

Scrum's roots lie in empiricism which is the idea that knowledge comes from experience.

Empirical approach means working in a fact-based, experience-based, and evidence-based manner, and in particular, progress is based on observations of reality. Engineers typically call this feedback, which is information that can be used immediately to correct a course of action when needed.

The three pillars of Scrum that uphold every implementation of empirical process control are: Transparency, Inspection, and Adaptation.

- Transparency: All people involved are transparent in their day-to-day dealings with others and they all trust each other. This promotes an easy and transparent flow of information throughout the organisation and creates an open work culture. Everyone strives and collectively collaborates for the common organisational objective.
- Inspection: The inspection can be done for the product, processes, people aspects, and practises with the purpose to identify whether the path the team has taken will indeed result in achieving the goals established.
- Adaptation: Adaptation in this context is about continuous improvement. Based on the results of the inspection, the team has the ability to adapt and might choose to modify its own behaviours to more closely meet the established goals, or perhaps to use empirical evidence to pick better goals.

The Scrum framework is also based upon 5 core values: commitment, focus, openness, respect, and courage. The Scrum values are more than words or abstract ideas. If they do not lead to concrete changes in behaviour, the team may as well not have them. Successful Scrum depends on the entire team embodying the core values because Scrum is not only about delivering quality work, it is about creating a great place to work, a place where people can take pride in their job and themselves. What follows is a short description of each Scrum value that will help teams incorporate them into their interactions.

COMMITMENT

Commitment here does not imply a contract, it implies instead a mindset or an attitude. The entire Scrum Team defines the team's objectives, owns up to their commitments and sticks to them. Each team member thus commits to delivering on this promise. They also commit to assisting each other wherever required and working as a team to deliver the best possible product. This commitment, when combined with other values, helps the team drive towards maximising value.

FOCUS

Focus can help teams to get work done by encouraging teams to zoom in on blocked or undone work and collaborate to get it complete. Focus is essential for a team to prioritise work when there are multiple options. And there are always multiple options because there is always more work that could be done compared to the capacity of the team. And most importantly, focus is what enables a team to ignore distractions and move in a coordinated fashion to achieve their goal.

OPENNESS

Openness is extremely important to succeed in Scrum and it is also closely related to one of the three core pillars: transparency. The Scrum team members should be open with each other and utilise information radiators as much as possible. The team should also be open about the work they are doing, about their progress, able to admit when they are wrong and open to honest feedback about the product, learning how to build better products, how to interact with each other better, and learning new ways of working.

RESPECT

Respect is completely fundamental to not just Scrum but any form of collaborative team-based work. Members on the team need to respect each other as people, respect each other's differences and respect that they won't always agree on methods but everyone is doing their best to reach the common goal of building a wonderful and successful product. Moreover, the overall organisation needs to respect Scrum, respect the team and respect the pillars of transparency, inspection and adaptation.

COURAGE

Courage is an interesting value, and enormously important. Courage is essential to transparency and safety. The team must have the courage to admit mistakes, otherwise they can never learn from their experiments. They also must have the courage to admit they are going down a wrong path and to change direction. For Scrum to really work, teams need to be empowered, they need to manage their own work, and not be micromanaged. If all those things hold true, then the team needs the courage to accept this responsibility and forge their own path.

When applied properly, the pillars and values help nurture and protect Scrum and give it its best chance of success. When ignored, Scrum will likely become a mechanical system of attempting to improve “productivity”, with little or no chance of succeeding.

Chapter 7

The Game of Scrum

The goal of Scrum is to help people, teams and organisations generate value through adaptive solutions for complex problems, as described in the Scrum Guide. The various roles of the Scrum team and Scrum events help the team break down large projects into manageable steps to optimise predictability and control risk.

Players and accountabilities

Scrum organises its players into Scrum Teams. Scrum Teams are cross-functional, meaning the members have all the skills necessary to create value each Sprint. They are also self-managing, meaning they internally decide who does what, when, and how.

The Scrum Team is responsible for all product-related activities from stakeholder collaboration, verification, maintenance, operation, experimentation, research and development, and anything else that might be required. They are structured and empowered by the organisation to manage their own work.

A Scrum Team consists of three roles, where each role complements the other roles is accountability, thereby turning collaboration into the key to success.

The Product Owner

The Product Owner is a one-person player role who has a clear view on the goals of the project, customer, market and organisation. The Product Owner is constantly working towards aligning the work with the product vision that captures why the product is being built, is responsible for defining the work and then prioritising those tasks. They communicate this to the scrum team and guide them through the project. It is important that the PO actively engages with the other players of the team regularly and repeatedly.

Rather than merely defining all the work at the start with a scope statement, like a traditional project manager, the Product Owner will review and reprioritize with feedback as needs change.

The Scrum Master

As the name implies, the Scrum Master, a one-person player role, protects the Scrum process. Since they're experts in Scrum and know how it should be applied, they are vigilant that the Product Owner and Developers are working within the Scrum framework. They are not merely Scrum police, but Scrum teachers who will coach team members on how to most effectively use the framework.

But the Scrum Master also provides a guiding light to lead the project to success. In a sense, they are the protector of the team in that they will make sure that everyone on the project can focus without distractions. That includes distractions from an overreaching Product Owner, and organisational or internal distractions, too.

Though some might see this as a project manager by a different name, it is not. Project managers manage the work of the project team members, whereas a Scrum Master guides his team but lets them work autonomously.

The Developers

The Developers are the heart of the Scrum Team, as they're the ones responsible for doing the actual project work. Each member of the team has a skill that, together with the other team members, combines to tackle all the needs of executing the project. The Product Owner sets the priorities, and the work is guided by the Scrum process and monitored by the Scrum Master, but all other responsibilities are laid at the feet of the Developers. That autonomy is the very core of the process. What comes from this approach is strong team bonds and a positive working environment where people feel empowered on the job. While this is not entirely alien to traditional project management, waterfall teams, for example, are managed by a project manager, not self-managed.

Time and events

The time-boxed iterations in the game of Scrum are called Sprints. A new Sprint starts immediately after the conclusion of the previous Sprint. Within a Sprint, a planned amount of work has to be completed by the team and made ready for review. In a sense, one Sprint within Scrum could be a project in itself, with start and finish. The team works towards a Sprint goal, they plan, refine, build, deliver, and review.
The Sprints can be as short as a few days and generally are no longer than 3 – 4 weeks.
As a container event, the Sprint encapsulates the Scum events, every event is timeboxed and is an opportunity to help the team work efficiently and closely together, adapt to changing conditions and to improve their knowledge and become more effective in the future.
The events are:

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

Sprint Planning

Every Sprint begins with the Sprint Planning. This ceremony helps to set up the entire team for the coming iteration, creating a smooth pathway for a successful Sprint. Sprint Planning requires the participation of all the scrum roles. It typically lasts for an hour or two.

The Product Owner, who proposes how the product could increase its value and utility in the current Sprint, comes to the meeting with a prioritised list of the product backlog items, which is presented to the group.

The whole Scrum Team then collaborates to define a Sprint Goal that communicates why the Sprint is valuable to stakeholders. The team selects the amount of work it considers feasible for the Sprint against the expectations of what it takes to make it releasable. The Product Owner also needs to be able to clarify any questions or assumptions that the Development Team has about the work.

Through a series of discussions and negotiations, the team should ultimately create a sprint backlog that contains all items they are committing to complete at the end of the sprint. The selected work is a forecast that represents the insights that the team has at the time of selection. The team might look at the amount of work they have, on average, completed in past Sprints and compare this to their capacity for the upcoming Sprint, to slightly increase the accuracy of the forecast.

Daily Scrum

To manage and follow up on its development work, the team holds a short daily meeting called the daily Scrum. The meeting is the team's chance to get together, define a plan for the day's work, assess progress towards achieving the Sprint Goal and identify any blockers to achieving the goal.

The Daily Scrum is timeboxed to 15 minutes. Standing up is not compulsory. However, many teams find this a useful technique to keep the event short and to the point, which is why it's also called daily standup.

One of the principles from the Agile Manifesto — "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly." — sums up the reason behind the next two events, the Sprint Review and the Sprint Retrospective. Both events take place at the end of the Sprint. The aim of Agile approaches is not necessary to get everything 'perfect' the first time around but to improve continuously. These events help make that possible.

Sprint Review

As the Sprint progresses, an increment of the product emerges from the team's collaborative work. At the end of the Sprint, the review allows the Scrum Team the opportunity to show the done increment to stakeholders (customers, management and anyone else considered relevant and interested). As well as inspecting working features produced during the Sprint, you are also after useful feedback that may help guide the work for future sprints. The team should feel empowered to show off the work they've been able to complete over the course of the sprint. It should not feel like they are on trial or defending the work they've done.

Sprint Retrospective

The final event in the Sprint is the Sprint Retrospective. This is when the Scrum Team inspects and reflects upon the process. The meeting covers all aspects of the work, i.e. technology, social aspects, the Scrum process, development practises, product quality, etc. The meeting is basically about what went well (to provide a space for the team to congratulate themselves on a successful sprint, which is important for morale), and agree where there is room for improvements and what experiments might be usefully conducted in order to learn and build a better product.

The ethos of Scrum dictates that no matter how good the Scrum team is, there will always be an opportunity to improve and the Sprint Retrospective gives the team a dedicated time in which to identify, discuss and plan this. The whole Scrum Team should take part. The event should be a collaborative effort, just like the entire Scrum and Agile process.

Artifacts

In the game of Scrum, artifacts represent work or value. The dynamic nature of the projects makes evident the need for each team to maximise transparency of key information. Scrum artifacts are ways to describe the work that must be done, and they can be seen as nuggets of vital information for the scrum team at a particular point in time.
There are three main scrum artifacts, according to the scrum guide. Each artifact contains a commitment to ensure it provides information that enhances transparency and focus against which progress can be measured:

- For the Product Backlog it is the Product Goal.
- For the Sprint Backlog it is the Sprint Goal.
- For the Increment it is the Definition of Done.

These commitments exist to reinforce empiricism and the Scrum values for the Scrum Team and their stakeholders.

Product Backlog

One of the most important things before investing all of the resources to complete a project is to first find out

what features and functionalities the end-user wants from the product that the team is trying to develop.
The tricky part is that the needs and requirements of the end-user never remain the same and the team has to keep track of all of that information to consult it regularly.
The answer to the problem is the Product Backlog.
The product backlog is a list of all the things that must be done to complete the whole project, and it evolves over time. For example, if there is a change in the business environment, marketing conditions or technical demands, the product backlog will reflect those changes. The product backlog is usually made up of three different types of items:

- **User stories**, which are high-level descriptions of a feature, told from the perspective of the end-user of the product.
- **Bugs**, which are problems that arise that the Product Owner wants fixed.
- **Tasks**, which describe the work the scrum team has to complete.

The backlog grows as the product is being built. When changes are added they can include more detail, estimates or a change in priority.

Commitment: Product Goal

The Product Goal describes a future state of the product which can serve as a target for the Scrum Team to plan against. The Product Goal is in the Product Backlog and is the long-term objective for the Scrum Team. The rest of the Product Backlog emerges to define “what” will fulfil the Product Goal.

Refining the Product Backlog

The product owner and the rest of the team are regularly working on refining the product backlog. This can occur at any time.

Refining the product backlog includes activities such as reviewing the user stories of highest priority at the top of the backlog, and asking the product owner questions about them. This includes, if necessary, deleting user stories and then writing new ones. This is followed by reprioritizing the product backlog.

When the items in the product backlog are chosen for the next iterative sprint, they are further refined to be developed during the sprint.

Sprint Backlog

The sprint backlog is the part of the product backlog that the team will be working on in their sprint, a sort of to-do list for the sprint in order to achieve the Sprint Goal.

The sprint backlog is further broken down into tasks for the team to execute. Every item on the sprint backlog needs to get developed, tested and documented. The product owner helps the developers come up with a sprint backlog during their Sprint Planning.

The sprint backlog is often represented as a task board, which is broken up into columns that represent the workflow. If there are items left unfinished by the end of the sprint, they will be added back to the product backlog and may be addressed during the next sprint.

Commitment: Sprint Goal
The Sprint Goal is the single objective for the Sprint. Although the Sprint Goal is a commitment by the Developers, it provides flexibility in terms of the exact work needed to achieve it. The Sprint Goal also creates coherence and focus, encouraging the Scrum Team to work together rather than on separate initiatives.

Refining the Sprint Backlog

The sprint backlog, like the product backlog, is a living document and can be changed by the scrum team. As the Developers work during the Sprint, they keep the Sprint Goal in mind. Work is discussed regularly at the daily scrum and the sprint backlog is modified if the work turns out to be different than expected, the Developers collaborate with the Product Owner to negotiate the scope of the Sprint Backlog within the Sprint without affecting the Sprint Goal.

Product Increment

Finally, we get to the most important Scrum artifact that is the Product Increment. This is the product that is worked on and delivered to the end-user. In order to provide value, the Product Increment must be usable. An increment is basically a working, potentially-deliverable version of the product. Each sprint is potentially creating shippable product increments, and so work cannot be considered part of an Increment unless it meets the Definition of Done.

Commitment: Definition of Done

For scrum teams, it’s really important to have a solid definition of what “done” means. They work in sprints, and need some way of deciding whether a work item is actually finished. The Definition of Done represents the formal definition of quality for all work items. Every Scrum team should set its own.

A good Definition of Done builds a common understanding within the team about quality and completeness, provides a checklist of criteria to check work items against, and ensures that the increment shipped at the end of the spring meets the quality level that the team agreed upon.

I have described the rules to playing the game of Scrum, but these rules leave room for different tactics to play the game. It’s important to understand that Scrum is not a methodology, it is a flexible framework that helps people, teams and organisations generate value through adaptive solutions for complex problems.

In Part III, I describe a collection of good practises that teams can use to serve the purpose of Scrum and reinforce the Scrum values.

Part III

Collection of Practical Tips and Tricks

Chapter 8

Monitoring progress towards a goal

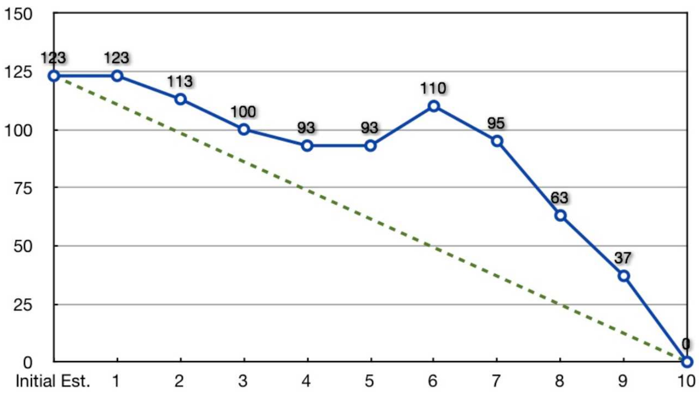
Looking at the rules of Scrum, including the need for transparency, which is crucial to the process of inspection and adaptation, and self-organisation, it is clear why it is important to track progress. Self-correction is difficult to achieve without it.

When using Scrum, we measure progress by what we have delivered and validated, not by how we are proceeding according to the predefined plan or how far we are into a particular phase or stage of development.

Therefore, during the evolution of the Scrum framework towards more lightness, we observe the elimination of burn-down charts as mandatory.

A sprint burn-down chart is a graphical representation that shows the progress the team is making and is a powerful tool for visualising the work remaining. The chart shows:

- The outstanding work (in the unit of measure chosen by the team, i.e. hours, story points, other) on the first vertical axis.
- Time, in days, along the horizontal axis.



There is an ideal work remaining line, which is a straight line connecting the start point to the end point. At the start point, it shows the sum of the estimates for all the tasks that need to be completed. At the end point, the ideal line crosses the x-axis and shows there is no work

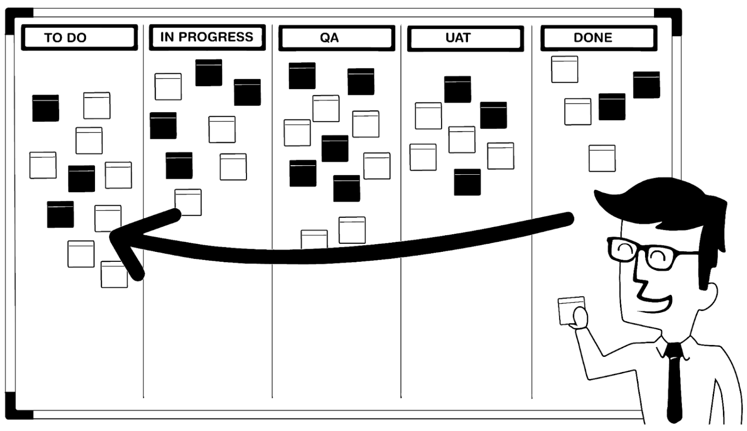
left to be done. This line is based on estimates and therefore not always accurate.
Then there is the actual work remaining line that shows the actual work that remains in the project or iteration. At the beginning the actual work remaining and the ideal work remaining are the same, but as the project or iteration progresses the actual work line will fluctuate above and below the ideal work line. Each day a new point is added to this line until the project or iteration is done to make sure it’s as accurate as possible. If the actual work line is above the ideal work line, it means there is more work left than originally thought. In other words, the project might be behind schedule.
The advantage of having a visual representation of an updated status report on the progress of the project is that it keeps everyone on the same page. Moreover, it keeps everyone involved and encourages the team to deal with issues before they become problems.
But there are also a few limitations. For example, the burn-down doesn’t offer any indication of which work items have been completed. Therefore, a burn-down chart might show progress but not whether the team is working on the right thing. So, these charts are a way to show trends rather than whether the team is delivering the right product backlog items.
Another issue with burn-down charts concerns the accuracy of the ideal work line. Whether the actual work line is above or below the ideal work line depends on the accuracy of the original estimates for the tasks. Therefore, if a team is overestimating requirements, progress will appear on track if not ahead of schedule.

But if they are underestimating requirements, it will appear that they are behind schedule.

Burn-down charts are still a great way to play the game and are suitable in many situations. Yet, they have been turned into a non mandatory practice. At the same time we still need the backlogs to exist and a visualisation of their progress being available, accessible and clear.
The good news is that there are multiple different good practises for that, one may be as simple as a Scrum board.

A Scrum board shows the team's work split across different stages of their workflow. It is made up of columns that teams use to identify categories that fit their workflow. These columns are typically labelled on top as “to do”, “work in progress (WIP)”, “to verify” or “test” and “done”. It’s best to keep the number of columns small. You want a board that is easy to use.
Each work item moves from column to column, usually from left to right, and it is represented by a card that moves as it gets worked on and eventually completed. The Scrum board is a tool, but if you don’t know how to use it then it’s not going to be effective.
Clear communication is the stepping stone to almost any successful venture. When working in a Scrum framework, the basic platform for communication is the Daily Scrum.
In the next chapter I will describe a good practice to keep the attention on the work the team wants to deliver and have effective communication.

Chapter 9
A better stand-up by walking the board



The Daily Scrum (or stand-up) could be the only time in the day when the whole team is assembled with the Product Owner, therefore it is the perfect opportunity to review progress and set goals for the day. However, this ceremony must be fast and efficient to ensure there is plenty of day left to put the planning into action, so it is necessary to establish a set of rules to keep it fast paced without losing the details.
For the stand-up meeting, teams commonly use a format which aims to be both quick and comprehensive:

- The team stands around the task board.
- Each team member, in turn, describes what they achieved yesterday, what they intend to achieve today, and what blockers they may be facing.
- Aim to keep the meeting short and less than 15 minutes.

Using this format, people are often more focused on being busy than actually progressing the work, so let's switch to a model where Work Items attend rather than people.

What is a snapshot of the current project progress? The task board (or Scrum board)! So, instead of hearing each team member in turn, let's hear a summary of each task on the board.

How do we decide what order to go through the tasks on the board? Walk the board from right to left!

The goal of each story is to get it done; so what does this story need to jump to the next column?

For example, a common question could be: “What needs to be done to move this story to QA? Do you need any help?”

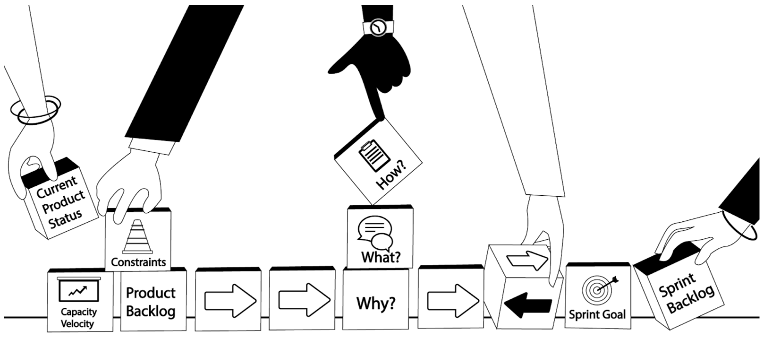
By asking these questions to move the project along, you’re creating a more organised workflow so your team can accomplish its goals in a step-by-step manner.

This approach ensures work that is closest to completion, and has had the most time invested in it (therefore most valuable in terms of effort), is discussed first.

As each work item is covered, the board walker can ask the team members working on the story to report on progress and if there are any issues.
The whole team can join the discussion if they can see any way to help progress.
Walking the board also helps reduce the time it takes for work to get across the board and into production.
Whenever a team member completes a task they can view the board and "walk" it in their head. When they get to a task they can help with, they can jump on it.
For example a developer may notice that a release is ready and that they can arrange to deploy to production immediately rather than picking up a new task from the backlog.
Another example might be a bug found on UAT that is holding up the next release. The developer can start fixing the bug immediately to get UAT back on track rather than picking up new work from the backlog.
Obviously having a board is a prerequisite which not all teams will have. In that case, a person-by-person structure is more appropriate.
Moreover, the difference in the process is subtle if only one person is working on that particular story, however if multiple people are actively working on the story you will see the difference. Changing to this different format will dramatically improve the speed and usefulness of the daily sync. It will shift the focus to the story and promote coordination.

Chapter 10

9 common Sprint Planning anti-patterns



I have already mentioned earlier in the book that the work to be performed in the Sprint is planned at the Sprint Planning. This plan is created by the collaborative work of the entire Scrum Team. The Sprint Planning usually starts with the Product Owner describing the highest priority features to the team. Then, the Developers and the Product Owner adjust the discussed scope of the upcoming Sprint to the available capacity. During this meeting the Scrum Team also crafts a Sprint Goal. The Sprint Goal is an objective that will be met within the Sprint through the implementation of the

Sprint Backlog and it provides guidance to the developers on why it is building the Increment.
Having set the Sprint Goal and selected the Product Backlog Items for the Sprint, the developers usually start by designing the system and the work needed to convert the Product Backlog into a working product Increment.
By the end of the Sprint Planning, the developers should be able to explain to the Product Owner and Scrum Master how they intend to work as a self-organising team to accomplish the Sprint Goal and create the anticipated Increment. After defining the context, let's consider 9 Sprint Planning anti-patterns in detail.

Sprint Planning Anti-Patterns of the Scrum Team

- The Scrum Team has irregular Sprint lengths. This happens when the Sprint length is adapted to the size of the tasks or the Sprint Goal. Instead of changing the Sprint length to accommodate the Sprint Goal, the Scrum Team should invest more effort into sizing tasks in the right way.
- The Scrum Team takes on Product Backlog Items that do not meet the Definition of Ready. It could be that the Scrum Team has not created at all a Definition of Ready that Product Backlog Items need to match before becoming selectable for a Sprint. A simple checklist, that a user story must meet before being accepted into an upcoming iteration, is enough to increase the

quality of user stories and the general way of working as a team.

- The Sprint commitment is not a Scrum Team decision. This could happen when just one of the developers, the tech lead for example, represents the rest of developers in the meeting. According to the Scrum Guide, the whole Scrum Team needs to participate and collaborate, otherwise Scrum values will suffer.

Sprint Planning Anti-Patterns of the Developers

- The Developers are not forecasting future velocity by combining recent velocity with the team's capacity for that Sprint, therefore they may take on too many tasks. The team should instead take into account public holidays, Scrum ceremonies and other meetings, new team members and team members quitting, just to name a few examples. (If you don't have the concept of velocity, you can look at chapter "metrics").
- The Developers are not demanding any capacity to address technical debts and bugs. In case the Product Owner ignores this practice and the developers accept this violation, future product delivery capability will decrease.
- The Developers do not work collaboratively on a plan to deliver on its commitment. In some cases

the senior developer assigns tasks to the individual members who they feel should complete the work. Rather than pushing tasks to individuals, team members should be allowed to pull tasks into their in-progress work, this improves communication and skills transfer.

Sprint Planning Anti-Patterns of the Product Owner

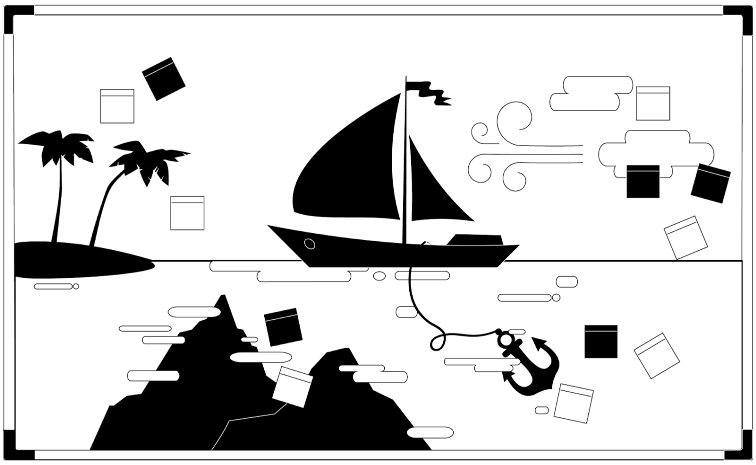
- The Product Owner is absent. In Scrum, the Product Owner is part of the team and responsible for driving the value of the product for the customer through the work of the developers, therefore it is important that the Product Owner actively works with the team throughout the entire Sprint to answer questions and avoid misunderstandings and rework.
- No Sprint goal is defined. If the Sprint backlog appears a random assortment of tasks, it may be a signal of a weak Product Owner who listens too much to stakeholders instead of prioritising the product backlog appropriately. According to the Scrum Guide, the Product Owner is the sole person responsible for managing the Product Backlog and this includes ordering the items in the Product Backlog to best achieve goals and missions. No one can force the developers to work from a different set of requirements.
- The Product Owner tries to include some last-minute Product Backlog Items that are not

ready yet. It is true that only the Product Owner can make such kinds of changes to ensure that the developers are working only on the most valuable tasks at any given time. However, if the Scrum Team is practising Product Backlog refinement sessions regularly, these occurrences should be a rare exception. In case those episodes happen frequently, it indicates that the Product Owner needs help with ordering the Product backlog and team communication.

Anti-patterns in Scrum are habits that are frequently exhibited but overall ineffective or maybe even harmful, so it is important to recognize and eliminate such behaviour.

Chapter 11

Sailboat retrospective



As described in the Scrum Guide, the Sprint Retrospective is the ceremony that occurs after the Sprint Review and prior to the next Sprint Planning.
It is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.
If done well, this ceremony can highlight opportunities for change and generate meaningful process improvements.
If done poorly, it can turn into a blame game, without suggestions to make things better, so it's important to

have a skilled facilitator to assure that retrospectives become effective.

What is great about this ceremony is that it happens right as a Sprint closes, meaning fresh ideas are usually top of mind and able to be teased out by the whole team.
There are several great retrospective techniques in the Agile community, the one I want to describe today is called Sailboat Retrospective as it uses a sailboat as a metaphor for the team.
The idea is that the team is on a sailboat, heading towards their goal while dealing with winds and rocks along the way.
Winds help propel the boat forward, but obstacles such as rocks and stubborn anchors represent the risks that the team might encounter and that could slow them down or even stop them from getting to where they need to be.
I will describe how to facilitate it step by step and you can try with your team when it is appropriate.

First, set the stage. The facilitator draws a large picture of a sailboat floating in the water, with about half of the space above and half below the water/boat, then adds anchors, wind, rocks and an island.
You then explain that you are going to use the sailboat as a visual metaphor for the team. Moreover the facilitator explains that, on a sailboat, there are things that slow it down (anchors), and things that propel it forward (wind).

One variation you could apply is that the facilitator could let the team draw the picture of the boat. It helps to act as an icebreaker and get all of the group participating and on the boat.

Now add goals. The facilitator asks the team to write down what their vision is, what they want to achieve, and what is their goal (this is represented by the island).
Examples can be: the best development environment they can imagine, an effective team who can take pride in their work, filled with happy customers, opportunity for learning and growth.

After that, gather data. The facilitator then asks the team to think of what is anchoring the team down and what is propelling it forward, and to start writing one anchor/wind per sticky note. Sometimes people will be unsure if they should gather a bunch of stickies and then come up, or just bring them up as soon as they have one. I encourage the latter.
As a facilitator, just keep an eye out for the energy in the room - you may need to prompt someone to go ahead and put their items on the board. When the energy starts to die down a bit, give people a fair warning that we'll wrap this part up in a moment.

Now Generate insights. The facilitator asks the team to come up to the board and group sticky notes that seem related somehow. As they do it, the team is asked to read the sticky notes out loud.

This part is a bit of a self-organising activity, it may need a bit of facilitation to make sure that people are getting some value out of the grouping and that one person's opinion isn't dominating when creating the groups.
The key here is engaging group discussions, awareness and consensus on what the sail/anchor is and how it impacts the team.

Finally plan the next steps. Remember to celebrate success and plan action items for current and future obstacles that have been identified during the above brainstorming.
As a decision making method, the facilitator can ask team members to "dot vote" for the group or individual sticky notes they think should be worked on.
Total up the sticky/group with the most dots, and move into some root cause analysis and proposed changes to make!
I typically give everyone three votes, and they are allowed to use them however they please: place all votes on one sticky/group, distribute them around, or even don't use one.

With this chapter I conclude my practical suggestions about how to facilitate three of the main Scrum events and we start exploring new concepts not included in the Scrum Guide. We start with User Stories and Story Points.

Chapter 12

User Stories and Story Points

I have already mentioned User Stories, earlier in the book, and described them as one of the types of items we can have in the Product Backlog. Now we look into them in detail.

In Agile, a User Story is a short, informal, plain language description of what a user wants to do within a software product to gain something they find valuable.

They typically follow a simple template:

As a <type of user>, I want <some goal> so that <some reason>.

With User Stories you put users at the centre of the conversation around what to add to or change in a software product.

We could say they are the expression of the first principle behind the Agile Manifesto: *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
Moreover, with User Stories you give developers the context and the why of what they're creating. Doing so helps them understand how they're providing value for the business and to keep the user/customer top of mind.

User stories are often written on index cards or sticky notes, and arranged on walls or tables to facilitate planning and discussion. As such, they strongly shift the focus from writing about features to discussing them. In fact, these discussions are more important than whatever text is written.
Anyone can write user stories. It is the Product Owner's responsibility to make sure a Product Backlog of Agile User Stories exists, but that doesn't mean that the Product Owner is the one who writes them. Over the course of a good Agile project, you should expect to have user stories written by each team member.
Also, note that who writes a User Story is far less important than who is involved in the discussions of it. While a Product Backlog can be thought of as a replacement for the requirements document of a traditional project, it is important to remember that the written part of an Agile User Story is incomplete until the discussions about that story occur.
During discussions, details can be added to User Stories by adding "conditions of satisfaction". They are Acceptance Criteria that clarify the desired behaviour.

In other words, they are a high-level acceptance test that will be true after the Agile User Story is complete. The concise nature and user-focus of User Stories also helps in separating who deals with what you'll make (customer or product manager) and who deals with how you'll make it (developers).
And finally, because User Stories are small self-sustained units of work, the team will enjoy lots of small wins as they complete one after the other. That's good for building momentum.

During discussions, estimates can be added to User Stories as well.
With traditional estimation techniques we measure the expected effort of a work item in hours, or days. With the Agile approach, we use relative estimates about the effort it might take to complete each item in the backlog, and we do this by using Story Points.
Story Points are a unit of measure for expressing an estimate of the overall effort that will be required to fully implement a Product Backlog item or any other piece of work.
When we estimate with Story Points, we assign a point value to each item. The raw values we assign are unimportant. What matters are the relative values. For example, an item that is assigned two Story Points is expected to be twice the effort as an item estimated as one point.
Instead of assigning 1, 2 and 3, that team could instead have assigned 100, 200 and 300. Or 1 million, 2 million

and 3 million. It is the ratios that matter, not the actual numbers.

It is important to understand that Story Points are a relative abstract measurement of the effort to do something. Effort is time, it is the number of time units it will take to do some work, it is how much time you need to complete the work, but Story Points cannot be time as we used before. Story Point estimation includes the following main components:

- Amount of work: if there is more to do of something, the estimate of effort should be larger.
- Repetition: this component is determined by how familiar the team member is with the feature and how monotonous certain tasks are within development.
- Complexity: This component is determined by how difficult the feature is to develop. If it is more complex, it will take more time. Moreover, there is more chance the developer makes a mistake and has to back up and correct it.
- Risk: The risk of a particular item includes vague demands, or dependence on a third party, for example. If a team is asked to estimate a work item and the stakeholder asking for it is unclear about what will be needed, that uncertainty should be reflected in the estimate. If implementing a feature involves changing a particular piece of old, brittle code that has no

automated tests in place, that risk should be reflected in the estimate too.

By incorporating the aspects above, teams can more accurately plan Sprints, include cushion for uncertainty, better estimate issues, and avoid leaning too heavily on time commitments.
When estimating with Story Points, most teams use a predefined set of values that doesn't include every possible number. For example, teams commonly use powers of two (1, 2, 4, 8, 16,...) with each number doubling the preceding, or a Fibonacci sequence (1, 2, 3, 5, 8, 13, …) with each number the sum of the two preceding numbers.
By intentionally leaving some numbers out of the set of acceptable estimates, teams avoid bogging down in discussions of, for example, 25 versus 26. Indeed, estimating to that level of precision would be extremely difficult and time-consuming, and most likely not even possible.
Mike Cohn, one of the contributors to the Scrum method and expert Scrum trainer, states that teams can estimate successfully with either set of values as each exhibits attributes of Weber's Law.
According to Weber's Law, if we can distinguish a 60% difference in effort between two estimates, we can distinguish that same percentage difference between other estimates. The values in the Fibonacci sequence work well because they roughly correspond to Weber's Law. After the "2" (which is 100% bigger than "1"), each number is about 60% larger than the preceding value.

The concept of Story Points is simple yet difficult to apply. Almost every Scrum team uses them, but they are not part of the official Scrum Guide. Because of this, people have different opinions on how you should use them. Being aware of mistakes that can be made when using Story Points helps to apply them the right way. This is what we are going to explore in the next chapter.
Later in the book, I will also describe the most common estimation techniques using Story Points.

Chapter 13

Common mistakes when using story points

Story Points are widely misunderstood and misused. Based on my experience, the main challenges with estimating in Story Points Scrum teams encounter are:

- Equating story points to time,
- Averaging Story Points,
- Story Pointing unfinished issues again.

I would like to start with some suggestions on how to overcome those challenges, starting from that natural inclination people have to link Story Points to units of time.

We already learnt, in the previous chapter, that Story Points are abstract. Defining a Story Point as 6 hours of work, it is the worst thing you can do with story points as you end up estimating a task differently according to who is the person who develops it, but the number of points assigned shouldn't depend on who did the work. Estimates should speak relatively. The key is to compare items.

I will try to explain the concept with an example. Two developers try to estimate a work item in hours, but one is faster and another one is slower, so they have different opinions. They will never agree on a number as each estimate, 10 or 20 hours, is right only for that individual. What they can do, instead, is to compare that item with another one and they will both agree that this item will take twice or 3 times as long as the other item.

If your team is not doing it, you might be still estimating in days and just calling them points.

The whole point is that team members with different skill sets and abilities can agree on the effort to do something and estimate with a common scale.

Another common mistake I have encountered is that teams average Story Points. When, in an estimating session, half of the team estimates a work item at 3 Story Points and the other half at 5 Story Points, it is easy to resolve the discussion by just putting 4 Story Points as the estimate.

The reason why I would suggest teams not to do this is that it attempts to provide a false sense of accuracy. The point is not to be 100% accurate. The point is to be accurate enough to plan ahead of time. Plus, you may lose a valuable discussion by averaging. Maybe 5 Story Points was a better estimate. And I will add more on this last point in the next chapters when we explore Agile estimation techniques.

Lastly, I would like to add that when moving an unfinished work item to the next Sprint, it is not necessary to re-estimate. The old estimate may not reflect the remaining effort to complete the work, but that is not any problem. As a result of Sprint Planning, the team will know all necessary tasks to complete the work item. So the next Sprint, the team will know how much effort is still necessary to complete the work. And the Story Points for that item will be part of the next velocity.

To help solve these problems, when they occur, there are a few things a SM can do immediately.

The SM can fully reset the team's understanding about story points and use questions to reinforce the relative nature of estimates.

When the team is very close to agreeing on the estimate, the SM can ask them to compare the item they are estimating to another item. Mainly the Scrum Master reminds them that this is a relative estimate, but does this by asking questions rather than just telling.

This is a coaching technique, when you ask a question, you ask a person to engage and work with you to solve the problem.

A technique called triangulating can help a lot as well. When triangulating, you compare one item the team is estimating to 2 other items, ideally one bigger and one smaller.

The SM might still hear the team is estimating while considering hours. The developers may say: *"This might be 12 hours work so 2 story points"*. In that case, the SM can say: *"OK, 2 points means this is very similar to this other item and it is a little smaller than this other one"*, and ask the team to confirm that those two comparisons are right.

Misunderstanding of the meaning of story points will cause people to spend 4 times the amount of time recommended to estimate and they become frustrated.

Following the advice above, there will be fewer disagreements about the size of an item, because people will not think about how long it will take to finish and they can estimate items together even if they have different levels of expertise.

Chapter 14

I.N.V.E.S.T. in good stories

I will start with listing a few common mistakes teams make when working with User Stories and then I will describe the characteristics of a good User Story.
Some examples of what not to do with User Stories are:

- Starting from a requirements document created in a non-Agile way and ending up with contrived stories.
- Explaining the "how" and not the "why". The story can easily become a way of simply describing a feature you want to be implemented rather than a story explaining a user's needs. The result is a User Story that is overly technical

and focused on specifics, reading more like a description of the software than a story.
- Assigning a story without discussing it first. Having a conversation about the story with everyone concerned, before you start implementation, is essential to collaboratively add the details, the acceptance criteria, that'll prevent misunderstanding and rework.

In order to create good User Stories, my suggestion is to start by remembering to "invest" in good user stories. INVEST is an acronym by Bill Wake, back in 2003, to help us remember the characteristics of a good User Story. It encompasses the following concepts:

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

Let's cover each of them with a simple explanation.

Independent. You want user stories to be independent of each other so you can freely move them around your product backlog as priorities shift. When dependencies come into play it may not be possible to implement a valuable story without implementing other much less valuable stories.

Negotiable. A story is not a contract, it is an invitation to a conversation. Therefore, the story captures the essence of what is desired. The ultimate goal is to meet customer needs, not develop something to the letter of the user story. So the actual result needs to be the result of collaborative negotiation between the customer (or customer proxy like the Product Owner) and developers.

Valuable. A good user story has value to the “user” in the user story. It also includes internal value which is useful for things which are normally called “non-functional requirements” or something similar. Without that value, there’s no point in putting any effort into the story. Finally, remember the “so that <value>” clause of the user story. It is there for a reason, it represents the exact value we are trying to deliver by completing the story.

Estimable. A story has to be able to be estimated or sized so it can be properly prioritised. Key factors for estimation are domain knowledge and technical knowledge. When a story cannot be estimated, the team can gain more clarity, it may be necessary to do some research about the story first. Sometimes also splitting the story can help to estimate .You don’t need exact estimates, but when you can estimate a story it’s also more negotiable.

Small. You want the effort to implement a user story to be small. but how small should they be? The answer depends on the team and the methodology being used.

In general small enough to allow the team to complete all work to get the story to a “done” state within the iteration. Plus big stories are harder to estimate, and thus less negotiable.

Testable. Stories should be testable in order to help determine completeness. I like to think that testable acceptance criteria can be written before implementing the story. Thinking this way encourages more collaboration up front and builds quality in. Writing criteria (that can be measured and tested, ideally automated) also makes a team more productive by avoiding rework as a result of misunderstandings.

INVEST encourages good habits which eliminate some of the bigger problems of User Stories like dependencies, being too big, hard to test, etc.
The Definition of Ready, which I will describe in next chapter, is very closely related to what makes a good User Story, and therefore to the INVEST concepts that I have just described.

Chapter 15

Definition of Ready

We are already familiar with the Definition of Done and we appreciate its value in helping teams remain transparent in how they get work completed.
A similar but less commonly used concept is that of a Definition of Ready. A Definition of Ready is used to determine whether work is ready to be started in the first place and whether a user story or product backlog item is ready to be accepted into the next iteration. The goal is to prevent problems before they have a chance to start. The DoR reduces the chances of a Sprint starting where team members immediately shake their heads at work items they do not sufficiently understand.

It is important that the team has a good discussion and agreement of what they need to do with a story in the product backlog before it can be added to a Sprint. Therefore, many teams define a template with a checklist format to start that discussion.
Common items considered for a Definition of Ready are:

- **Action**. Does the team know what they need to do, and can they do it now? Is the item free from external dependencies?
- **Refine**. Has the item been through a process of refinement before sprint planning? Is there a common understanding amongst the team on what the item is and how it will be implemented?
- **Value**. What is the business value of the item? What is the value to the end user? Is the value clear to everyone on the team?
- **Estimate**. Has the item been estimated by the team? And, is the item agreed to be of a size that the team is comfortable can be completed within an iteration?
- **Acceptance Criteria**. Has the item got clear conditions of satisfaction?
- **Demo**. Does the team understand how they might demo the item or discuss it in the sprint review once complete?

As with the Definition of Done, the Definition of Ready should be collaboratively created and agreed by the whole team and be seen as a living document that grows with the team as they mature.

To be clear, the recommended use of a definition of ready is as a guideline for the team to use to determine if stories are ready to be worked on. It should be used as a checklist of things to consider, rather than a stage-gate that has to be fully satisfied for every story.

A stage-gate approach will hamper the team's ability to be agile as it is, after all, another way of describing a waterfall process.

Agile practitioners are usually aware that a user story is meant to represent an ongoing and evolving conversation with stakeholders, and not a fixed specification.

Therefore, I would suggest always considering the context or situation for every story/item rather than blindly applying every aspect of the Definition of Ready to every item the team works on.

Chapter 16

Agile Estimation

In traditional software project management, estimates are based on the question “How long will it take?”

To answer this question, traditional estimation uses methods that follow ‘bottom-up’ estimating. This means that teams inspect the smallest tasks at the bottom, estimate the hours or days required to complete them in order to determine the cost of each feature, and then use this information to develop a schedule for the project.

When you are estimating Agile projects, you will start with a broad estimate for different parts of the project, and then continually refine as more information is available. It is called a top-down approach because we

are not interested in the detail of the tasks, instead we are much more interested in quick-fire estimates of higher level features. Therefore, Agile estimates are based on the question “How big is it?”
Indeed, Agile estimation is about evaluating the effort required to complete each work item listed in the backlog, which, in turn, can give Product Owners new insight and can support decisions about the priority of an item or whether it can be promised to users in 3 months, and maybe even whether to do the work item at all.
The main principles behind agile estimation are:

- allow discussion to derive more information about the items,
- create mutual understanding about the solutions,
- create team commitment on the work agreed upon
- strengthen team relationships by collaborating.

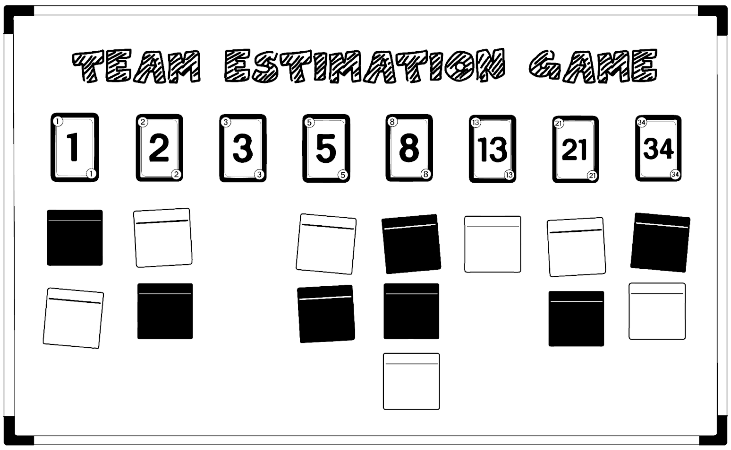
The team members share responsibility and are collectively committed to the work of each iteration, so typically the features are estimated by the entire team. According to some studies on the accuracy of estimation of effort between individual and group, the estimates based on group discussions are more accurate than the individual estimates. Having everybody involved means that the combined expertise of the whole team can be utilised.
After all, an estimation is nothing more than a well educated guess.
When estimating, we use all the knowledge and experience at hand to make a guess about the amount

of effort it is going to take. So, instead of looking at every new work item separately, why not compare it to previously finished work items? It’s easier for humans to relate to similar items than to guess the actual size of things anyway. That’s why in Agile environments, all estimations are done in relative terms rather than absolute terms.

In the next chapter we will explore 3 different techniques using relative estimates.

Chapter 17

Techniques for estimation



In this chapter, I am going to run through three of the most used Agile estimation techniques.

Planning Poker

Planning Poker was invented by James Grenning, an experienced Agile coach and trainer, and later it was popularised by Mountain Goat Software's Mike Cohn in his book Agile Estimating and Planning. It is an agile estimating and planning technique that is based on consensus. Planning poker starts with the team members involved in the estimation process sitting together for the session. Each team member gets a set of numbered cards, usually based on the Fibonacci sequence. Then the Product Owner reads a user story or describes a feature to the team. The team members discuss the feature and ask the Product Owner questions if needed. When the members have finished their discussion, each member selects one poker card to represent the estimate. The cards are then revealed simultaneously.

If all members selected the same value, that becomes the estimate. If not, the team discusses their estimates. The high and low estimators should especially share their reasons. After further discussion, each member reselects a poker card, and all cards are again revealed at the same time. The poker planning process is repeated until consensus is achieved.

Being asked to justify estimates results in estimates that better compensate for missing information. This is important on an agile project because the user stories being estimated are often intentionally vague. Whereas sometimes the team members might decide that agile estimating of a particular item needs to be deferred until additional information can be acquired.

The estimating session (which may be spread over multiple days) is used to create initial estimates useful in scoping or sizing the project. As product backlog items (usually in the form of user stories) will continue to be added throughout the project, most teams will find it helpful to conduct subsequent agile estimating and planning sessions once per iteration.

Teams estimating with Planning Poker consistently report that this technique leads to better estimates because it brings together multiple expert opinions.

The Bucket System

The Bucket System is a way to do estimation of large numbers of items with a small to medium sized group of people.
The game is based on the idea that estimating is hard and it is extremely hard to estimate precisely. It's good then to make estimating easier by not requiring teams to put exact values on every item they estimate.
The group estimates the items by placing them in "buckets" and each bucket can hold estimates up to its size. The facilitator starts the session by creating multiple sections on a table called placement 'buckets', of the chosen scale (a team might choose to estimate using 1, 2, 4, 8 and 16, another team might use the Fibonacci sequence of 1, 2, 3, 5, 8 and 13). The PO comes to the session with a collection of work items that need to be estimated.
The team chooses an item at random from the collection. Read it to the group. The group discusses its relative position on the scale. Once consensus has been reached, they put the item in the appropriate bucket. Then they choose a second item at random and, after discussion and consensus is reached, place it in the appropriate bucket. And they do the same for a third item.

After that, there is a “divide and conquer” phase. All the remaining items are equally allocated to all the participants. Each participant places items in buckets that they believe the items should sit in, without discussion with other participants. If a participant has an item that they truly do not understand, it can be transferred to someone else.

When they can't decide whether a particular product backlog item should be an 8 or a 13 (or a 3 and a 5), part of the answer can be found by thinking about water buckets.

Suppose you have 10 litres of water you need to store. You also have an 8-litre bucket and a 13-litre bucket.

Ten litres of water doesn’t fit in an 8-litre bucket, the water would overflow and spill out. So you would store the water in the 13-litre bucket. The goal is to find a number (a bucket) just large enough to hold the whole story.

When all the items have been “thrown” into the buckets, Everyone quietly reviews the items on the scale. If a participant finds an item that they believe is out of place, they are welcome to bring it to the attention of the group. The group then discusses it until consensus is reached and places it in one of the buckets.

This technique helps team members move away from the feeling that estimates need to be perfect and precise. It is also collaborative as everyone in a group participates roughly equally and the results are not traceable to individuals which encourages group accountability.

The Team Estimation game

This technique was originally created by Steve Bockman and described in the book The Elements of Scrum.
It is faster than the Planning Poker, has the same accuracy and is fun.
With this technique, the team utilises people's natural ability to compare things against each other. Rather than dealing with individual stories, they estimate a set of stories in the aggregate. Silent relative sizing needs to occur.
The facilitator places the team's story cards in a pile on the table, selects the top card from this pile and places it in the middle of the playing surface (or sticks it on a wall).
After placing the first card, each team member plays in turn doing one of the following:

- Play the top card from the pile as described above and place it to the right if it requires more effort to be completed than the other(s), to the left if it requires less effort, or under another card if the user stories are of about the same effort.
- Move a card already on the playing surface, declaring disagreement about its relative size.
- Pass.

During play, the players may talk about why cards are being moved or about what they think about the size of the stories. Other team members may ask clarifying questions, but they must not express their own opinions during another player's turn.

The goal is to get clarification and not to get too hung up on the exact sizes of the estimates.

The first part of the play ends when there are no more cards in the pile and there is a general consensus on the ranking of the cards.

In preparation for round two of the game, the facilitator produces a deck of Fibonacci cards. Each card in this deck has one of the Fibonacci numbers on it. The Fibonacci sequence reflects the general principle that risk increases geometrically in proportion to complexity.

The team now works together to assign points to each stack to indicate the size (level of effort) of stories that are in that stack.

Each team member plays in turn doing one of the following:

- Pick up a card with a number on it and place it on one of the columns.
- Use their turn to change a number assignment made by another team member.
- Pass.

This continues until a consensus is reached. When everyone feels confident enough in the sizes to pass on their turn, round two is over.

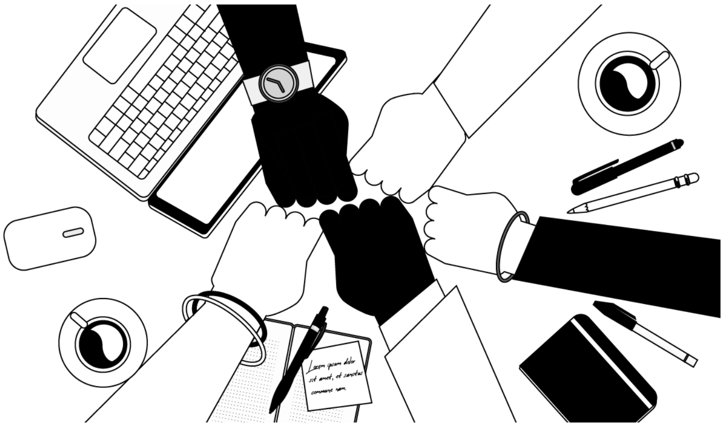
As a result of the Team Estimation Game, the team gets an estimated package of stories.

When playing it on your team, note that you don't have to start with a "1" as your smallest story size.

If a player thinks there may be future stories that will be significantly smaller than the smallest story that is currently on the playing surface (or on the wall),
they may opt to start with the “2” or “3” above the first story instead of the one.
This leaves room for future stories to be sized smaller than the smallest story in the current set.

When estimating Agile projects, having everybody involved means that the combined expertise of the whole team can be utilised and they produce better estimates.
But, it might happen that the team gets hung up trying to create perfect estimates, which is not ideal. In the next chapter I will describe how to avoid this situation.

Chapter 18
4 practical suggestions for better estimates



When teams estimate well, they can do it quickly and accurately. Moreover, the organisation can comfortably base decisions on those estimates.
Most teams I know want to do good work and have high standards, but it is important to avoid teams getting stuck in the pursuit of perfect estimates as this can damage relations with stakeholders and compromise quality.
When there is an urgency to deliver results and deliver them on-time, often the stakeholders take estimates as

guarantees and hold team members to every estimate they provide. If the team takes longer than what they estimated, the team gets in trouble.

As a consequence, teams start padding estimates and that exacerbates the lack of trust between stakeholders and teams.

Sometimes also higher estimates are not enough and the team finishes late anyway as, knowing that the estimates are too high, they wait too long to start the work and they fail to finish on-time.

As a consequence of this, teams don’t want to estimate at all, or they become obsessed with perfect estimates, as it seems the only way to be right, and discuss each item to a tedious level of detail.

All of this is caused by miscommunication about the meaning of an estimate.

Fortunately we can solve these problems with a shared understanding about estimates, their accuracy, and what they are good for.

I am going to describe four techniques that every team can try and that will help to uncover hidden assumptions and encourage communication both for people who want the answers and for those responsible for creating them.

First thing to do is agreeing with the team which type of estimates they want to go for and share that choice with the stakeholders.

Usually, teams estimating based on the worst case will give higher estimates (and might be right 99% times), on the contrary, teams estimating based on the best case

will give lower estimates (and might be right only 10% of times), and finally with a median estimate, roughly half time they will finish faster than estimated and half time it will take longer.
Without a shared understanding on the type of estimate being given, not only the team will struggle to agree on an estimate and estimating takes longer than it should, but also it could happen that stakeholders think they are given the worst case estimate, so they think the estimates could be wrong in 1% of cases, instead the team is providing a median estimate which in 50% of cases is expected to take longer.
The Scrum Master can and should help the team to agree on the type of estimate, then the SM should talk to stakeholders and tell them first which type of estimates the team chose and then remind them that estimates are just guesses and not guarantees.

Even if stakeholders understand that the team is providing an estimate that might be wrong 50% of times they might still ask for some level of guarantee. In that case a second step for the teams is to add some margin of safety to their project estimate.
If a project is made up of 10 items, with the median estimate 5 might take longer and 5 might take less, my suggestion for the team is don’t just sum up the 10 estimates and tell the stakeholders the delivery date, instead sum the 10 estimates and add some safety margin and communicate that as the plan.

The third challenge is that many teams have a tendency to underestimate and they need techniques to balance underestimating with overestimating. If that is the case, the technique I suggest is to try the bucket estimation with the Fibonacci sequence which reframes the question from ”what estimate to give to the item”, into “which bucket does this item belong in”.

Each bucket can hold estimates up to its size. This way if the team thinks an item is 6 story points, that doesn’t fit in the 5 bucket, it would make the 5 point bucket overflow, so the item has to go into the 8 bucket. Using this technique, the team will have a slight tendency to round up and they are more likely to balance under and over estimates.

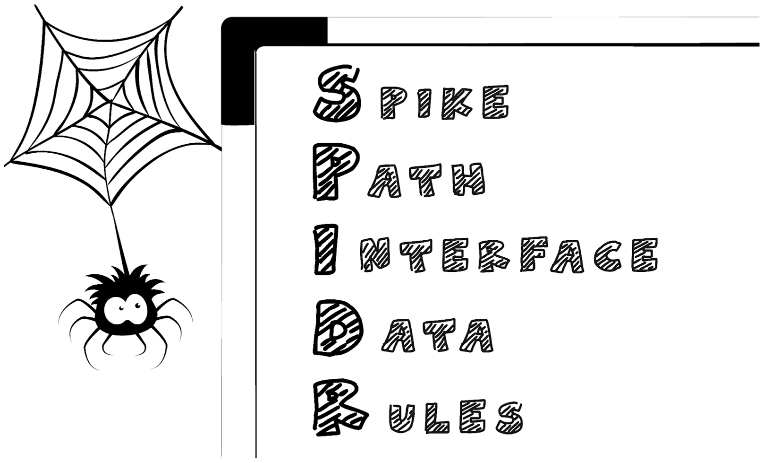
My last suggestion is related to the scale to use when estimating. Product backlog items are usually big and the team struggles to choose between two estimates that are too close to one another. They cannot tell the difference between 42 or 43 story points even if they are good at estimates. A good choice could be the Fibonacci series, as in this sequence, for numbers above 3, each number is roughly ⅔ larger than the previous. And it might be a big enough difference to be discernible.

Moreover, as the precision decreases with larger features, teams might find it useful to break the feature into smaller stories or tasks.

In the next chapter we will explore a few simple techniques to split big stories.

Chapter 19

S.P.I.D.R.



We already know by now that a User Story describes a software feature from the end-user perspective. Indeed, in a User Story, you try to describe who the story is for, what is going to be delivered, and why it matters in the first place.
Moreover, stories can and should be broken down into smaller chunks by the development team when it makes sense, the reason is that not only smaller stories are easier to understand and therefore easier to estimate, but also their smaller size makes them inherently less risky for a team to take on.

However, when teams first start splitting stories, they’re often tempted to split them horizontally rather than vertically.

This concept was first espoused by Bill Wake, in 2003. He says: *A story needs to be valuable. We don't care about value to just anybody; it needs to be valuable to the customer. This is especially an issue when splitting stories. Think of a whole story as a multi-layer cake, e.g., a network layer, a persistence layer, a logic layer, and a presentation layer.*
When we split a story, we're serving up only part of that cake. We want to give the customer the essence of the whole cake, and the best way is to slice vertically through the layers. Developers often have an inclination to work on only one layer at a time (and get it "right"); but a full database layer (for example) has little value to the customer if there's no presentation layer.

It is true that many teams struggle to split large user stories into smaller stories in a useful way, but, fortunately, story splitting is a skill that can be learned in a relatively short time.
There are now several techniques and tools that can be used to split user stories. I would like to introduce you to the simple and fast SPIDR method from Mike Cohn (Agile coach and co-founder of the Scrum Alliance).
He summarises five techniques with which almost every large user story can be divided.

SPIKE

If the subject on a User Story is not clear or too complex and needs extra research, it’s time for a Spike. <br />
A goal of a Spike can be gaining more information on the topic by doing research or it can be gaining experience on the subject by building a prototype, typically used for the evaluation and feasibility of new technologies.
With the help of such newly acquired knowledge, a story can then be better understood and possibly split more easily with other SPIDR methods.

PATH

If there are several possible alternative paths (or scenarios) in a User Story, one option is to create separate User Stories for some of the paths.
For example, let's take a User Story in which the shop owner wants to send the customer his new password by text message or email in case the customer forgets his password.
You can split the User Story into two separate paths to recover the customer's password.
It is not absolutely necessary to write a story for each individual path, just where it makes sense.

INTERFACE

Interfaces in this context can be different devices, for example if you have a website and an app.
It might be useful to split a User Story into separate stories for the different interfaces, you might change the website first, because of the probably larger target group, and after that the app.

DATA

If a User Story needs more than one source of data for all requested functionality, you can often start with only one source of data.
For example, if you are building a website intended to attract tourists to a particular city, the first story could include information about the different museums.
After that you can add hotels, public transport options, outdoor activities, etc.

RULES

Business rules or technological standards can be another splitting factor.
Take the example of online purchase of items. There are often constraints that are for example based on business requirements of the respective shop, such as an online purchase limit of a maximum of 3 items per

basket (similarly to dry pasta and toilet rolls limit during coronavirus pandemic).
With the first User Story it would be conceivable that the development team omit this restriction, allowing every user to buy as many items as they wish, not considering the purchase limit. The restriction could then be added in a second iteration step.
Incremental delivery such as this means that initial stories are not immediately implemented completely, but instead are delivered in several smaller steps.
Sometimes it makes sense to neglect technical specifications or business rules, if by doing so you are able to discuss the early results with the stakeholders.

User Story splitting is not always easy at the beginning, but practice makes it perfect.

Chapter 20
The Shu-Ha-Ri model and Scrum maturity

Shu Ha Ri is a Japanese martial art concept that is used to describe the progression of training or learning. It is roughly translated to “first learn, then detach, and finally, transcend.”
In recent years, it has been abstracted and applied to the cycle of learning in general.
I got familiar with this concept while reading the book “Coaching Agile Teams” by Lyssa Adkins.
In this chapter, I want to present this concept applied to Scrum teams and use it as a tool to help teams to identify in which stage they are.

The team is at the Shu phase when they are new to Scrum or new to each other. At this stage it is not rare for the team to change or drop Agile practises and lose the intention behind them. In some cases they might mash up Scrum with something else so that their practises are not even clear to them. Therefore they need the Scrum Master to guide them to practice at Shu.
In this beginning stage, the teams follow the teachings precisely. They concentrate on how to do the task, without worrying too much about the underlying theory. When a team is first learning something, a variety of ideas is not usually the most helpful place to start. If there are multiple variations on how to do a practice, they concentrate on just the one way their master teaches them.

The team is at the Ha phase when they live by ideals in the Agile Manifesto. In all they do, they stand on the side of individuals and interactions, working software, customer collaboration, and responding to change. They have the basic practises working well and start producing new insights that let them improve each Sprint. They pause to consider the ramifications before they alter, drop, or add an Agile practice. At this point, with the basic practice working, teams start to learn the underlying principles and theory behind the technique. Once teams get the basics down, they move on to experimenting and looking to integrate new thoughts or ideas. They keep learning from the Scrum Master and integrate that learning into their practice. They need the

Scrum Master to coach them to a deeper expression of Agile.

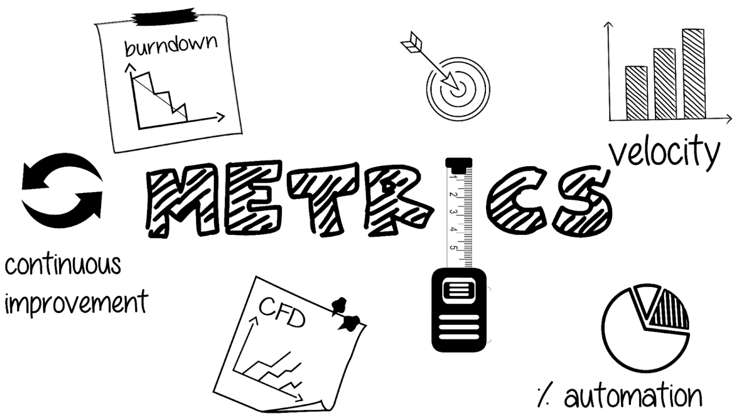
The team is at the Ri phase when they altered their practice of Scrum and did so consciously, keeping the values and the principles of Agile alive, their practice of Agile leads to progressively better and faster delivery and higher satisfaction. They took in the skills and mind-sets necessary to be truly self-monitoring and self-correcting and they are now not learning from others, but from their own practice. They create their own approaches and adapt what they have learned to their own particular circumstances. Eventually they will move beyond the specific practises and evolve their own way of doing things. At this stage, the Scrum Master needs to let them go.

All the different levels require different coaching styles. The Scrum Master should understand in which stage the team is to help them to perform in a more efficient way.
Below I summarise what the various coaching styles should be.
At Shu stage the SM must teach the rules. The teams that are at this level have a basic knowledge of Agile values/principles/practises. They need to have someone to guide them.
At the Ha stage, teams can come up with their solutions, they just need a coach to help them find different ways to achieve what they need. Teams have a good understanding of agile values/principles/practises, they start to interiorise them from their past experiences. The

role of the SM works as a coach while they start to understand how they can use different approaches to achieve the same result.
In the Ri stage, the team has fully internalised the values, principles, and practises. Everything runs quite well, the role of the SM works as an advisor.

One important thing I would like to add is that each successive stage contains the others. For example, if a team is in “Ha”, but you want to introduce a new practice or idea, remember to use a teaching approach because the team is new to that practice so that they will be in Shu for that idea. This is important because most probably you will be changing coaching styles depending on the practice or idea that you want to feed into the team.

Chapter 21
Metrics



Many teams have an uneasy relationship with metrics. In the majority of cases it happens because they have the misfortune of being on a project where stats are used for comparison across teams and, in consequence, set one team against another. In other cases metrics are used to measure engineering output and pretend this translates into a teams' productivity.
Many agilists, in an effort to defy this trend, affirm that measurements should not be used at all, and that only the production of software itself should be considered the criterion for success.

As the legendary engineer W. Edwards Deming wrote "without data, you're just another person with an opinion", I think, on a project where no data of any kind is tracked, it is tough to optimise something you don't measure and it is hard to tell whether you are on track for release.

When talking about metrics, I think the first question we should ask is: What are Agile Metrics for?
Agile Metrics are meant to help teams better analyse and understand their workflow, discover shortcomings, and improve the development process, work quality, product being developed, predictability and health of the team.

The second question I would ask is: Who are Agile Metrics for?
Agile metrics are primarily a feedback loop for the team. Teams should regularly review agile metrics to tune and adjust their behaviour, they need to understand how they are doing and use the data to become more productive. Most of these metrics are supposed to be starting points for conversations, they have meaning because of the context around them, and the only people who have experienced and will understand that context are the people in the team.

Now that we've covered the basics, it's time to get into 10 metrics I find useful for Agile teams.

I will start with metrics that relate to the pace of delivery, value and quality delivered, followed by metrics used to measure satisfaction.

Velocity. It simply measures the amount of work that a team completes during a set amount of time, usually a Sprint or iteration. It is primarily used to encourage consistency and prevent burnout on the team.
When calculating Velocity, remember that something half done, is not done at all.
It is crucial to follow how your team’s Velocity changes over time, Velocity will most likely increase as teams eliminate bottlenecks, learn to work and communicate better, and inspect and adapt to their process. However, don’t expect constant linear improvement.
As you track it, look also for any erratic moments. If these become the norm rather than the exception, help the team investigate the root cause, possibly during a retrospective.
Moreover, Velocity can be extrapolated to longer time frames for release/quarterly planning as we will see in the next chapter.
Be sure to focus on being accurate and not overly precise, and be careful not to compare Velocity across teams because story points and definition of done can vary between teams.
This is one of the most misunderstood metrics. A team’s Velocity is unique to them, it simply cannot be compared to another team.

Cycle Time. It measures how long it takes something to get done from start to finish. It is a very simple metric that describes how work is flowing into and through a system.

Cycle Time tells you how quickly your team can process a piece of work. You want it to be consistent and short, regardless of the kind of work being done.

If your work items are User Stories and your sprints are two weeks, You want your Cycle Time to be under two weeks, well under.

When Cycle Times are longer than a Sprint, teams are not completing work they committed to and this is not a good thing.

Moreover, consistent Cycle Time means you can accurately predict when you will be able to deliver individual pieces of work, whether you are using continuous flow or sprint-like timeboxes.

Cycle Time also gets you immediate feedback, as you can see the results of any changes right away.

Indeed, when you adjust the system, Cycle Time will either increase or decrease right away, so in a short time you know if the experiment succeeded or failed.

WIP (Work In Progress). It measures the number of tasks that the team is currently working on.

Lots of work in progress could mean the team is switching focus constantly and maybe working as a group of individuals rather than working as a team.

It is really important to assess how, as a team, you could collaborate or help each other in moving a user-story out of the window.

In such cases, you might find useful limiting work in progress that is simply restricting the maximum amount of work items in the different stages of the workflow.
Teams should agree on WIP limits and focus on helping members of the team who get blocked rather than starting new tasks while one of their team struggles.
The “Stop starting, start finishing” philosophy is not limited to Lean and Kanban world, it is applicable in Scrum too.
Moreover, WIP limits ensure that your team will keep an optimal work pace without exceeding its work capacity.

Throughput. It represents the number of work items delivered in a given period of time, it could be measured monthly, quarterly, per release, iteration, and so on.
The value in this metric is that it can be used to track the consistency of delivery from a team and organisational perspective.
It can also be used to determine how much software can be delivered for a specific time frame.
Indeed, empirical analysis of historical data can be used to forecast delivery performance. The more historical data available, the more accurate the projections are likely to be.
It is important to remember that the Throughput strictly counts work items, it does not take into account the fact that they might be of different sizes.

Sprint Burndown Chart. Before starting a Sprint, a team forecasts how many Story Points they can complete in its course.

The Sprint Burndown Chart visualises how many Story Points have been completed during the Sprint and how many remain, and helps forecast if the Sprint scope will be completed on time.
This metric allows you to track closely the progress of a Sprint and it shows how agile your team really is.

Committed vs. Completed. It is the percentage of story points completed during the Sprint divided by those committed at the planning meeting.
It provides insight about the team's ability to predict its capabilities.
If the percentage is higher than 100%, it might mean that the team ran out of work during the Sprint and brought forward items from the backlog, the metric indicates the team needs to focus on better planning.
If, instead, it is much less than 85%, still the team has to improve their forecasts with better grooming sessions, or splitting stories better to improve estimations and using spikes for research.
There are several techniques to split stories, I have already described some of them earlier in the book.
One common mistake you should avoid is to use this metric to compare teams or to evaluate their performance, this is not the purpose of it.

Release Frequency. The first principle of the Agile Manifesto is: "Our highest priority is to satisfy the customer through early and continuous delivery of working software".

Therefore, one of the aims of agile teams is to deliver in short cycles and this metric can help understand whether teams are really building potentially shippable software.
If teams are failing to deliver code to production, then they are failing to meet their primary purpose.
This metric also gives qualitative insight into the release process, if it is easy for the team or it requires heroics, and how solid the environments are.

Automated Test Coverage. It captures how much of your codebase is covered by automated tests.
Test automation brings speed, efficiency and reliability to testing and it is a prerequisite to continuous delivery in fast-paced agile development environments, as it contributes to reducing the amount of time required to release working software.
This metric can have a crucial role in measuring the effectiveness and accuracy of your team's automated testing efforts as it can help to reveal parts of the software that do not have sufficient test coverage.

Escaped Defects. It measures the number of defects found in the product once it has been delivered to the user.
You can capture this metric per unit of time (per Sprint, or release) and your rate is a constant feedback loop to how your team is doing.
It is also a measure of deployed software quality and can provide insight into what went wrong with development or testing in a specific part of the project,

and you can use the Escaped Defects rate to know when you need to slow down or improve testing.
Spikes or a steady climb in this metric would encourage further analysis as with a high escaped defect rate, even the most awesome product is going to have a lot of unsatisfied customers.

Team Morale. My final metric, not strictly related to the agile process, it’s a look at the team's well-being.
The ultimate goal of this indicator is to ensure the team is successful. There is a direct relationship between morale and productivity, happy teams create better work, which will deliver more satisfied customers.
The difficult step in managing morale is measuring it.
It is true that there are some signals not difficult to spot such as an increasing turnover rate. When more people than usual leave your team, it may be a sign of dissatisfaction.
However, you can also talk regularly with your team, they can often tell you how they feel. You could have a quick survey during a retrospective with the team writing their happiness scores, using a 1-5 scale or smiley face indicators.
Moreover, you can track these numbers from Sprint to Sprint to see the trends.

Whether you use all of the measurements discussed or only a subset, it is important that any solution consider the audience for the data. Make sure you use the metrics as a support to improve the process and not to punish the teams. Always be sure to fully understand

and communicate accurately what the metrics are saying, and follow where the data lead.
Comparing the evolution for each indicator over time and connecting them to the business metrics will help you empower the teams and achieve the expected outcome.

Chapter 22

Release Planning

In Agile environments, planning takes place at multiple levels. Formally Scrum defines only the Sprint Planning event and the “daily planning” which happens with the Daily Scrum. However, most organisations benefit from release planning which is a longer term planning. It is a high-level planning of multiple Sprints (three to six iterations in most cases).

One of the most important steps in the release planning process is defining the vision for your product. The vision will guide subsequent decisions on which features to prioritise, where to focus effort and resources, and how to adapt if the project requires adjustment during development. All the above aspects contribute to maximising the chances of achieving the desired outcome.

As a first step, an organisation must determine the proper cadence for releasing features to its customers. Some organisations decide to release every Sprint, while others combine multiple Sprints into one release and others release just after the completion of each feature, this practice is called Continuous Deployment or Continuous Delivery.
Whether the organisation intends to deploy every Sprint, every few Sprints, or continuously, they find some amount of longer-term, higher-level planning to be useful.
The process involves determining the work to be done, understanding scope, date and budget constraints, monitoring progress from Sprint to Sprint and making adjustments as required. It involves the entire Scrum team and the stakeholders. At some point, the involvement of all these people is necessary to maintain a good balance between value and quality.

Before starting the release planning and creating a release plan, the following must be available:

- A product backlog, which has been prioritised and estimated.
- How much work the Scrum team can complete per iteration (Velocity).

- The agreed goals for the scope, schedule and resources.

As I have already mentioned, the constraints of scope, date, and budget are important variables that affect how we will achieve our goal. These constraints are either fixed or flexible. I will describe two realistic options to create our release plan, depending whether the project is feature-driven or date-driven.

Feature-driven

A feature-driven model is appropriate where the scope is more important than the date. In this model, when we run out of time and we haven't completed all of the features, we extend the date to ensure that we get everything required.

If we are doing a feature-driven release, we must know what the features are at the start of the release.

This is usually true when we are building a simple or familiar product. In case instead we are developing innovative products, many features will emerge and evolve during the development effort. We certainly have some idea of the desired features up front, so we will use them in our initial release planning. However, we must be prepared to continuously revise our release plan as our understanding of the required features changes.

If an emergent must-have feature appears, we will simply add it to the scope of the release and push out the release date.
During feature-driven planning, we need to calculate the number of Sprints required to deliver the fixed set of features. The number of Sprints needed to complete the release is determined by using the sum of all features divided by the expected Velocity of the Scrum Team.

Date-driven

Many people consider this to be the approach most closely aligned with Scrum principles. We can fix both the date and the budget, but the scope must be flexible.
The Scrum principle of creating the highest-priority features first should alleviate any pain of having to drop features. When we run out of time on a date-driven release, whatever has not yet been built should be of lower value than what has already been built, therefore it is much easier to make a decision to ship if the features that are missing are low value.

Assuming that the length of each Sprint is the same throughout this release, which is the normal case with Scrum, to find out how much work can be completed within a given time-frame, the Velocity of the Scrum Team is multiplied by the number of Sprints.

The outcome of the Release Planning is a release plan that communicates when we will finish the product, what features we will get at the end, and how much will be the cost. Of course, assuming that we proceed with the release, we must revisit our release plan every Sprint to update it based on our current knowledge.

In order to have the best release plan possible, I would suggest to follow these five tips:

Keep the Focus on Goals. There's a lot to take into account when developing the release plan. You can easily get lost in the weeds. You want to keep your eyes on the priorities: goals, benefits and results. Features contribute to a goal. Focus on the goal and the feature will follow.

Identify Task Dependencies. Dependencies are tasks and User Stories in the Product Backlog that cannot start or end until another starts or ends. If you're not aware of the dependent User Stories in your release plan, you are going to suffer delays and block your team. By identifying these User Stories beforehand and making sure you stay aware of them, you're going to keep the Scrum Team working without unnecessary interruption.

Release Often. The mandate of any release planning is to release your product to customers. Only then will you be able to determine if the User Stories you released were of value to them. Therefore, release often. Smaller releases are easier to digest for customers than having a couple of big ones per year.

Release Done Work. It might sound obvious, but often work in the Product Backlog is moved forward through production without being completed. These incomplete User Stories can involve a lot of time and money to fix. That will take away from your main goal, which is delivering value to your customers. Have a Definition of Done for your User Stories and product deliverables and stick to it.

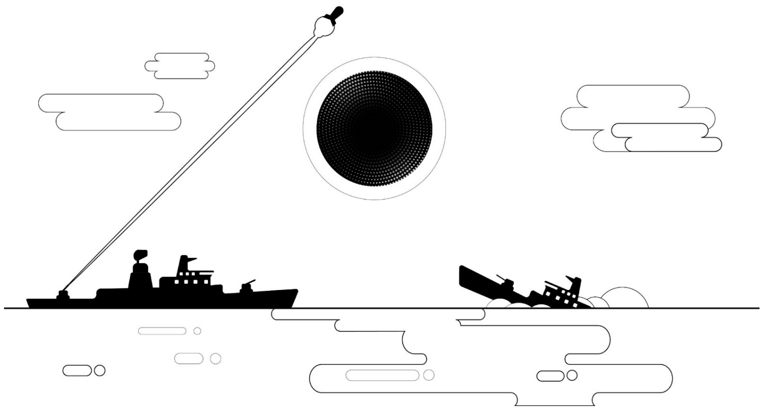
Continuously Improve. Good enough isn't good enough. There will always be room for improvement, but like release planning these improvements should be applied incrementally. Give them time to prove themselves.

Chapter 23

Agile games

Most people think that games and business work doesn't seem like a natural pairing, I think instead games are serious instruments to explain key principles in an interactive way. Playing games for learning is especially helpful for concepts which can be easily misunderstood because they are different from existing ways of working.

Battleship game



The first Agile game I would like to introduce is called "Battleship". This game is generally used to experience the difference between the Waterfall design process and the Agile process. It introduces people to iterative development and explains the concepts behind it. The idea is to get people to understand that up-front, large plans are just not recommended. I am going to describe how to play the game.

Description of the game

Split the group into two teams: the Grand Plan team (Team A) and Frequent Feedback team (Team B).
The game is run in a single round during which each team will take 30 shots.
To score the game, each team gets 1 point for a hit and an additional 2 points for each ship sunk.
Each team starts by laying out their ships on the grid.
Team A is told to place all their planned attacks up front.
Team B plans and takes one shot at a time.
Once Team A has determined all 30 shots, they inform Team B of their shots. After Team A has shared all of their shots, Team B lets Team A know which shots were hits and if any ships were sunk. After Team A completes their turn, Team B responds. Team B takes a single shot, gets feedback from Team A (Miss / Hit / Sunk), and decides their next shot. They do this 30 times.
For the basic game, the win is usually quite clear with Team A scoring much less than Team B. The learning point is that responsiveness is vital to success, the

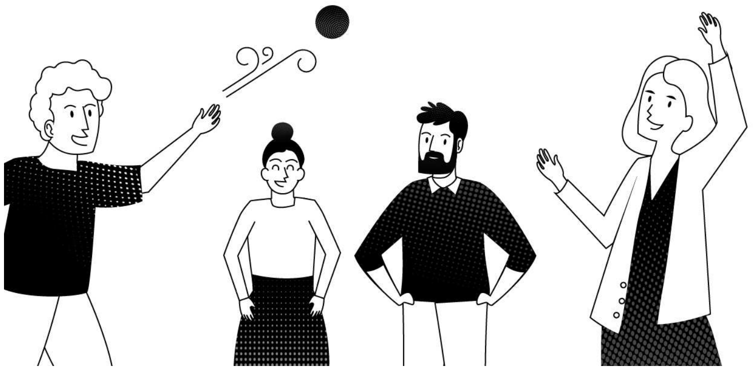
tighter we make the learning feedback loop, the more successful we can be.

Variation

In this variation, split the group into two teams: the Small Releases team (Team A) and Continuous Delivery team (Team B).
The game is played in 4 rounds of 8 shots each.
To score this variation of the game, each team gets a single point for a hit, an additional 2 points for each ship sunk, and 2 bonus points for each ship discovered and sunk in a single round.
Each team starts by laying out their ships on the grid.
Team A still pre-plans the 8 shots per round and executes them all at once, receiving feedback after each round.
Team B still takes a single shot at a time for 8 shots, getting immediate feedback.
Mark shots with the round number in which they were taken. This will not only help keep track of the activity per round, but will help with the scoring.
In the variation game, the difference in score is smaller, but still present. The learning point is that too fast a feedback loop is not always better than a small batch; having a pause to consider what we're going to focus on next and designing a short term strategy for it ensures that there actually is a strategy.

Agile games build a safe-to-fail environment and let participants get a first hand experience of new concepts. For example, suppose you need to introduce a new team to the Agile mindset and suppose managers will participate too. Do not start it by giving a standard presentation with slides about agile values and principles, self-organising teams and so on, use a game instead. More precisely, use a physical agile game: the Ball Point Game (invented by Boris Gloger).

The Ball Point Game



The second game I want to describe simulates an agile production process. It is basically an analogue of the Scrum process. The team will self-organise and form a process based on the rules provided. The objective is to pass as many balls as possible in the given timeboxes through the team by following certain rules.

In order to play the Ball Point Game, you'll need a large open space with enough room for everyone to stand. You'll also need a large number of brightly colored ping pong balls (about 60 for a group of 50 people) and you may want a whiteboard to do the debriefing and a stopwatch.

The Rules

The rules are quite simple and the more people you have, the more exciting it can be. You play the game best with more than 6 participants and it would be an excellent game for larger groups up to 50 people.

- You are one big team.
- Each ball must be touched at least once by every team member.
- Each ball must have air-time, in other words, it must not be passed directly from hand to hand.
- You cannot pass the ball to the person immediately to your left or right.
- Each ball must return to the same person who introduced it into the system. For each ball that does, the team scores 1 point.
- If you drop a ball, you cannot pick it up.
- There will be a penalty (points deducted) if you break any of the rules.
- If you've played this game before, please participate silently so you don't spoil it for others.

In the game, there are two roles, only: the team (included PO) and the facilitator.
Here's how to play the game:

- Allow the team to prepare and to determine how they will organise themselves. (2 min)
- Ask the team for an estimate how many balls they can pass through the system at the first run (each run is 2 minutes).
- Run the first iteration. (2 min) – at the end check if someone counted the balls
- Allow the team to discuss how to improve the process. (1 min)
- Repeat for five iterations (recording the estimate, actual and changes each time).

Hints for the facilitator: after a couple of iterations, during the learning minute, you might want to give the team clues, such as eliminate waste, maximise resources. Later you might want to hint that they should use both hands, and later still that they could cup their hands together to drop fewer balls (less waste). Try not to make the hints too obvious too early in the game.

At the end of the exercise, debrief for five to ten minutes. There are a number of basic agile principles and values that are worth talking about.

- Trust. See how to build trust in the team and in individuals.
- Self-organisation. See how the team makes decisions to work best, without control from outside managers.
- Inspect & Adapt. See how the team steps back and reflects in retrospectives on a regular basis to improve their own work.
- Timeboxed, incremental delivery. See how the team estimates, plans, and improves quality in an iterative manner.
- Agile events. Getting acquainted with "Sprint", "Retrospective", "Planning", "Estimation".
- Learning. See how fast the team succeeds.

All in all, the Ball Point Game is a great ice-breaker, good fun and very thought-provoking.

Before closing this part of the book, I would like to remind you that, because practises come and go, it is important for agile teams to understand the principles that lead to the creation of specific practises.
The same is true of our methods, methodologies, frameworks, or whatever we choose to call collections of practises such as Scrum, XP, Safe, and LSS.
Rather than devotion to practises or frameworks, the priority is keeping the various principles of agile in mind.

Part IV

Scrum played at scale

Chapter 24

Main challenges when scaling Agile

The success of Agile methods for small, co-located teams has inspired use in new domains. The number of companies that apply Agile practises to large-scale projects is increasing, but this raises new challenges as well.

Fundamental assumptions in Agile development are severely challenged when using these practises in large-scale projects. The good news is that in addition to the basic rules to play the game of Scrum, described earlier in the book, there are some tactics that allow Scrum, and more in general Agile, to be played at a larger scale.

Adapting to these scaling Agile models is not done overnight and is surely not easy.
During the journey of adapting to scaling Agile models, the most common implementation challenges organisations face are listed below.

Forming a New Mindset

A major prerequisite when adapting to any scaling Agile model is having a Lean Agile mindset. It is not just one person who needs to have it, but it has to slowly and steadily be incorporated with everyone in the entire organisation. Leaders need to be taught the concept of servant leadership where the priorities of the teams are kept first. Teams need to be taught on how to take ownership of their work and should be empowered to make their own decisions.

Going through a Culture Shift

When working with the scaled Agile model, you need to dismantle the age-old concept of top-down hierarchies and work within your small circle. Agile propels working in collaboration with multiple teams in synchronisation believing that project demands and requirements keep changing. To bring fuel to the rage of cultural shift, Agile leaders have to play a significant role. The leaders must embrace the idea of failing fast and learning from the change. Instead of achieving milestone over the

milestone, leaders should encourage and prioritise value and flow above all. They should adjust their management style.
Transforming to this culture takes time and is certainly not an easy task. Many people oppose the idea of a more self empowered team. Individuals are accustomed to the traditional ways of command style management and refuse to give up that mentality. Accepting change is not easy, but once you take a leap of faith, the future is golden.

Experience a Work management shift

Traditional work and project management approaches start with a fixed scope and estimate the time and resources (people) necessary to achieve that scope. This idea assumes that, by defining requirements up front, organisations can reduce risk and increase success. The Lean-Agile model, on the other hand, flips that paradigm. Time and resources become more fixed through established iteration windows while scope becomes more fluid, influenced by constant learning and change. Teams experiment and receive feedback quickly and adjust the scope accordingly so that organisations can adapt in an agile way. When scaling Agile, organisation can shift their work management approach by:

- Adjusting budgeting practises from being project-driven to being determined by value stream.
- Modifying team structures to enable rapid experimentation and active collaboration amongst teams.
- Practising horizontal communication patterns instead of the top-down approach.

Support a technology shift

Finally, organisations working towards scaling Agile must address their technology stack. Scaling Agile across the enterprise both requires and creates increased visibility, transparency, and information flow. For most organisations, that means evaluating and potentially augmenting or replacing technology solutions.

What technology tools can facilitate scaling Agile? It depends, in part, on the organisation’s Agile maturity.

If the business already supports multiple Agile teams, scaling Agile may mean implementing a solution that can connect them for improved transparency and flow. Other organisations may need something more robust that can go beyond teams’ and teams of teams’ visibility to map Agile work to the greater portfolio.

In both cases, they should look for a tool that allows information and collaboration to flow in both directions, mapping strategic plans down to Agile teams and rolling work, impact, and financial contributions up to strategic objectives.

Chapter 25

Major frameworks for scaling Agile

People expect that implementing a new scaled agile framework will instantly do wonders. But it is not like that. These practises are easier said than done. Transforming one individual is one thing, aiming to improve the entire organisation is another. This is nothing to be worried about. It takes time, patience and consistency to achieve the level of perfection that everyone hopes for.

A framework helps you avoid reinventing the wheel with regard to the structure, processes and the principles to follow for your organisation to become agile.

It is also good to hire an agile coach. Their knowledge attained from numerous training sessions and real-time experience can prove to be extremely valuable in guiding every individual. An agile coach can also connect and completely understand the pain that most individuals may be going through during this transition.

So, let's look at some of the most important characteristics of five frameworks for scaling your agile adoption.

Scrum@Scale (SaS)

It seeks to do at scale what Scrum does for a single team: keep the what (product) and the how (process) separate. To that end, it defines two distinct but overlapping cycles: the Scrum Master Cycle for product delivery and the Product Owner Cycle for product discovery. Two new roles facilitate the scaled versions of the Scrum events: the Scrum of Scrums Master and the Chief Product Owner.
The main goal is to align growing organisations around one common and shared set of goals. Cross-team coordination is managed through a Scrum of Scrums. This technique introduces the concept of a team of teams with its own backlog for everything needed to coordinate the teams' work and resolve impediments that a single team can't tackle on their own. It requires continuous communication between different Scrum teams, especially in those areas where they overlap.

Scrum of Scrums meetings are usually attended by the Scrum Masters of every team.

Large Scale Scrum (LeSS)

LeSS is a framework for scaling agile product delivery. Like other Scrum-based frameworks, LeSS is single-team Scrum with a few adaptations.
It truly keeps adaptations to a minimum. It is one of the least prescriptive frameworks around. There is one Product Owner and one Product Backlog for the complete shippable product and all teams work on the same product.
It only adds a preliminary part to the Sprint Planning, called Sprint Planning One, which is a meeting between the Product Owner and rotating team representatives. It happens before the multi-team Sprint Planning Two that is between all members of each team.
It also adds an overall retrospective and replaces the per-team Sprint reviews with an all-teams one.
For organisations with more than eight teams, there's LeSS Huge. It divides the product into requirement areas and adds just a single role: the Overall Product Owner. This role is responsible for product-wide prioritisation and deciding which teams work in which requirement area.
All areas follow the same Sprint and produce a single integrated product increment.

Nexus

Nexus builds upon Scrum's foundation, and it minimally extends the Scrum framework only where absolutely necessary to enable a group of 3 to 9 scrum teams, called Nexus, to work with a single product owner and a single product backlog (similar to Scrum@Scale and Less) to build an integrated Increment that meets a goal. Nexus seeks to preserve and enhance Scrum's foundational bottom-up intelligence and empiricism while enabling a group of Scrum Teams to deliver more value than can be achieved by a single team. It does this by introducing a Nexus Integration Team and Cross-Team Refinement to coordinate the work and helping teams to reduce the complexity created by cross-team dependencies that they encounter as they collaborate to deliver an integrated, valuable, useful product Increment at least once every Sprint.
The Nexus Integration Team deals with integration issues that would prevent the Nexus from delivering an integrated product Increment. It consists of the Product Owner, plus a Scrum Master and members from the development teams.
Cross-Team Refinement is an ongoing activity to identify cross-team dependencies and surface early which team will likely work on an item. Who attends can vary with the items up for refinement.

Other adaptations to single-team Scrum events, are:

- Nexus Sprint Planning to coordinate the work of all teams in the Nexus.
- Nexus Daily Scrum in addition to each team's daily scrum to keep tabs on (newly discovered) dependencies and integration issues.
- Nexus Sprint Review that replaces the individual team's sprint reviews.
- Nexus Sprint Retrospective to review how the teams and their shared processes and tools functioned. It closes the Sprint, so it follows the individual teams' retrospectives.

Disciplined Agile (DA)

Disciplined Agile started as Disciplined Agile Delivery (DAD), with a focus on product delivery. It evolved from there and was renamed Disciplined Agile to reflect its widening scope. DA covers everything from project conception to delivery to clients.

It is lightweight. Indeed, it shines a light on the "what" and the tools you need to make it happen, but it leaves the "how" up to you.

The Disciplined Agile is a hybrid model, which is formed by a collection of the world's proven Lean-Agile methods such as Kanban, XP, Agile Modelling, Unified Process and many more.
The DA process works in three stages – Inception, Construction, and Transition.
To make this regular process more Agile, the method also weaves in four different life cycles that describe how to complete the work at hand.

The four Agile DA lifecycles are:

Agile Delivery lifecycle. It is based on Scrum. It helps to turn goals into a work item list and then into short milestones.
Lean lifecycle. It creates a continuous stream of workflow throughout the project. It ensures the processes are optimised and there are little to no bottlenecks.
Continuous Lean and Agile Delivery lifecycle. It ensures that teams use iterations to work and deliver fast and often. It focuses mainly on the construction and transition phases.
Exploratory (Lean Startup) lifecycle. It aims to brainstorm new solutions based on the gathered feedback.

Teams can choose which life cycle works best for them and an Agile coach helps them in understanding when to use the chosen cycle.

Scaled Agile Framework (SAFe)

It is a set of principles, practises and workflows enabling larger companies to move towards an agile way of working and offers guidance at the Portfolio, Value Stream, Program, and Team levels.
SAFe combines Lean, Agile, and DevOps practises for business agility. It promotes alignment, collaboration, and delivery across large numbers of agile teams.
Many agile practitioners consider SAFe overly prescriptive and complex.
It does indeed prescribe many roles, events, and practises. They add quite some complexity and require significant investment and commitment to adopt. And they do detract from the flexibility that Agile holds dear.
However, for very large enterprises this can be a blessing. SAFe's prescriptive nature provides concrete guidance without forcing you to immediately remodel your enterprise's organisational structure or your product's architecture to help minimise team dependencies.
Since one team is rarely enough to finish a product, several teams working on the same goal are grouped together and called Agile Release Trains (ART).
The train works similarly to an iteration. However, it takes longer (usually around 5 iterations of a single team) and the process is facilitated by a dedicated Release Train Engineer (RTE).
One of the tools it uses for this is its quarterly planning event: the Program Increment Planning (PI planning).

It's a top-down collaborative event and planning cycle on top of and overarching the standard Scrum Sprint cycle or cadence in Kanban.
PI planning allows you to align everyone (at your level of adoption) on the strategic goals for the coming three months. It helps surface the dependencies between teams and departments involved, and come up with a prioritisation that allows you to move efficiently towards the PI goal.

All teams take part in the planning, testing, and retrospectives, while product management provides the vision and the backlog.
If one release train is not sufficient to cover the whole organisation, multiple trains are grouped together into something called a Solution Train.
To conclude, I would like to add some notes on the SAFe levels:

- At the Team level, teams can choose to follow Scrum or Kanban plus several XP (eXtreme Programming) practises.
- The Program level coordinates team efforts with quarterly Program Increment Planning (PI Planning) and a team of teams called the Agile Release Train (ART).
- If you have a large product that more than 150 people work on, at Value Stream level, SAFe adds a Solution Train to coordinate the various ARTs and a Solution Train Engineer whose role

is similar to the RTE’s but at a more integrated level.

- The Portfolio level manages development streams and coordinates with the other levels to ensure that Agile Release Trains and Solution Trains align with strategic goals.

Scaling Agile can be an intimidating prospect, but each of these Scaled Agile Frameworks and approaches is designed to help you on your journey. If none of the frameworks and approaches mentioned here appeal to you, other frameworks and approaches abound, and about 1 in 6 companies roll their own.

Glossary

Agile: A specific set of values and principles, as expressed in the Manifesto for Agile Software Development.

Agile modelling (AM): A methodology for modelling and documenting software systems based on best practises. It is a collection of values and principles that can be applied on an (agile) software development project. This methodology is more flexible than traditional modelling methods, making it a better fit in a fast changing environment. It is part of the agile software development tool kit.

Artifact: A tangible by-product produced during product development. The product backlog, sprint backlog, and potentially shippable product increment are examples of Scrum artifacts.

Burn-down Chart: a chart showing on the vertical axis the quantity of work remaining over time, which is shown on the horizontal axis.

Ceremony: A ritualistic or symbolic activity that is performed on well-defined occasions. Some people refer to the core Scrum activities of Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective as ceremonies.

Cross-functional team: A team composed of members with all the functional skills (such as UI designers, developers, testers) and specialties necessary to complete work that requires more than a single discipline.

Daily Scrum: Time-boxed and adaptive planning activity that the team performs each day during a Sprint. It serves for the team to inspect the daily progress and update the Sprint Backlog. This event is also called daily stand-up or simply stand-up.

Definition of Done: A checklist of the types of work that the team is expected to successfully complete by the end of the sprint, before it can declare its work to be potentially shippable.

Definition of Ready: A checklist of conditions that must be true before a product backlog item is considered ready to pull into a Sprint during Sprint Planning.

Developers: A self-organising, cross-functional team of people who collectively are responsible for all of the work necessary to produce working, validated assets. One of the three roles that constitute every Scrum team.

Empiricism: A process control type in which decisions are based on observation, experience and experimentation. It supports the principles of inspection, adaptation, and transparency.

Increment: The sum of all the Product Backlog items completed during a Sprint and all previous Sprints.

Product Backlog: A list of all the things that must be done to complete the whole project.

Product Owner (PO): A member of the Scrum Team who has a clear view on the goals of the project, customer, market and organisation.

Pull system: A lean technique that is used to control the flow of work by only replacing what has been consumed.

Scrum Master (SM): A member of the Scrum Team who is responsible for ensuring the team lives Scrum values and principles and follows the processes and practises that the team agreed they would use.

Scrum Team: A group of collaborators, typically between five and nine individuals, who work toward completing projects and delivering products. The fundamental Scrum Team comprises one Scrum Master, one Product Owner and a group of developers.

Sprint: Short and time-boxed period when a Scrum Team works to complete a set amount of work.

Sprint backlog: The part of the product backlog that the Scrum Team will be working on in their Sprint.

Sprint goal: A shared high-level explanation of what the Scrum Team plans to achieve during the course of a Sprint.

Sprint Planning: The event that kicks off the Sprint by defining what can be delivered in the Sprint and how that work will be achieved.

Sprint Retrospective: A recurring meeting held at the end of a Sprint, used by the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next iteration.

Sprint Review: A recurring meeting that takes place at the end of the Sprint, used by the Scrum Team to inspect the outcome of the Sprint and to consider the plan for the product in the future.

Stakeholders: Stakeholders are people and organisation units who frequently interface with the product owner, scrum master and scrum team to provide them with inputs and facilitate creation of the project's products and services, influencing the project throughout the project's development.

Story Point: Unit of measure for expressing an estimate of the overall effort required to fully implement a piece of work.

Unified Process: Is based on the enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation. The system is developed incrementally over time, iteration by iteration, and thus this approach is also known as iterative and incremental software development.

User Story: short, informal, plain language description of what a user wants to do within a software product to gain something they find valuable.

Meet the Author

Katya is passionate about helping others love their job, adopting contemporary work practises (like Agile, Scrum, Kanban, XP, DevOps) and bringing meaningful change into organisations that will improve performance and quality of life in the workplace. She strongly believes that today's leaders must inspire and collaborate, not micromanage, to bring agility and innovation in their orgs.

She is an excellent team worker with strong coaching and mentoring skills, has a servant leadership approach to the team, provides extended support whenever possible and leads by example when needed.

In March 2016 she became a Certified Scrum Master and since then she has been practising Agile methods. Having worked in Information Technology for over 20 years, her knowledge of Computer Science makes it easier to collaborate with teams of talented product developers.

For more information, visit katyaagosti.co.uk.

You can also find Katya on Linkedin as https://www.linkedin.com/in/katya-agosti-08086569/

Thank-you notes

As any author will tell you, there may be one name on the front cover, but a book is only possible due to the hard work of multiple people. I would like to call your attention to the few who supported me.

First and foremost, many thanks to my lovely partner, Rosario, for your valid advice and encouragement on the book project. I am so lucky to have you in my life.

A very special thank you goes to my sister, Valeria Agosti, for the amazing collaboration on creating the cover and the pictures inside the book.

Moreover, I would like to offer deep gratitude to my coach and friend, Kelly Cook, who kindly donated her time to review drafts of this book and has been there for the entire journey.

And, lastly, to you, the reader. Thanks for reading!

Bibliography

Accelerate, building and scaling High Performing Technology Organisations, by Nicole Forsgren, PhD Jez Humble, and Gene Kim, ISBN 978-1942788331, (2018) IT REVOLUTION

Actionable Agile Metrics for Predictability, by Danie Vacanti, ISBN 978-0986436338, (2015) Leanpub

Agile Retrospectives: Making Good Teams Great, by Esther Derby and Dian Larsen, ISBN 978-0977616640, (2006) Pragmatic Bookshelf

Coaching Agile Teams, A companion for Scrum Masters, Agile Coaches, and Project Managers in Transition, by Lyssa Adkins, ISBN 978-0321637703, (2010) Addison - Wesley

Essential Scrum, a practical guide to the most popular agile process, by Kenneth S. Rubin, ISBN 978-0137043293, (2013) Addison Wesley

Scrum, A Pocket Guide, A Smart Travel Companion, by Gunther Verheyen, ISBN 978-9087537203, (2013) Van Haren Publishing

SPRINT, how to solve big problems and test new ideas in just five days, by Jake Knapp with John Zeratsky and

Braden Kowitz, ISBN 978-1501140808, (2016) Simon & Schuster Paperbacks

The Art of Agile Development, by James Shore and Shane Warden, ISBN 978-0596527679, (2007) O'REILLY

The elements of Scrum, by Chris Sims and Hillary Louise Johnson, ISBN 978-0982866917, (2011) DYMAXICON

The Lean Startup: How Constant Innovation Creates Radically Successful Businesses , by Eric Ries, ISBN 978-0670921607, (2011) Penguin Business

Online References

Chapter 1
Manifesto for Agile Software Development (n.d.). Available at https://www.agilealliance.org/agile101/the-agile-manifesto/
The History of Agile (n.d.). Available at https://www.planview.com/resources/guide/agile-methodologies-a-beginners-guide/history-of-agile/

Chapter 2
Agile & Waterfall Methodologies – A Side-By-Side Comparison (n.d.). Available at http://www.base36.com
The Traditional Waterfall Approach (2009). Available at https://www.umsl.edu/~hugheyd

Chapter 3
Agile and Lean (n.d.). Available at https://www.planview.com/resources/guide/lean-principles-101/agile-and-lean/
Creating customer value with Lean Thinking(n.d.). Available at https://www.winman.com/blog/creating-customer-value-with-lean-thinking
Ford Production System: A successful adoption of Lean manufacturing (2013). Available at http://cmuscm.blogspot.com/2013/02/adoption-of-lean-manufacturing-ford.html
From supermarkets to software: history of Kanban (2018). Available at https://getnave.com/blog/kanban-history/
History of Lean (n.d.). Available at https://sixsigmastudyguide.com/history-of-lean/

How lean and agile relate and how you can win by using both (2021). Available at
https://www.plutora.com/blog/how-lean-agile-relate
Lean and Agile - Ideas that work together (2019). Available at
https://www.leaneast.com/lean-and-agile-work-together
Lean Management History (n.d.). Available at
https://www.wevalgo.com/know-how/lean-management/lean-management-history
The Origins and Evolution of Lean Management Systems (2012). Available at
https://www.jois.eu/files/DekierV_5_N1.pdf
What is Lean? (2017). Available at
https://theleanway.net/what-is-lean

Chapter 4
Agile frameworks (n.d.). Available at
https://www.toolsqa.com/agile/agile-methodology/
Agile framework comparison: Scrum vs Kanban vs Lean vs XP (2017). Available at
https://dzone.com/articles/agile-framework-comparison-scrum-vs-kanban-vs-lean
Crystal Clear - Human Powered Methodology For Small Teams (n.d.). Available at
https://www.agilest.org/scaled-agile/crystal-clear/
Kanban vs Scrum vs XP – an Agile comparison
(n.d.). Available at
https://manifesto.co.uk/kanban-vs-scrum-vs-xp-an-agile-comparison/
Scrum (software development) (n.d.). Available at
https://en.wikipedia.org/wiki/Scrum_(software_development)
The Kanban Method (2018). Available at
https://getnave.com/blog/what-is-the-kanban-method/

Chapter 5
Agile Games (n.d.). Available at
https://www.plays-in-business.com/agile-games-facilitation/
Agile Games - ball point game (2011). Available at
https://www.101ways.com/2011/09/27/agile-games-ball-point-game/
Agile Games - Battleship! (2019). Available at
https://onbelay.co/articles/2019/2/6/agile-game-battleship

Chapter 6
A short history of Scrum (2017). Available at
https://www.exin.com/article/short-history-scrum
Scrum Framework History (n.d.). Available at
https://agileforgrowth.com/scrum-history/
Scrum, the essential agile method for software development (2019). Available at
http://www.aligntechsolutions.com/2019/02/scrum-the-essential-agile-method-for-software-development/
Scrum, what's in a name?(2017). Available at
https://dzone.com/articles/scrum-whats-in-a-name
The brief history of Scrum (2019). Available at
https://warren2lynch.medium.com/the-brief-of-history-of-scrum-15efb73b4701
The evolution of the Scrum Guide (2019). Available at
https://medium.com/serious-scrum/the-evolution-of-the-scrum-guide-10-to-19-f3ac4d82cfcb

Chapter 7
The five Scrum values and why they matter (n.d.). Available at
https://nira.com/scrum-values/
The Three Pillars of Empiricism (2016). Available at
https://www.scrum.org/resources/blog/three-pillars-empiricism-scrum

What are Scrum's Three Pillars? (n.d.). Available at https://www.visual-paradigm.com/scrum/what-are-scrum-three-pillars/
What are the Scrum values? (2018). Available at https://www.extremeuncertainty.com/what-are-the-scrum-values/

Chapter 8
A beginner's guide to Scrum Ceremonies (2019). Available at https://www.projectmanager.com/blog/guide-to-scrum-ceremonies
A quick guide to Scrum artifacts (2020). Available at https://www.projectmanager.com/blog/scrum-artifacts
Five steps to find your Definition of Done (2020). Available at https://plan.io/blog/definition-of-done/
Product Backlog and Sprint Backlog: A Quick Guide (2018). Available at https://www.projectmanager.com/blog/product-backlog-sprint-backlog
Scrum Artefacts (2020). Available at https://www.ntaskmanager.com/blog/scrum-artifacts
Scrum Roles: The anatomy of a Scrum Team (2018). Available at https://www.projectmanager.com/blog/scrum-roles-the-anatomy-of-a-scrum-team
The Definition of Done: what does done actually mean? (2017). Available at https://medium.com/@dannysmith/the-definition-of-done-what-does-done-actually-mean-ef1e5520e153
The five Scrum Events (n.d.). Available at https://www.thescrummaster.co.uk/scrum/the-five-scrum-events/
The four Agile Scrum Ceremonies Explained (2021). Available at

https://thedigitalprojectmanager.com/scrum-ceremonies-made-simple/
The Scrum Guide (2020). Available at
https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf
What are the time-boxed events in Scrum? (n.d.). Available at
https://www.visual-paradigm.com/scrum/what-are-scrum-time-boxed-events/
What is a Sprint in Scrum? (n.d.). Available at
https://www.visual-paradigm.com/scrum/what-is-sprint-in-scrum/

Chapter 9
Burndown chart: What is it and how do I use it? (2019). Available at
https://www.projectmanager.com/blog/burndown-chart-what-is-it
Five tips for using Scrum boards (2020). Available at
https://www.projectmanager.com/blog/what-is-a-scrum-board

Chapter 10
A better stand up by walking the board (2017). Available at
https://www.audiodog.co.uk/blog/2017/12/10/better-stand-up-by-walking-the-board/

Chapter 11
Scrum: 19 Sprint Planning Antipatterns (n.d.). Available at
https://dzone.com/articles/scrum-19-sprint-planning-anti-patterns

Chapter 12
What is Agile estimation? (n.d.). Available at
https://www.visual-paradigm.com/scrum/what-is-agile-estimation/

How do waterfall and agile project estimates differ? (2018). Available at https://swarmonline.com/how-do-waterfall-and-agile-project-estimates-differ/

Chapters 13, 14
Agile estimation with the bucket system (n.d.). Available at https://www.101ways.com/agile-estimation-with-the-bucket-system/
How can we get the best estimates of story size? (n.d.). Available at https://www.mountaingoatsoftware.com/blog/how-can-we-get-the-best-estimates-of-story-size
Planning Poker (n.d.). Available at https://www.mountaingoatsoftware.com/agile/planning-poker
Seven Agile estimation techniques (2016). Available at https://technology.amis.nl/agile/8-agile-estimation-techniques-beyond-planning-poker/
Story Points estimates are best thought of as range (n.d.). Available at https://www.mountaingoatsoftware.com/blog/story-point-estimates-are-best-thought-of-as-ranges

Chapter 15
Definition of Ready Template: what needs to be completed before adding a story to a sprint (2018). Available at https://www.linkedin.com/pulse/definition-ready-template-what-needs-completed-before-ignacio-paz/
The dangers of a Definition of Ready (n.d.). Available at https://www.mountaingoatsoftware.com/blog/the-dangers-of-a-definition-of-ready
Walking through a definition of ready (2017). Available at https://www.scrum.org/resources/blog/walking-through-definition-ready

What is a definition of ready? (n.d.). Available at https://agility.im/frequent-agile-question/what-is-a-definition-of-ready/

Chapters 16, 17
Agile estimation with the bucket system (n.d.). Available at https://www.101ways.com/agile-estimation-with-the-bucket-system/
How can we get the best estimates of story size? (n.d.). Available at https://www.mountaingoatsoftware.com/blog/how-can-we-get-the-best-estimates-of-story-size
How do waterfall and agile project estimates differ? (2018). Available at https://swarmonline.com/how-do-waterfall-and-agile-project-estimates-differ/
How to play the team estimation game (2012). Available at https://agilelearninglabs.com/2012/05/how-to-play-the-team-estimation-game/
Planning Poker (n.d.). Available at https://www.mountaingoatsoftware.com/agile/planning-poker
Seven Agile estimation techniques (2016). Available at https://technology.amis.nl/agile/8-agile-estimation-techniques-beyond-planning-poker/
Story Points estimates are best thought of as range (n.d.). Available at https://www.mountaingoatsoftware.com/blog/story-point-estimates-are-best-thought-of-as-ranges
What is Agile estimation? (n.d.). Available at https://www.visual-paradigm.com/scrum/what-is-agile-estimation/

Chapter 18
Are people problems creating estimation problems? (n.d.). Available at
https://www.mountaingoatsoftware.com/blog/are-people-problems-creating-estimating-problems
Does the perfect estimate exist? (n.d.). Available at
https://www.mountaingoatsoftware.com/blog/does-the-perfect-estimate-exist-free-video-training
The five possible estimates and which one your team should use (n.d.). Available at
https://www.mountaingoatsoftware.com/blog/the-five-possible-estimates-and-which-one-your-team-should-use

Chapter 19
Better user stories (n.d.). Available at
https://mountain-goat-software.thinkific.com/courses/better-user-stories-professional
SPIDR, five simple techniques for a perfectly split user story (2017). Available at
https://blogs.itemis.com/en/spidr-five-simple-techniques-for-a-perfectly-split-user-story

Chapter 20
Agile maturity assessments: going back to Shu Ha Ri (2021). Available at
https://zenexmachina.com/agile-maturity-assessments-going-back-to-shu-ha-ri/
Shu Ha Ri Agile, a fantastic tool for Agile Coaches (2019). Available at
https://luis-goncalves.com/shu-ha-ri-agile-coaches/

Chapter 21
Five agile metrics you won’t hate (n.d.). Available at
https://www.atlassian.com/agile/project-management/metrics

How to get value measuring agile team health metrics (2019). Available at https://www.agileconnection.com/article/how-get-value-measuring-agile-team-health-metrics
Seven useful agile metrics that optimise for learning (n.d.). Available at https://www.solutioneers.co.uk/agile-metrics/

Chapter 22
Five tips for better agile release planning (2021). Available at https://www.projectmanager.com/blog/agile-release-planning-tips
How to create an agile release plan (n.d.). Available at https://www.lucidchart.com/blog/agile-release-planning
Overview agile release planning (n.d.). Available at https://www.pmmajik.com/overview-agile-release-planning/
Release planning (n.d.). Available at https://www.knowledgehut.com/tutorials/scrum-tutorial/release-planning

Chapter 23
Agile game - battleship (2019). Available at https://onbelay.co/articles/2019/2/6/agile-game-battleship
Agile games - ball point game (n.d.). Available at https://www.101ways.com/agile-games-ball-point-game/
Ball point game - introducing agile by the fun way (2017). Available at https://www.plays-in-business.com/ball-point-game-introducing-agile-by-the-fun-way/
You sunk my methodology (n.d.). Available at https://www.tastycupcakes.org/2012/02/you-sunk-my-methodology/

Chapter 24
Essential SAFe: most common challenges moving to Scaled Agile Framework (2021). Available at https://www.bacancytechnology.com/blog/scaling-agile-implementation-challenges
Five common challenges faced when scaling agile (2021). Available at https://kendis.io/scaling-agile/challenges/
Scaling Agile: how to overcome 3 common challenges when scaling agile (n.d.). Available at https://www.planview.com/resources/guide/what-is-agile-program-management/scaling-agile-common-challen/

Chapter 25
Agile at scale (n.d.). Available at https://www.atlassian.com/agile/agile-at-scale
Disciplined Agile Delivery (2018). Available at https://kendis-io.medium.com/disciplined-agile-delivery-dad-cf0d1b6ffb13
Scaled Agile Approaches (n.d.). Available at https://teamhood.com/agile/safe-vs-sos-vs-dad-vs-less/
Scaling Agile: an overview of popular frameworks (n.d.). Available at https://www.actonic.de/en/scaling-agile-an-overview-of-popular-frameworks/
Scaling Scrum with Nexus (n.d.). Available at https://www.scrum.org/resources/scaling-scrum
Six Scaled Agile Frameworks - which one is right for you? (n.d.). Available at https://www.digite.com/blog/scaled-agile-frameworks/

Scrum Vocabulary
Agile tutorial (n.d.). Available at https://www.visual-paradigm.com/tutorials/agile-tutorial/how-to-identify-scrum-project-stakeholders/

Agile modelling (n.d.). Available at
https://en.wikipedia.org/wiki/Agile_modeling
Unified process (n.d.). Available at
https://www.sciencedirect.com/topics/computer-science/unified-process
Kanban pull system (n.d.). Available at
https://kanbanzone.com/resources/kanban/kanban-pull-system/