

GUIDE SCRUM DEVELOPER

2025

v.1.0



www.europeanscrum.org

Authors: David Marti and Yvonne Agnes

Table of Contents

1	INTRODUCTION	- 4 -
1.1	INTRODUCTION TO SCRUM AND THE ROLE OF THE SCRUM DEVELOPER	- 4 -
1.2	SCRUM PILLARS AND VALUES	- 4 -
2	SCRUM COMPONENTS	- 5 -
2.1	ROLES.....	- 5 -
2.2	ARTIFACTS.....	- 5 -
2.3	EVENTS.....	- 5 -
2.4	RULES AND VALUES.....	- 5 -
3	ROLES IN SCRUM	- 6 -
3.1	SCRUM MASTER: THE PROCESS FACILITATOR	- 6 -
3.2	WHO IS THE SCRUM MASTER?	- 6 -
3.3	SCRUM MASTER ROLES.....	- 6 -
3.4	¿WHAT SHOULD THE SCRUM MASTER NOT DO?.....	- 6 -
3.5	PRODUCT OWNER.....	- 6 -
3.6	WHO IS THE PRODUCT OWNER?.....	- 6 -
3.7	PRODUCT OWNER ROLES	- 7 -
3.8	WHAT SHOULD THE PRODUCT OWNER NOT DO?	- 7 -
4	DEVELOPMENT TEAM: THE HEART OF SCRUM	- 7 -
4.1	WHO ARE THE DEVELOPMENT TEAM?	- 7 -
4.2	DEVELOPMENT TEAM FEATURES	- 7 -
4.3	DEVELOPMENT TEAM FEATURES	- 7 -
4.4	WHAT SHOULD THE DEVELOPMENT TEAM NOT DO?.....	- 8 -
5	RELATIONSHIP BETWEEN DEVELOPMENT TEAM, SCRUM MASTER AND PRODUCT OWNER	- 8 -
5.1	RELATIONSHIP BETWEEN THE DEVELOPMENT TEAM AND THE SCRUM MASTER.....	- 8 -
5.2	RELATIONSHIP BETWEEN THE DEVELOPMENT TEAM AND THE PRODUCT OWNER	- 8 -
6	SPRINTS: ITERATIVE CYCLES IN SCRUM	- 8 -
7	SCRUM EVENTS AND THE ROLE OF THE SCRUM DEVELOPER	- 9 -
7.1	SPRINT: THE DEVELOPMENT CYCLE	- 9 -
7.2	DAILY SCRUM: DAILY TEAM SYNC	- 9 -
7.3	SPRINT PLANNING.....	- 9 -
7.4	SPRINT REVIEW: PRODUCT DEMO.....	- 10 -
7.5	SPRINT RETROSPECTIVE: REFLECTION AND CONTINUOUS IMPROVEMENT	- 10 -
8	SCRUM ARTIFACTS AND THE ROLE OF THE SCRUM DEVELOPER	- 10 -
8.1	PRODUCT BACKLOG: PRODUCT REQUIREMENTS LIST.....	- 10 -
8.2	KEY FEATURES OF THE PRODUCT BACKLOG	- 11 -
8.3	STRUCTURE OF A PRODUCT BACKLOG ITEM.....	- 11 -
8.4	PRODUCT OWNER'S RESPONSIBILITY IN THE PRODUCT BACKLOG	- 11 -
9	SPRINT BACKLOG: SPRINT WORKLIST	- 12 -
9.1	KEY FEATURES OF THE SPRINT BACKLOG	- 12 -
9.2	SPRINT BACKLOG STRUCTURE.....	- 12 -
9.3	DEVELOPMENT TEAM RESPONSIBILITY IN THE SPRINT BACKLOG	- 13 -
9.4	INCREMENT: THE IMPROVED FUNCTIONAL PRODUCT	- 13 -
9.5	DEFINITION OF "DONE" AND THE ROLE OF THE SCRUM DEVELOPER	- 13 -
10	PLANNING AND ESTIMATING IN SCRUM.....	- 13 -
10.1	PLANNING AND ESTIMATING IN SCRUM: DETAILED EXPLANATION.....	- 13 -
10.2	AGILE ESTIMATING IN SCRUM.....	- 14 -

11	AGILE ESTIMATION TECHNIQUES	- 14 -
11.1	STORY POINTS: RELATIVE EFFORT MEASUREMENT	- 14 -
11.2	PLANNING POKER: COLLABORATIVE ESTIMATION TECHNIQUE.....	- 14 -
11.3	THE PLANNING POKER PROCESS IS AS FOLLOWS:	- 14 -
12	PRODUCT BACKLOG REFINEMENT	- 15 -
12.1	THE MAIN ACTIVITIES OF THE PRODUCT BACKLOG REFINEMENT INCLUDE:	- 15 -
12.2	EXAMPLE OF BACKLOG REFINEMENT:.....	- 15 -
13	ROLE OF THE SCRUM DEVELOPER IN PLANNING AND ESTIMATING	- 16 -
14	AGILE PRACTICES FOR DEVELOPERS: DETAILED EXPLANATION WITH EXAMPLES	- 16 -
15	EXTREME PROGRAMMING (XP) AND ITS RELATIONSHIP WITH SCRUM	- 16 -
15.1	WHAT IS EXTREME PROGRAMMING?.....	- 16 -
15.2	MAIN XP PRACTICES APPLIED IN SCRUM	- 17 -
15.3	RELATIONSHIP BETWEEN XP AND SCRUM	- 17 -
16	CODE REFACTORING: KEEPING CODE CLEAN AND SCALABLE	- 17 -
16.1	INTRODUCTION TO CODE REFACTORING	- 17 -
16.2	BENEFITS OF REFACTORING.....	- 17 -
16.3	REFACTORING STRATEGIES.....	- 17 -
17	METRICS-DRIVEN DEVELOPMENT	- 18 -
17.1	INTRODUCTION.....	- 18 -
17.2	MAIN METRICS USED	- 18 -
17.3	KEY TOOLS	- 18 -
17.4	BENEFITS OF METRICS-DRIVEN DEVELOPMENT	- 18 -
18	AGILE ARCHITECTURES	- 18 -
18.1	INTRODUCTION.....	- 18 -
18.2	MICROSERVICES	- 19 -
18.3	KEY FEATURES.....	- 19 -
18.4	PROCEEDS	- 19 -
19	DOMAIN-DRIVEN DESIGN (DDD)	- 19 -
19.1	DDD PRINCIPLES	- 19 -
19.2	PROCEEDS	- 19 -
20	TEST-DRIVEN DEVELOPMENT (TDD)	- 20 -
20.1	WHAT IS TEST-DRIVEN-DEVELOPMENT?	- 20 -
20.2	BENEFITS OF TDD.....	- 20 -
20.3	TDD PRACTICAL EXAMPLE	- 20 -
21	SOLID PRINCIPLES: GOOD SOFTWARE DESIGN PRACTICES	- 20 -
21.1	WHAT IS SOLID?	- 20 -
21.2	THE FIVE SOLID PRINCIPLES EXPLAINED WITH EXAMPLES.....	- 21 -
22	TECHNICAL DEBT: THE COST OF LOW-QUALITY CODE	- 21 -
22.1	WHAT IS TECHNICAL DEBT?	- 21 -
22.2	HOW TO MANAGE TECHNICAL DEBT IN SCRUM:.....	- 21 -
22.3	EXAMPLE	- 21 -
23	TESTING AND QUALITY CONTROL	- 22 -
23.1	TESTING AND QA IN SCRUM: DETAILED EXPLANATION	- 22 -
23.2	TYPES OF TESTS IN SCRUM	- 22 -
23.3	UNIT TESTS.....	- 22 -

23.4	INTEGRATION TESTING	- 22 -
23.5	REGRESSION TESTING	- 22 -
23.6	BLACK BOX AND WHITE BOX TESTING.....	- 23 -
24	TEST AUTOMATION IN SCRUM	- 23 -
24.1	ROLE OF THE SCRUM DEVELOPER IN SOFTWARE TESTING AND QUALITY.....	- 23 -
25	TOOLS FOR THE SCRUM DEVELOPER.....	- 24 -
25.1	JIRA WITH CONFLUENCE: PLANNING AND DOCUMENTATION	- 24 -
25.2	KEY FEATURES OF JIRA FOR THE SCRUM DEVELOPER.....	- 24 -
25.3	CONFLUENCE: DOCUMENTATION AND COLLABORATION.....	- 24 -
25.4	HOW A SCRUM DEVELOPER USES JIRA AND CONFLUENCE	- 24 -
25.5	TRELLO: VISUAL TASK MANAGEMENT.....	- 24 -
25.6	KEY FEATURES OF TRELLO FOR THE SCRUM DEVELOPER	- 24 -
25.7	MONDAY.COM: COLLABORATIVE PLANNING	- 25 -
25.8	KEY MONDAY.COM FUNCTIONS FOR THE SCRUM DEVELOPER.....	- 25 -
26	SCRUM BOARD: THE TEAM'S VISUAL TOOL.....	- 25 -
26.1	DASHBOARD COLUMNS.....	- 25 -
26.2	CARDS OR BACKLOG ITEMS:	- 25 -
26.3	BENEFITS OF THE SCRUM BOARD	- 26 -
26.4	TYPES OF SCRUM BOARDSPHYSICAL BOARD.....	- 26 -
26.5	HOW THE SCRUM DEVELOPER USES THE SCRUM BOARD	- 26 -
27	THANKS.....	- 26 -

1 Introduction

1.1 Introduction to Scrum and the Role of the Scrum Developer

Scrum is a widely used framework in the software development industry for agile project management. It is based on principles of transparency, inspection, and adaptation, enabling development teams to deliver high-quality functional products in iterative cycles called Sprints. Its flexibility and focus on collaboration make it easy to respond quickly to changes in customer requirements and improve development efficiency.

The Scrum Developer Guide is designed for developers who work in Scrum environments and want to optimize their skills in programming, collaboration, and agile methodologies. Through this guide, developers gain knowledge about software development best practices in agile teams, including continuous integration, automated testing, and clean design principles.

This guide covers all the essential topics to effectively prepare for and pass the Scrum Developer certification, addressing both the theoretical foundations and the practical application of Scrum in real projects.

1.2 Scrum Pillars and Values

These values complement the pillars and help create an efficient and collaborative work environment:

1. **Commitment**
 - Team members must be committed to the goals of the Sprint and to the delivery of a quality product.
 - **Example:** A developer makes sure to complete his task in the Sprint and helps others if necessary.
2. **Courage**
 - It takes courage to tackle complex problems, try new ideas, and take responsibility.
 - **Example:** A team recognizes that a feature is not well implemented and decides to refactor it, rather than ignore the problem.
3. **Focus**
 - Each team member should focus on what really brings value to the product and avoid distractions.
 - **Example:** During a Sprint, developers focus on completing prioritized tasks and not on adding unnecessary features.
4. **Respect**
 - All team members should value each other's opinions and abilities.
 - **Example:** A junior developer is listened to and supported by his more experienced peers.
5. **Aperture**
 - The team should be open to new ideas, changes, and feedback.
 - **Example:** During the Sprint Retrospective, team members propose improvements in communication and are willing to adapt to optimize work.

These pillars and values are the foundation of Scrum and ensure that the team works efficiently, collaboratively, and adaptable to change.

2 Scrum Components

Scrum is an agile framework made up of key elements that ensure your effectiveness and success in project management. Its main components are presented below in schematic form:

2.1 Roles

- **Scrum Master:** Facilitates the Scrum process, removes impediments, and guides the team toward continuous improvement.
- **Product Owner:** Represents the customer's interests, manages the Product Backlog and prioritizes the value of the product.
- **Development Team:** Self-organized and multidisciplinary team in charge of developing the product.

2.2 Artifacts

- **Product Backlog:** Prioritized list of product requirements and functionalities, managed by the Product Owner.
- **Sprint Backlog:** Set of Product Backlog items selected for a Sprint, along with the execution plan.
- **Increment:** Finished and functional product at the end of each Sprint, ready to be delivered or validated.

2.3 Events

- **Sprint:** A fixed-duration work cycle where an increase in output is developed.
- **Sprint Planning:** Planning meeting where the team defines what will be developed in the Sprint.
- **Daily Scrum:** Daily synchronization meeting where the team reviews their progress and plans the next actions.
- **Sprint Review:** Demonstration of the increase to Product Owner and stakeholders for feedback.
- **Sprint Retrospective:** Final reflection of the Sprint to identify improvements in the process and apply them in future cycles.

2.4 Rules and Values

- **Commitment:** Individual and collective responsibility to achieve the objectives of the Sprint.
- **Focus:** Concentration on planned work to maximize the value delivered.
- **Openness:** Transparency in the process and willingness to receive comments.
- **Respect:** Effective collaboration and recognition of the value of each team member.
- **Courage:** Ability to face challenges and make difficult decisions to improve the product and the team.

3 Roles in Scrum

Scrum defines three main roles within its framework, each with clear roles and responsibilities:

1. **Scrum Master**
2. **Product Owner**
3. **Development Team**
4. **All of them make up the Scrum Team.**

Unlike other traditional project management approaches, there are no **project manager** or **technical lead** roles in Scrum. The team is self-organized, which means it has autonomy to make decisions about how to carry out the work.

3.1 Scrum Master: The Process Facilitator

3.2 Who is the Scrum Master?

The **Scrum Master** is a facilitator and servant leader who is responsible for ensuring that the development team and the organization understand and adopt Scrum effectively. **It's not a manager or a boss**, but a person who guides the team to peak performance by removing obstacles and improving collaboration.

3.3 Scrum Master Roles

The **functions of the Scrum Master** are as follows:

- **Facilitate Scrum events** (Daily Scrum, Sprint Planning, Sprint Review, Sprint Retrospective).
- **Eliminate impediments** that hinder the team's work.
- **Ensure that the team follows the principles of Scrum**, encouraging self-organization.
- **Collaborate with the Product Owner** to ensure the Product Backlog is clear and prioritized.
- **Train and coach the team** in agile practices and in the correct use of Scrum.
- **Measure and improve team productivity** through retrospectives and process adjustments.
- **Protect the team** from external interruptions that may affect the Sprint.

3.4 What should the Scrum Master NOT do?

- You should not act as a boss or supervisor.
- He should not make decisions for the team.
- You should not manage individual tasks within the Sprint.

3.5 Product Owner

3.6 Who is the Product Owner?

The **Product Owner (PO)** is responsible for defining the value of the product and ensuring that the development team is working on what matters most to the business and customers.

3.7 Product Owner Roles

The **Product Owner's functions** are as follows:

- **Define the vision of the product** and make sure the team understands it.
- **Manage and prioritize the Product Backlog**, ensuring that the most valuable tasks are implemented first.
- **Work closely with the development team** to clarify requirements and answer questions.
- **Communicate customer and stakeholder needs** to the team.
- **Define acceptance criteria** for user stories and validate that work meets these criteria.

3.8 What should the Product Owner NOT do?

- You should not micromanage the development team.
- You should not change the Sprint goals after the iteration has started.
- You should not act like a boss who assigns specific tasks.

4 Development Team: The Heart of Scrum

4.1 Who are the Development Team?

The **Development Team** is a group of multidisciplinary professionals who work together to deliver a functional product increment at the end of each Sprint. **It is the core of Scrum**, since it is the team in charge of the actual execution of the product, the ideal composition of members of the Development Team is less than 10, between 3 and 9.

4.2 Development Team Features

The **Development Team** has the following features:

- **Self-organized:** You don't need someone to tell you how to do your job; you decide the best way to complete the sprint.
- **Multidisciplinary:** Includes developers, designers, testers, analysts, and any other profile necessary for product delivery.
- **Responsible for Increment:** Their goal is to deliver a functional product in each Sprint.

4.3 Development Team Features

The **functions of the Development Team** are as follows:

- **Transform Product Backlog user stories** into functional software increments.
- **Break down the User Stories** from the Sprint Backlog into tasks.
- **Estimate the effort of each task** and commit to a realistic amount of work in sprint planning.
- **Collaborate with the Product Owner** to understand requirements and propose improvements.
- **Participate in the Daily Scrum** to synchronize work and resolve blockages.
- **Ensure that the increment meets the "Done Definition"** to maintain product quality.
- **Continuously improve** your processes through Sprint retrospectives.

4.4 What should the Development Team NOT do?

- You should not wait for orders from a boss.
- You should not rely exclusively on the Scrum Master to solve problems.
- You should not work on tasks outside of the Sprint Backlog without discussing it with the team.

5 Relationship between Development Team, Scrum Master and Product Owner

Scrum is not a hierarchy where one role is superior to another, but a system in which each role collaborates equally to achieve a common goal.

5.1 Relationship between the Development Team and the Scrum Master

- The **Scrum Master acts as a mentor** to the team, helping them to apply Scrum correctly.
- The **Scrum Master removes impediments** for the team to focus on their work.
- **It doesn't dictate what to do, but rather helps the team to continuously improve.**
- If the team encounters difficulties in the Scrum methodology, the Scrum Master guides them and facilitates solutions.

Example: If the team has problems with internal communication, the Scrum Master organizes dynamics to improve it.

5.2 Relationship between the Development Team and the Product Owner

- The **Product Owner defines WHAT should be built, but the team decides HOW to do it.**
- The **Development Team collaborates with the PO** to ensure that they have a good understanding of the requirements.
- The team helps refine the Product Backlog by proposing improvements and estimating tasks.

Example: If the Product Owner requests a new feature, the team analyzes its feasibility and proposes the best way to implement it.

6 Sprints: Iterative Cycles in Scrum

A **Sprint** is the core of Scrum and represents a work cycle with a fixed duration (1, 2, 3, 4) weeks, the duration of the Sprints is fixed, and is determined before the project starts.

The decision on the length of the Sprint is up to the entire Scrum Team, during the Sprint the length of the Sprint should not be modified.

If the Development Team, during a Sprint, finishes all the tasks of the Sprint Backlog, and their respective User Stories, the duration of the Sprint is not shortened, the Product Backlog is refined, and new User Stories from the Product Backlog are incorporated into the Sprint Backlog.

If the Development Team, during a Sprint, cannot finish the tasks of the Sprint Backlog, or does not have time to start some in the Sprint Backlog, and their respective User Stories, the duration of the Sprint is not extended, the corresponding User Stories are moved from the Sprint Backlog to the

Product Backlog, and it is refined. **If some User Stories are half started**, that is, some tasks are done and others are not, the User Stories half started are re-estimated, and those User Stories are moved from the Sprint Backlog to the Product Backlog.

At the end of each Sprint, the team must deliver a **functional and potentially releasable** product increment.

The **Scrum Developer** within the Sprint is responsible for building and delivering the product. Their role is to transform user stories into real functionalities, making sure to follow good development practices, perform tests and ensure the quality of the software. It is also key in estimating and planning work within the Sprint.

7 Scrum Events and the Role of the Scrum Developer

7.1 Sprint: The Development Cycle

The **Sprint** is the time frame where all the development work happens. It starts with **Sprint Planning**, continues with the team's daily work, and ends with **Sprint Review** and **Sprint Retrospective**.

Role of the Scrum Developer in the Sprint: The Scrum Developer is the **key executor** within the Sprint, since he is the one who transforms the requirements into finished functionalities. During the Sprint, the developer must collaborate with the team to solve problems, adjust functionalities based on Product Owner feedback, and ensure that the final product meets the expected quality.

7.2 Daily Scrum: Daily Team Sync

The **Daily Scrum** is a short meeting of **maximum 15 minutes** where the Development Team synchronizes its work. The participation of the Development Team is mandatory, that of the Scrum Master optional, as a facilitator, and that of the Product Owner, although optional, is often not advisable. Each member of the Development Team answers three key questions to the entire team:

1. What did I do yesterday that contributed to the goal of the Sprint?
2. What am I going to do today to get closer to the goal of the Sprint?
3. Are there any impediments that are blocking my progress?

Role of the Scrum Developer in the Daily Scrum: The Scrum Developer should share their progress, communicate any blockages that are affecting their work, and coordinate with the team to make sure everyone is aligned. It's also a key time to ask for support from other members if there are technical issues.

7.3 Sprint Planning

In this meeting, the team decides which items of the **Product Backlog** will be developed in the Sprint and how the work will be executed. It is usually divided into two parts, the first part is decided what the objective of that Sprint is, and its Definition in fact, and the items (User Stories) of the Product Backlog are selected, which will be implemented in that Sprint, that is, those items are passed from the Product Owner's artifact, the Product Backlog, to the Development Team artifact, Sprint Backlog. In the second part of Sprint Planning, the Development Team decides how to implement those items, User Stories, which are already in Sprint Planning, and divide it into tasks, and often distribute them among the components of the Development Team.

The maximum duration of the event depends on the duration of the Sprint, **if the Sprint lasts one month, its maximum duration is 8 hours, if it is fewer hours, proportional to the duration of the Sprint.**

The entire Scrum Team is required to participate, i.e., Scrum Master, Product Owner, and Development Team. The Scrum Developer is key in this meeting, as he or she participates in estimating the effort required for each task and defines how the functionalities will be implemented. It is your responsibility to ensure that the selected work is **realistic and achievable within the Sprint.** You can also ask questions of the Product Owner to clarify technical details before committing to the work.

7.4 Sprint Review: Product Demo

At the end of the Sprint, the team presents the work done to stakeholders and the Product Owner. The functional increase of the product is shown and feedback is collected.

The participation of the entire Scrum Team is mandatory. **Its maximum duration is 4 hours in one-month Sprints,** in case of less Sprint length, its duration is proportional. The Scrum Developer must demonstrate how the developed functionalities work, explain the technical decisions made and receive feedback for future improvements. It is also the time to validate that the product meets the requirements of the Product Owner and stakeholders.

7.5 Sprint Retrospective: Reflection and Continuous Improvement

It is an internal team meeting where the positive aspects and areas for improvement of the Sprint are analyzed. The aim is to optimize the way of working for the next Sprint.

They analyze what things are being done well, and can be improved, what things are not done and should be improved, and what things are done badly, and should be done well.

The participation of the entire Scrum Team is mandatory. **Its maximum duration is 3 hours in one-month sprints, in case of less sprint length, it is proportional.**

The Scrum Developer should provide their perspective on what worked well and what can be improved. It can suggest changes in methodology, team communication, or tools used. It's also the time to talk about blockages that could have been avoided and how to improve the team's efficiency in the upcoming Sprint.

8 Scrum Artifacts and the Role of the Scrum Developer

8.1 Product Backlog: Product Requirements List

The **Product Backlog** is one of the fundamental artifacts within the Scrum framework. It is an ordered and prioritized list that contains all the requirements, functionalities, improvements and fixes necessary for the product under development. Its main goal is to ensure that the development team works on the tasks of greatest value to the business in a structured and efficient manner. These components of the Product Backlog are called **PBIs, Product Backlog Items,** and are represented in the Product backlog through **User Stories, Themes, and Epics.**

8.2 Key Features of the Product Backlog

- The Product Backlog has the following features:
- **Evolutionary and Dynamic**
 - The Product Backlog is not a static document; It is constantly changing as new needs are discovered, feedback is received, or a better understanding of the product and market is acquired.
- **Prioritized by Value**
 - Each item within the Product Backlog is sorted according to its importance and the value it brings to the business. It is the responsibility of the **Product Owner** to set this priority to maximize the impact of the development team's work.
- **Backlog Grooming**
 - The Product Backlog should be refined regularly in refinement sessions. During these sessions, large items are broken down, acceptance criteria are redefined, and it is ensured that the highest priority items are ready to be worked on in the following Sprints.
- **Contains Different Types of Elements**
 - User Stories
 - Technical requirements
 - Bug fixes
 - Performance Improvements
 - Technical debts
- **Clarity and Definition**
 - Each item in the Product Backlog should have a clear description and well-defined acceptance criteria, so that the development team can understand and work on it without confusion.

8.3 Structure of a Product Backlog Item

Each item within the **Product Backlog (PBI)** can be structured as follows:

- **Identifier:** A unique number or code for tracking.
- **Title:** A brief description of the item.
- **Description:** Detailed explanation of the need or functionality.
- **Priority:** Level of importance (High, Medium, Low).
- **Estimated Effort:** A measure of complexity or time required to complete the task.
- **Acceptance Criteria:** Conditions that must be met to consider the task completed.

They are represented through User Stories, Epics, and Themes. The Product Owner, with the help of members of the Development Team, if necessary, writes and details them.

8.4 Product Owner's Responsibility in the Product Backlog

The **Product Owner** is primarily responsible for the management and maintenance of the Product Backlog. Its main functions include:

- Define and prioritize Product Backlog items based on business value.
- Refine the elements so that they are clear and understandable to the development team.
- Adapt the Product Backlog based on feedback from stakeholders and the development team.

- Ensure that high-priority items are ready to be included in the Sprint Backlog.

The Scrum Developer does not manage the Product Backlog, but does participate in its refinement, providing their technical vision on the effort required for each task and possible challenges.

9 Sprint Backlog: Sprint Worklist

Sprint Backlog is a subset of the Product Backlog that contains the items selected to be developed during a Sprint. These components are **called User Stories**. The Sprint Backlog is a key artifact within the Scrum framework, as it provides transparency on the work to be done in each iteration.

9.1 Key Features of the Sprint Backlog

The Sprint Backlog has the following features:

- **Defined During Sprint Planning**
 - The Sprint Backlog is created in the **Sprint Planning** meeting, where the development team selects the Product Backlog items that it commits to completing during the Sprint.
- **Owned by the Development Team**
 - Once defined, the Sprint Backlog is managed exclusively by the development team, which has the authority to adjust the tasks needed to achieve the Sprint goal.
- **Divided into Tasks and Activities**
 - Each item in the Sprint Backlog is broken down into specific, manageable tasks for easy execution and tracking.
- **Focus on a Sprint Goal**
 - All selected items must contribute to **the Sprint Objective**, which defines the value to be delivered at the end of the iteration.
- **Continuous Update**
 - The Sprint Backlog is not static; may be updated during the Sprint as needed, as long as the changes do not compromise the goal of the Sprint.

9.2 Sprint Backlog Structure

Each item in the Sprint Backlog, called a User Story, can include the following information:

- **Identifier:** Unique code or number for tracking.
- **Title:** Brief description of the element.
- **Description:** Details on what needs to be developed.
- **Priority:** Importance within the Sprint.
- **Estimated Effort:** A measure of the work required to complete it.
- **Responsible:** Team member assigned to the task.
- **Status:** Task Progress (Pending, In Progress, Completed).

Each User Story is broken down into Tasks, and these are **defined, estimated and assigned, by the Development Team**.

9.3 Development Team Responsibility in the Sprint Backlog

The **development team** is responsible for the management of the Sprint Backlog. Its main functions include:

- Divide the selected elements, User Stories, into specific tasks.
- Update and adjust the Sprint Backlog according to progress and needs.
- Monitor progress during the **Daily Scrum**.
- Ensure that completed tasks contribute to the goal of the Sprint.

9.4 Increment: The Improved Functional Product

The **Increment** is the upgraded version of the product delivered at the end of each Sprint. It must be in a condition to be launched or used by users.

Role of the Scrum Developer team in the Increase: The Scrum Developer is responsible for delivering functional and well-tested code that meets quality standards. It is your duty to ensure that the product increment is stable and ready for deployment.

9.5 Definition of "Done" and the Role of the Scrum Developer

The **Done Definition** is the set of acceptance or validation criteria that determine when a job is completely finished. The Done Definition is **defined for each Sprint goal, and for the respective User Stories**. It includes aspects such as:

- Fully developed and revised code.
- Successfully executed unit and integration tests.
- Up-to-date technical and user documentation.
- Approval by the Product Owner.

Role of the Scrum Developer in the Done Definition: The Scrum Developer must ensure that all tasks comply with the **Done Definition** before marking them as completed. This means that you must not only develop the functionality, but also write tests, document the code, and make sure everything is ready for deployment.

10 Planning and Estimating in Scrum

10.1 Planning and Estimating in Scrum: Detailed Explanation

Planning and estimating in **Scrum** are critical to ensuring that the team can work effectively, delivering value continuously in every **Sprint**. Unlike traditional planning methods, where you try to predict exactly how long it will take to complete each task, Scrum uses an agile approach based on **relative estimates**, which allows for greater flexibility and adaptation to changes.

In this section, we'll explore in detail agile **estimation**, the techniques used, and the Product **Backlog refinement** process, with a focus on how these practices help improve team efficiency.

10.2 Agile Estimating in Scrum

In Scrum, estimates are not based on exact time calculations, but on the **relative complexity of the work**. This means that instead of asking "how many hours will it take to complete this task?", the team asks "how difficult or big is this task compared to others?".

The goal of agile estimation is for the team to be able to:

- **Determine how much workload you can handle in a Sprint.**
- **Ensure that user stories are actionable within the Sprint.**
- **Identify tasks that are too large that need to be broken down** into smaller, more manageable units.

To achieve this, various estimation techniques are used in Scrum, the most common being **Story Points** and **Planning Poker**.

11 Agile Estimation Techniques

11.1 Story Points: Relative Effort Measurement

Story Points are a unit of measurement used in Scrum to estimate the **relative complexity of a task** compared to others. Instead of measuring in time, Story Points consider factors such as:

- **Technical difficulty**
- **Amount of work involved**
- **Risks and uncertainty associated with the task**

On a Story Point scale, a task with **1 point** is the simplest possible, while one with **8, 13, or more points** is considerably more complex and may need to be broken down into smaller tasks.

Example:

- A user story to add a button to the interface could be worth **1 Story Point**.
- A story to develop a new authentication module with multiple functionalities could be worth **8 or 13 Story Points**, depending on its complexity.

Story Points allow the team to focus on the difficulty of the tasks instead of worrying about how long exactly it will take to complete them.

11.2 Planning Poker: Collaborative Estimation Technique

Planning Poker is a technique used in Scrum to collaboratively estimate user stories. In this dynamic, each team member assigns an estimate based on Story Points without external influences.

11.3 The Planning Poker process is as follows:

1. The Product Owner presents a user story to the team.
2. Each team member selects a Story Point value based on their perception of the complexity of the task.
3. All estimates are revealed simultaneously.

4. If there are significant discrepancies between the estimates, team members explain their reasons.
5. The vote is repeated until a consensus is reached on the final estimate.

This technique allows the team to arrive at a more accurate estimate, encouraging discussion and alignment of expectations.

Example: If a team is estimating a user story to add a new form to the app, one developer can assign it **2 Story Points** while another gives it **8 Story Points**. In this case, the team discusses the difference in perception and adjusts the estimate according to the actual complexity of the work.

12 Product Backlog Refinement

Product Backlog Refinement is an ongoing process in Scrum in which the team reviews and improves user stories before they are selected for a Sprint.

This process is critical to ensure that tasks are:

- **Clear and well-defined** before Sprint Planning.
- **Divided into smaller units if they are too large.**
- **Correctly prioritized** based on the value they bring to the product.

Backlog refinement is not a formal event within Scrum, but a best practice that is done regularly during the Sprint.

12.1 The main activities of the Product Backlog refinement include:

- **Break down big stories** into smaller, more manageable tasks.
- **Add acceptance criteria** to user stories so developers know what's expected.
- **Update priorities** based on new business needs or product changes.
- **Clarify technical doubts** to avoid problems during implementation.

12.2 Example of Backlog Refinement:

- A team reviews a user story titled *"As a user, I want to receive email notifications when my order has been shipped."*
- During refinement, developers realize that this task requires adjustments to the database, configuration to the mail server, and changes to the interface.
- They decide to divide the story into three smaller tasks:
 - Notification service settings.
 - Creating the trigger in the database.
 - Implementation of the user interface to enable or disable notifications.

This refinement allows the team to address the user story more efficiently and without unexpected obstacles during the Sprint.

13 Role of the Scrum Developer in Planning and Estimating

The **Scrum Developer** has a key role in planning and estimating work within Scrum. Although the Product Owner defines the requirements of the product, it is the developers who determine how much effort and complexity each task entails.

The Scrum Developer's responsibilities in planning include:

- **Actively participate in Sprint Planning**, ensuring that estimates are realistic.
- **Bring their technical expertise** to the table to estimate the difficulty of each user story.
- **Identify risks or technical dependencies** that may affect implementation.
- **Break down large stories into more manageable tasks**, ensuring that each one can be completed within a sprint.

In the refinement of the Product Backlog, the Scrum Developer contributes by reviewing the user stories with the Product Owner and clarifying technical doubts before they reach Sprint Planning. This avoids surprises during development and improves the team's efficiency.

Example of the day-to-day life of a Scrum Developer:

- During a refinement session, a developer notices that a user story for integrating a new payment method doesn't have clear acceptance criteria.
- He suggests adding technical details about the payment provider's API and proposes dividing the task into two parts: one for technical integration and one for user testing.
- This allows the story to be clearer and more manageable within the Sprint.

14 Agile Practices for Developers: Detailed Explanation with Examples

Agile practices for developers seek to improve code quality, development efficiency, and collaboration within the team. Within Scrum, it is essential for developers to adopt techniques that facilitate the continuous delivery of functional software, with a focus on **code quality and sustainability**.

In this section, we will explore in detail **Extreme Programming (XP)**, **Test-Driven Development (TDD)**, **SOLID principles**, **Technical Debt**, and many more, along with practical examples of their application in agile projects.

15 Extreme Programming (XP) and its Relationship with Scrum

15.1 What is Extreme Programming?

Extreme Programming (XP) is an agile methodology focused on code quality and development team efficiency. It is based on the continuous delivery of functional software through improved communication, simplicity in design, and rapid feedback.

Although XP and Scrum are different agile methodologies, **they can complement development teams** looking to improve their code quality within an agile framework.

15.2 Main XP Practices Applied in Scrum

- **Iterative Development**
 - In XP, as in Scrum, development is organized in **short iterations**, ensuring continuous delivery of value to the customer.
 - Each iteration in XP can be equivalent to a Sprint in Scrum.
 - **Example:** A Scrum team that follows XP principles delivers working versions of the product every Sprint, making sure that each version is stable and tested.
- **Pair Programming**
 - Two developers work together on the same code, where one writes and the other reviews in real time.
 - **Benefits:** Improves code quality, reduces errors, and allows knowledge to be better distributed within the team.
 - **Example:** A Scrum team decides that each critical functionality is programmed in pairs to reduce defects and improve collaboration.
- **Continuous Testing**
 - XP emphasizes the need for **automated tests** that are constantly running to catch bugs early.
 - **Example:** In a Scrum team using XP, each new feature must have automated testing before being integrated into the core code.

15.3 Relationship between XP and Scrum

Scrum defines **how to organize and manage work**, while XP focuses on **how to write quality code within an agile framework**. Both can be combined to optimize both team management and software quality.

16 Code Refactoring: Keeping Code Clean and Scalable

16.1 Introduction to Code Refactoring

Code refactoring is the process of improving the internal structure of the software without changing its external functionality. This process is critical to maintaining code quality as the product evolves, allowing it to be more understandable, maintainable, and efficient.

16.2 Benefits of Refactoring

- **Improved Readability:** Well-structured code makes it easier for other developers to understand the system.
- **Simplified Maintenance:** Clean code reduces the time and effort required to make changes or correct errors.
- **Reduced Complexity:** Reduces code duplication and improves modularity.
- **Optimized Performance:** Better organization can reduce computational load and improve efficiency.

16.3 Refactoring Strategies

- **Remove Duplicate Code:** Consolidate repeated logic into reusable functions.
- **Divide Large Classes and Methods:** Separate responsibilities into smaller classes and functions.
- **Use of Meaningful Names:** Name variables, methods, and classes in a descriptive manner.

- **Simplify Conditional Expressions:** Reduce excessive nesting and the use of complex operators.
- **Eliminate Unnecessary Dependencies:** Apply principles such as Dependency Reversal and Interface-Oriented Programming.

17 Metrics-Driven Development

17.1 Introduction

Metrics-driven development involves the use of tools that allow code quality to be evaluated automatically and objectively. These tools analyze multiple aspects of the code and provide detailed reports on potential problems.

17.2 Main Metrics Used

- **Cyclomatic Complexity:** Measures the number of execution paths within a program. A high value indicates that the code can be difficult to understand and test.
- **Test Coverage:** Indicates the percentage of code that is covered by automated testing.
- **Code Duplication:** Detects similar code segments that could be refactored to reduce redundancies.
- **Maintenance and Debt Ratio:** Assesses how easy it is to modify and maintain code over the long term.

17.3 Key Tools

These are examples of some of the tools that can be used.

- **SonarQube:** Provides real-time analysis on quality, vulnerabilities, and errors in code.
- **Code Climate:** Generate reports on code health, test coverage, and suggestions for improvement.
- **Linting Tools:** Tools like ESLint and Pylint help detect bugs and bad practices in the source code.

17.4 Benefits of Metrics-Driven Development

Here are some benefits:

- Reduction of errors in production.
- Increased reliability and stability of the software.
- Improved team collaboration by sharing quality standards.

18 Agile Architectures

18.1 Introduction

Agile architectures allow software to be developed in a scalable, flexible, and easy-to-maintain way. In this context, two popular approaches are **microservices** and **Domain-Driven Design (DDD)**, which allow for better code organization and greater independence in development.

18.2 Microservices

Microservices are an architectural approach in which an application is divided into small, independent services that communicate with each other.

18.3 Key Features

Here are some key features:

1. **Decomposition into Independent Services:** Each microservice handles a specific functionality.
2. **Scalability:** They can be scaled individually according to demand.
3. **Standalone Deployment:** They can be updated without affecting the rest of the application.
4. **Resiliency:** A failure in a service does not affect the entire system.

18.4 Proceeds

Here are some key benefits:

- Reduced dependency between teams.
- Easier to make changes and updates.
- Flexibility in the selection of technologies for each microservice.

19 Domain-Driven Design (DDD)

DDD is a domain-model-centric approach to software design, ensuring that the application structure reflects the rules and needs of the business.

19.1 DDD Principles

Mention of DDD principles:

1. **Domain Modeling:** The application is divided into domains that represent the real entities of the business.
2. **Ubiquitous Language:** Use of a common language between developers and domain experts.
3. **Bounded Contexts:** Separating business logic into well-defined modules.
4. **Repositories and Aggregates:** Facilitate access and management of domain data.

19.2 Proceeds

Some of the benefits that DDD brings:

- It facilitates the understanding of the business within the development.
- Improves software maintainability and scalability.
- Reduce complexity in large, distributed systems.

20 Test-Driven Development (TDD)

20.1 What is Test-Driven-Development?

Test-Driven Development (TDD) is a development technique that involves writing tests before writing functional code. It is based on a three-step iterative cycle known as **Red-Green-Refactor**:

1. **Write a test** (Network)
 - Before writing the code, the developer writes an automated test that defines the expected behavior of the functionality.
 - Since the functionality does not yet exist, the test **will initially fail** (hence the term "Network").
 - **Example:** If we are developing a login feature, we would first write a test to verify that a user with correct credentials can log in.
2. **Implement the minimum code necessary** (Green)
 - The simplest possible code is written for the test to pass.
 - It doesn't matter if the code isn't optimized at this point, the goal is to make it work.
 - **Example:** The authentication feature is implemented with minimal credential validation.
3. **Refactoring the Code** (Refactor)
 - Once the test passes, the code is optimized and improved without changing its functionality.
 - **Example:** The authentication function is improved by ensuring that it complies with clean code principles and is easily maintainable.

20.2 Benefits of TDD

Some of the benefits of TDD include:

- Ensure that each new feature is properly tested.
- Reduce the number of errors in production.
- It forces developers to write more modular and reusable code.

20.3 TDD Practical Example

If a Scrum team adopts TDD, each user story selected in the **Sprint Planning** should include **automated testing** before starting the implementation, ensuring that the code delivered in the **Sprint Review** is of high quality.

21 SOLID Principles: Good Software Design Practices

21.1 What is SOLID?

The SOLID principles are a set of best practices that help design **modular, maintainable, and flexible** code. They are essential for ensuring the quality of software within an agile framework like Scrum.

21.2 The Five SOLID Principles Explained with Examples

- **Single Responsibility Principle (SRP) - Principio de Responsabilidad Única**
 1. Each class or module must have a **single reason for changing**.
 2. **Example:** In a procurement application, the Invoice class should only be responsible for calculating the price, while another InvoicePrinter class should be responsible for printing the invoice.
- **Open/Closed Principle (OCP) - Principio de Abierto/Cerrado**
 1. A module must be **open for extension**, but **closed for modification**.
 2. **Example:** In a payment system, if we want to add PayPal as a method without modifying existing code, we can use interfaces to extend functionalities without changing the codebase.
- **Liskov Substitution Principle (LSP) - Principio de Sustitución de Liskov**
 1. A derived object must be able to **replace its base class without altering its behavior**.
 2. **Example:** If we have a bird class, and a penguin subclass, we can't make Penguin inherit a fly() method, as it doesn't comply with the Liskov principle.
- **Interface Segregation Principle (ISP) - Principio de Segregación de Interfaces**
 1. It's better to have **small, specific interfaces** rather than a single interface with a lot of unnecessary features.
 2. **Example:** A notification system may have IEmailNotifier and ISMSNotifier instead of a single INotifier interface with methods that some objects will not use.
- **Dependency Inversion Principle (DIP) - Principio de Inversión de Dependencias**
 1. High-level modules **should not rely directly on low-level modules**, but on **abstractions**.
 2. **Example:** A payment controller in an application must rely on a **generic interface** rather than relying directly on PayPalPay or Stripe.

In Scrum, developers must apply **SOLID** to ensure that the code delivered in each Sprint is **clean, extensible, and maintainable**.

22 Technical Debt: The Cost of Low-Quality Code

22.1 What is Technical Debt?

Technical **debt** is the cost that is generated when quick solutions are chosen instead of well-designed solutions, leading to long-term problems.

22.2 How to manage technical debt in Scrum:

- Identify it in the **Sprint Retrospective**.
- Allocate time in each Sprint for **refactoring**.
- Prioritizing quality over speed.

22.3 Example

A Scrum team, pressed for time, decides to **copy and paste code** instead of reusing existing features. In the short term, this allows them to deliver functionality on time, but over time the code becomes difficult to maintain and fix.

23 Testing and Quality Control

23.1 Testing and QA in Scrum: Detailed Explanation

Testing **and quality control** are fundamental in Scrum, as they guarantee that the software developed in each **Sprint** meets functional and business requirements, minimizing defects and ensuring an optimal user experience.

Since **development in Scrum is incremental**, it is crucial that testing is **continuous and automated**, allowing each change to be validated without affecting the stability of the product.

In this section, we'll explore in detail the **types of testing, test automation, and its benefits**, with practical examples of how they can be applied in agile teams.

23.2 Types of tests in Scrum

In an agile environment, testing is not a separate phase of development, but is **integrated into the workflow**. There are different types of tests that help ensure software quality at different levels

23.3 Unit Tests

Unit tests are the first line of defense against errors in your code. They focus on testing **individual units of code**, such as functions or methods, ensuring that they behave correctly in **isolation**.

Unit tests allow you to detect errors at an early stage, help refactor code with confidence, and can be automated to run every Sprint.

23.4 Integration Testing

Integration testing verifies that different modules of the system work together correctly.

If an application has an **authentication module** and another **database** module, an integration test would validate that a user can successfully authenticate and that their data is retrieved from the database.

Integration testing ensures that different parts of the system interact correctly, detects errors in communication between modules, and is essential in systems with microservices.

23.5 Regression Testing

Regression testing is used to make sure that recent changes to your code **don't introduce errors into previously developed functionality**.

A Scrum team develops a **payment system**. In a Sprint, a new cryptocurrency payment functionality is introduced. Regression tests are run to ensure that credit card and PayPal payments **continue to work properly** after this change.

Regression testing ensures that new code **doesn't break existing functionality**, can run automatically on every code integration, and reduces the risk of errors in production.

23.6 Black Box and White Box Testing

These two testing approaches help evaluate software quality from different perspectives.

Black-box testing evaluates software behavior **without knowing the internal logic of the code**. They focus on the inputs **and outputs** of the system.

A tester tests the UI without knowing how the backend code works.

White-box testing evaluates the source code and internal structure of the system. They are used to analyze **execution paths and code coverage**.

A developer checks whether all the conditions of an if have been properly tested.

Black-box testing focuses on the **functionality of the software**, and white-box testing focuses on the **structure of the code and its implementation**.

24 Test Automation in Scrum

In agile environments, **test automation** is critical to ensuring software quality without slowing down development.

Test automation allows tests to be executed efficiently and quickly, ensuring that every increment of the product is **of high quality and stable in production**.

Manual testing is prone to errors. Automating them ensures that every run is accurate and consistent.

Automated tests allow tests to be run in seconds, instead of relying on manual testers, and facilitate **continuous integration** into each Sprint.

With tools such as Selenium, Cypress, or JUnit, hundreds of tests can be run simultaneously in different environments.

Automated tests can be run on each new code integration, catching bugs before they reach production.

24.1 Role of the Scrum Developer in Software Testing and Quality.

The Scrum Developer is responsible for ensuring that the software meets quality standards in each delivery. To this end, he is actively involved in test testing and test automation.

The Scrum Developer should write unit tests for each new feature, collaborate with the QA team to define automated testing strategies, ensure that the code delivered in each Sprint complies with the Done Definition, and review automated test reports for bug fixes before the Sprint Review.

During the **Daily Scrum**, a developer reports that he has completed a new API. Before you mark the task as finished, you need to make sure that it **passes all the automated tests**. If a test fails, you must fix the code and run them again. Only when the code passes all the tests can it be reviewed by the team in the **Sprint Review**.

25 Tools for the Scrum Developer

In an agile environment, Scrum Developers rely on digital tools to manage work efficiently, collaborate with the team, and keep documentation organized. These tools help in planning, task tracking, code integration, and project documentation. Below are some of the most commonly used tools and how each of them impacts the developer's daily work.

25.1 Jira with Confluence: Planning and Documentation

Jira is a widely used Atlassian tool for project management in Scrum teams. It provides a robust system for planning Sprints, assigning tasks, tracking work, and visualizing progress using Kanban and Scrum boards.

25.2 Key features of Jira for the Scrum Developer

- Product Backlog and Sprint Backlog Management: Allows you to create, prioritize, and estimate user stories.
- Scrum and Kanban boards: They make it easy to visualize the status of each task within the Sprint.
- Workflow automation: Reduce administrative burden with predefined rules.
- GitHub and Bitbucket integration: Match commits and pull requests to user stories.
- Reports and metrics: Allows you to generate burndown charts and velocity reports to measure the team's performance.

25.3 Confluence: Documentation and collaboration

Confluence complements Jira by providing a space where teams can document everything related to the project, such as technical decisions, retrospectives, implementation guides, and functional requirements.

25.4 How a Scrum Developer uses Jira and Confluence

- Log in to Jira to check your tasks in the Sprint Backlog and update their status.
- See Confluence to review technical documentation and add notes about code changes.
- Link user stories in Jira with relevant documentation in Confluence.
- Create retrospectives in Confluence after each Sprint to improve the way you work.

25.5 Trello: Visual Task Management

Trello is a management tool based on Kanban boards, ideal for the visual organization of tasks and workflows. Its intuitive interface allows teams to organize tasks into columns that represent different stages of work.

25.6 Key features of Trello for the Scrum Developer

- Customizable boards and lists: Organize tasks into lists like "To Do," "In Progress," and "Completed."
- Labels and priorities: Assign colors and categories to tasks based on their importance.
- Automation with Butler: Create automatic rules to move cards or send notifications.

- Integration with external tools: Slack, Google Drive, GitHub, among others.
- How a Scrum Developer uses Trello to organize individual tasks within the Sprint.
- Attach bug captures or comments to specific tasks.
- Update the status of each task as it progresses in its development.
- Collaborate with other team members by adding notes and checklists.

Trello is a great choice for small teams or developers looking for a fast and flexible way to organize their work without the complexity of Jira.

25.7 Monday.com: Collaborative Planning

Monday.com is a highly customizable project management platform that allows for the planning and organization of work in Scrum teams.

25.8 Key Monday.com Functions for the Scrum Developer

- Custom workflows: It adapts to the Scrum methodology, allowing each Sprint to be structured.
- Visual Work Tracking: Use dynamic boards with timelines and Gantt charts.
- Automation of notifications and reminders: Reduces the need for manual follow-ups.
- Integrations with developer tools: Compatible with GitHub, Slack, and Jenkins.
- How you use a Scrum Developer Monday.com plans and prioritizes your tasks in each Sprint.
- Coordinate with the team using shared boards.
- Visualize the development status of each feature in real-time.

Monday.com is ideal for teams looking for an intuitive and adaptable interface for managing their Scrum projects.

26 Scrum Board: The Team's Visual Tool

The Scrum Board is a visual representation of the current state of the Sprint, designed to facilitate transparency and team alignment. It is used to keep track of user stories and tasks, allowing all team members to know what is being worked on and what is pending. Let's mention the key elements of the Scrum Board.

26.1 Dashboard Columns

The Scrum board can contain as many columns as the Development Team needs or wants, the most common being the following:

- **To Do:** Contains the to-dos selected in the Sprint Schedule.
- **In Progress:** Shows the tasks that the team is currently working on.
- **Review:** Indicates completed tasks that are being reviewed by other team members or testers.
- **Done:** Contains tasks that are completely finished and ready to be turned in.

26.2 Cards or Backlog Items:

Features of the Cards in the Scrum Board are as follows:

- Each card represents a task, user story, or bug.
- They contain details such as description, responsible, priority, and status.
- Swimlanes or Categories:
- Some versions of the Scrum Board allow you to organize tasks by priority, functionality, or work team.

26.3 Benefits of the Scrum Board

List of the benefits of the Scrum Board:

- Increases transparency within the team.
- It allows visual tracking of the progress of the Sprint.
- Facilitates early detection of blockages.
- Improve team collaboration and alignment.

26.4 Types of Scrum Boards Physical Board

Types of Scrum Boards:

- Used in teams that work in person, with sticky notes on a whiteboard.
- Benefit: Tangible interaction and immediate visibility.
- Digital Board:
- Used in remote or hybrid teams using tools such as Jira, Trello or Monday.com.

Benefit: Integration with other tools and accessibility from any location.

26.5 How the Scrum Developer Uses the Scrum Board

- Update the status of your tasks as you progress in their development.
- Review user stories and their acceptance criteria before starting your deployment.
- Identifies blockages or impediments and communicates them to the team or Scrum Master.
- Participate in the Daily Scrum by reviewing the board to coordinate the day's work.

27 Thanks

- David Marti and Yvonne Agnes for the development of this guide in its entirety, for any improvement do not hesitate to contact, yvonne@europeanscrum.org, david@europeanscrum.org
- LinkedIn Author: <https://www.linkedin.com/in/davidmarti>
- All information can be found at: www.europeanscrum.org