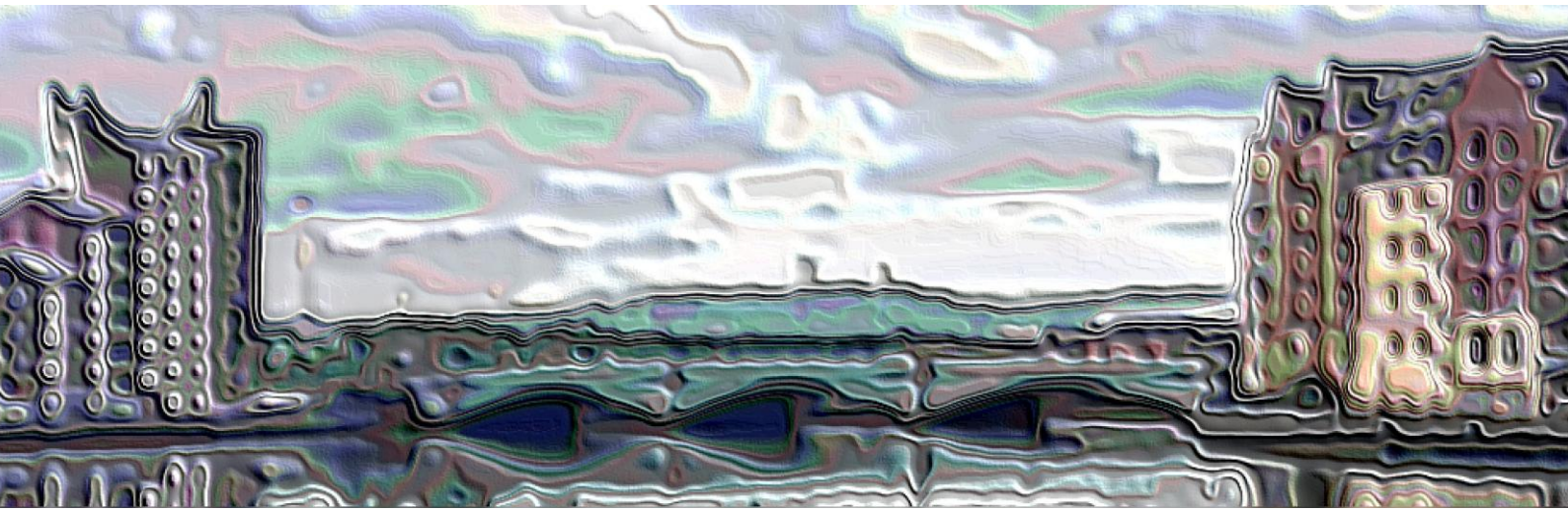
Scrum Manager®



Scrum Master

Scrum Manager - Core I

v. 2.6.1

Scrum Master

Scrum Manager: Core Subject Area I

Version 2.6.1 – January 2019

DISCLAIMER

THE AUTHORS OF THIS BOOK ARE ACTIVE MEMBERS AND CONTRIBUTORS OF THE SCRUM MANAGER PROFESSIONAL COMMUNITY, EXPERTS IN AGILE PROJECT MANAGEMENT. THEY CARRY OUT THIS JOB TO SHARE THEIR PROFESSIONAL KNOWLEDGE AND EXPERIENCE WITH PEOPLE AND ORGANIZATIONS THAT COULD CONSIDER IT USEFUL.

THIS WORK IS OFFERED AS IT IS, WITHOUT GUARANTEE OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING GUARANTIES ABOUT ADEQUACY FOR ANY PARTICULAR PURPOSE. IN NO EVENT THE AUTHORS OR THE RIGHTS HOLDERS SHALL BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITIES.

Cover Design: Scrum Manager.
Original picture: The Albert Bridge 04 – Belfast de William Murphy (http://www.streetsofdublin.com/)

Collective work created and coordinated by Iubaris Info 4 Media SL.
Authors of the Spanish version: Alexander Menzinsky, Gertrudis López, Juan Palacio.
Author of the English translation: José López

Table of content

Preface

Purpose of this book

This is a training text for the implementation and the improvement of scrum in the agile management of projects, teams and organizations.

Parts I and II comprise the official Scrum Master certification in Scrum Manager® training framework.

Audience

The intended audience of this book includes all those interested in the knowledge of the agile management model called scrum.

Organization of the book

Introduction

Putting scrum in context.
Situation and reasons for the emergence of scrum at the end of the last century as an alternative to sequential development based on processes.

Part I: Standard Scrum.

All the information needed to start working with scrum.
Roles, events and artefacts that form the framework of the standard scrum. Techniques, and guidelines for its implementation and operation.

Part II: Advanced Scrum

The keys to boosting fluency and results in the projects planned and developed with scrum.
In order to maintain a sustainable and constant production rhythm, indiscriminately delivering in sprints, or following a continuous flow of development.
Adapting practices and roles to the organization characteristics, to achieve self-organization, based on scrum principles, rather than applying standard practices.

Part III: Extensions.

New section included in this version of the book as a complement to the Scrum Manager core agenda with specialization information or extension of certain topics.
This version includes:

Agile requirements engineering: Epics, themes, user stories and tasks.

Continuous improvement and quality control

Thank you for choosing Scrum Manager training.

Your evaluation is the main feedback and the most important criterion of quality control for Scrum Manager, it helps us to improve and decide the validity and the evolution of materials, courses, training partners and teachers.
If you have participated in a training activity audited by Scrum Manager, we ask you and appreciate your evaluation about the quality of the material, teacher, agenda, etc. as well as your comments and suggestions.

Scrum Manager guarantees the anonymity of the information received, therefore it shares with the teachers and authorized partners the assessments and aspects of improvement, but never the names of the students who have provided the information.
You can perform the assessment at the "MEMBER AREA" of Scrum Manager

http://www.scrummanager.com

INTRODUCTION

Agility

The working environment of knowledge companies is very different to the one that gave rise to predictive project management disciplines. Nowadays corporative strategies are needed for products launching aimed at early delivery of tangible results, and the agile and flexible response needed to work in rapidly evolving markets.

Now the product is being built while it is simultaneously being modified and introducing new requirements. The client starts from a fairly clear vision, but the level of innovation required, and the speed at which the business environment moves, do not allow him to predict in detail how the final result will be.

Perhaps there are no "final products", but products in continuous evolution and improvement to maximize the client's satisfaction

Agile project management is not formulated on the need for anticipation, but on the need for continuous adaptation

Is predictive project management, also known as waterfall, the only one possible? Do the criteria for meeting deadlines and costs always determine success? Can there be projects whose management does not seek to carry out a previously planned work, with a budget and in a previously calculated time?

Today there are product managers who do not need to know what the 200 functionalities that the final product will be, nor whether it will be finished in 12 or 16 months.

There are customers who prefer a first version with minimal functionality in a matter of weeks, instead of a complete product in one or two years. Customers whose interest is to quickly market a new concept, and to continuously develop its value.

There are projects that do not need to manage the follow-up of a plan, and whose failure may be the consequence of an inappropriate management model.

Most of the fiascos in project management are produced by applying sequential engineering and predictive management in both the acquisition process (requirements, contracting, monitoring and delivery) and the project management, to develop products that do not need predictability guarantees in execution, as much as rapid response and flexibility to operate in rapidly changing and evolving business environments.

The Agile Manifesto

In March 2001, 17 software professionals, critics of process-based production models, were summoned by Kent Beck, who had published a couple of years earlier the book explaining the new Extreme Programming (Beck, 2000) methodology.

They met in Salt Lake City to discuss the processes used by programming teams.

At the meeting, the term "Agile Methods" was coined to define those emerging as an alternative to formal methodologies: CMM-SW, (precursor to CMMI), PMBOK, SPICE (initial project of ISO 15504), which they considered excessively "Heavy" and rigid because of their normative nature and heavy reliance on detailed, pre-development planning.

The members of the meeting summarized in four postulates what has been called the "Agile Manifesto", which are the values on which these methods are based.

Up to 2005, among proponents of process models and agile models, radical stances were more prevalent, more preoccupied with disqualifying the other than knowing their respective methods.

Agile Manifesto

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

- ***Individuals and their interaction, above processes and tools.***
- ***Software that works, over and above exhaustive documentation.***
- ***Collaboration with the client, above contractual negotiation.***
- ***The response to change, beyond tracking a plan.***

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

We value individuals and their interaction over processes and tools.

This is the most important postulate of the manifesto.
Of course, the processes help to do the job. They are an operating guide. Tools improve efficiency, but there are always tasks that require talent and need people to contribute and work with an appropriate attitude.

The process-based production aims to ensure that the quality of the result is a consequence of the "explicit" know-how deployed in the processes, rather than the knowledge provided by the people who execute them. However, in agile development processes the processes are only an aid. A support to guide the work. The extreme defence of the processes leads to the assertion that they can achieve extraordinary results with mediocre people, and the truth is that this principle is not true when creativity and innovation are needed.

We value software that works over comprehensive documentation.

Being able to anticipate how the final product will work, observing previous prototypes, or already elaborated parts, offers stimulating and enriching feedback, generating ideas that could not be conceived at first, and which could hardly be included in the drafting of a requirements document detailed at the beginning of the project.

The agile manifesto does not consider the documentation useless, only the unnecessary one. The documents support facts, they allow the transfer of knowledge, record historical information, and for many legal or regulatory issues are mandatory, but their relevance should be much less than the final product.

Communication through documents lacks the wealth and production of value that is achieved through direct communication between people and through interaction with prototypes of the product.

That is why, whenever possible, the use of documentation that consumes work effort without giving a direct value to the product should be minimized

If the organization and the teams communicate through documents, in addition to hiding the richness of the interaction with the product, they form bureaucratic barriers between departments or between people.

We value customer collaboration over contractual negotiation.

Agile practices are indicated for products that are in continuous evolution. In this kind of products, it is not possible to define in a closed requirements document how the final product should be, and it is more appropriate to take feedback continuously, and in parallel to product development, to redefine and improve the requirements of parts not yet developed according to the lessons learned with previous versions delivered to the stakeholders.

The goal of an agile project is not to control the execution to guarantee the fulfilment of the planning, but to continuously provide the highest possible value to the product.

It is therefore more appropriate a relationship of involvement and continuous collaboration with the client, than a contractual delimitation of responsibilities.

We value the response to change over following a plan

To develop products according to unstable requirements, where change and rapid and continuous evolution is inherent, responsiveness is much more valuable than tracking and assuring plans.

The main values of agile management are anticipation and adaptation, different from those of ortho

The 12 Principles of the Agile Manifesto

The agile manifesto, after the postulates of these four values on which it is based, establishes these 12 principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity --the art of maximizing the amount of work not done-- is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

The origin of scrum.

Scrum is an agile development model characterized by:

- ***Adopting an incremental development strategy, rather than full product planning and implementation.***
- ***Basing the quality of the result more on the tacit knowledge of the people in self-organized teams, than on the quality of the processes used.***
- ***Overlapping the different phases of the development, instead of performing them one after another in a sequential or waterfall cycle.***

This model was identified and defined by Ikujiro Nonaka and Hirotaka Takeuchi in the early 1980s, analysing how new products were developed by the major technology companies as: the Fuji-Xerox, Canon, Honda, Nec, Epson, Brother, 3M and Hewlett- Packard (Nonaka & Takeuchi, The New New Product Development Game, 1986).

In their study, Nonaka and Takeuchi compared the new form of teamwork that they were identifying with the scrum formation of Rugby players, and for that reason they called it "scrum".

Although this form of work arose in technological products companies, it is appropriate for projects with unstable requirements and for those that require speed and flexibility, frequent situations in the development of certain software systems but also in many other sectors and industries.

In 1995 Ken Schwaber presented "Scrum Development Process" in OOPSLA 95 (SCRUM Development Process), a framework of rules for software development, based on scrum principles, He had employed it in the development of Delphi and Jeff Sutherland in his company Easel Corporation (company that in a series of purchases and fusions, would integrate in VMARK, later in Informix and finally in Ascential Software Corporation).

What we mean now as "scrum"

1.- Rugby

Rugby scrum formation

Scrum is the term that defines the most recognizable formation of rugby, in which both teams, crouching and gripping each other, push to get the ball, and remove it from the formation without touching it by hand.

Illustration 1: Rugby scrum formation.

2.- Working methods

Agile self-organizing work environment

In 1986 the researchers Nonaka and Takeuchi give a polysemic dimension to the term scrum originally used in sports as rugby, when used it to baptize the principles of development that they discovered in the most innovative technological companies (Takeuchi & Nonaka, 1986).

Illustration 2: Scrum as a framework: 1986, Hirotaka Takeuchi, Ikujiro Nonaka "The New Product Development Game "

Scrum, in the original conception of Nonaka and Takeuchi, is characterized by the role of brilliant, self-organized and motivated teams, which approach the development of complex systems starting from a general vision and overlapping development phases.

Methodology for Software Development

In 1995 Ken Schwaber presented at OOPSLA (Schwaber, SCRUM Development Process, 1995) a software development methodology, based on a scrum environment and they used the same term to define the process.

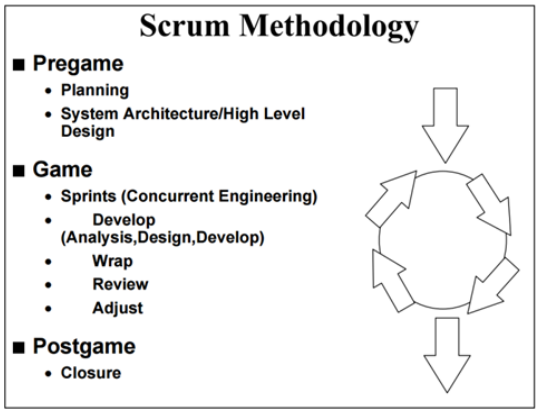


Illustration 3: Ken Schwaber, "Scrum Development Process"

Framework for software development based on the scrum methodology of Ken Schwaber

In 2005 Mike Cohn, Esther Derby and Ken Schwaber formed the "Scrum Alliance" organization to disseminate a specific framework for software development based on this methodology. It was also called scrum.

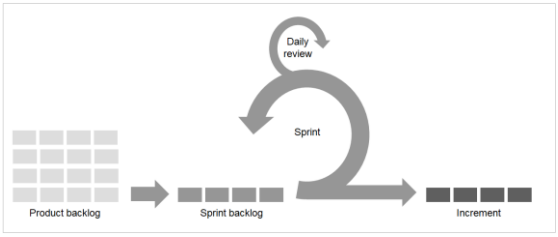


Illustration 4: Scrum framework.

Scrum Manager uses the term in the original meaning given by Nonaka and Takeuchi, as a work environment characterized by the composition of self-organized teams that work together in an agile way: with autonomy and overlap of development phases, sharing knowledge and learning in an open way

PART ONE

The Scrum rules.

Scrum: technical and advanced scrum

To start with scrum, it is advisable to adopt the standard framework: as it is explained in this first part, where you can find the roles, artefacts and events that configure its lifecycle model (see diagram page 20).

Once a continuous and iterative advance flow is achieved, you must consider if the goal is to go beyond what is a concurrent engineering model or to adopt scrum practices, or others that may be more appropriate to the characteristics of the project or team. It is time to unlearn standard practices, and rely on scrum values, rather than only in their technique.

This part of the book shows the basic scrum techniques: the application of rules and roles, and the events and artefacts that are used.

Technical Scrum

Rules

Software Development Rules Framework

Authors: Ken Schwaber y Jeff Sutherland
"Scrum Development Process OOPSLA'95" 1995

Application of defined rules

Roles

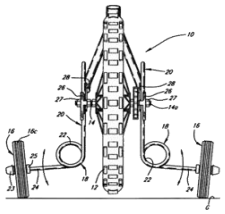
- Product owner
- Development team
- Scrum Master

Events

- Sprint
- Planning meeting
- Daily Scrum
- Sprint review
- Sprint retrospective

Artefacts

- Product backlog
- Sprint backlog
- Increment



To learn scrum rules

Advanced Scrum

Values

Original Scrum Concept

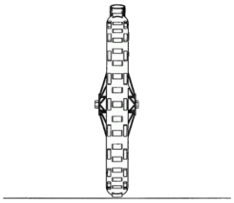
Authors: Hirotaka Takeuchi e Ikujiro Nonaka
"The New New Product Development Game" 1986

Application of agile values

- People > processes
- Outcome > documentation
- Collaboration > negotiation
- Change > planning

… "To advance in scrum"

- Uncertainty
- Self-organization
- Overlapping development phases
- Multilearning
- Subtle control
- Knowledge dissemination



To learn to advance in scrum without rules

Introduction to the technical framework

The technical framework of scrum is formed by a set of practices and rules that respond to the following principles of agile development:

- Evolutionary management of the product, instead of the traditional or predictive ones.
- Quality of the result based on the **people's tacit knowledge**, rather than on the explicit knowledge represented by the processes and the technology used.
- **Incremental development strategy through iterations** (sprints) .

It begins with the general vision of the desired result, and thereupon the functionalities that are desired to obtain in the first place are specified and details are given to build them.

Each cycle of development or iteration (sprint) ends with the delivery of an operative part of the product (increment). The duration of each sprint can be from one, up to six weeks, although it is recommended not to exceed one month.

In scrum, the team monitors the evolution of each sprint in brief daily meetings where the work done by each member on the previous day and the one planned for the current day are reviewed together. These daily meetings are closed for a maximum of 5-15 minutes, and are held standing next to a board or blackboard with the information of the tasks included in the sprint and the work pending on each of them by every team member. This meeting is called "standing meeting", "daily scrum", “stand-up meeting”, and: "daily scrum" or "morning rollcall".

Management of the evolution of the project

Scrum empirically manages the evolution of the project with the following tactics:

Review of Iterations

At the end of each sprint, the result is functionally reviewed, by all those involved in the project. It is therefore the duration of the sprint, the maximum period to discover errors, unverifiable, or misinterpretations in the functionalities of the product.

Incremental development

You do not work with designs or abstractions. Incremental development provides at the end of each iteration a piece of operational product that can be used, inspected and evaluated.
Scrum is suitable for projects with uncertain and / or unstable requirements.
Why predict the definitive version of something that is going to be evolving continuously? Scrum considers instability as a premise and adopts working techniques not only to facilitate the evolution without degrading the quality of the architecture, but also allowing it to evolve during development.
During construction, the design and the architecture are debugged, not being closed in the first phase of the project. The different phases that cascade development performs sequentially, in scrum will be overlapped and performed continuously and simultaneously.

Self-organization

There are many factors in a project that could be unpredictable. Predictive management assigns to the role of project manager the global responsibility for its management and resolution.
In scrum, the teams are self-organized, with a sufficient scope of decision to adopt resolutions they deem appropriate.

Collaboration

It is an important and necessary component so that the work can be reliably managed through the self-organization without the intervention of the project manager.
All members of the team collaborate openly with each other, in accordance to their abilities and not depending of their role or position.

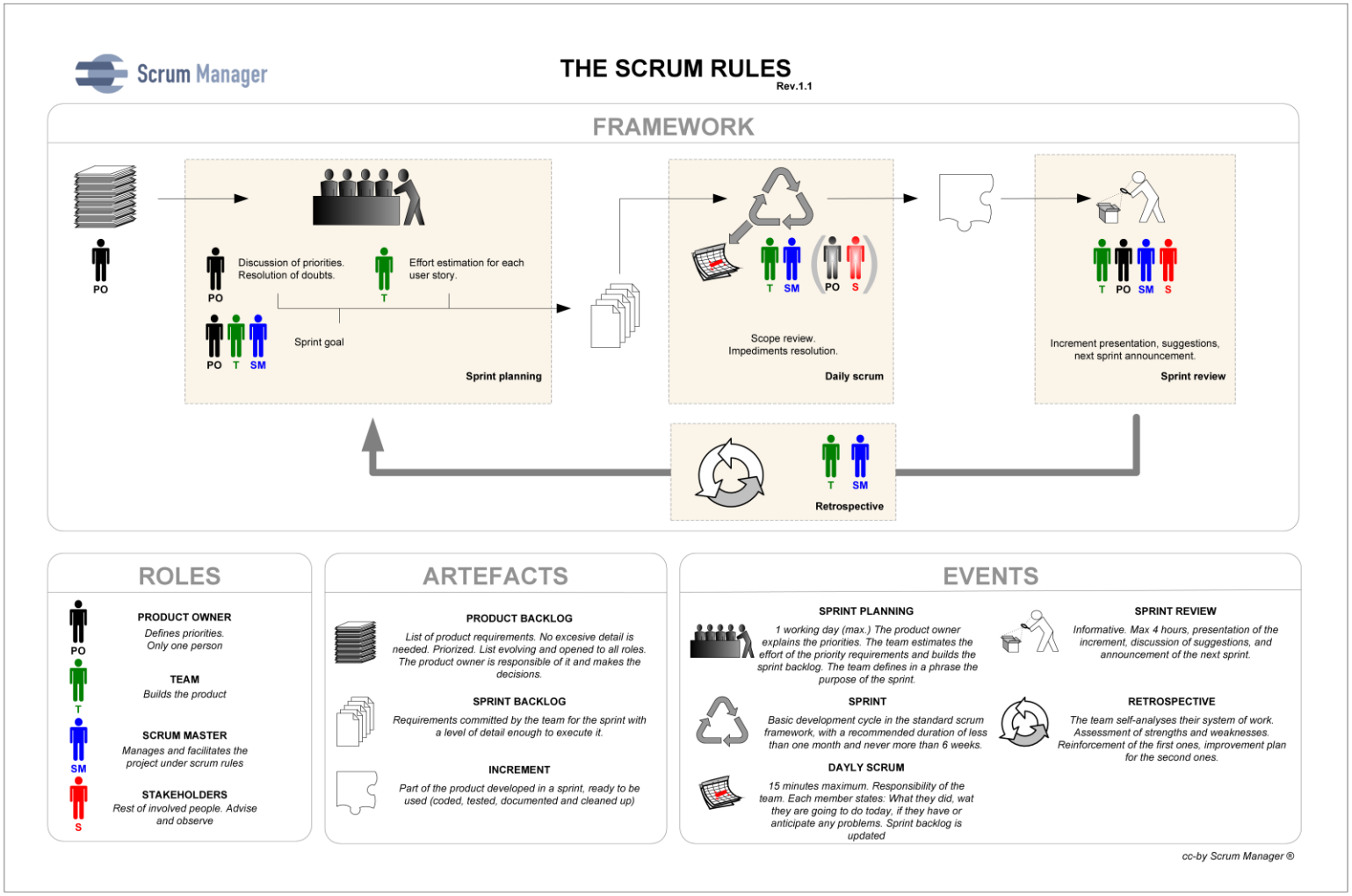


Illustration 5: Technical scrum framework.

Technical scrum

Technical scrum

The technical framework of scrum is formed by:

- ***Roles:***
  - The scrum team.
  - The product owner.
  - The Scrum Master.
- ***Artefacts:***
  - Product Backlog.
  - Sprint Backlog.
  - The increment.
- ***Events***
  - Sprint.
  - Sprint Planning Meeting.
  - Scrum daily.
  - Sprint review.
  - Sprint retrospective.

And the key is the sprint.

It is called sprint to each cycle or iteration of work that produces a significant part of the product finished and functionally operative (increment)

As will be seen later, when dealing with advanced scrum, more flexible scrum implementations may adopt two different tactics to maintain a continuous advance in the project:

- ***Iterative increment: based on time-based pulses (timeboxing)***
- ***Continuous increment: based on the maintenance of a continuous flow, not marked by pulses or sprints.***

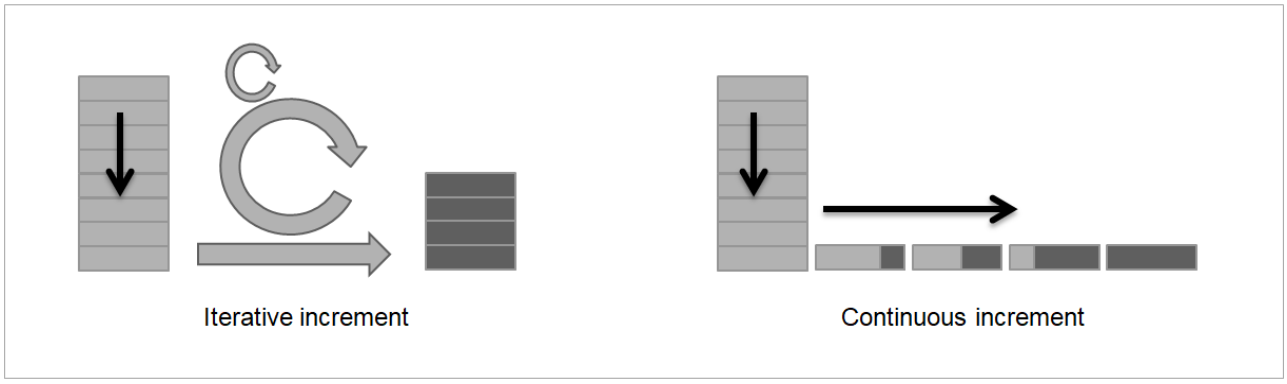


Illustration 6: Iterative / continuous Increment.

Technical scrum works with pre-established time pulses that are called sprints. It therefore uses iterative increase to maintain a constant rate of progress.

Artefacts

- Product backlog: list of user requirements, which from the initial vision of the product grows and evolves during development.
- Sprint backlog: list of tasks that the team must perform during the present sprint to generate the expected increase.
- Increment: result of each sprint.

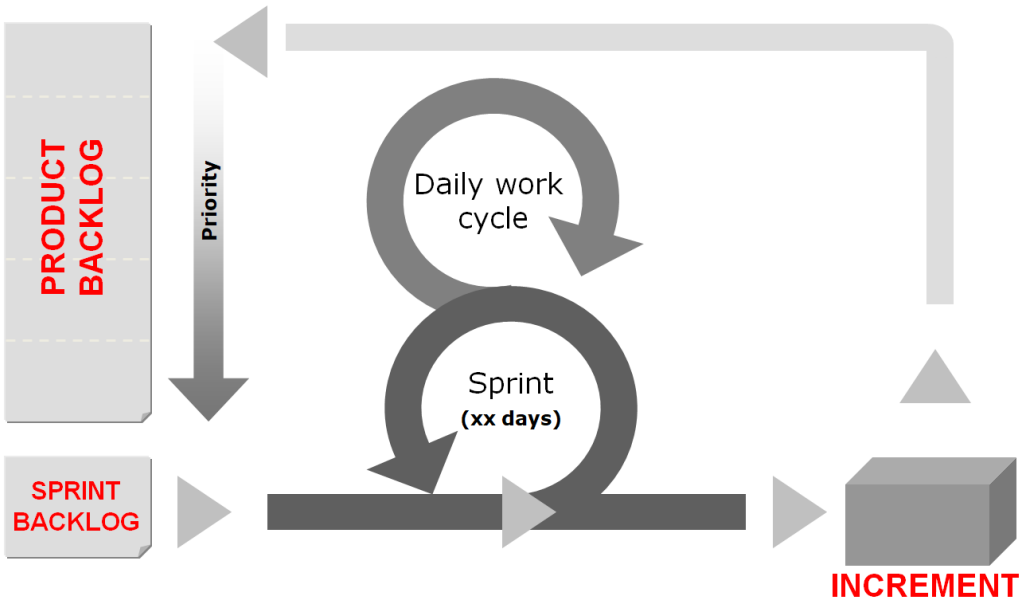


Illustration 7: Diagram of the scrum iterative cycle.

Another artefact typical of the standard scrum model is the advance graphic or burn down chart that the team updates daily to check its progress. This item, along with poker estimation practice and the burn up or product chart is included in the Agile Metrics chapter (page 47)

Product backlog and sprint backlog: the requirements management in agile development.

In traditional software engineering, the system requirements are part of the acquisition process, and it is therefore the responsibility of the customer to define the problem and the functionalities that the solution must provide.

It does not matter if it is traditional or agile management. The product backlog is the responsibility of the customer, although it is dealt with a different approach in each case.

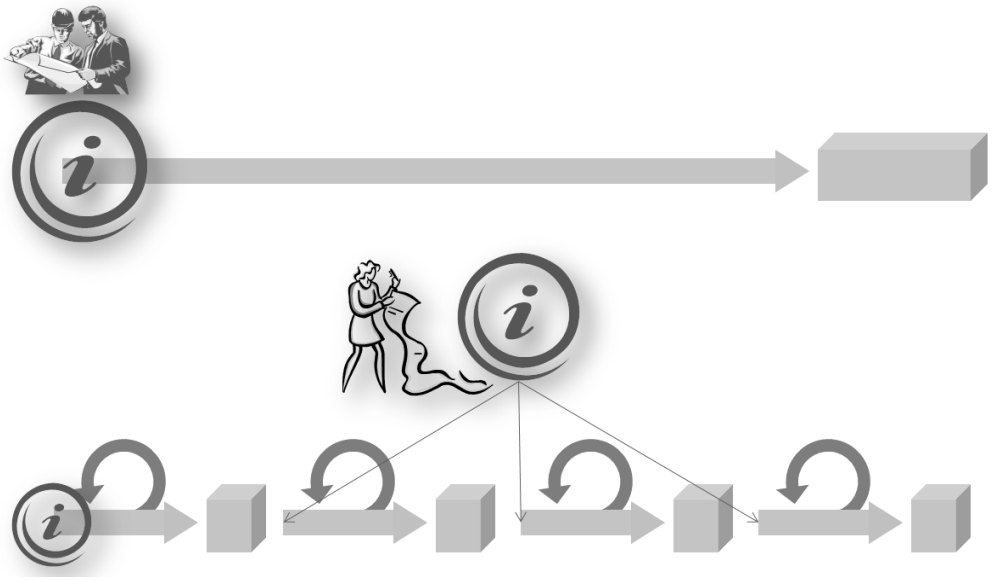


Illustration 8: Traditional / Evolutionary Requirements.

- In predictive projects, system requirements are usually specified in formal documents; While in agile projects they take the form of a product backlog or list of user stories.
- Formal system requirements are specified completely and closed at the beginning of the project; However a product backlog is a living document, which evolves during development.
- The requirements of the system are developed by a person or team specialized in requirements engineering through the process of obtaining (elicitation) with the customer. In scrum the customer (product owner) shares his vision with the whole team, and the product backlog is built and evolves continuously with the contributions of all the project members .

Scrum, uses two formats to record the requirements:

- Product Backlog
- Sprint Backlog

The product backlog records the requirements seen from the customer's point of view. An approach like that of the system requirements or "ConOps" of the traditional software engineering. It consists of the list of functionalities or "user stories" that the client wants to obtain, ordered by the priority given by him to each one.

The sprint backlog reflects the requirements seen from the point of view of the development team. It consists of the list of tasks in which the user stories that must be carried out in the sprint are broken down.
In the development and maintenance of the product backlog, the most important thing is not the format, but to give attention to:

- The user stories, which include a defined and well-known view of the product by the whole team expressed from a subjective point of view, related with one of the groups of users identified
- User stories are individually defined, prioritized, and pre-estimated.
- The product backlog is carried out and managed by the customer (product owner).

Product Backlog: Customer Requirements

The product backlog is the ordered list of everything that the product owner believes the product needs.

It is the inventory of functionalities, improvements, technology and correction of errors that must be incorporated to the product through the successive sprints.
It represents everything that the customer, the users, and in general the interested parties expect. Everything that involves a task that must be done by the team must be reflected in this backlog.

Some examples of possible inputs to a product backlog could be:

- Offer to the users the consultation of the works published by a certain author.
- Reduce the time of installation of the program.
- Provide the query of a work through a web API.

The product backlog is never completed; It is in continuous growth and evolution. At the beginning of the project includes the requirements initially known and better understood, and evolves as development progresses.

Thanks to its dynamic nature it reflects what the product needs to incorporate in order to suit the circumstances, at any time.

Before starting to iterate the product it is necessary to:

- Let the product owner have the vision of the business objective he wants to achieve, and he must share it with the team.
- Have enough user stories within the product backlog to perform the first sprint.

Usually the product backlog is created as the result of a brainstorming session, or "cross-fertilization", or an "eXtreme Programming" exploration process where the whole team collaborates, understanding and sharing the Vision of the product owner.

The product owner maintains the product backlog ordered by the priority of the elements, being the highest priority those that give more value to the product, or for some reason are more necessary, and determine the immediate development activities.

The degree of concretion of the user stories in the product backlog should be proportional to the priority: The highest priority should have sufficient detail to be able to decompose into tasks and move on to the next sprint.

The items of the product backlog that can be incorporated into a sprint are called "prepared" or "actionable" and are those that can be selected at the sprint planning meeting.

Preparing the product backlog

It is called “refinement” or "grooming" or preparation of the product backlog to the activities of prioritization, detail and estimation of the items that compose it. It is a process that is performed when is timely, at any time, in a continuous and collaborative way by the product owner and the development team. It should not consume more than 10% of the working capacity of the team.

The responsibility of estimating the foreseeable effort for each item corresponds to the team members who must do the work.

Format of the product backlog

Scrum prefers verbal or direct way to writing communication. The product backlog is not a requirements document, but an information tool for the team.

If a list format is used, the minimum information that is usually included for each user story is:

- Description of the functionality / requirement, called "user story".
- Priority assigned.
- Prediction of the effort required.

And sometimes also a code or unique identifier of the user story.

Due to the characteristics of the project or the team, additional information can be included such as:

- Observations, comments and remarks
- Validation criteria.
- Person assigned.
- Sprint Number in which it is planned to be performed.
- System module or area to which it belongs ...

An example of the format that a product backlog might have:

Id	Priority	Description	Est.
1	Very High	Technology platform	30
2	Very High	User Interface	40
3	Very High	An user registers into the system	40
4	High	The operator defines the flow and text constants of a file	60
5	High	Xxx	999

Illustration 9: Product backlog example.

Sprint Backlog

The sprint backlog is the list of tasks needed to build the user stories to be performed in a sprint.
It is made by the team at the sprint planning meeting, indicating for each task the planned effort to do it. To calculate the effort of each task (in points or ideal time, see p. 40), it is common to use techniques such as poker estimation (page 49).

The sprint backlog breaks down user stories into units of adequate size to monitor progress on a daily basis, and identify risks and problems without complex management processes.
It is also a tool for direct visual communication of the team.

Conditions

- Performed jointly by all team members.
- Covers all tasks identified by the team to achieve the sprint goal.
- Only the team can modify it during the sprint.
- Too large tasks should be broken down into other tasks. In no case can a task be of such a size that it needs more than a day's work.
- It is visible to the whole team. Ideally on a board or wall in the same physical space where the team works.

Format and support

They are usual supports:

- Physical board or wall.
- Spreadsheet.
- Collaborative or project management tool.

And according to the project and team characteristics, it is appropriate to design the most comfortable format, considering the following criteria:

- Include the following information: Backlog of the sprint, person responsible for each task, state in which it is and remaining work time to complete it.
- Include only the strictly necessary information.
- It should serve as a means to record at each daily sprint meeting the time remaining for each task.
- Facilitate consultation and daily and direct communication of the team.

Example:

	B	C	D	E	F	G
	Details					
	Epic	**Sprint**	**User Story Role**	**User Story Name**	**Story Details**	**Acceptance Test Criteria**
	New Art Gallery Website	1	Gallery Owner	Create web page for each artist	There are 20 artists currently exhibiting at the gallery. Each artist should be provided 1 page to showcase their art	1. 20 web pages must be on the website 2. Each artist must be listed on the page with their bio 3. At least one piece of art should be described and a picture included on each page
	New Art Gallery Website	1	Gallery Owner	Create main Gallery Webpage	New gallery wants to build a community between the artists, the gallery & the customers.	1. The main page of the website should describe the gallery and its mission. 2. The main page of the website should have a link to each of the artist pages that are currently exhibiting at the gallery
	New Art Gallery Website	2	Gallery Owner	Artist & Customer communication	Customers should be able to chat directly with artists to learn more about any art pieces they may be interested in.	1. Customers should be able to send a message to the artist. There is no requirement that the artist respond immediately. 2. Artists must have a way to respond to

Illustration 10: Example of sprint backlog on worksheet.

During the sprint, the team daily updates the pending effort for each task. At the same time, with these data, they trace the burn-down graph, described later, in the agile metrics chapter.

The increment

The increment is the part of product produced in a sprint, and has the characteristic of being completely finished and operative, able to be delivered to the customer.
Prototypes, modules or sub-modules, or pending parts of tests or integration should not be considered as an increment.

Ideally in scrum:

- Each item of the product backlog refers to deliverable functionalities, not internal works of the "database design" type.
- There is an "increment" in each iteration.

However, it is a common exception to the first sprint, which is often referred as "sprint 0". It can include "verify platform and design" objectives, which are necessary when starting some projects, they involve design or prototype development to validate the expectations of the platform or technology to be used.
Taking into consideration this usual exception

> Increment is the part of product realized in a potentially deliverable sprint:
> **finished and tested**.

If the project or system requires documentation, or documented validation and verification processes, or levels of independence involving third-party processes, these must also be performed to consider that the increment is "done."

Events

- **Sprint**: name that receives each development iteration. It is the central nucleus that generates the pulse of advance to the rhythm of "predetermined times" (time boxing).
- **Sprint Planning Meeting**: A working meeting that marks the beginning of each sprint, determining the purpose of the sprint and the tasks necessary to achieve it.
- **Daily Scrum**: brief daily team meeting, in which each member answers three questions:
  1.- The work done the day before.
  2.- The work planned to perform.
  3.- Things he may need, or impediments that must be eliminated to get the job done.
  Each person updates in the sprint backlog the time or effort pending of their tasks, and with this information, the graph with which the team monitors sprint progress (see page 47) (burn-down)
- **Sprint review:** analysis and inspection of the increase generated, and adaptation of the product backlog if necessary.

A fourth meeting was incorporated into the standard scrum framework in the first decade of 2000:

- **Sprint Retrospective**: review of what happened during Sprint. Meeting in which the team analyses operational aspects of the way of work and creates a plan of improvements to apply in the next sprint.

Sprint

The key scrum event to maintain a continuous pace of advancement is the sprint: the time-box period with a maximum duration of 4 weeks, during which an increment of the product is built.
The increment made during the sprint must be finished, that is being completely operational and useful for the client, in conditions to be deployed or distributed.

Scope of the sprint

When starting to work with scrum it is advisable to consider the sprint as the event container for all events: the sprint planning meeting, the daily scrum, the sprint review and the retrospective.
In this way, in addition to setting a daily rhythm of progress and visibility of the tasks (daily scrum), it also marks a fixed rhythm to check the progress and visibility of the product (planning and revision of the sprint), which is also the same to give visibility and a point of reflection and improvement to the mode of work (retrospective).

In advanced scrum implementations, it is possible to consider however that the scope of the sprint is only the construction of the increment, leaving out the planning and the review meetings of the sprint, and the retrospective.

The implications of narrowing the scope, and for which the team may be interested, are:

- To calculate the speed of the sprint considering only the working time, not including the meetings of beginning, closing and retrospective
- To have more flexibility to perform sprints of different durations, and
- To separate the frequency of retrospectives from that of sprints.

Sprint planning

Description

This meeting is based on the client's business priorities and needs, and determines what and how the features will be incorporated into the product in the next sprint.

This is a meeting led by the person in charge of the operation of the scrum framework (Scrum Master in technical scrum, or a member of the team, in advanced scrum) to which the product owner and the complete team must attend, and any other people involved in the project can assist as well.

The meeting can last up to a full working day, depending on the volume or complexity of the user stories that are wanted to be included in the next increment.

This meeting should address two issues:

- What will be given at the end of the sprint.
- Which is the work required to carry out the planned increment, and how the team will be carried out.

The meeting is organized in two parts of similar duration, whose aim is to answer one of these questions in each one.

Preconditions

- The organization has determined and assigned the resources available to carry out the sprint.
- The highest priority user stories in the product backlog are already "prepared", so they already have a sufficient level of accuracy and a prior estimate of the work they require.
- The team has a knowledge of the technologies used, and the product business sufficient to make estimates using his expert judgment, and to understand the business concepts that the product owner exposes.

Inputs

- The product backlog with the collection of user stories groomed.
- The product already developed in the previous increments (except for the first sprint).
- Team speed or performance data in the last sprint, which is used as a criterion to estimate the amount of work that is reasonably expected for the next sprint.
- Circumstances of the client's business conditions and the technological scenario employed.

Results

- Sprint backlog.
- Duration of the sprint and date of the review meeting.
- Purpose and objectives of the sprint.

Meeting format

This meeting marks the beginning of each sprint.
Maximum duration: one day.
Attendants: Product owner, development team and Scrum Master.
Others who can attend: all those who provide useful information, since it is an open meeting.
It consists of two parts separated by a pause (coffee break or lunch, depending on the duration).

First part: What will be given at the end of the sprint.

The product owner introduces the product backlog, exposing the highest priority user stories he needs and expects to be developed in the next sprint. If the product backlog has had significant changes since the previous meeting, it explains the causes that have caused them.
The goal is that the whole team knows the right level of detail and would be able to understand the work of the sprint.

Product Owner:

- Presents the user stories of the product backlog that have the highest priority and that can be done in the sprint.
- The presentation is made with a level of detail sufficient to transmit to the team all the information necessary to build the increment.

The team

- Ask the questions and ask for the necessary clarifications.
- Proposes suggestions, modifications and alternative solutions.

The inputs of the team may involve modifications to the product backlog or its user stories.
This meeting is a scrum hot spot to favour cross-fertilization of the team ideas and to add value to the original vision of the product.

After rearranging and rethinking the stories of the product backlog, the team defines the "sprint goal," which is the phrase that synthesizes the value that will be delivered to the customer.

With the exception of sprints dedicated to cluttered task collections, the elaboration of this motto together in the meeting is a guarantee that all the team understands and shares the purpose of the work, and during the sprint it serves as a reference criterion in decisions that the team self-manage.

Part Two: How you will get done the increment.

The team breaks down each functionality into tasks, and estimates the effort for each of them, thus composing the tasks that make up the sprint backlog. In this breakdown, the team takes into account the elements of design and architecture that the system must incorporate.

Team members establish what the tasks will be for the first few days of the sprint, and self-assign them based on their knowledge, interests and a homogenous distribution of work.

This second part should be considered as a "team meeting", in which all its members must be present, and they are the ones who decompose the estimation and assign the work.

The role of the product owner is to answer questions and verify that the team understands and shares its purpose.

The Scrum Master acts as moderator of the meeting.

Functions of the Scrum Master

The Scrum Master, or the moderator of the meeting, is responsible and guarantor for:

1. Holding this meeting before each sprint.
2. Ensuring that there is a product backlog prepared by the product owner.
3. Helping to maintain the dialogue between the product owner and the team.
4. Ensuring that an agreement is reached between the product owner and the team regarding what will include the increment.
5. Helping the team to understand the vision and the business needs of the client.

6. Ensuring that the team has performed a realistic decomposition and estimation of the work, and has considered the possible tasks of analysis, research or support.
7. Ensuring that at the end of the meeting are objectively determined:
   - The elements of the product backlog to be executed.
   - The purpose of the sprint.
   - The sprint backlog with all estimated tasks.
   - The duration of the sprint and the date of the review meeting.

Example of operational board for the meeting

It is recommended that the product owner use a spreadsheet or a similar tool to digitally store the product backlog. Although it is not advisable to use it as a basis to work on it at the meeting.

At the meeting it is preferable to use a blackboard or a board and removable tokens or labels.
The board facilitates communication and teamwork during the meeting.

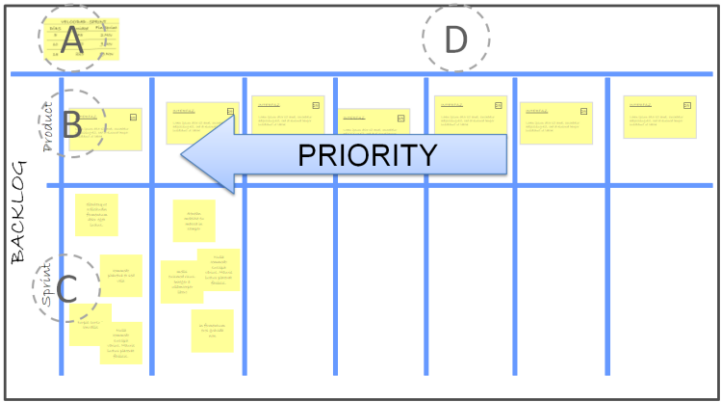


Illustration 11: Example of working board.

Some common supports:

- Whiteboard and Post-it® type adhesive notes.
- Laminated corkboard and thumbtacks to hold notes.
- Vitrified steel panel and magnetic supports to hold notes.

With removable tape different areas are marked to delimit:

- An upper area where the team notes:
  - (A) the units of work that according to the average speed of the team could be made in sprints of 2, 3, 4 and 5 weeks.
  - (D) Duration that the sprint will finally have, as well as the target set, duration, time set for the daily meetings and scheduled date for the sprint review meeting.
- B.- A strip to sort the elements of the product pile from highest to lowest priority.
- C.-A parallel strip to break down each element of the product backlog into the corresponding tasks of the sprint backlog.

Daily Scrum

Description

Short daily meeting, no more than 15 minutes standing, in which the team synchronizes the work and establishes the plan for the next 24 hours.

Inputs

- Up-to-date sprint backlog and burn-down graph with information from previous meeting.
- Information of the progress of each team member.

Results

- Updated sprint backlog and burn-down graph.
- Identification of possible needs and impediments.

Meeting format

It is recommended to be performed standing next to a board with the sprint backlog and the sprint advance graphic, so everyone can share the information and the score.
At the meeting the whole team is present, and other people related to the project or organization can also attend, although they cannot participate.

At this meeting each member of the development team explains:

- What he has achieved since the previous daily scrum.
- What he's going to do until the next day's daily scrum.
- If he is having any problems, or if he foresee that he may encounter some impediment.

And it updates on the sprint backlog the stiffness that it estimates pending in the tasks assigned to him, or he marks as finalized those already completed.
At the end of the meeting:

- The team refreshes the sprint's advance chart, with updated estimations of pending work,
- The Scrum Master takes the appropriate steps to solve identified needs or impediments.

The team is responsible for this meeting, not the Scrum Master; and it is not an "inspection" or "control" meeting, but a communication within the team to share the status of the work, check the progress and collaborate on possible difficulties or impediments.

Sprint review

Description

Meeting held at the end of the sprint to check the increment.
It should not last more than 4 hours, in the case of reviewing long sprints, but for regular sprints should be enough with one or two hours of duration.

Goals:

- The product owner checks the progress of the system. This meeting marks, at regular intervals, the rhythm of construction, and the path that is taking the vision of the product.
- The product owner identifies user stories that can be considered "made" and those that cannot.
- When reviewing and testing the increment, the product owner, and the whole team obtain relevant feedback to check the product backlog.

- Other engineers and programmers of the company can also attend to know how the technology used works.

Preconditions

- The sprint is concluded as planned.
- The entire development team, the product owner, the Scrum Master and all the people involved in the project who wish to do so are present.

Inputs

- The finished and deployable increment.

Results

- Feedback for the product owner: milestone tracking system construction, and information to improve the value of the product vision.
- Call for the next sprint meeting.

Meeting format

It's an informal meeting. The goal is to see the increment built. Graphics presentations and power points are forbidden.

The team should not invest more than an hour in developing the meeting, and what is shown is the final result: finished, tested and operating in the client environment (increment).

Depending on the characteristics of the project, it may also include user documentation, or technical documentation.

It's an informational meeting**. Its mission is not to make decisions or criticisms about the increment**. With the information obtained, the product owner will subsequently discuss possible changes to the product vision.

Recommended protocol:

1. The team explains the purpose of the sprint, the list of functionalities that were included and those that have been developed.
2. The team makes a general introduction of the sprint and demonstrates the performance of the built parts.
3. A round of questions and suggestions is opened. This part generates valuable information so that the product owner and the team in general, can improve the vision of the product.
4. The Scrum Master, in accordance with the agendas of the product owner and the team, closes the date for the next sprint preparation meeting.

Retrospective

Meeting that takes place after the review meeting of each sprint, and before the planning meeting of the next, with a recommended duration of one to three hours, depending on the duration of the sprint.
Within it the team performs his own self-analysis about its way of working, and identifies strengths and weaknesses. The objective is to consolidate the positive factors, and plan improvement actions on the negative.
The fact that it was performed normally at the end of each sprint sometimes leads to mistakenly being considered as "sprint review" meetings, when it is advisable to treat them separately, because their goals are different.
The purpose of sprint review is to analyse "WHAT" is being built, while a retrospective meeting focuses on "HOW" we are constructing: "HOW" we are working, with the aim of analysing problems and improving aspects.
The "retrospective" meetings held periodically by the team to improve the way they work are increasingly considered as a component of the scrum technical framework, although it is not a meeting to follow the evolution of the product, but rather to improve the Framework.

Roles

All people involved, with a direct or indirect relationship with the project, are classified into two groups: committed and involved. In scrum circles it is often called the first (without any pejorative connotation) "pigs" and the second "hens".
The origin of these names is in the following metaphor that graphically illustrates the difference between "commitment" and "involvement" in the project:

> *A hen and a pig walked along the road. The hen asked the pig, "Do you want to open a restaurant with me?"*
> *The pig considered the proposal and replied: "Yes, I would like to. And what would we call it? "*
> *The hen answered: "eggs with ham".*
> *The pig stopped, paused and replied, "On second thought, I do not think I'm going to open a restaurant with you. I would be really engaged, while you would only will be only involved. "*

<table>
<tr><th>COMMITTED (Pigs)</th><th>INVOLVED (Hens)</th></tr>
<tr><td>Product Owner</td><td rowspan="2">Other stakeholders (Direction, management, vendors, marketing, etc.)</td></tr>
<tr><td>Team members</td></tr>
</table>

Illustration 12: Scrum standard roles.

- **Product owner**: is the person responsible for achieving the highest product value for customers, users and other people involved.
- **Development team**: group or working groups that develop the product.

An observation on this point, on the role of Scrum Master, for being often frequent the doubt to consider if it is a "committed" or "involved" role.

Based on the fact that the division between committed people and people involved is more "conceptual" than "relevant", but when working with this present role, his responsibility is the functioning of the technical scrum framework in the organization.

His direct responsibility, his mission, is therefore the form of work, being the product elaborated as a second level objective, or indirect.

For this reason in the previous table the role of Scrum Master is not considered, although in any case it is not an especially relevant question. If one were to force a response, from the criterion that it is not engaged in the project (but rather in the improvement of the way of working), it should be considered as an "involved"

Product Owner

The product owner is the decision maker of the customer. His responsibility is to maximize the value given to the product.
In order to simplify communication and decision making, this role must fall on a single person.
If the client is a large organization, or with several departments, it can take the form of internal communication that they consider appropriate, but only one person representing the client is integrated in the development team and the client must have sufficient knowledge of the product and the necessary power and authority to take the decisions that correspond to this role.
In summary, the product owner is who:

- Decides ultimately how the final result will be, and the order in which successive increments are constructed: what is put in and what is removed from the product backlog, and which is the priority of user stories.
- Knows the plan of the product, its possibilities and investment plan, as well as the expected return on the investment made, and is responsible for dates and functionalities of the different versions of the same.

In the internal developments for the company itself, the product manager or the marketing manager usually takes on this role.
In developments for external clients, the person responsible for the customer acquisition process takes it on.
Depending on the circumstances of the project, product owner may even delegate his duties to the development team, or to someone he trusts, but responsibility is always his.

To exercise this role it is necessary to:

- **Know perfectly the business environment of the client**, the needs and the objective that is pursued with the system that is being built.
- Have the **vision of the product**, as well as the concrete needs of the project, in order to efficiently prioritize the work.
- Have sufficient **authority and knowledge over the product plan** to make the necessary decisions during the project, including to cover expected project return.
- Receive and analyse continuously **feedback of the business environment** (market evolution, competition, alternatives) and of the project (team suggestions, technical alternatives, tests and evaluation of each increment).

It is also recommended that the product owner:

- Has a knowledge of scrum enough for solving tasks as:
    - Development and management of the product backlog.
    - Exposition of vision and user stories, and participation in the planning meeting of each sprint.
- Knows and had previously worked with the same team.

The organization must respect its decisions and not modify priorities or elements of the product backlog.

Development team

It is formed by the group of professionals who perform the increment of each sprint.
It is recommended that a scrum team has no less than 3 or more than 9 people. Beyond 9 it is difficult to maintain direct communication, and the habitual frictions of group dynamics (which begin to appear from 6 people) are more strongly manifested. In the count of the number of members of the development team neither the Scrum Master nor the product owner are considered.

It is not a work group formed by an architect, designer or analyst, programmers and testers. It is a multifunctional team, in which all members work in solidarity with shared responsibility. It is possible that some members are specialists in specific areas, but their responsibility is the increment of each sprint and is taken on by the development team as a whole.
The main responsibilities, beyond the self-organization and use of agile technologies, are those that make the difference between "work group" and "team".
A working group is a set of people who perform a job, with a specific assignment of tasks, responsibilities and following a process or implementation guidelines. The operators of a chain form a working group: although they have a common boss, and work in the same organization, each one is responsible for his own work.
A team has a spirit of collaboration and a common purpose: achieving the highest possible value for the client's vision.
A scrum team responds as a whole. It works in a cohesive and self-organized way. There is no manager to delimit, assign and coordinate tasks. The members themselves perform that.
In the team:

- Everyone knows and understands the vision of the product owner.
- Contribute and collaborate with the product owner in the development of the product backlog.
- Share together the purpose of each sprint and the responsibility for achievement.
- All members participate in decision-making.
- The opinions and contributions of all are respected.
- Everyone knows scrum.

Scrum Master

He is responsible for compliance with the rules of a technical scrum framework, ensuring that they are understood in the organization, and work according to them.
He provides the necessary advice and training to the product owner and the team.
He performs his work with a servile leadership model: to the service and to the help of the team and the owner of the product.
He provides:

- Counselling and training to the team, in order to work self-organized and with team responsibility.
- Review and validation of the product backlog.
- Moderation of meetings.
- Resolution of the impediments that in the sprint can hinder the execution of the tasks.
- Management of group dynamics difficulties that can be general in the team.
- Configuration, design and continuous improvement of the scrum practices in the organization.
- Respect for the organization and those involved, for the patterns of times and forms of scrum.

As the organizational fluency grows and evolves into a more advanced scrum framework, the role of the Scrum Master may not be necessary when these responsibilities are already institutionalized in the organization.

Scrum Values and Principles

Technical Scrum defines a framework that helps to organize people and workflow. It is the "body" or the visible interface, but the agile values are the authentic agility engine the principles that govern their way of working.
The rules of a scrum team may be those of this technical framework or others. Agility is not provided by the fulfilment of practices, but by values.

- **Respect between people**. Team members must trust each other and respect their knowledge and skills.
- **Responsibility and self-discipline** (no discipline imposed).
- Focused and **value-oriented work for the client**.
- **Commitment.**

And the principles that govern their way of working are:

- **Empowerment** to the team and its members so they can self-organize and make decisions about development.
- Information, **transparency** and visibility of project development.
- Frequent **inspection and adaptation** to detect deviations and make necessary adjustments.

Agile measurement and estimation

Why measure?

Information is the raw material for decision-making, and the one that can be quantified, provides objective management and monitoring criteria.

From the specific level of programming, to the most general of the overall organizational management, there are three scales or zoom levels with which the work can be measured:

- Development and management of the technical solution.
- Project management.
- Management of the organization.

In the first one, it is possible to measure, for example, the proportion of polymorphism in the code of a program, in the second, the percentage of the project plan carried out, and in the third, the level of job satisfaction.

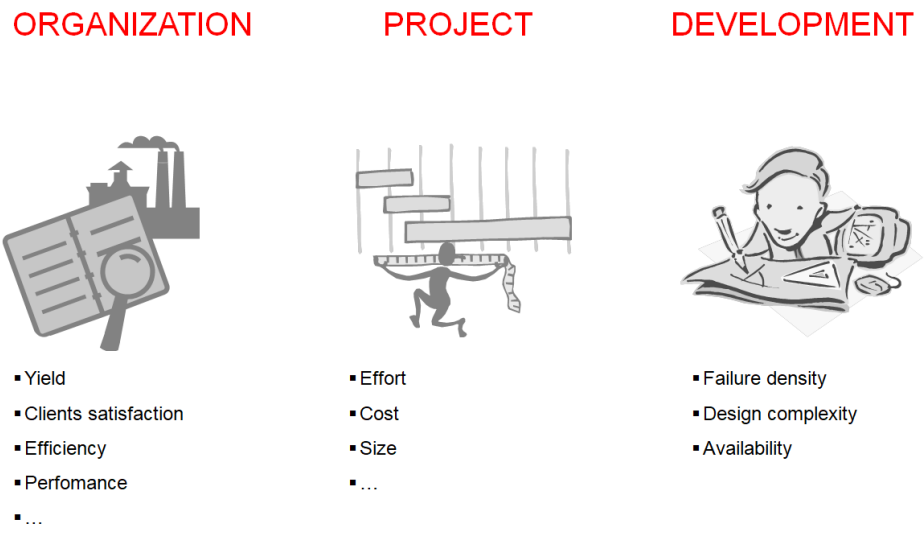


Illustration 13: Measurement areas.

This text covers the agile measurement in the project scope, although the general considerations of this introduction are common to all three.

Flexibility and common sense

> Measuring is expensive

Before incorporating a measurement procedure, you should question its suitability, and the way in which it will be applied.

> Measuring cannot be an end itself.

Measuring processes should not be implemented simply by measuring.

Criteria for the design and application of metrics

The fewer the better.

- Measuring is expensive.
- Measuring adds bureaucracy.
- The goal of a scrum team is to offer the best value / cost ratio.

Issues to be questioned before monitoring and measuring an indicator:

- **Why?**
- **What is the value of this measurement?**
- **How much value is lost if omitted?**

The goal of scrum is to produce the highest possible value continuously, and the key issue for incorporating indicators into project management is:

How does the use of the indicator contribute to the value that the project provides to the client?

Is the indicator appropriate for the purpose to be achieved?

Measuring is not, or should not be, an end itself.
What is the end?
Meeting agendas, improving team efficiency, forecasts...?
Be critical. If, after analysing it, you are not convinced, if you prefer not to incorporate an indicator, or change it: you are the manager, you decide it.
To determine what to measure is the hardest part. In the best-case scenario, erroneous decisions will only lead to avoidable management costs; but they can also worsen what is intended to be improved.

Example: Measuring the efficiency of programming jobs

The XYZ organization has adopted standard metrics for Software Engineering efficiency:

- LOC / Hour: Total number of new or modified lines of code in each hour.

To increase productivity, the performance paid to programmers has been linked to the results of this metric, so that it has managed to produce more lines of code without increasing the workforce.

To avoid being a "hollow" increment of lines of code, or an increase in the number of errors to program faster, the metric also incorporates:

- Test Defects / KLOC,
- Compile Defects / KLOC and
- Total Defects / KLOC, to control that the number of errors in the code does not grow, and
- "appraisal time" indicators to measure time and costs of designing and executing code reviews.

And for fear that the measurement system may prove to be excessively costly, quality cost indicators (COQ) are included, which measure the review times and contrast them with the improvements in times eliminated by failure reduction.

What will we measure is a valid indicator of what we want to know?
There are relatively mechanical programming tasks, oriented more to the integration and configuration than to the development of new systems.
For those, it may be reasonably accurate to consider efficiency as the volume of work performed per unit of time.

For the latter, however, it is more appropriate to think of the amount of integrated value per unit of development; Expressed in hours, iterations or function points.

What do we want to know: the number of lines, or the value delivered to the customer?
Are they related to one another?
Can the value delivered to the customer be measured objectively?
In our work environment, there are many parameters that can be measured with objective and quantifiable criteria: task time, delta and interrupt times, number of function points, requirements instability, coupling ratio, the number of errors per line of code...

Are we measuring this often simply because it is quantifiable and easy to calculate?
Are we measuring the number of lines that people develop when we really want to know the value of their work?
Is it happening the same when we try to measure: ease of use, ease of maintenance, flexibility, transportability, complexity, etc.?

Speed, work and time

Speed, work and time are the three variables that make up the formula of speed in agile project management, defining it as the amount of work done per unit of time.

$$Speed = Work / Time$$

For example, it can be said that the speed of a team of 4 members is 20 points per week or 80 points per sprint.

Time

To maintain a continuous pace of progress, agile development employs two possible tactics: iterative increment, or continuous increment.

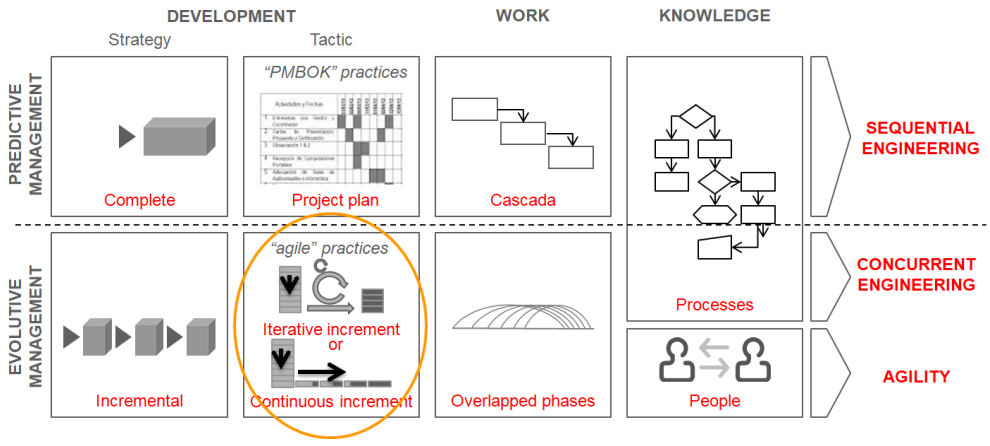


Illustration 14: Agile management with iterative or continuous increment.

Progress through iterative increments keeps pace by leaning on sprint pulses. For this reason, it usually uses sprint as a unit of time, and expresses the speed as work or tasks performed in a sprint.

Note: technical scrum uses iterative increment, and therefore defines the speed as the amount of work done in a sprint.
Progress through a continuous increment maintains a constant forward flow without dead points or bottlenecks. There are no sprints, and therefore the units of time are days, weeks or months, so that the speed is expressed in "points" (amount of work) per week, day, or month

Real time and ideal time

Illustration 15: Ideal time and real time.

An important observation: the difference between "real" time and "ideal" time.
Real time is the working time. Equivalent to the working day.
For a team of four people with an eight-hour working day, the actual time in one week (five working days) is:
4 * 8 * 5 = 160 hours

Ideal time, however, refers to working time under ideal conditions, that is, eliminating everything that is not strictly "work", if there are not interruptions, pauses or attention to non-task issues and that the person is in his best conditions of concentration and availability.
The ideal time is usually used in estimates, such as unit of work or effort required. Ex: "That task has a size of 3 ideal hours".
It is a concept similar to what PSP[1] calls "Delta Time" as the part of working time that is actually effective working time.

> Ideal time is not a unit of time, but of work or effort required.

Work

Measuring the work may be necessary for two reasons: to record the work already done, or to estimate in advance, what should be done.
In both cases a unit, and an objective criterion of quantification is required.

Work already done

Measuring the work already done is not particularly difficult.
It can be done with units related to the product (e.g. lines of code) or the resources used (cost, working time...).
To measure it, you need just count what has already been done with the unit used: lines of code, function points, hours worked, etc.
Agile project management does not measure the work already done to calculate the progress of the work by subtracting it from the expected time

> Agile management does not determine the degree of progress of the project for the work done, but for the pending. It results a better predictor of the effective progress

Other business processes in the organization may need to record the effort invested, and therefore it need to be recorded, but should not be used to calculate project progress.

Work to be done

Scrum measures pending work to:

- Estimate the effort and time needed to do a job (tasks, user stories or epics).
- Determine the progress of the project, and especially in each sprint.

[1] Personal Software Process

Determining precisely, quantitatively and objectively the work that will need to build a requirement, is a questionable endeavour.

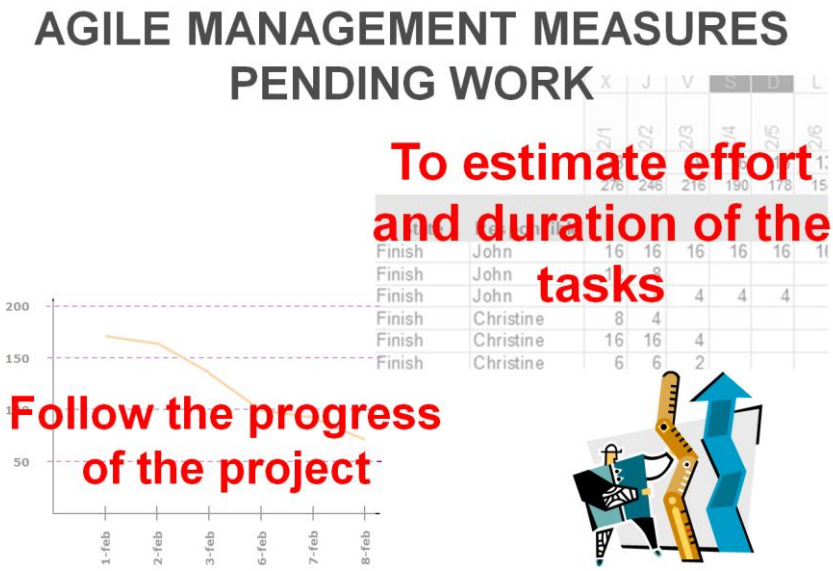


Illustration 16: : Pending work measurement.

The work required to fulfil a requirement or a user story cannot be absolutely predicted, because the functionalities are not unique solution realities, and if it could, the complexity of the measurement would make a metric too heavy for the agile management.
And as it is not possible to accurately estimate the amount of work in a requirement, you cannot know how much time you will need, because in addition to the uncertainty of work, there are these other uncertainties inherent in "time":

- It is not realistic to talk about the quantity or quality of work performed by a person per unit of time, because the differences between people are very large.
- The same task, performed by the same person will require different times in different situations.

On these premises:

- It is not possible to estimate accurately, nor the quantity of work of a requirement, nor the time necessary to carry it out.
- The complexity of estimating techniques grows exponentially to the extent that:
  - They try to increase the reliability and accuracy of the results.
  - Increases estimated job size.

The strategy used by agile management is:

- Work with approximate estimates.
- Estimate with the "expert judgment" technique.
- Break down the tasks into smaller subtasks, if the estimates exceed average ranges, or a day of real time.

Work Units

A work can be dimensioned by measuring the product that is constructed, like the traditional Function Points or COCOMO; Or the time it takes to complete it

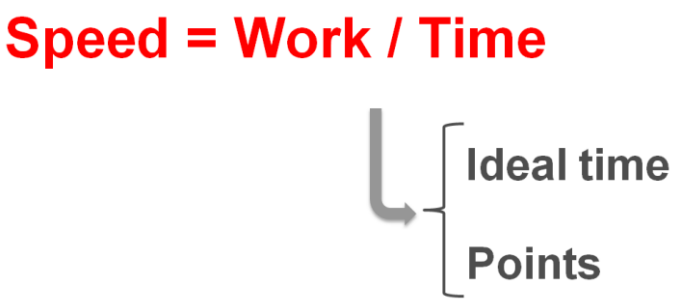


Illustration 17: Speed.

In agile management, "points" are often used as units of work, using denominations as "points of story" or simply "points".
EXtreme Programming's "Story Point" unit is defined as the amount of work done on an "ideal day".

Each organization, according to its circumstances and its criteria, institutionalizes its working metric defining the name and the units, so that it can define its unit, its "point":

- As the relative size of familiar tasks you normally employ.
  Ex: The team of an internet sales system, could determine that a "point" would represent the size of a "list of invoices of a user".
- Based on the ideal time required to do the job.
  Ex: A team can determine that a "point" is the work done in 4 ideal hours.

It is important that the metric used, its meaning and the form of application was consistent in all measurements of the organization, and known by all people:
That must be an institutionalized work procedure.

Speed

Speed is the magnitude determined by the amount of work done into a period of time.
Speed in technical scrum is the amount of work done by the team in a sprint. For example, a speed of 150 points indicates that the team performs 150 work points in each sprint.
When working in advanced Scrum implementations, which can perform sprints of different durations, or not always with the same number of members in the team, the speed is expressed indicating the unit of time and if it refers to the total of the Team, or the average per person, if applicable. For example: "The average speed of the team is 100 points per week." "The average speed of a team person is 5 points per day."

Measurement: uses and tools

Product chart.

The "burn up" or product chart is a product owner planning tool that visually displays the foreseeable evolution of the product.
Project in time the construction, based on the speed of the work team.
The projection is carried out on a Cartesian diagram that represents the estimated effort to construct the different stories of the product backlog, and, on the abscissa axis the time, measured in sprints or in real time.

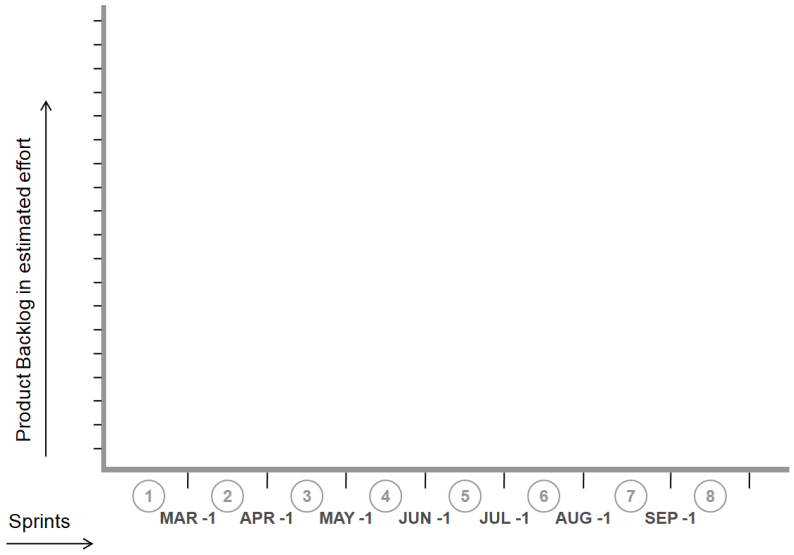


Illustration 18: Product Chart.

Example
Conventions used by the team:

- Unit to estimate the work: scrum points.
- It is planned to work with sprints of fixed duration: monthly (20 working days).
- The team consists of 4 people, and develops an average speed of 400 points per sprint.

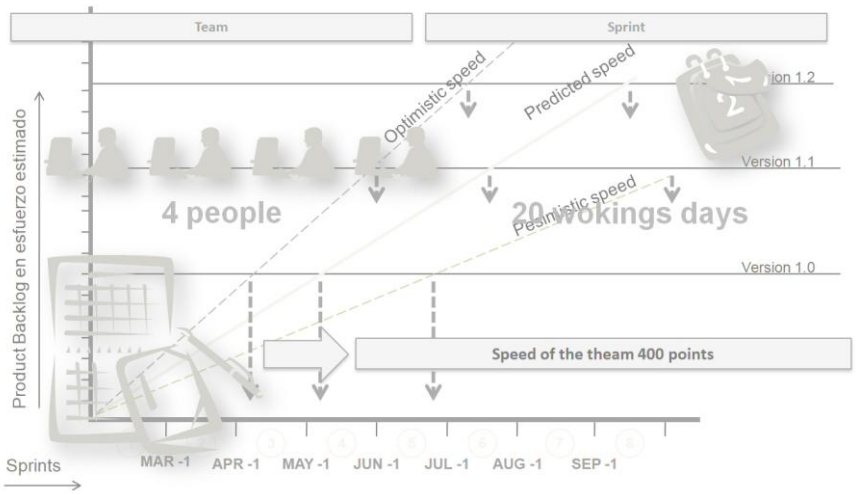


Illustration 19: Product Chart as product plan.

The line representing the expected pace of progress is plotted on the graph, according to the average speed of the team (in this example 400 points per sprint).

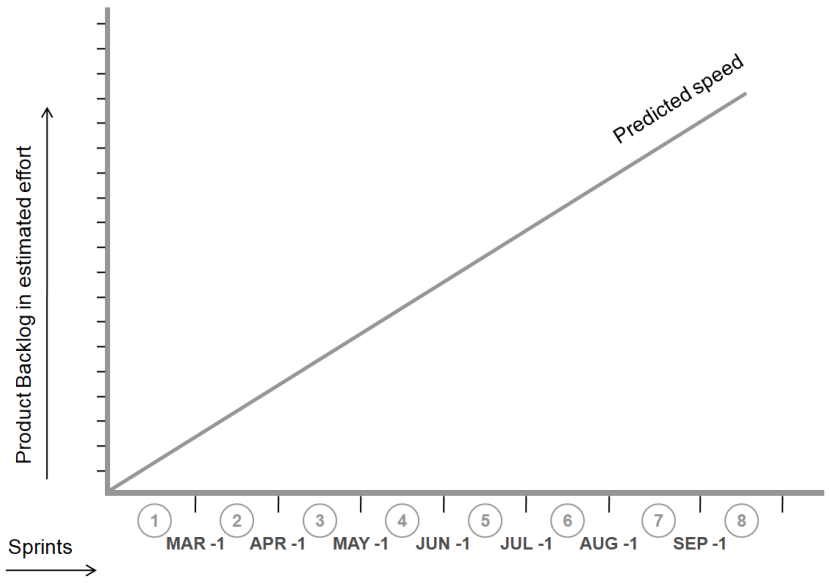


Illustration 20: Product chart: planned speed.

It is advisable to also trace the pace of progress with a pessimistic and optimistic forecast. They are drawn based on the speed obtained in the previous sprints with the worst and best results, or establishing a margin according to the work team criterion (e.g. + - 20%).

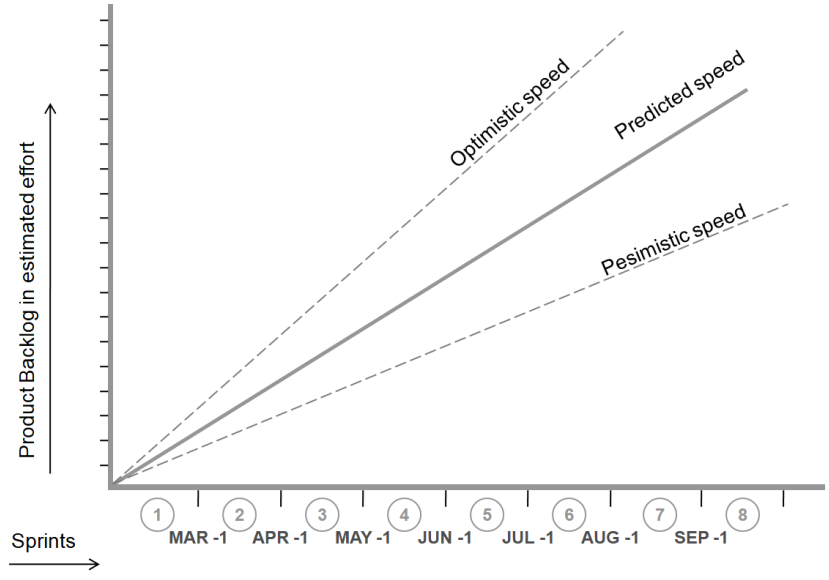


Illustration 21: Product chart: optimistic and pessimistic speed.

Afterwards we must take the product backlog prioritized. The following figure represents the one used in this example:

Id	Stories	Work	Validation criteria
1	Story A 1.0	150	Lorem ipsum dolor sit amet
2	Story B 1.0	250	consectetuer adipiscing elit
3	Story C 1.0	250	Aliquam vehicula accumsan tortor
4	Story D 1.0	300	Pellentesque turpis
5	Story A 1.1	250	Phasellus purus orci
6	Story D 1.1	350	penatibus et magnis dis parturient
7	Story E 1.0	150	Quisque volutpat ante sit amet velit
8	Story B 1.1	500	Cras iaculis pede eu tellus
9	Story C 1.1	150	Vestibulum vel diam sed pede blandit
10	Story E 1.1	200	Suspendisse aliquam felis et turpis
11	Story F 1.0	TBD	Nullam imperdiet lorem vitae justo
12	Story A.1.2	TBD	Suspendisse potenti. In nec nunc
13	Story B 1.2	TBD	Nam eros tellus, facilisis sed, pretium
14	Story F 1.1	TBD	Morbi arcu tellus, condimentum

Illustration 22: Example of product backlog.

In this case, the product owner plans to release the version 1.0 when he has available the first four stories, which have an estimated effort of 950 points (150 + 250 + 250 + 300).
In addition, the forecasts for later versions (1.1 and 1.2) have been also outlined, as shown in the following figure:

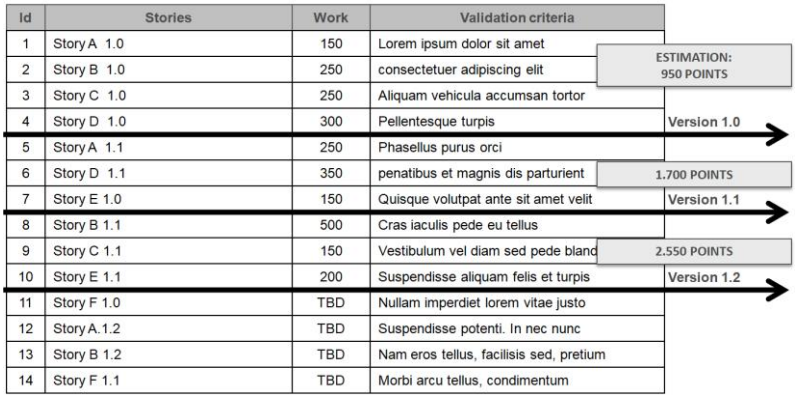


Illustration 23: Planned product versions.

To plot the forecast, each version is placed on the vertical axis in the position corresponding to the calculated effort to construct all the stories that it includes.
Following the example, the position of version 1.0 would be on the value 950 of the ordinate axis:

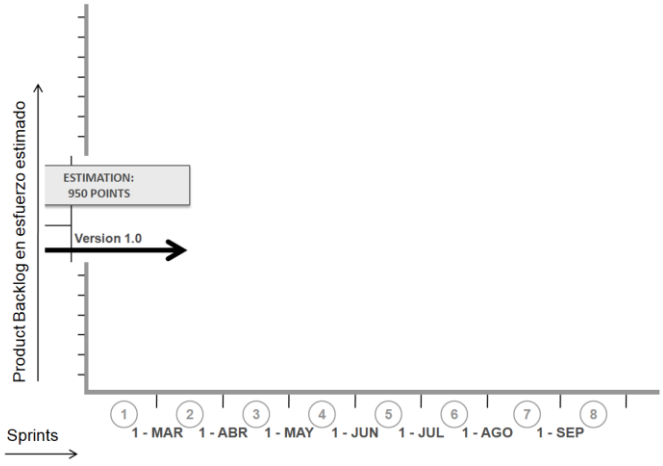


Illustration 24: Representation of version 1.0 on the product chart.

The cut points that mark this position with the speed lines of the team (pessimistic, realistic and optimistic) project on the horizontal axis the date or sprint in which it is expected to complete the version.

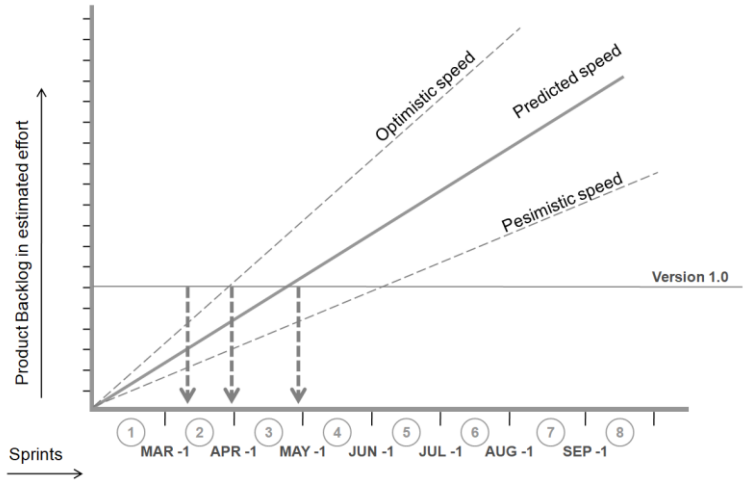


Illustration 25: Forecast of dates on the chart product.

Similarly, early estimates of future planned versions can be projected.

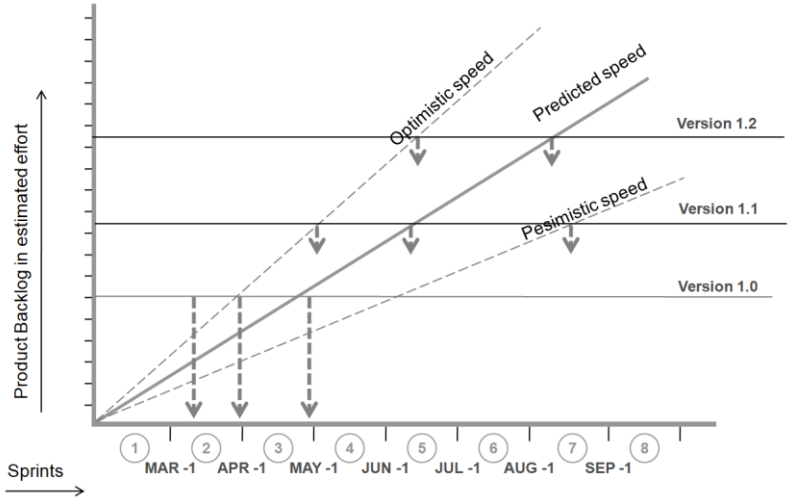


Illustration 26: Preview forecast for future versions on product chart.

This tool projects the forecast of the product backlog, which is a living document whose evolution determines the future development of the product

Burn-down Chart: sprint monitoring

It is updated by the work team in the daily scrum to check whether the pace of advance is foreseen or the delivery of the sprint can be compromised.

The agile strategy for monitoring the project is based on:

- Measure the missing work, not the work done.
- Close, frequent and direct feedback and progress tracking (daily if possible).

And this chart works with both principles:

- Record the pending job on the Y axis.
- It must be updated daily.

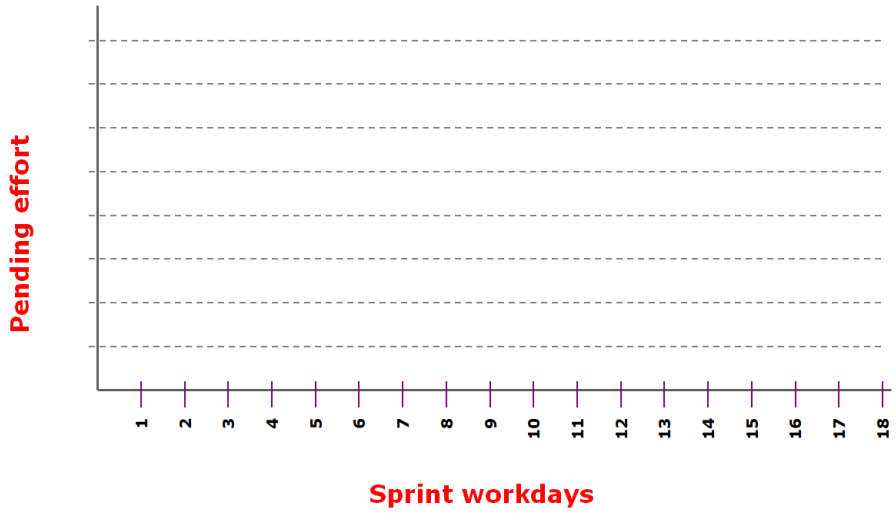


Illustration 27: : Burn-down chart.

The team has in the sprint backlog, the list of tasks that they will perform, and in each one records the pending effort.
This is: on the first day, in the sprint backlog, for each task the effort is estimated, since no work has been done yet.
Day by day, each team member updates in the sprint backlog the time remaining for the tasks they are developing, until they are completed and 0 is left as pending time.
The following figure shows an example of a sprint backlog on the sixth day of the sprint: the finished tasks no longer have an outstanding effort, and the total effort planned for the sprint: 276 points (A), is currently 110 (B).

SPRINT	START
1	Feb-06

			X	J	V	S	D	L	M	X	J	V	S	D	L
			2/1	2/2	2/3	2/4	2/5	2/6	2/7	2/8	2/9	2/10	2/11	2/12	2/13
			23	23	19	16	16	13	12	11	9	9	6	3	
		A ➡	276	246	216	190	178	158	143	122	108	78	46	16	
Task	State	Responsible							B ⬆						EFFORT
task 1 description	Finish	John	16	16	16	16	16	16	16	16	16	4			
task 2 description	Finish	John	12	8											
task 3 description	Finish	John	4	4	4	4	4								
task 4 description	Finish	Christine	8	4											
task 5 description	Finish	Christine	16	16	4										
task 6 description	Finish	Christine	6	6	2										
task 7 description	Finish	Antonio	16	4											
task 8 description	Pending	Antonio	16	16	20	12	4								
task 9 description	Pending	Antonio	12	2											
task 10 description	Pending	John	12	12	12	12	12	12	12	12	12	12	6		
task 11description	Pending	John	8	8	8	8	8	8	8	8	8	8	8	4	
task 12 description	In progress	John	14	14	14	14	14	14	14	14	14	14	14	10	
task 13 description	Pending	Antonio	8	8	8	8	8	6							
task 14 description	Pending	Antonio	16	16	16	16	16	16	12	4					
task 15 description	Pending	Antonio	16	16	16	16	16	16	16	16	10	4			
task 16 description	Pending	Christine	8	8	8										
task 17 description	Pending	Christine	12	12	12	8	4								
task 18 description	Pending	Christine	16	16	16	16	16	10	5						
task 19 description	Pending	Christine	12	12	12	12	12	12	12	4					
task 20 description	Pending	Christine	16	16	16	16	16	16	16	16	16	4			
task 21 description	Pending	Christine	12	12	12	12	12	12	12	12	12	12	4		
task 22 description	Pending	Antonio	8	8	8	8	8	8	8	8	8	8	2		
task 23 description	Pending	Antonio	12	12	12	12	12	12	12	12	12	12	12	2	

Illustration 28: Sprint backlog.

With this sprint backlog information, the graph is updated by putting the total pending effort of all tasks that have not been completed yet.

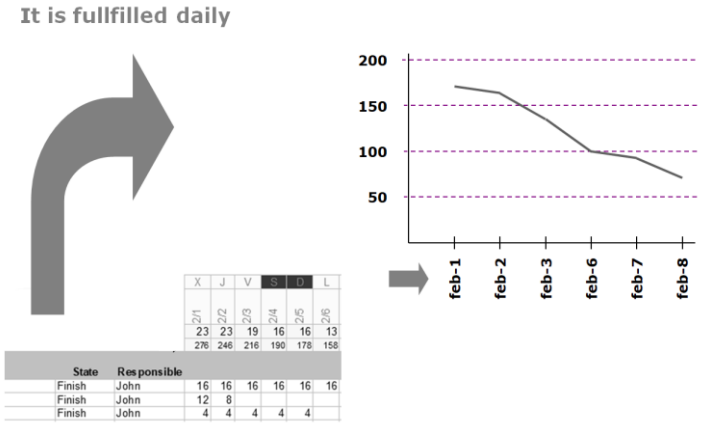


Illustration 29: From Sprint backlog to Burn-down chart.

The ideal advance of a sprint would be represented by the diagonal that reduces the pending effort continuously and gradually until completing it on the day that the sprint ends.

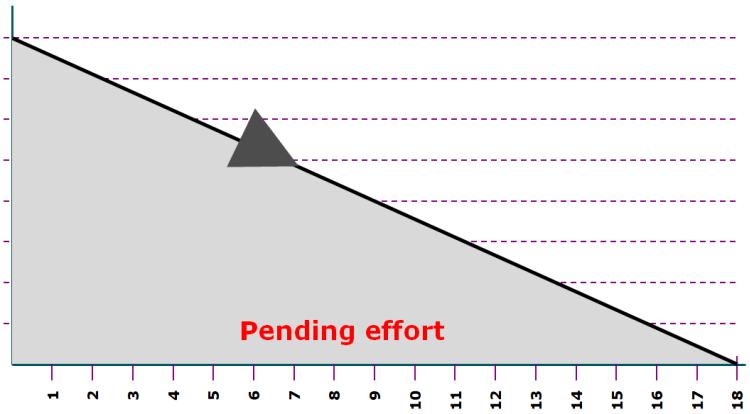


Illustration 30: Burn-down chart planned.

Perfect diagonal charts are very unusual, and the following image is an example of a more frequent forward pattern.

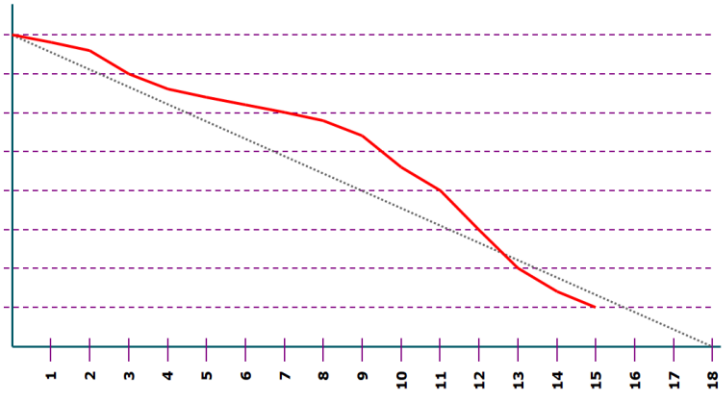


Illustration 31: Burn-down chart actual.

The next would be the aspect of the graph in an "underestimated sprint". The team's estimate of the sprint start meeting is less than the actual effort required by the tasks.

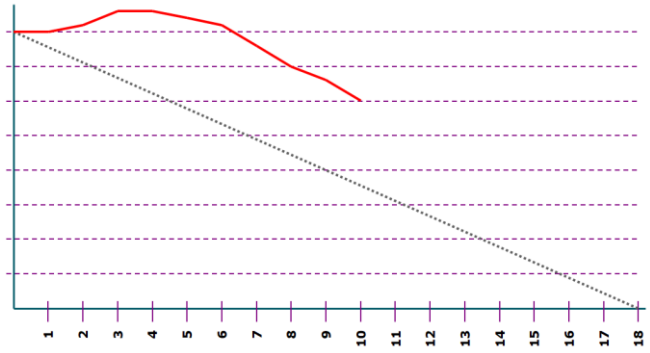


Illustration 32: : Burn-down chart of an underestimated sprint.

And the next would be the graphic pattern of an "overestimated sprint".

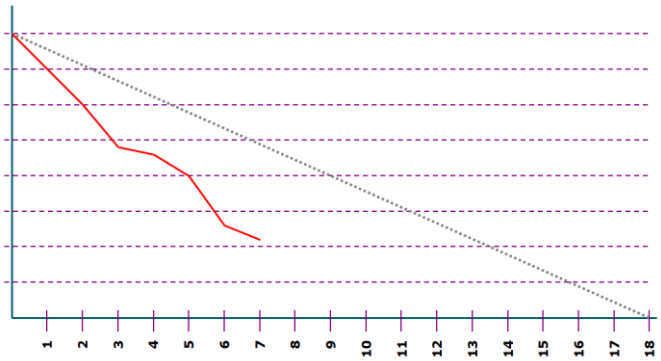


Illustration 33: Burn-down chart of an overestimated sprint.

Planning Poker: An agile technique.

It is an agile practice to conduct the meetings in which the effort and duration of tasks is estimated.
James Grenning devised this planning game to avoid lengthy discussions that do not end with concrete conclusions.
The initial Grenning model consists of 7 cards, with numbers 1,2,3,5,7,10 and infinity (Grenning, 2002).

The operation is very simple: each participant has a set of cards, and in the estimation of each task, all face up the combination that adds the estimated effort.
When it is considered that this is greater than x ideal hours (the maximum size considered by the team for a story), the card “∞” is lifted.

Tasks that exceed maximum size must be decomposed into smaller subtasks.
Each team or organization can use a set of cards with the appropriate numbers to the unit of effort they work with, and the maximum size of task or story to be estimated.

Variant: Fibonacci sequence

Based on the fact that increasing the size of the tasks increases uncertainty and margin of error, this variant has been developed, which consists in using only Fibonacci sequence numbers, so that:

- The card game consists of numbers in succession of Fibonacci.
- The estimation is not done by lifting several cards to compose the exact figure (as in the original version of Grenning), but by facing up the card with the figure closest to the estimate.

For example, if a person believes that the appropriate size of a task is 6, he is forced to reconsider and either accepts that the size can be 5, or accepts a more conservative estimate and raise the 8.
To estimate tasks can be valid a card game like this:

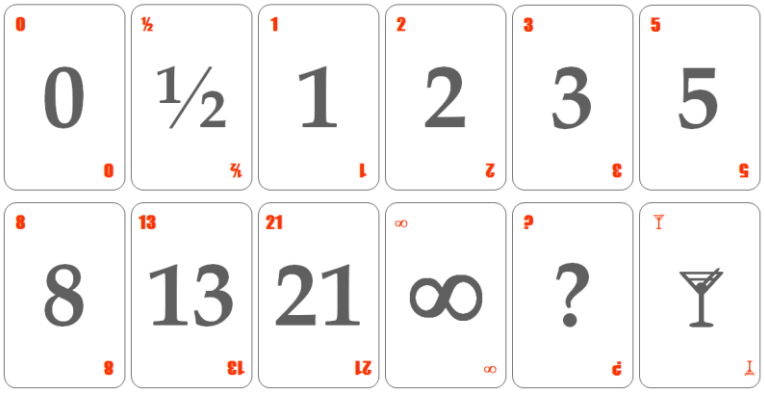


Illustration 34: Cards for poker estimation (Fibonacci sequence).

It is common to use a card with a symbol of doubt or interrogation to indicate that, for whatever reason, an estimate cannot be specified.
It is also possible to include another card with an allusive image, to indicate that a break is needed.

Operative

- Each meeting participant has a set of cards.
- For each task (user story or functionality, depending on the level of requirements to be estimated), the client, moderator or product owner exposes the description using a time to no exceed.
- Another additional time is set for the customer or product owner to answer the possible questions from the team.
- Each participant selects the card, or cards that represent his estimate, depending on the estimation system used, and separates them from the rest, face down.
- When everyone has made their selection, they are shown faced up.
- If the estimate is "infinite", exceeding the established maximum limit, the task must be divided into smaller subtasks.
- If the estimates are very different, who assumes responsibility for managing the meeting, with its management criteria, and based on the characteristics of the project, team, meeting, number of items still to be evaluated… can choose from:
  - Asking people of extreme estimates: Why do you think it takes so much time?, and why do you think it takes so little time? After hearing the reasons, the team must repeat the estimate.
  - Setting aside the estimate of this task and resume at the end or at another time those that have remained pending.
  - Asking the customer or product owner to decompose the functionality and evaluate each of the resulting functionalities.
  - Taking the lowest, highest, or average estimate.

This protocol of moderation prevents in the meeting the jams of circular analysis between diverse options of implementation, it makes all the attendees participate, it reduces the estimation time of a functionality from fifteen or thirty minutes to a few minutes, it facilitates to reach consensuses without discussions, and is also fun and energizes the meeting and the team.

PART TWO

Advanced Scrum.

1.- Knowledge in continuous evolution

Agile practice frameworks do not reach ICT projects as a "thesis" of knowledge, but as a "antithesis" to the project management practices that Software Engineering was usually developing.

In order to analyse what this means, it is necessary to put in perspective the reasons why IT projects embraced agility at the end of the last century, and their differences with process engineering; Not from the concrete practices, but from the principles they are based on, and understanding their agility strengths and weaknesses.

Achieving a vision of the reasons and principles of each methodology, beyond the concretion of a model, is the key to make the leap from technical management to expert management. That is, move from management based on the application of practices and procedures to management based on the application of our own knowledge.

Technical management: Management based on the application of practices and processes models.
Expert management: Management based on manager's tacit knowledge.

The dialectical pattern

By questioning knowledge, its evolution begins, following a dialectical pattern of: thesis, antithesis and synthesis.

In a schematic way the dialectical pattern can be defined as the flow of advance that opposes an antithesis to a previous conception, understood as thesis. The antithesis shows the problems and contradictions of the thesis, and from their confrontation comes a third moment called synthesis, a resolution or a new understanding of the problem.

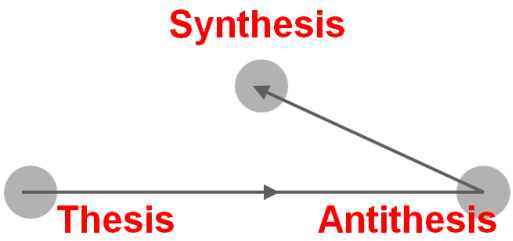


Illustration 35: Dialectical pattern.

In this way, the strategy based on trying to solve the challenges of software projects with process engineering was the first thesis to respond to the "software crisis", and its problems and contradictions have been highlighted by its antithesis: agility.

In 1968, at the first conference on software development held by the NATO organization, the problems of software programming were analysed, and it was coined the term "software crisis" to refer to them.

The conclusion of the conference (Bauer, Bolliet, & Helms, 1969) was the need to create a scientific discipline that, as in other areas, allowed a disciplined and quantifiable systematic approach to the development, operation and maintenance of software systems, that is the application of process engineering to software. It was the birth of Software Engineering.

The first strategy of software engineering (thesis) was based on two pillars:

- Process engineering:

- Predictive management:

The first one to apply the basic principle of quality successfully contrasted in industrial production environments: "the quality of the result depends on the quality of the processes used."

The second one to ensure compliance with agendas and budgets.

While this discipline had evolved and was perfected through different process models and knowledge bodies for project management (MIL-Q9858, ISO9000, ISO9000-3, ISO 12207, SPICE, SW-CMM ...) in the software industry doubts emerged and this strategy was questioned.

Is predictive planning appropriate for any project? Are the success criteria always guaranteeing the fulfilment of dates, costs and pre-established functionalities?

On the one hand, a new type of project appeared whose purpose was not to build a system previously defined and planned in its entirety. For this new type of project was not realistic to draw a closed plan from the beginning. These are projects in which it is not interesting to know if the final system will have 20 or 200 functionalities, nor to know how these will be in detail: Its interest is to put a novelty in the market as soon as possible, and from that moment evolve the vision and value continuously.

On the other hand, it was also questioned if the software could be produced with patterns of industrial processes, and it was begun to accept that in the quality of the result could be more important the tacit knowledge of the person who realizes it than the know-how contributed through the process and the technology used.

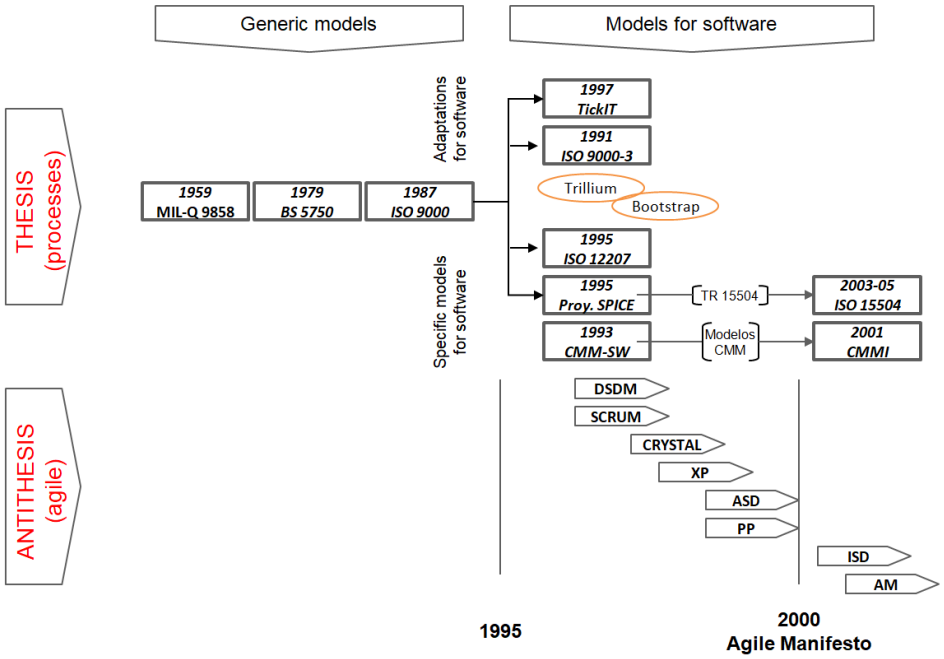


Illustration 36: Evolution of the first improvement models.

From the origins of agility, in the mid-90s, until 2005-2010, it had been frequent radical positions between proponents of process models (thesis) and proponents of agile frameworks (antitheses) who were more likely to focus on disqualifying the others than in reviewing and debugging the methods themselves.

Some examples of this tension:

"The difference between a bank robber and a CMM theoretician is that with the robber you can negotiate" ...

"The evaluation in CMM depends more on a good presentation in paper than on the real quality of the software product. It has more to do with the blind follow-up of a methodology than with the development and production of a system in the technological panorama ".
(Orr., 2003)

"If you ask a typical software engineer if you think CMM can be applied to agile methods, it will respond either with a look of surprise or with a hysterical laugh."
(Turner & Jain, 2002)

Dialectical spiral of knowledge.

Professional knowledge evolves continuously because the reality in which it is applied is in permanent movement, and also because improvement is always possible.

The implementation of new techniques, processes or models generates their own antitheses by revealing the weaknesses, contradictions and improvement points, and the confrontation of both leads to a synthesis, which will become a new thesis whose subsequent use will produce its antithesis, developing through this dialectical pattern a spiral of continuous knowledge evolution (Nonaka & Takeuchi, 2004)

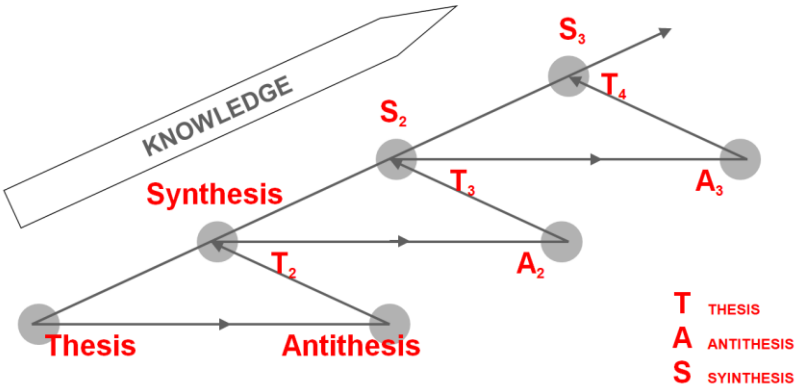


Illustration 37: Dialectical spiral of knowledge.

In non-technical disciplines and in previous generations the pace of advancement on this dialectical spiral had allowed professionals to perform with the knowledge acquired along their entire professional career. However today this is not possible, especially in the IT sector.

There are no methods, practices or models of work that guide us with solvency for a long time, but knowledge in evolution. This is a key consideration in Scrum Manager's knowledge base, because it does not define a fixed model, but an up-to-date knowledge base for expert management instead of technical management. It is based more on the manager’s documented and expert criteria than on the application of practices or processes.

2.- Company as a system

Companies are not made up by a variable number of independent departments or areas. They are systemic realities, and their efficiency is proportional to the harmony of the work modes employed at the different departments. The consideration of scrum as the framework of work rules of the project management field, whose practices can be adopted without implications in the rest of the organization produces limited and even counterproductive results.

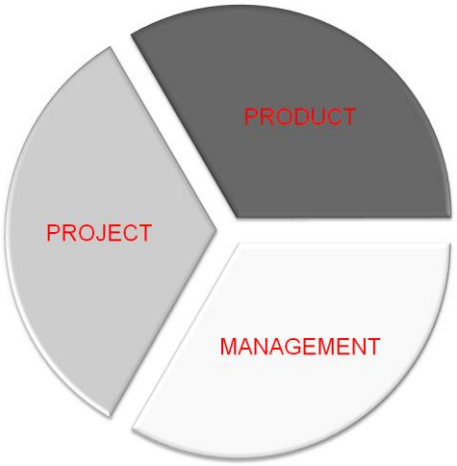


Illustration 38: The company as a system.

For example, in an organization whose management is based on industrial production models, and the engineering area accordingly works with process-based models with sequential or cascade life cycles, the adoption of agile practices in the area of project management will generate friction.

3.- Flexibility

The goal is not to implement a scrum framework based on rules. The goal is to reach an agile organization as a whole, being it able to "advance in scrum". Being it able to respond in work scenarios that evolve rapidly, or have high doses of uncertainty, where there are no stable requirements when designing new products or services. These are customers who need to start using a product as soon as possible and improve it continuously. These are products in which innovation is a key value.

A basic principle of the pragmatic implementation of scrum is flexibility, which is to adapt scrum practices to the organization and not the other way around. It is a question of expert management rather than technical management. A management directed from the knowledge, experience and criterion of the manager and not a management oriented to the search and set up the best model. A management focused on the person rather than the model.

Knowledge of the different techniques and methodologies broadens the manager's resource criteria and fund.

To follow the evolution of professional knowledge and to continuously expand and improve the criteria and inventory of our own professional resources, it is advisable to:

- Overcome resistance to change and avoid attitudes of adoption or dogmatic defence of a model.
- Have a critical-constructive spirit: To continually question in an "antithetic" way the current ways, with professional knowledge and criterion, to adapt the work system itself to the characteristics of the project, team and organization.

Advanced Scrum

Adapting scrum practices to the circumstances of the organization itself allows the use of continuous or iterative techniques; Kanban boards with the format most appropriate to each project, and in general the practices and rules that best fit the circumstances of each case.
In this way the guide lines of the defined rules are abandoned, and the scrum values are directly applied.

Technical Scrum

Rules

Software Development Rules Framework

Authors: Ken Schwaber and Jeff Sutherland
"Scrum Development Process OOPSLA'95" 1995

Application of defined rules

Roles

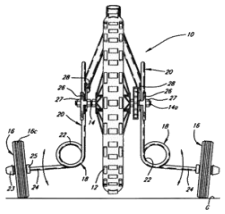
- Product Owner
- Development team
- Scrum Master

Events

- Sprint
- Sprint planning meeting
- Daily Scrum
- Sprint review
- Sprint retrospective

Artefacts

- Product backlog
- Sprint backlog
- Increment



To learn the scrum rules

Advanced Scrum

Values

Original Scrum Concept

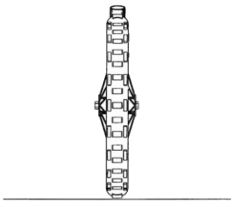
Authors: Hirotaka Takeuchi and Ikujiro Nonaka
"The New New Product Development Game" 1986

Application of agile values

- People > processes
- Outcome > documentation
- Collaboration > negotiation
- Change > planning

… "To advance in scrum"

- Uncertainty
- Self-organization
- Overlapping development phases
- "Multilearning"
- Subtle control
- Knowledge dissemination



To learn to advance in scrum without rules

Responsibilities

When moving from the rule-based technical scrum to the advanced scrum, which directly applies the principles of agile management with the knowledge and experience of the teams, the scope of the responsibilities to be covered goes beyond the project roles:
The organization, as a systemic reality, must respond, in a coordinated way and aligned with its vision, to responsibilities in three areas: Management, processes and production.

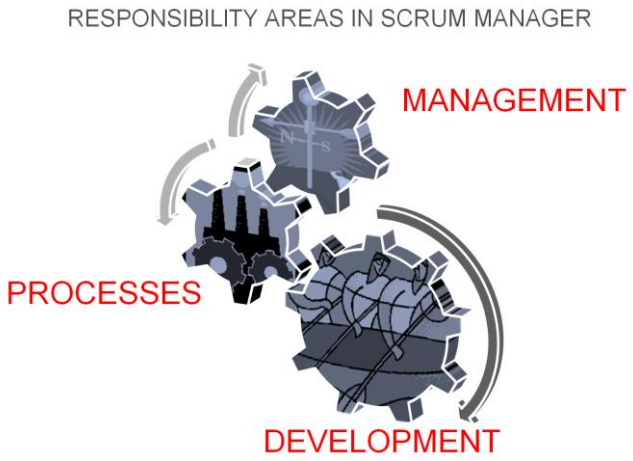


Illustration 39: Responsibility áreas in Scrum Management.

Management

- Systemic equilibrium of the organization.
- Coherence of the model.
- Means and training.

Processes

- Flexible scrum configuration.
- Continuous improvement.
- Guarantee of scrum operation in each project (in technical scrum assigned to the role of Scrum Master).

Production

- Product (in technical scrum assigned to the role of Product Owner).
- Self-organization (in technical scrum assigned to the team).
- Agile technology (in technical scrum assigned to the equipment).

The use of agile practices and technologies, working in self-organized teams, having a product vision defined and managed throughout the project, and ensuring the operation of scrum during execution, are responsibilities that belong to the scope of the project.
The different areas of the company are communicated and aligned with a common vision, consistent with an agile work model. Having the means to design and implement an agile implementation appropriate for the company, continuous improvement of the model and people training, are responsibilities within the organization.

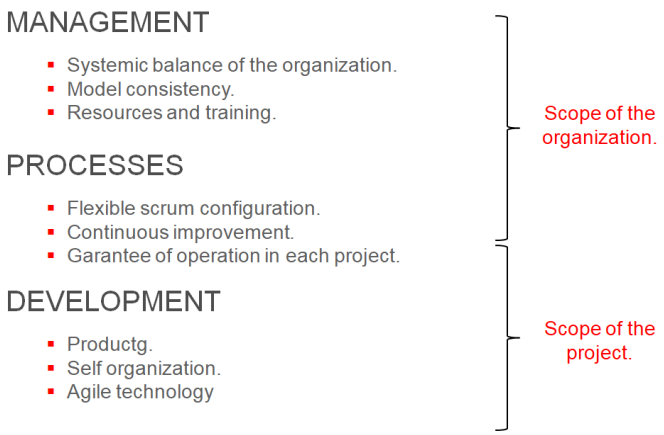


Illustration 40: Scrum responsibilities areas.

In technical scrum, the responsibilities of the project scope are assumed by defined roles:

- The scrum operation responsibility is assigned to a specific manager role for scrum operation: Scrum Master.
- The responsibility for product vision and management corresponds specifically to the role of product owner.
- The responsibility for self-organization and use of agile practices and technologies belongs to the team.

The most advisable in phases of implantation, in teams unfamiliar with agile development, is the adoption of the technical scrum role model.
In the evolution towards a more mature and global level of agility in the organization it is advisable to adapt the scrum framework to the reality of the organization, so that the relevant thing is not the presence of certain roles and rules, but to cover adequately all the responsibilities needed at the organizational level.
An example of a flexible assignment of the responsibilities of the project scope to the existing scheme of positions in the organization could be:

- Scrum operation guarantor => Quality or processes
- Product management guarantor => Product manager
- Self-organization and agile technology = Team

Whether it is in the implementation of agility, the necessary responsibilities are assigned to roles of the structure of the company, or new positions are created (Product Owner or Scrum Master), the important thing is that the people who perform them have the experience, knowledge and professional experience required.

Methodologies

Map of methodologies.

Since the 80s, so many process models, frameworks and work practices have been developed to improve the quality and efficiency of software projects, that is useful to transcend labels and to underpin the underlying principles and strategies to develop them; So that, coordinating three concepts: "development, work and knowledge", and two models of management: "predictive and evolutionary" clears up and simplifies the apparent labyrinth of process models, frameworks or work practices to which we refer : CMM-SW, CMMI, PMBOK, DSDM, Crystal, ISO 15504, RUP, XP, Scrum, ITIL, ASD, PRINCE 2, LEAN, KANBAN, TDD, etc.

The different practices and methodologies respond to combinations of three concepts and two patterns of project management.

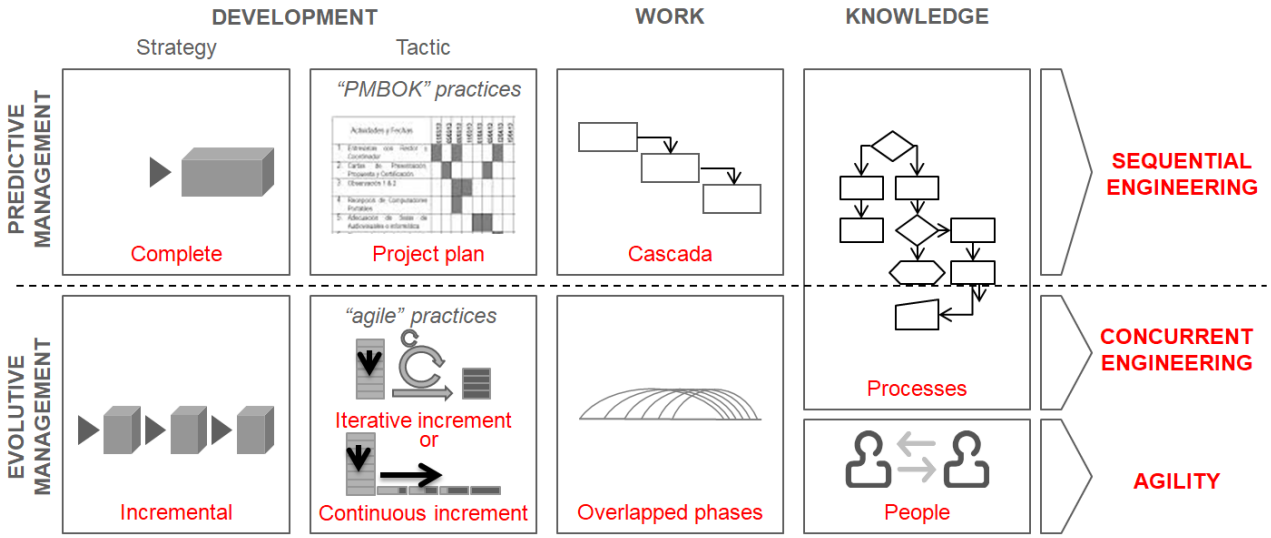


Illustration 41: Diagram of Project management concepts.

Concepts

1.- Development

Complete: The description of what is desired is available at the beginning of the project, is complete and detailed, serves as a basis for estimating the project plan: tasks, resources and work agenda. During execution, compliance is managed.
Incremental: The description of what you want to get is not available in a complete and detailed way at the beginning: it is complemented and evolves in parallel to the development, which generates the resulting product incrementally and can be managed with two different tactics:

- Continuous incremental development: Using techniques to achieve a continuous flow of development of functionalities or parts of the product, which are delivered to the customer on a continuous basis.
- Iterative development: Using time-stamping techniques or timeboxing to maintain the production of product increments in a cyclic and continuous way. This is the production framework used to apply the standard scrum framework, which defines as a sprint each development iteration, at the end of which there is a product increment.

2.- Work

Sequential (waterfall): Divide the work into phases, which begin at the end of the previous one. The most common example is the cascade cycle defined in Software Engineering with the requirements, analysis, design, coding, testing and implementation phases.
Concurrent: Overlap in time the different phases. Following the example of software engineering, the definition of requirements, analysis, coding and deployment of the result is performed and reviewed simultaneously and continuously.

3.- Knowledge

Where is the main knowledge used, in the correct execution of the process or in the know-how of the person who performs it?
Process-based production: The knowledge or know-how, responsible for the quality of the result is to a greater extent in the processes and technology used: "The quality of the result depends on the quality of the processes used".
People-based production: The knowledge or know-how responsible for the quality of the result is to a greater extent in the tacit know-how of the people who build it.

Project Management Patterns

Predictive management

Management model whose objective is to provide predictable results: the development of the expected product, in the expected time, and investing the expected resources. It employs a complete development strategy with traditional planning practices. The main references in the knowledge development for this type of management are PMI and IPMA and the process models CMMI, ISO 15504 and SPICE among others, which employ sequential engineering and process-based production.

Evolutionary management

Management model whose objective is to deliver as soon as possible a minimum viable product, and increase its value continuously. It uses a strategy of overlapping phases of work, and an incremental development, which can be obtained by maintaining a rhythm of brief and cyclical iterations or a constant development flow. It can be carried out with process-based production (concurrent engineering) or with people-based production (agility).
This distinction is important because without it, confusing situations are generated, which come to consider agility to the simple application of the standard rules of scrum (cycle of iterative increase with roles and defined artefacts), or the simple use of kanban visual management techniques to maintain a continuous flow of tasks.

> Agility and evolutionary management are not the same. Evolutionary management can be done using agility or using concurrent engineering

People, Processes and Technology

The body knowledge of Scrum Manager reconsiders two vertices of the classic triangle of production factors: People - Processes and Technology. **Process and people**.

Processes

In order to differentiate the procedures[2] into their two possible types, we can say that in one, people help the process, and in the other processes are the practices that help people.
In the first case the process is the protagonist, the one who knows how to do the work, and the person is integrated into the system as an instrument, as an operator or supervisor.
In the second, the artificer is the person and the process is an aid, a tool that simplifies "mechanical" or routine aspects.
That is why we call the first processes and second practices.

The main difference between them is the type of knowledge they work with.
Knowledge can be:

- Explicit: content in processes and technology
- Tacitus: The one applied by the person, based on his knowledge, practice, experience and ability.

Scrum Manager brings a consideration to the traditional people-process-technology triangle, considering that the working procedures can be:

- Processes: If their implementation provides key knowledge needed to achieve the result. Therefore the process and the technology that it employs are containers of "explicit" knowledge.
- Practices: If the procedure helps the person, who contributes with their tacit knowledge that is the "know-how" key in order to achieve the result.

It can be said that in the former the person helps the procedure, and in the latter the procedure helps the person.

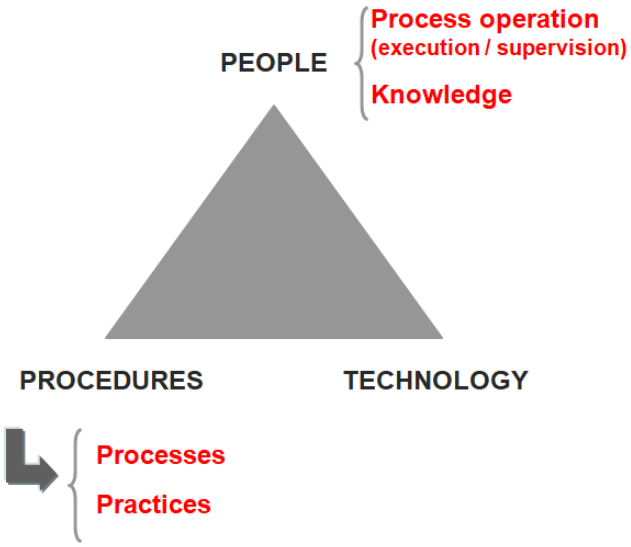


Illustration 42: People, procedures and tecnology.

[2] We will not call them processes but "procedures" leaving the name "process" for the procedure that has explicit rules in it the main knowledge for obtaining the result

Process engineering models consider the binomial "process-technology" as the main responsible for the quality of the result. Its antithesis, the agility, gives the prominence of the result to the people.
From the point of view of Scrum Manager, both options are valid, but for different types of jobs. In industrial production environments, people provide work to execute and monitor processes. However, for knowledge companies working in fast and innovative scenarios, people contribute with their talent the know-how that gives value to the result.

People

Organizations that need to continuously bring an innovative component, or that move in fast innovation sectors, obtain better results if they make people's talent responsible for value generation rather than process execution.
In this type of organization it is important to ensure, in addition to the level of creativity of the team, its ability to learn. The Nonaka and Takeuchi (Nonaka & Takeuchi, The Knowledge-Creating Company, 1995) model of knowledge conversion defines the process for the people's acquisition of tacit knowledge in 4 phases: sharing experiences, direct communication, documents/manuals and traditions, which add new knowledge to the collective basis of the organization.

Visual management kanban for continuous increment.

The following image shows where technical scrum is located, as described in the previous chapter:

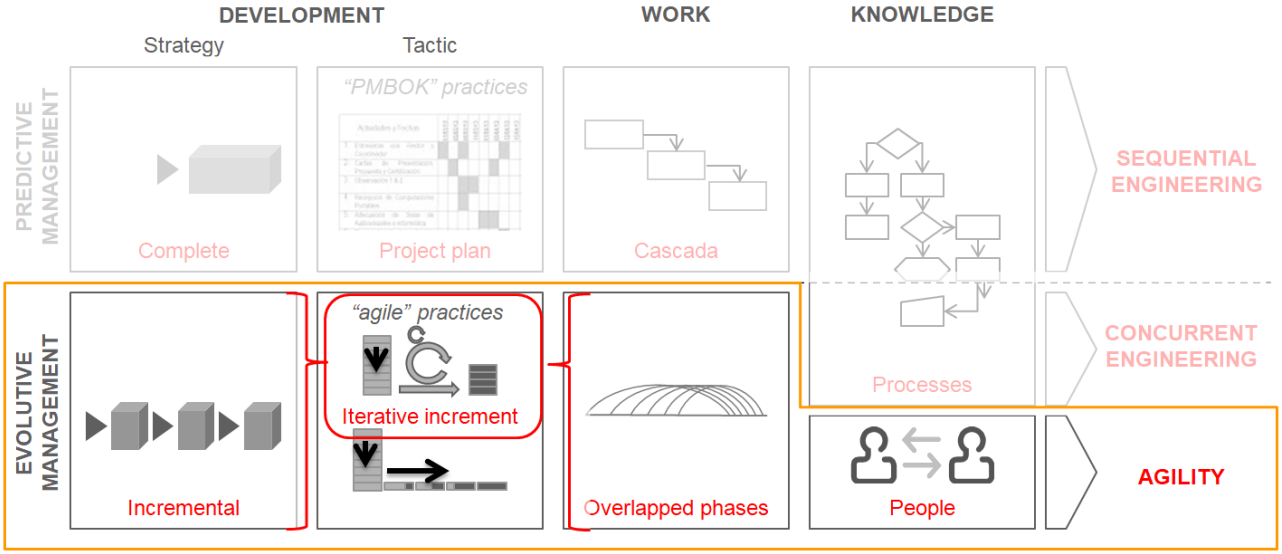


Illustration 43: Agility with iterative incremental development.

Its main characteristic is the use of sprint pulses; it uses timeboxing with a defined and fixed cadence as a motor of advance, the rhythm is marked by the sequence of sprints.
The timeboxing tactic helps the team to move forward, while mitigating the usual tendency to delay expected delivery times.
The original scrum teams observed and described by Nonaka and Takeuchi (Nonaka & Takeuchi, The New Product Development Game, 1986) and the principles of agility do not prescribe the use of a particular tactic for incremental development. In fact, it is also possible to work with a constant progress of tasks one after another, without increments.
Achieving a continuous flow of tasks without using sprints is not easy because bottlenecks often block the progress, causing in other areas of the team dead times in which they run out of tasks to perform.
Currently, the Kanban visual management is the most used technique to regulate a flow of continuous advance in IT projects and knowledge services managed evolutionarily without sprints.

Kanban: Origin and definition

The Japanese term Kanban was used by Taiichi Onho (Toyota), which refers to the visualization system used in production processes that coordinate in an assembly line the timely delivery of each part just in time, avoiding overproduction and unnecessary storage of product in intermediates states.
It can be translated as board or signage card, and its origin dates back to the late forties or early fifties.

> The term kanban applied to the agile management of projects refers to the techniques of visual representation of the information to improve the efficiency in the execution of the tasks of a project

.

Kanban in the IT sector

The use of Kanban boards shows and manages the flow of advance and delivery, and helps to avoid the two most important problems: bottlenecks and downtime.
Agile software development employs visual management practices because they are the best way to implement the principles of direct communication and simplicity in documentation and management.
Since 2005, it has become increasingly common to replace the product and the sprint backlog lists with sticky notes on boards, which are more versatile and allow to the team members to add new tasks, position changes, or the rearrangement of the user stories priorities in a product backlog, or can reflect through their position, which ones are being programmed, tested, are finished or in other possible states defined.
The Kanban practices are valid for evolutionary management with continuous delivery. They should be used with flexibility criteria, without considering prescriptions or exceptions in the working method and in order to achieve the personalized implementation that gives the best answer to the agile or the concurrent engineering, or synthesizes principles of both.

Technical Management vs. Expert management

Some authors consider scrum and Kanban as frameworks of different rules and practices.
According to Kniberg & Skarin, the following differences would be drawn (Kniberg & Skarin, 2009):

- Scrum prescribes roles, not kanban.
- Scrum works with fixed-time iterations, Kanban with cadences (simple, multiple, or event-driven).
- Scrum limits the WIP by iteration, Kanban limits the WIP by every state of the workflow.
- Scrum teams are multidisciplinary, in Kanban they can be specialists.
- Scrum does not allow to change the tasks of the sprint, in kanban the task can be altered until entering the flow.
- In scrum the product backlog must have the length of at least one sprint. In kanban the rate of tasks pulling must be attended.
- In scrum you must estimate the stories and tasks and calculate the speed, kanban does not measure tasks or speed.
- Scrum needs a prioritized product backlog, in kanban it is the next story or task dragged from the client.
- Scrum prescribes daily meetings, not kanban.
- Scrum uses burndown diagrams, not kanban.
- Scrum boards are reset at the end of each sprint, not in kanban.

By evolving towards an advanced scrum model, based on the application of agility values with the own experience and knowledge, and abandoning the rule-based models, it is learned to break

these models and to adapt practices, leaving as trivial the technical and methodological questions "that otherwise distort the reality and the focus of the management:
Situation A: "On the one hand you want to use kanban, but on the other hand you want to estimate the tasks (for example to register the speed for organizational reasons of the company) ..."
Situation B: "The company manages simultaneous projects with a project office organization and therefore fits better the establishment of roles; But nevertheless you want to work with kanban instead of scrum ... Should I do scrumban? what is that? Or what should be done is Scrumbut? Is it the solution of a bad manager? "

Kanban boards: concepts

The kanban visual management practices are useful for:

1. Managing the workflow.
2. "Broadcasting" information.

A kanban board can be used only as an information broadcaster, or as a visual workflow management tool.

1.- Characteristics of kanban to manage the workflow.

On a kanban board guidelines can be established to regulate the flow of progress of the tasks.

- The position of each card on the board reflects the state of the work it represents.
- The usual minimum states in a kanban board are "pending", "ongoing" and "completed"

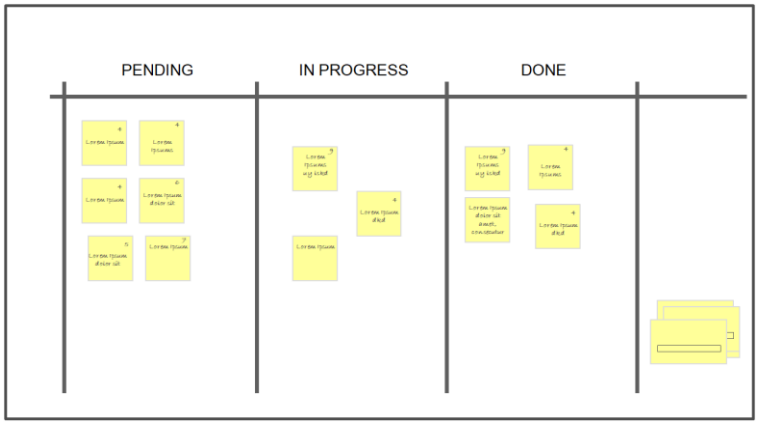


Illustration 44: Basic sctructure of a kanban board.

In some cases it is convenient to include additional states (for example, tested, validated).
The order of the works from the "pending" area reflects the expected sequence of tasks according to the priorities.
The monitored works can be tasks, user stories or epics, according to the use to which the board is dedicated

Kanban brings out the information of the problems.

Conflicts in the work prioritization, problems in the flow due to impediments or workloads, incidences in development, etc. are immediately evident when updating on the board the status of the works.

Kanban Facilitates a steady pace and avoids Parkinson's law

It generates a continuous advance of work whose rhythm is not "predestined" by a temporal planning: Gantt or Sprint (iterative increase).
The absence of temporary milestones avoids the usual tendency to lengthen the working time to complete the estimated time (Parkinson's law)

> The work expands to fill all the time that was planned
> Parkinson's Law.

On the other hand, the absence of temporary milestones, without kanban techniques to monitor and manage the progress, would generate lengthening of times and delays due to deferment and perfectionism.

> Agile processes promote sustainable development. Promoters, developers and users must be able to maintain a constant rhythm indefinitely.
> Agile Manifesto

2.- Information broadcaster.

Kanban acts as a broadcasting tool for the project progress information:

It promotes direct communication

- It facilitates direct communication within the team by updating information in meetings in front of a kanban board.
- It shares the visibility of the evolution of the project with all those involved.

It facilitates early detection of problems

- Kanban continuously monitors the progress of the project. The updating of just-in-time information helps to identify at first the possible impediments, problems and risks, which otherwise go unnoticed until they start to produce delays or repercussions already inevitable.

It favours a culture of collaboration and resolution.

- It is an open and transparent means of communication for the team and all the participants.

Kanban: Operative

Sequence and polyvalence

Two are the factors that combine in a work scenario and that determine the form and the way that the kanban board must be used:

- **Sequence of tasks**: Do the tasks have to be done in a certain order, or can they be done in any order?
- **Versatility of people**: Can the team members do any kind of task?

Sequence.

Do the works reflected in the sticky notes of the board have to be executed in a certain order or can they be carried out in any order?

It is not the same to design a board for the development of a new information system, that for the technological infrastructure maintenance of a company. The first case requires to perform the tasks in a particular order. For example, it is not possible to perform the test task if the programming has not been done before. However, the following could be the tasks of a maintenance team: "installing a new printer on the Director's computer" "updating the operating system on the web server", etc. This type of tasks can be performed in any order. There are no dependencies between them so that you can do anyone anytime, even if the other tasks are not completed.

Polyvalence

Is it a multipurpose team or are they specialists? Can any member perform any task?

Following with the previous examples, it is possible that in the maintenance team a person can indistinctly install a printer, or an operating system, or it may not; it is possible that there are hardware technicians and software technicians. In a similar way a programming project can include specific tasks of graphic design, programming, integration, testing, etc. that can only be performed by certain team members.

The following image reflects the key value of kanban: the management of a continuous flow of advance, and the factors that must be considered to configure the board with the most appropriate structure to our work and team.

Kanban visual management techniques are appropiate in order to avoid bottlenecks and downtimes.

Adjusting them to the circumstances of our workload and team with flexibility criteria.

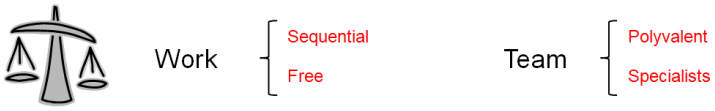


Illustration 45: Strength and key variables of kanban boards.

Four are the possible patterns according to the combination of sequential or free work with a multipurpose team or a specialist team:

1. Polyvalent team that performs non-sequential tasks.

This is the easiest environment to manage: anyone on the team can do any task, and tasks can be taken in any order.

In this situation, if there are bottlenecks or dead times, improvement measures should focus on optimizing the size of the team according to demand, and as far as possible, the distribution of the tasks types and times of the demand.

2.- Team of specialists who perform non-sequential tasks.

The specialization of the team brings a difficulty factor to solve bottlenecks or dead times.

If there are bottlenecks, the strategy to face the type of tasks that provoke them must go in one or both of the following lines:

- Sizing: either the number of people trained to perform this type of tasks, or the number of tasks of that type that can be compromised, or in the time of customer response.
- Optimizing the execution process of this type of tasks.

The presence of dead times should question the dimension of the demand, or its non-homogeneous distribution.

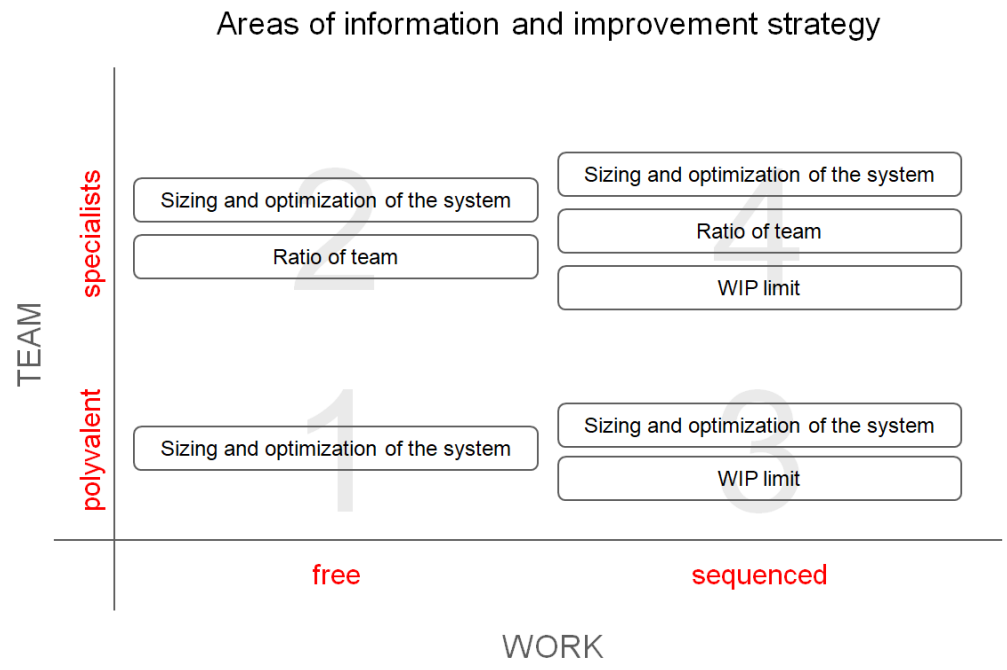


Illustration 46: of information and improvement revealed by the kanban boards.

3.- Polyvalent team that performs sequential tasks

In this case, the dependence between tasks is the main cause of tensions in the flow.

A common practice in kanban boards that manage sequential tasks is to limit the maximum number of tasks that can be found in a given phase. This is called "adjusting the WIP".

WIP is a term used in lean manufacturing, where it comes from, it is used to indicate the number of products in a step of the manufacture process that are not finished yet simultaneously.

By analogy, in kanban boards this term is used to indicate the tasks that are in a phase of the process, pending to pass to the next one or to be completed; and in this environment the term WIP indicates the limit or the maximum number of tasks that can accumulate in a certain area. For example, lets say that in a kanban board for software programming the testing area "has a WIP of 3", this means that there can be no more than three tasks simultaneously in that phase.

4.- Team of specialists and work that requires a sequenced order.

This is the environment in which it is more difficult to adjust a continuous workflow, because it requires analysing and fine-tuning all possible improvement lines: dimension and balance of specialists in the team, dimension or balance of response times in the commitment, and adjustment of WIP limits in each phase.

In each of these four possible situations, the board brings problems to the surface, and the team or manager can make the adjustments in the possible lines of work taking into account the individual case and according to their criteria and the circumstances of their organization.

Practical cases of kanban boards

The following are examples of different types of boards.

Dashboard to provide product development information.

Example of the board that could be found in the product manager's office showing the state in which is the construction of the product.

In this case it is not used as a visual management tool, but only as a "broadcaster" of information.

The dashboard displays the following information regarding the state of development of the product:

- Suggested user stories, which are under evaluation without determining yet if they will be incorporated into the product.
- Approved user stories: will be added to the product.
- User stories "prepared" (already valued and prioritized) planned to be scheduled.
- User stories that are currently being programmed.
- Scheduled user histories that can be evaluated in the test environment.
- User stories already evaluated, pending of deployment.
- User stories displayed in the last two versions.

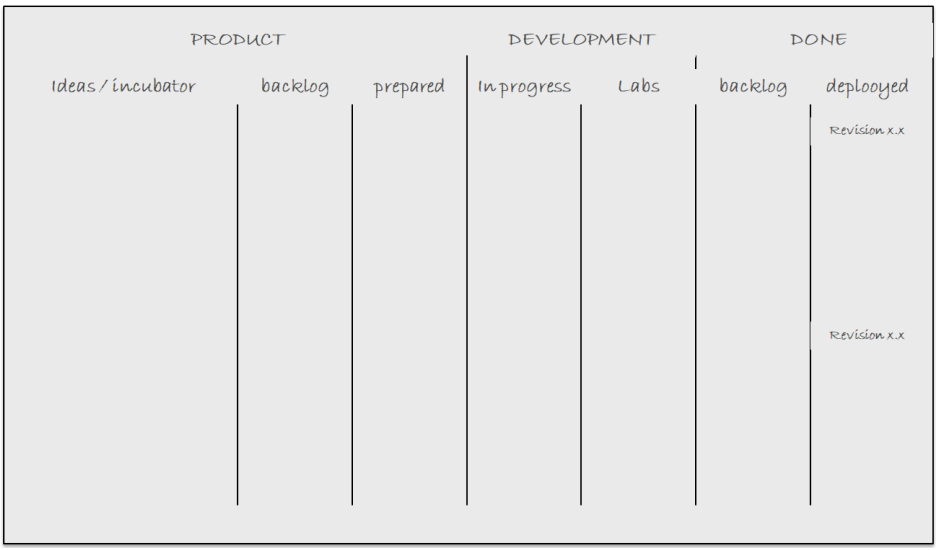


Illustration 47: Sample kanban board to monitor product status.

Boards for evolutionary development, with continuous increment, and with iterative increment.

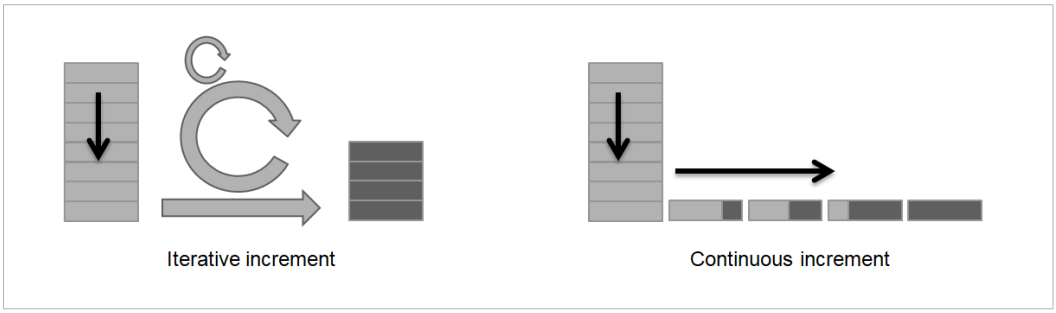


Illustration 48: Boards: continuous increase - iterative increment.

Kanban, in addition to providing visual information, allows the application of techniques such as limiting the WIP, and shows the areas that need to be improved to maintain a continuous development flow. It is useful to work with continuous increase.

But can kanban be used by a team working with scrum and iterative increments to visually represent the progress of tasks with sticky cards instead of using a forward graph?

Case 1: Continuous increase.

The following images represent possible boards to guide the work management of a team that is developing a product with continuous increases, and which shows the following information:

- Tasks backlog
- "Prepared" tasks.
- Tasks in analysis.
- Tasks in coding.
- Finished tasks.
- Tasks built into the development server (labs).
- Integrated tasks in production.

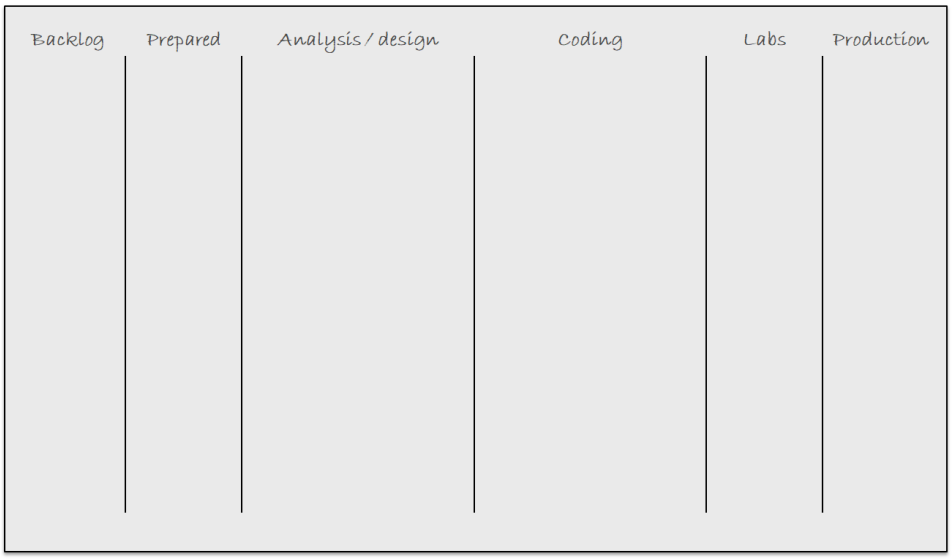


Illustration 49: Example of kanban board to monitor and manage continuous increment.

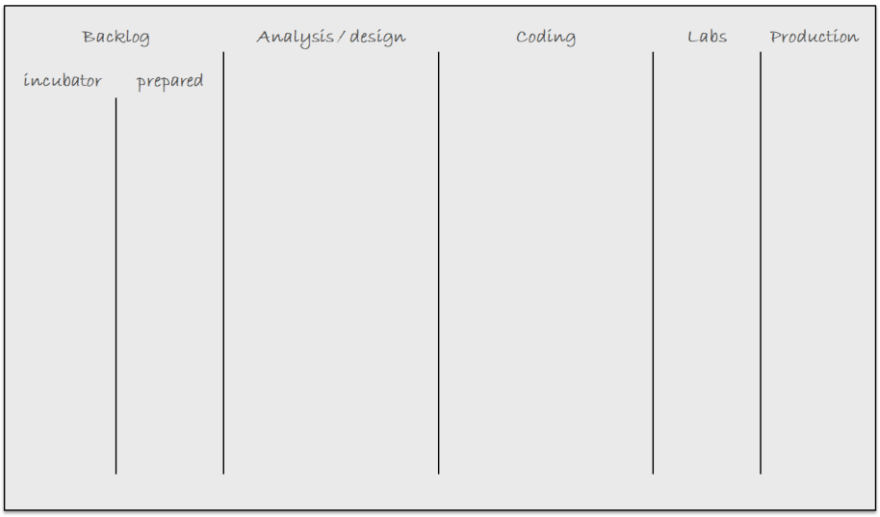


Illustration 50: Example of kanban board to monitor and manage continuous increment.

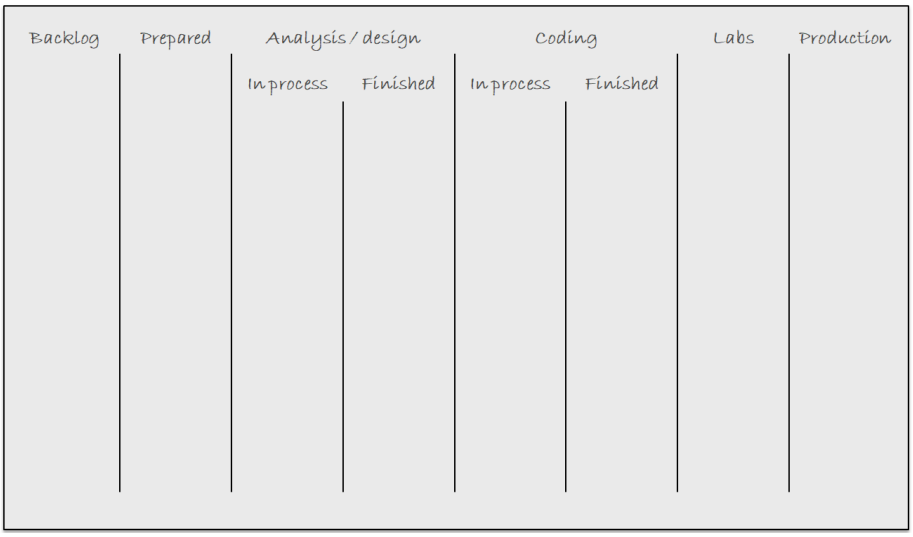


Illustration 51: Example of kanban board to monitor and manage continuous increment.

Case 2: Iterative increase.

Dashboard to guide the work management of a team working in iterative increments (sprints) and showing:

- Tasks backlog
- Prepared tasks.
- Tasks in analysis.
- Tasks in coding.
- Finished tasks.
- Integrated tasks on the development server (labs)
- Integrated tasks in production.

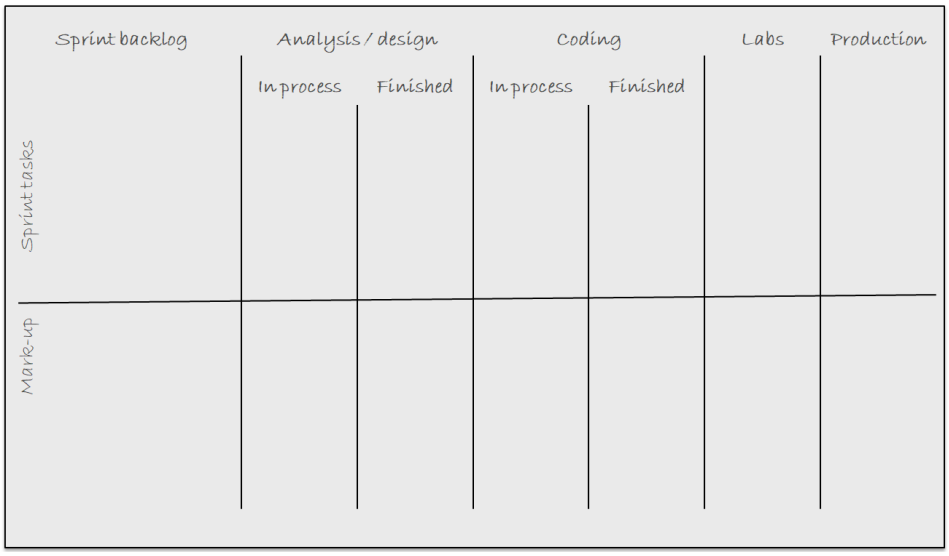


Illustration 52: Sample kanban dashboard to monitor and manage iterative increment.

Board for an operation and maintenance team.

Guidance board for the management of an operation and maintenance equipment that reflects:

- Status of scheduled tasks for the week and person who is working with each one.
- Status of unforeseen and urgent incidents, and people who are working in each one.

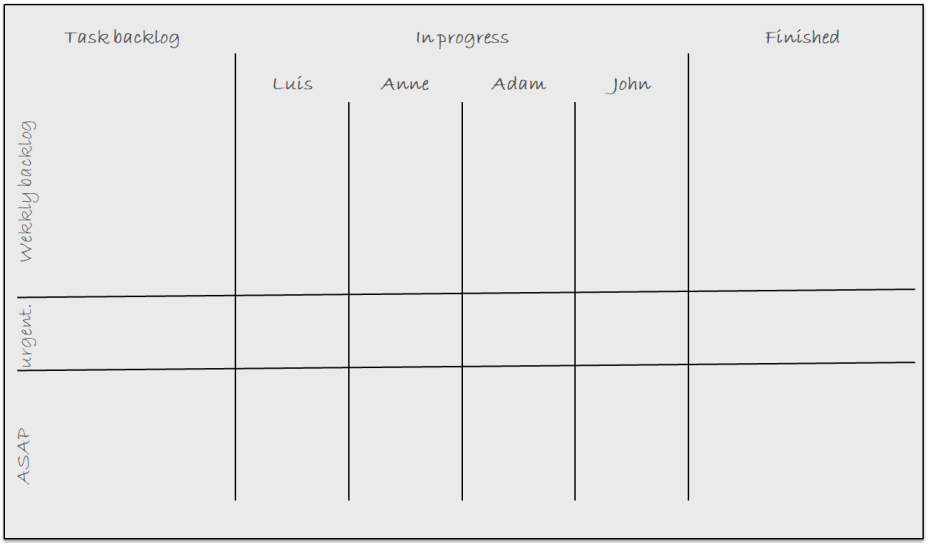


Illustration 53:Example kanban board to monitor and manage maintenance tasks.

Tips for adjusting the flow: Muda, Mura and Muri.

Muda, Mura and Muri are three key concepts of Kaizen continuous improvement, which Lean production incorporates as key variables to increase efficiency.

- Muda: Waste
- Mura: Discrepancy. Workflow variability. Interruptions in the workflow. Time-out.
- Muri: Tension. Work overload that produces bottlenecks.

Mudas

The most common mudas or wastes in IT projects are:

- Bureaucracy: Procedures, documentation and unnecessary paperwork that do not add value to the result.
- Overproduction: Develop more characteristics than necessary.
- Multiproject: Alternate the time of work between several projects and interruptions of the workflow.
- Waits: Waiting times due to lack of cadence in the workflow.
- "Start doing": Ordering work to go forward tasks not enough defined and only thus avoiding people stop working.
- Capacity misalignments: Highly talented people assigned to routine tasks, and vice versa.
- Errors: Rework for bugs.

The kanban boards detect and help to manage the other two kaizen variables: Mura and Muri.

Determinant factors of Mura and Muri

It should be noted that each organization or type of project has its own characteristics of:

- Sequence: the tasks must be performed sequentially or not.
- Versatility, polyvalence or specialization of the team members.

And these are the factors that in each case determine the greater or less difficulty to maintain a continuous flow of development.
As it has been seen, teams of polyvalent members working with non-sequential tasks are the ones that can achieve a constant forward flow more easily.
When difficulties arise, the variables to be combined, depending on the possibilities in each case, are:

- Volume of demand.
- Backlog order or user story stack: if a bottleneck is produced in one phase, care should be taken to ensure that the next story to be entered on the board requires little effort from that phase.
- WIP or task limit in a given phase.
- Staffing: team size and specialization or polyvalence.

Muri

The WIP is an important variable for adjusting bottlenecks (Muri):
When using kanban to maintain a continuous increase, the concept of sprint disappears. An increase is no longer the result of a sprint, but every user story that is completed and delivered to the customer. In order to maintain a continuous flow of deliveries that increase the product in a sustained way, bottlenecks (Muri) must be avoided: thus the accumulation of tasks in a certain phase of the process. One way to do this is to limit the amount of work that can be accumulated in each phase.

The parameter that indicates the maximum number of tasks in an area of the kanban board is the so-called WIP: Work In Process, or "in-process inventory". It should not be confused with Work in progress, term that designates a work that has begun but is not yet finished.

A "WIP" value too low can cause jams in other phases, especially if the system is too rigid (sequential tasks and specialist equipment).
Experience teaches you to get the best fit to achieve the most continuous flow possible.
If previous experience is not available, and considering that tasks should not be larger than 4 ideal hours, the team must establish a starting criterion, and from there adjust it.
In this sense a generally useful recommendation (in teams of multipurpose members) is to start with a WIP equal to the number of team members x 1.5, rounding the result off by excess.
It is not advisable to work with tasks whose size is expected to exceed one day of work, and if this happens it is advisable to divide them into smaller ones.
Example:
The following figure shows a kanban board with a work limit in the states "Product analysed" and "In progress".

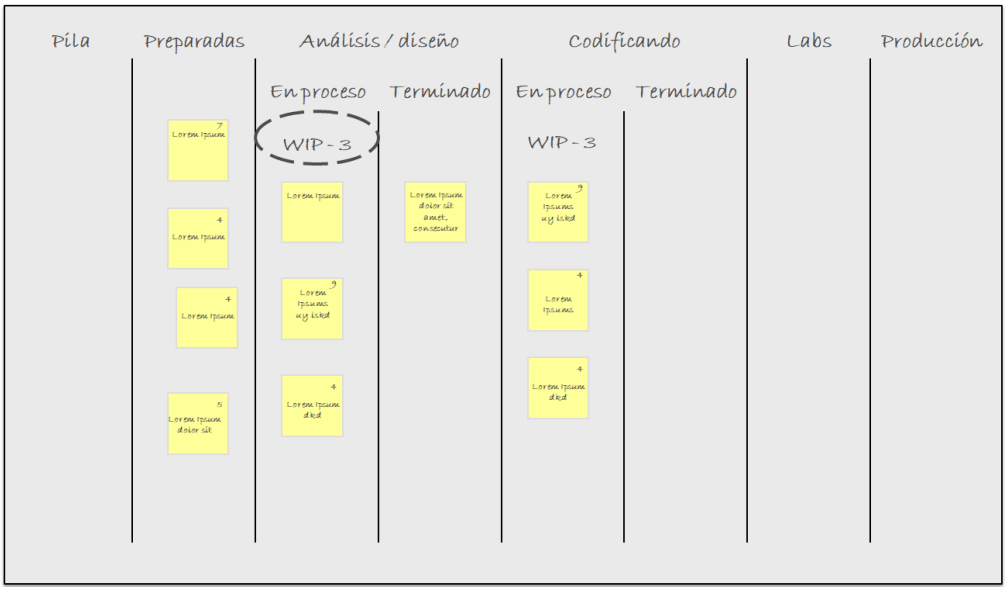


Illustration 54: WIP.

In this example, the product owner has a zone to sort the backlog (A). It is the area in which the product manager regularly adds, modifies, and prioritizes user stories.
But there are only three stories that can be in the "analysed" state to go to production. Three with which it is already planned to analyse and revise the estimate with the team.
Likewise, the "in progress" area has a limit of three stories. Until one goes to "DONE", any other can enter to production, and likewise as long as there are three in the "ANALYZED" zone it is not decided which will be the next story of the backlog to be picked.
This forces a workflow without jamming, continuous and focused.

Mura

The main factors responsible for the variability of the flow and the occurrence of Mura or dead times are:

- Degree of multifunctionality of the team members.
- Flexibility in the order in which the different phases of each task must be done.
- Flexibility to alter the entry order of the user stories from the product stack.

The greater these aspects are, the easier it is to avoid the occurrence of downtime.

EXTENSIONS

Agile requirements engineering

User Stories

User stories are used for specifying requirements in agile methods; they are a brief description of the software functionality as it is perceived by the user (Mike Cohn, 2004).

They describe what the client or the user wants to be implemented and are written with one or two sentences using the user's common language. People's thinking is structured following a narrative, a story, this is how we understand the world. We can understand characters, desires and motivations, so the easiest way to acquire and retain knowledge is through stories. We put ourselves inside the protagonists so that we live as the story that tell us, we empathize, and at all levels involved in decisions making, even at the developer level, we make better decisions.

Each user story should be limited, this should be easily memorable and write on a card or post-it (card). Shortly before being implemented, they are explained by discussions with users and the definition of their associated validation criteria. As the changes are welcome in agility, it is not worthwhile to go deeper before, since at the time of its implementation these may have changed since they were written. The validation criteria allow the product owner / business user to confirm that the team has correctly collected the requirements; the team performs the appropriate tests and / or develop guided by tests with TDD, and finally verify that the history has been completed.

User stories are part of the functionality capture formula defined in 2001 by Ron Jeffries referred as the three C's:

Card - Conversation - Confirmation

These are an agile way of managing the user's requirements without having to elaborate a large quantity of formal documents and without requiring a lot of time to administer them.

Advantages provided by user stories:

- Being very short, these represent business model requirements that can be implemented quickly (days or weeks).
- They need little maintenance.
- They allow to maintain a close relationship with the client.
- They allow to divide the projects in small deliveries.
- They make it easy to estimate the development effort.
- They are ideal for projects with volatile or not very clear requirements.

The origin of user stories comes from XP "eXtremeProgramming" or extreme programming, where user stories must be written by clients. This methodology was created by Kent Beck and was described for the first time in 1999 in his book "eXtreme Programming Explained".

The user stories are applied in most agile methodologies, being a very important tool also in scrum.

Epics, themes and tasks

The so-called epics are high level user stories that are distinguished by their large size, unlike user stories, which have low granularity, epics have a high granularity. It is a label that we apply to a large story, the effort of which is too great to complete in one go or in a single sprint. Epics often have an associated flow by which they can be divided into user stories, in other words, user stories resulting from the decomposition of an epic are intimately related to each other. As it increases its priority and the moment of its implementation approaches, the team breaks the epic down into user stories with a more adequate size to be managed with agile principles and techniques which are close estimation and monitoring (usually daily).

Above the epics are the themes that represent a collection of epics and / or related user stories to describe a system or subsystem. For example, in a software system for accounting management, the set of epics: "create, update, and delete customer's file", "One time and recurring invoices preparation", "Navigation queries and loyalty actions", "Orders preparation", "Returns treatment" Could be referred to as the subject of customer management.

Below the user stories are the tasks. These are the result of the team's decomposition of user stories into appropriate work units to manage and track the progress of their execution.

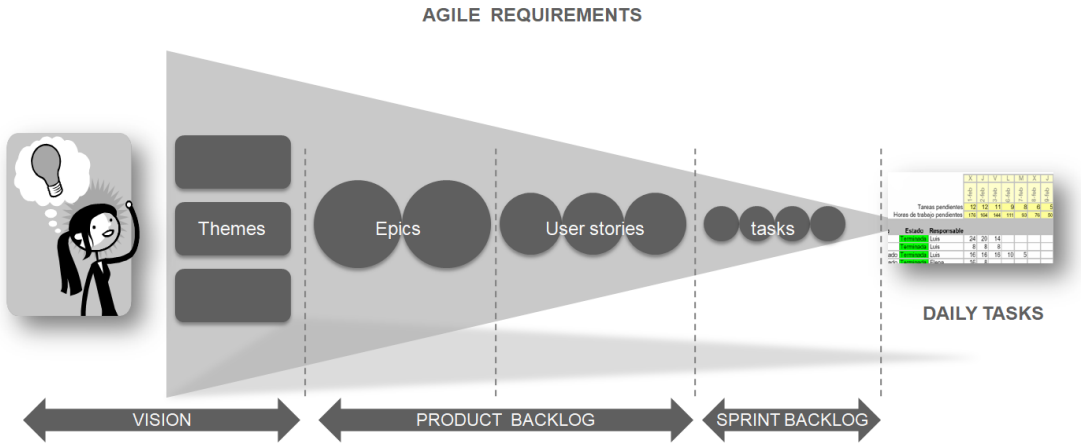


Illustration 55: Graph with the four levels of size with which agile manages the requirements.

In scrum, and in agile methodologies in general, a product backlog can contain both user stories and epics. The product backlog must be sorted by priority and the level of detail of each item must be relative to the position of the item inside the backlog. In a long list, it does not make any sense to have in detail something with very low priority, because probably it will change throughout the project, and it may even not be developed. From the middle to the bottom of the list, where the least priority is, the epics are placed. As we move closer to the higher priority elements, detail must increase, so user stories are the elements that should be at the top of the list.

Information in a user story

Necessary and optional information

When deciding what information is included in a user story, it is preferable not to adopt rigid formats. The results of scrum and agility do not depend on the forms, but on the institutionalization of its principles and the proper implementation of the characteristics of the company and the project. Therefore, apart from 4 fields that are considered essential, any field that provides useful information for the project can be included.

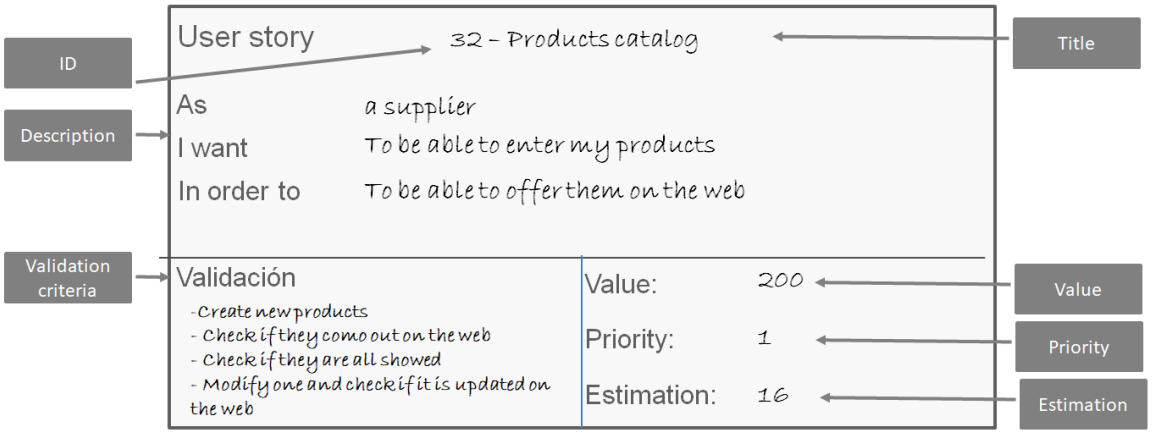


Illustration 56: Sample of a user story card.

The fields that are considered essential to adequately describe user stories are:

- **ID:** individual identifier of the user history, unique to the functionality or job.

- **Description**: synthesized description of the user story. The style can be free, the most appropriate, but three questions must be answered: Who benefits? What do you want? And What is the benefit? Mike Cohn recommends following the next pattern which ensures that the functionality is described at high level and in a not too extensive way:

As [user role], I want [feature] to be able to [benefit]

- **Estimation**: Estimation of the effort required expressed in ideal time to implement the user story. Depending on the team's suitability, it is also possible to use development units, known as story points (these units represent the theoretical development time / person stipulated at the beginning of the project).
- **Priority**: prioritization system that allows us to determine the order in which user stories should be implemented.

Depending on the type of project, the internal organization of the team and the structure and procedures of the organization, other fields may be advisable, such as:

- **Title**: descriptive title of the user story.
- **Validation criteria**: acceptance tests agreed with the client or the user. These are the tests that the code must surpass to consider as finished the implementation of the user story.
- **Value**: value (usually numeric) that brings the user story to the client or user. The objective of the team is to maximize the value and the customer perceived satisfaction in each

iteration. This field together with the estimate will serve to determine the priority with which the user stories should be implemented.

- **Dependencies**: a user story should not be dependent on another story, but sometimes it is necessary to maintain their relationship. In this area, we would indicate the identifiers of the other stories on which it depends.
- **Person assigned**: when we want to suggest the person, who can implement the user story. Remember that in scrum is the self-managed team who ultimately distributes and therefore assigns the tasks.
- **Termination Criteria**: The definition of finished / done includes the criteria or activities necessary to terminate a user history (developed, tested, documented ...), which are agreed by the team and the owner of the product.
- **Sprint**: It may be useful for the product owner's organization to include the sprint number where the story is likely to be made.
- **Risk**: technical or functional risk associated with the implementation of the user story.
- **Module**: module of the system or product to which it belongs.
- **Remarks**: to enrich or clarify the information or any use that may be useful.

User story			
Number: 1	User: customer		
Story name	Change shipping address		
Business priority	High	Development risk	Low
Estimated points	2	Iteration assigned	1
Programmer responsable	John Smith		
Description As a customer I want to change the shipping address of an order so I can get it at home or at the office			
Validación The customer can hange the delivery address in any of the orderes that have pending shipment			

Illustration 57: : Example of a user story.

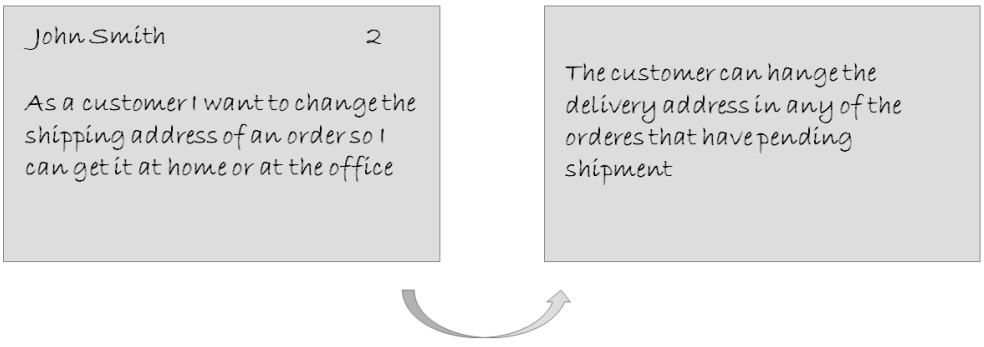


Illustration 58:Example of a user story.

Mike Cohn maintains that while user stories are flexible enough to describe the functionality of most systems, they are not appropriate for it at all. If for any reason, a need must be expressed in a way different from a user story, it is recommendable that it is done. For example, the layout of screens is usually described with screenshots, so this is the best way to convey the design that we want to give to a specific application.

Validation criteria

After 50 years of software engineering history we have arrived at the conclusion that the validation criteria, which are translated into tests, are an excellent language to detail functional requirements, and that is why they take on a great importance in the user stories.

To measure the quality of an acceptance criterion the SMART method is used, in which the following criteria must be met as much as possible:

- S - Specific
- M - Measurable
- A - Achievable
- R - Relevant
- T - Time-boxed

They are written as soon as the user stories that are likely to enter a sprint are obtained and refined in the sprint planning. Acceptance criteria help the development team to understand the operation of the product, so they will estimate better the size of the underlying story, and when the team is in the developmental phase and the developer needs to make a decision, he will take the most appropriate. Finally, the product owner checks in the sprint review, thanks to these acceptance criteria, if each of the user stories can be considered as done and finished. Acceptance criteria can be written in natural language, just as the product owner expresses them. Another option is to write them with the Behaviour Driven Development (BDD) own behavioural description technique or with gherkin, a specific language specially created for software behaviour descriptions. The syntax of gherkin is as follows:

(Scenario) Scenario [Scenario Id] [Scenario title]:
(Given) In case [Context] and additionally [Context]
(When) when [Event],
(Then) then the system [Result / Expected Behaviour]

Derived from this syntax, the elements of the acceptance criteria are:

- **Scenario Id**: Number (example 1, 2, 3 or 4), which identifies the scenario associated with the story.
- **Scenario Title**: Describes the scenario context that defines a behaviour.
- **Context**: Provides a more detailed description of the conditions that trigger the scenario.
- **Event**: Represents the action that the user executes in the context defined by the scenario.
- **Result / Expected behaviour**: Given the context and the action performed by the user, the consequence is the behaviour of the system in that concrete situation.

```
1 Scenario: Some determinable business situation
2 Given some predondition
3 And some other precondition
4 When some action by the actor
5 And some other action
6 And yet another action
7 Then some testable outcome is achieved
8 And some thing else we can check happens too
```

Illustration 59: Example of gherkin code.

Quality in user stories

In 2003 Bill Wake developed a method called INVEST to ensure quality when writing user stories. The method is used to check the quality of a user story by checking if it fulfils a series of characteristics:

- **I - Independent**
- **N - Negotiable**
- **V - Valuable**
- **E - Estimable**
- **S - Small**
- **T - Testable**

Independent
It is advantageous that each user story can be planned and implemented in any order. Therefore, the stories prepared should be totally independent (what facilitates the later work of the team). It must be highlighted the fact that the dependencies between user stories can be reduced combining them into one or dividing them differently.

Negotiable
A user story is a short description of a need that does not include details. Stories must be negotiable, as their details will be agreed with the client or the user during the conversational phase. Therefore, user stories with too many details should be avoided because it would limit the conversation about them.

Valuable
A user story must be valuable to the customer or the user. One way to make a valuable story is to write it yourself.

Estimable
A good user story must be able to be estimated with sufficient accuracy to help the customer, user, or product owner to prioritize and plan their implementation. The estimation will usually be performed by the work team and it is directly related to the size of the user story (a large user story is more difficult to estimate) and with the team's knowledge of the expressed need (in the case of lack of knowledge, more phases of conversation will be needed).

Small
User stories should encompass a few weeks of work at most. There are even teams that restrict them to days/person. A short description helps to reduce the size of a user story and facilitates its estimation.

Testable
The user stories should be able to be tested (confirmation phase of a user story). If the client or user does not know how to test the user story it means that it is not completely clear or it is not valuable. If the computer cannot test a user story it will never know if it has finished or not.

There are companies that have designed their own cards like these from Braintrust that, apart from giving a professional appearance and Include all the required information and each field has the blank space enough to be filled

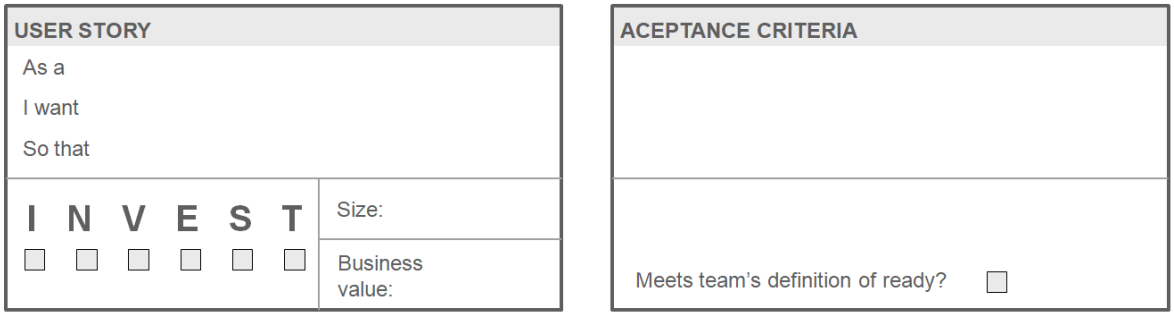


Illustration 60: Example of user story cards designed by Braintrust.

Good practice advices:

- Always write the stories with the "what", avoid the "how".
- Do not write an exhaustive description, just the right one.
- Write the validation criterion and be explicit enough.
- Estimate all the stories, not doing so can create false expectations.
- Do not trust all the information on the cards, sometimes an external documentation such as a wiki is a good idea.
- Never give a story as finished when it is "almost done".

Prioritisation of user stories

Although all user stories can be important, to focus on the work efficiently, it is necessary to highlight those that give a greater value to the system, thus, user stories should be prioritized. The owner of the product must assign a value to them; this value is involved in the prioritization system and basically it will be based on the following variables:

- Benefits of implementing the functionality.
- Loss or cost that requires postponing the implementation of the functionality.
- Risks of implementing it.
- Consistency with business interests.
- Differential value with respect to competitive products.

One of the aspects to keep in mind is that the definition of "value" may vary for each of our customers. Beyond a classification system of high, medium or low priority it is highly recommended to use qualitative scale, one that has an intrinsic meaning. This is the case for the MoSCoW technique, in which the user responsible for assigning the priority is aware of the real effect that his choice will produce. This technique was first defined in 2004 by Dai Clegg of Oracle UK Consulting in the book Case Method Fast-Track: A RAD Approach. Its purpose is to obtain a common understanding between the client and the project team, specifically on the importance of each user story. The classification is as follows:

- M - MUST HAVE (it is necessary): You must have the functionality implemented in the solution, else this will fail or the solution cannot be considered a success.

- S - SHOULD HAVE (it is recommended): You should have the functionality implemented in the solution as it is a high priority functionality. The solution is expendable, it will no fail if it does not exist but there should be justified causes for not implementing it.

- C - COULD HAVE (could be implemented): It is a feature desirable; therefore, it would be convenient to have this functionality implemented in the solution, it will depend on the time and the project budget.

- W - WILL NOT HAVE (we do not want it ... maybe in the future): This is a very low priority or discarded functionality at this moment, but it may be relevant in the future. Subsequently, when it becomes important, it can move to one of the previous states.

It is important to distinguish between priority and value for the customer. It may be that a user story has no value for the client or user, but that it is absolutely required, therefore of high priority. For example, the infrastructure necessary for the implementation of a software does not bring value to the client itself, but without it cannot develop or execute the solution developed.

User Stories Division

As part of the ongoing flow of requirements taken through the user stories, there are refinement meetings held to keep the product backlog updated. A refinement meeting is a meeting hold by the product owner who, together with the team, works on the refinement of the product backlog. In this meeting, as seen in the image on the right, elements of the pile are added, reprioritized, deleted and divided. Its aim is to ensure that the user stories that can be developed in the short term have a level of detail and an effort estimation enough for the team commitment

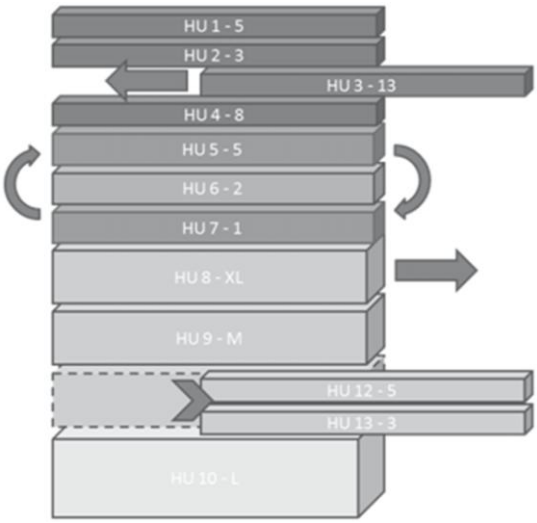


Illustration 61: Refining the product backlog.

Experience shows that smaller user stories improve the flow and that large user stories imply greater functional uncertainty and greater estimation difficulty. Dividing stories leads to a better understanding of them, increasing the accuracy of estimates and making them easier to prioritize.

We can divide user stories horizontally and vertically. Horizontal means division according to the type of work, by technologies for example, typical of traditional methodologies. This way of dividing horizontally generates stories that do not have business value individually, only the set of all of them has value. The horizontal division favours "thinking silos" in which each member of the team focuses only on those of their specialty, a situation that tends to produce bottlenecks. The agile requirements engineering avoids this type of problems with the multifunctionality of its members, every member participates to a greater or lesser extent in the different types of tasks. Finally, the stories from horizontal divisions cannot be prioritized as they do not provide any business value individually.

Unlike horizontal division, the vertical division is more useful, a history vertically divided generates stories that have business value themselves; the functionality is not divided according to technical layers but to functional layers. The sprint increments and the resulting stories are like a portion of a pie that includes all the necessary technical layers.

Christiaan Verwijs describes 10 strategies for dividing user stories vertically:

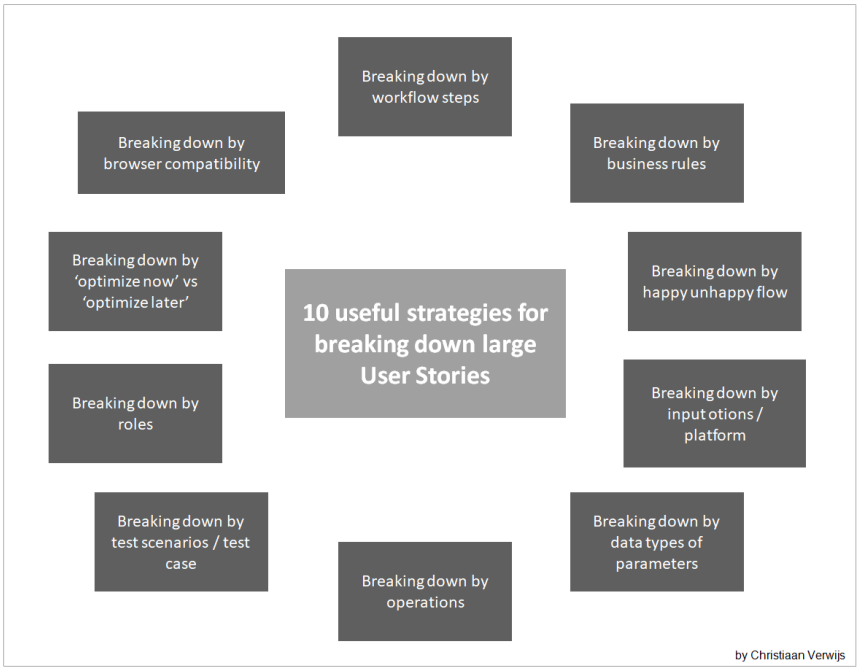


Illustration 62: Schema with 10 strategies for dividing user stories.

- Strategy 1 - Workflow Step Breakdown: For user stories that include some type of workflow, they can be divided according to the individual steps of the flow.
- Strategy 2 - Division by business rules: user stories that implicitly or explicitly involve business rules can be divided by these rules. Often test cases involve important business rules, so for this division we can rely on the tests.
- Strategy 3 - Division by happy / unhappy flow: functionalities usually describe a flow in which all goes well and other flows in which deviations, exceptions or problems are dealt, therefore these flows are also a way of dividing big stories.
- Strategy 4 - Division by entry options / platforms: in case of products that must work on different platforms, like portable, tablets, mobile ... the user stories can be divided by its entrance platform.
- Strategy 5 - Division by data types or parameters: some user stories can be divided by their input or output parameters, such as the different options of a search.
- Strategy 6 - Division by operations: There are stories that involve the typical operations of high, reading, modification and low (CRUD - create, read, update & delete), these operations can be another form of division.
- Strategy 7 - Division by case / test scenarios: sometimes there are stories that are difficult to divide by function, in this case it can help to ask what the test scenarios of the story will be and divide it by them. Scenarios can be a combination of business rules, flows that go well and with exceptions, entry platforms, etc.
- Strategy 8 - Division by roles: the user stories that cover different roles can be divided by the functionalities of each role.
- Strategy 9 - Division to optimize now or later: user stories can be implemented in different degrees of perfection and optimization of the functionality described.
- Strategy 10 - Browser Compatibility Division: User stories for web applications often must work on a wide variety of browsers, the more modern ones tend to be more compliant with standards, and the older ones often need customizations so that everything works correctly.

In any of these strategies, the division reduces uncertainty and allows the development of the important stories leaving the less important stories to future developments.

Comparison with other ways for requirements management

User Stories versus Use Cases

Whenever use cases and user stories are mentioned, some confusion occurs. Some have managed to include use cases in their agile processes but that does not mean that user stories are equivalent to use cases.
A use case is based on UML, a descriptive language initially intended for simplicity in communication but which is not close to human communication. However, a user story is written in a colloquial language, which functions as a reminder of the conversation with the client.
As Alistair Cockburn commented in 2001 in his book “Agile software development”, user stories are closer to the requirements capture, the phase that serves to extract the user's needs, whilst use cases are closer to the requirements specification.
We could say that a user story says "what" the customer or user needs and a use case is a "how" you want it.
In the validation criterion, there are also conceptual differences: unlike the use cases that require requirements tracking matrices with finished percentages, user stories that include the validation criterion include the terminated binary form, or value or they are not worth it.
Agility itself is evident since user stories are alive. The functional and technical analysis is done shortly before the development, in the meeting at the start of sprint in case of scrum. The breakdown in tasks is done by the team, whose level of detail and foresight far exceeds what can be done by a single architect or functional analyst in use cases.

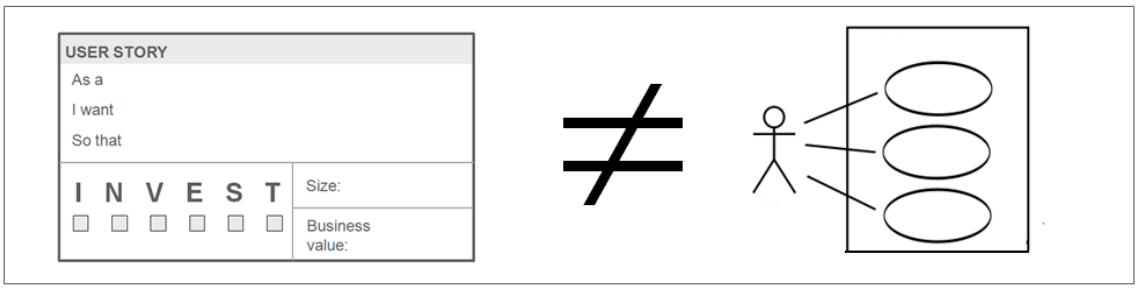


Illustration 63: User stories are not like use cases.

When a project begins to follow up an agile methodology should completely forget the use cases and the team should focus only on the realization of the user stories.
For those who are interested in supplementing user stories with use cases I invite you to read Cockburn's book, in which he describes the problems that may arise and how he gave a solution to deal with them.

User Stories vs. Functional Requirements

Usually, user stories are associated with functional requirements, some of which are artefacts of agile methodologies and others of traditional methodologies. However, behind the user stories there are aspects that differentiate them, not knowing these differences lead to very common confusions.

We must be aware that although user stories describe functionalities that will be useful to the client or user, and are usually written on cards or post-its, they are much more than that, since they involve a subsequent conversation in which the team details along with the user or client the functionality to be developed.

As functional requirements, user stories tell the "what" but not the "how" functionality will be developed. Therefore, user stories should not have the level of specification details of a functional requirement.

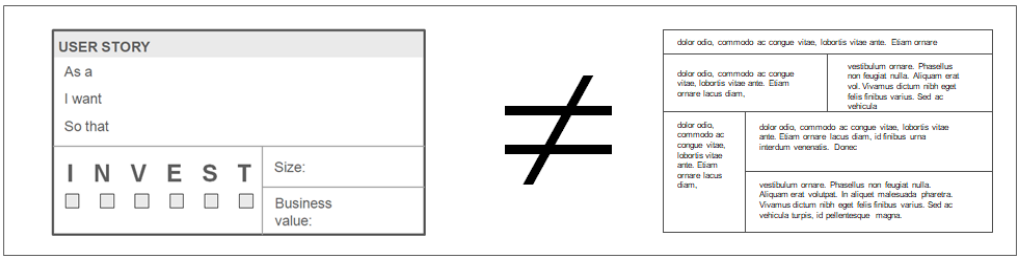


Illustration 64: User stories are not like functional requirements.

Bibliography

Bauer, F., Bolliet, L., & Helms, H. (1969). Software Engineering. Report on a conference sponsored by the NATO SCIENCE COMITEE. *Software Engineering* (pág. 136). Garmisch: Peter Naur & Brian Randell.

Beck, K. (2000). *Extreme Programming Explained.* Addison-Wesley.

Grenning, J. (2002). *Planning Poker or How to avoid paralysis while release planning.*

Hino, S. (2006). *Inside the Mind of Toyota: Management Principles for Enduring Growth.* Productivity Press.

Kniberg, H., & Skarin, M. (2009). *Kanban and Scrum, making the most of both.* crisp.

Nonaka, I., & Takeuchi, H. (1995). *The Knowledge-Creating Company.* University Press.

Nonaka, I., & Takeuchi, H. (1986). The New New Product Development Game. *Harvard Business Review* .

Nonaka, I., & Takeuchi, I. (2004). *Hitotsubashi on Knowledge Management.* Singapore: John Wiley & Sons.

Ohno, T. (1988). *The Toyota Production System: Beyond Large-scale Production.* Productivity Press.

Orr., K. (2003). CMM versus Agile Development: Religious wars and software development. *Cutter Consortium, Executive Reports 3(7)* .

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit for Software Development Managers.* Addison Wesley.

Schwaber, K. (1995). *SCRUM Development Process.* Burlington: OOPSLA 95.

Schwaber, K. (1995). SCRUM Development Process. *OOPSLA 95* , 17.

Smith, A. (1776). *An Inquiry into the Nature and Causes of the Wealth of Nations.* Londres: W. Strahan & T. Cadell.

Takeuchi, H., & Nonaka, I. (1986). *The New New Product Development Game.* Cambridge: Harvard Business Review.

Taylor, F. W. (1911). *The Principles of Scientific Management.* New York: Harper & Brothers.

Turner, R., & Jain, A. (2002). Agile Meets CMMI: Culture Clash or Common Cause? *XP/Agile Universe 2002* , 153-165.

Illustrations table

Index