Agile Software Development

UA.DETI.IES

Topics covered

- Agile principles
- Agile development techniques
- Agile project management
- Scaling agile methods

Topics covered

- **Agile principles**
- Agile development techniques
- Agile project management
- Scaling agile methods

Why Agile?

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a **fast–changing requirement** and it is practically impossible to produce a set of stable software requirements.
  - Software must evolve quickly to reflect changing business needs.
- Agile principles:
  - **Focus on the code rather than the design**
  - Are based on an **iterative approach** to software development
  - Are intended to **deliver working software quickly** and evolve this quickly to meet changing requirements.

Abordagem iterativa!

12 Agile Principles

@OlgaHeismann

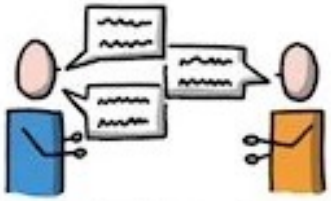
Satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development.

Deliver working software frequently.

Business people and developers must work together.

Build projects around motivated individuals. Give them the support they need. Trust them.



The most efficient and effective method of conveying information is face-to-face conversation.

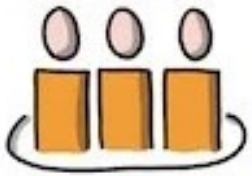
Working software is the primary measure of progress.

The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design.

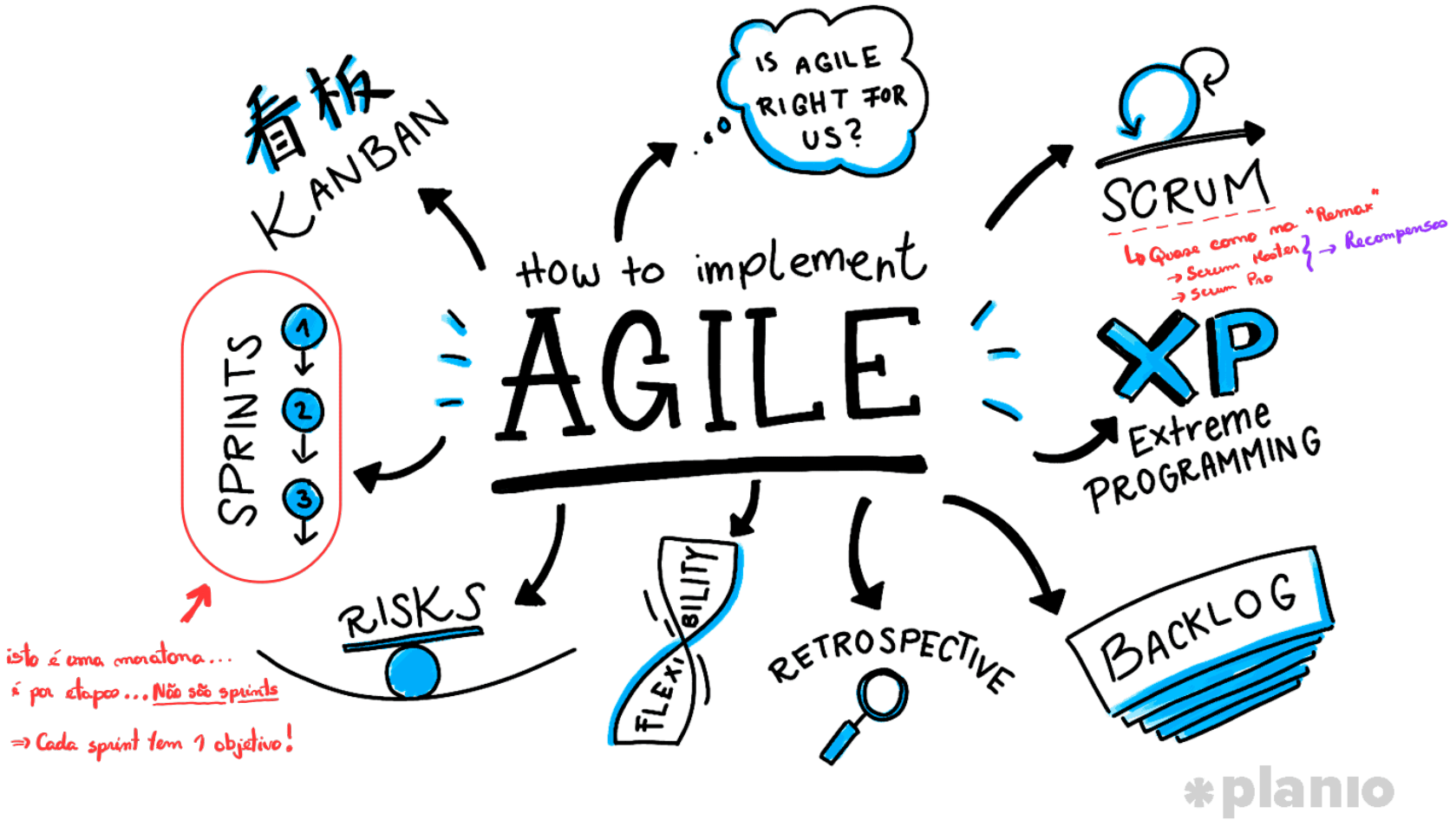
Simplicity—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

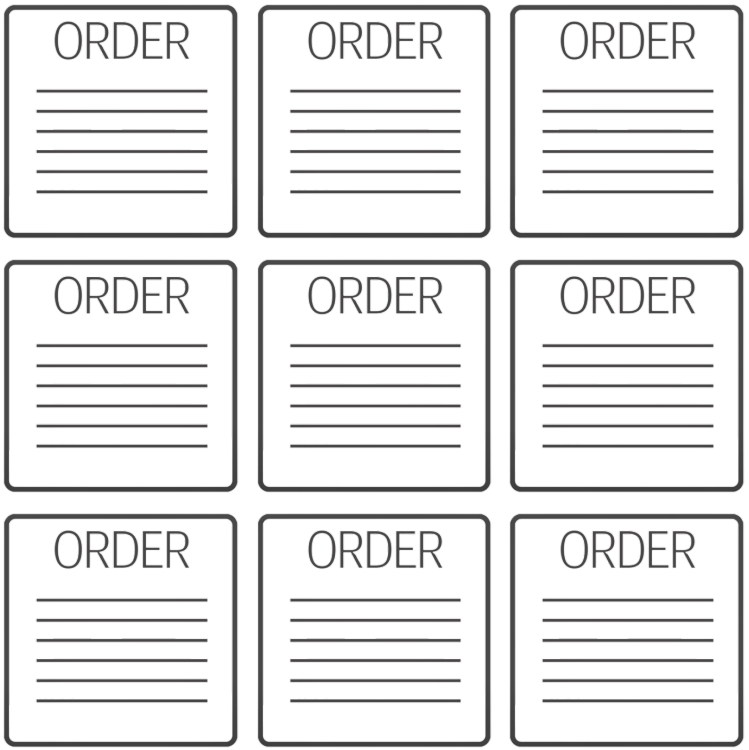


The team reflects on how to become more effective and adjusts its behavior accordingly.

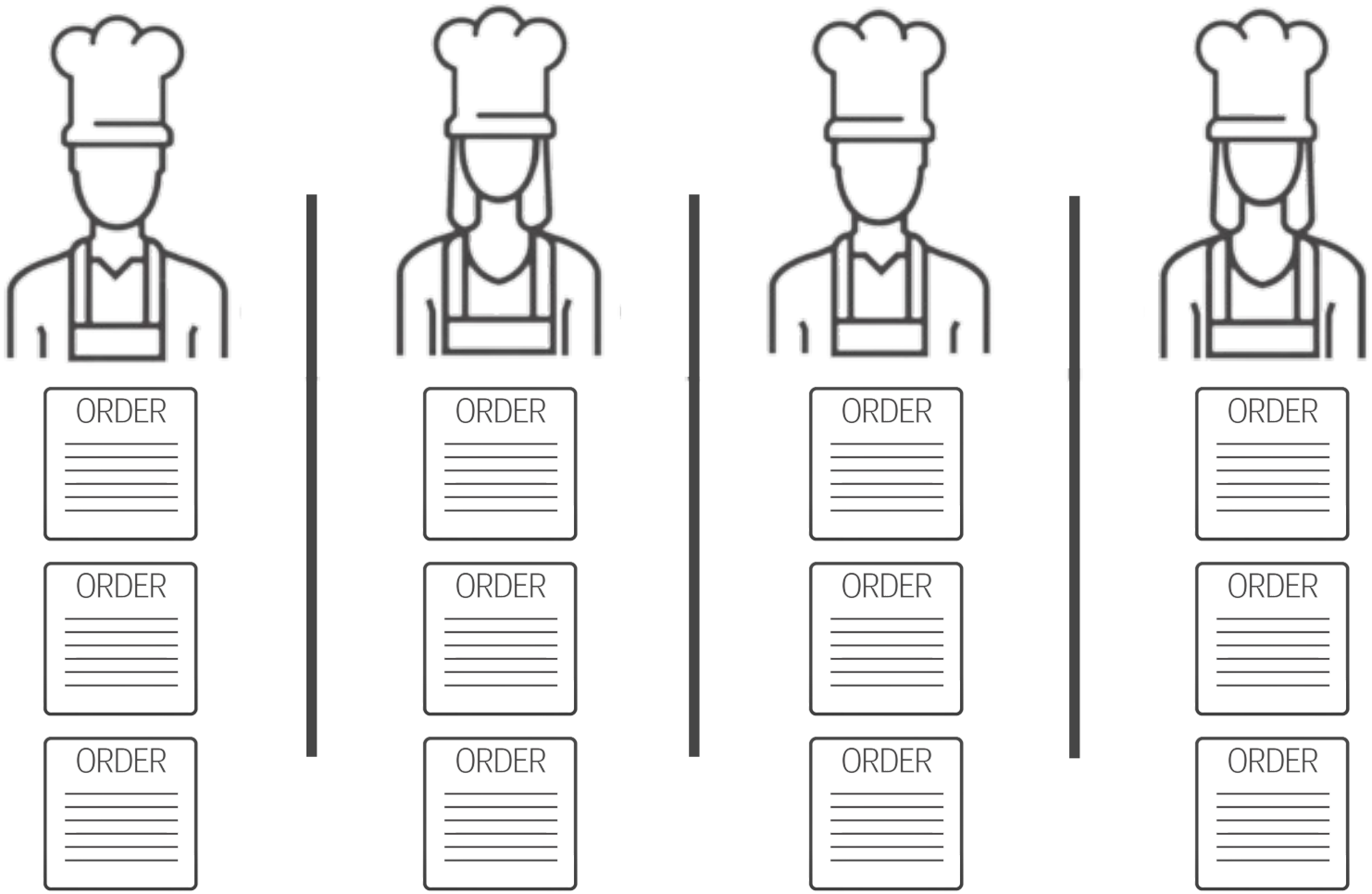
How to implement Agile?



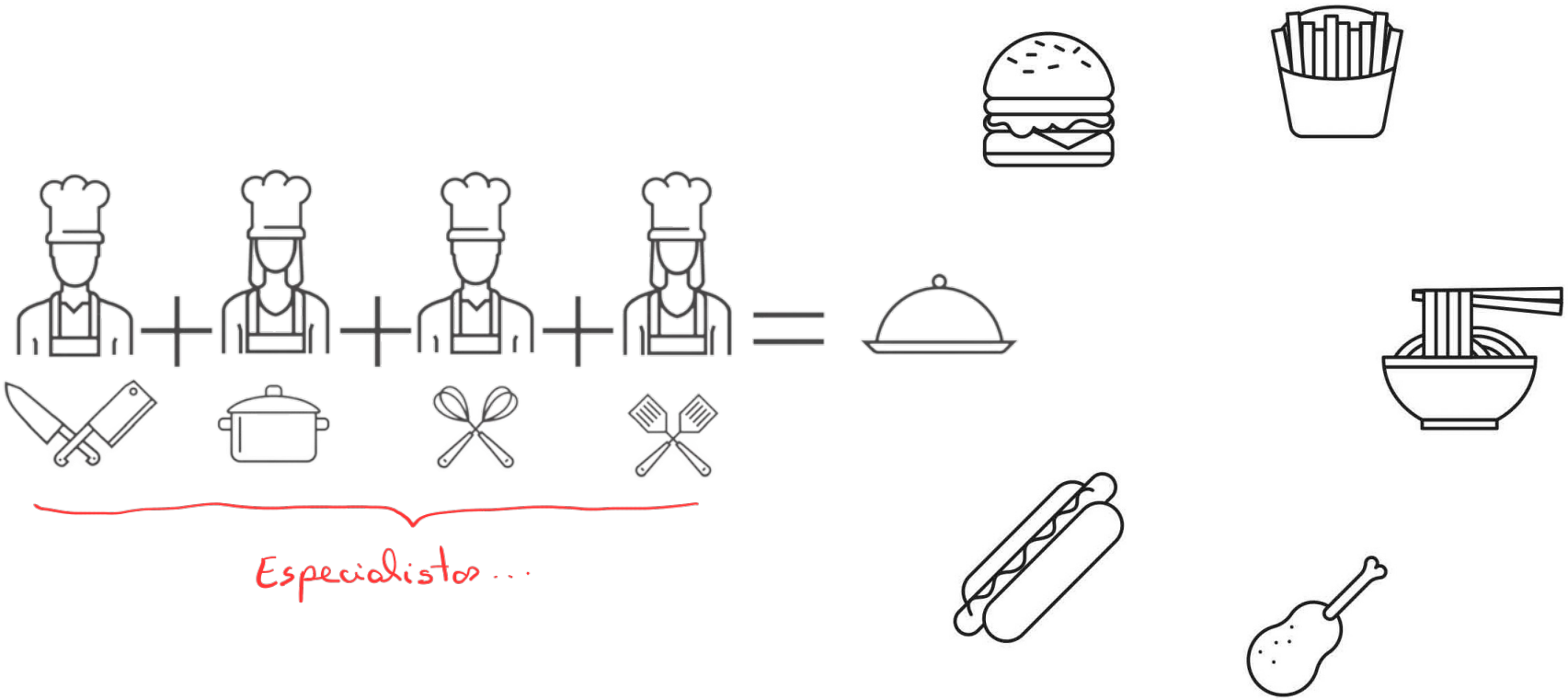
A simple an analogy - restaurant



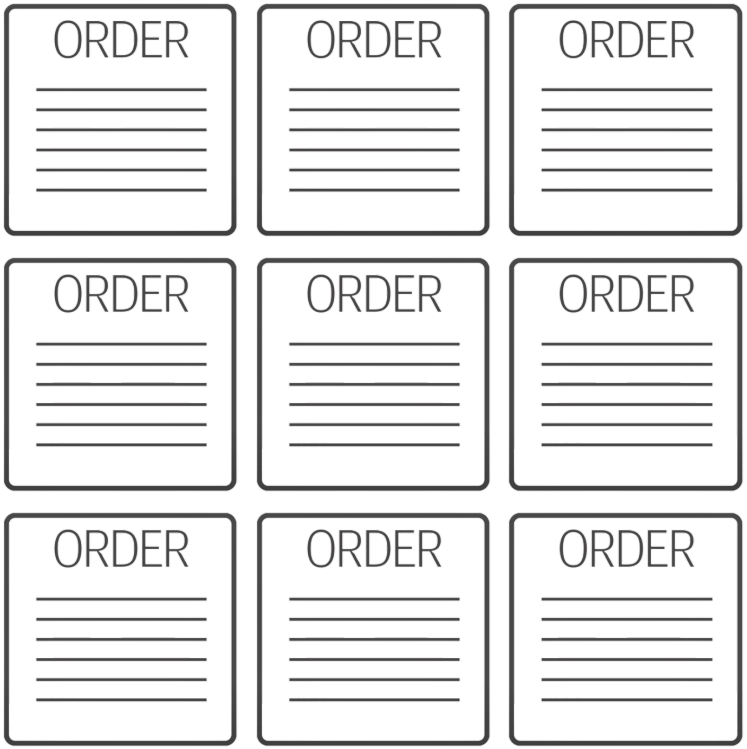
A simple an analogy - restaurant



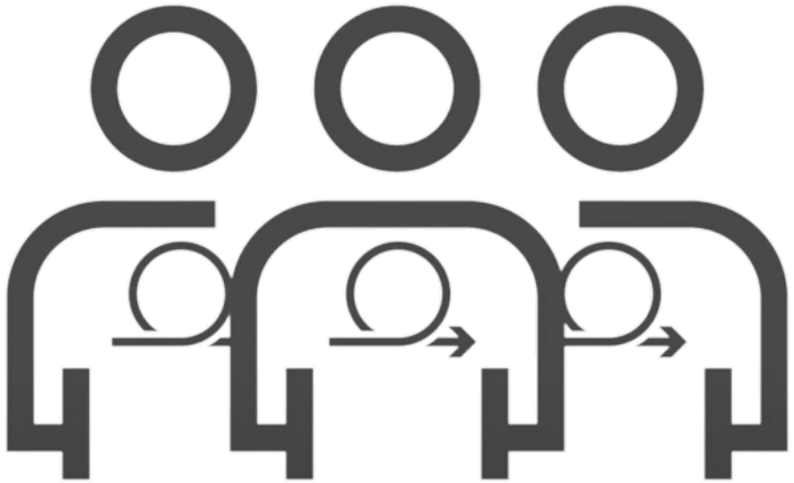
A simple an analogy - restaurant



A simple an analogy – software



A simple an analogy – software

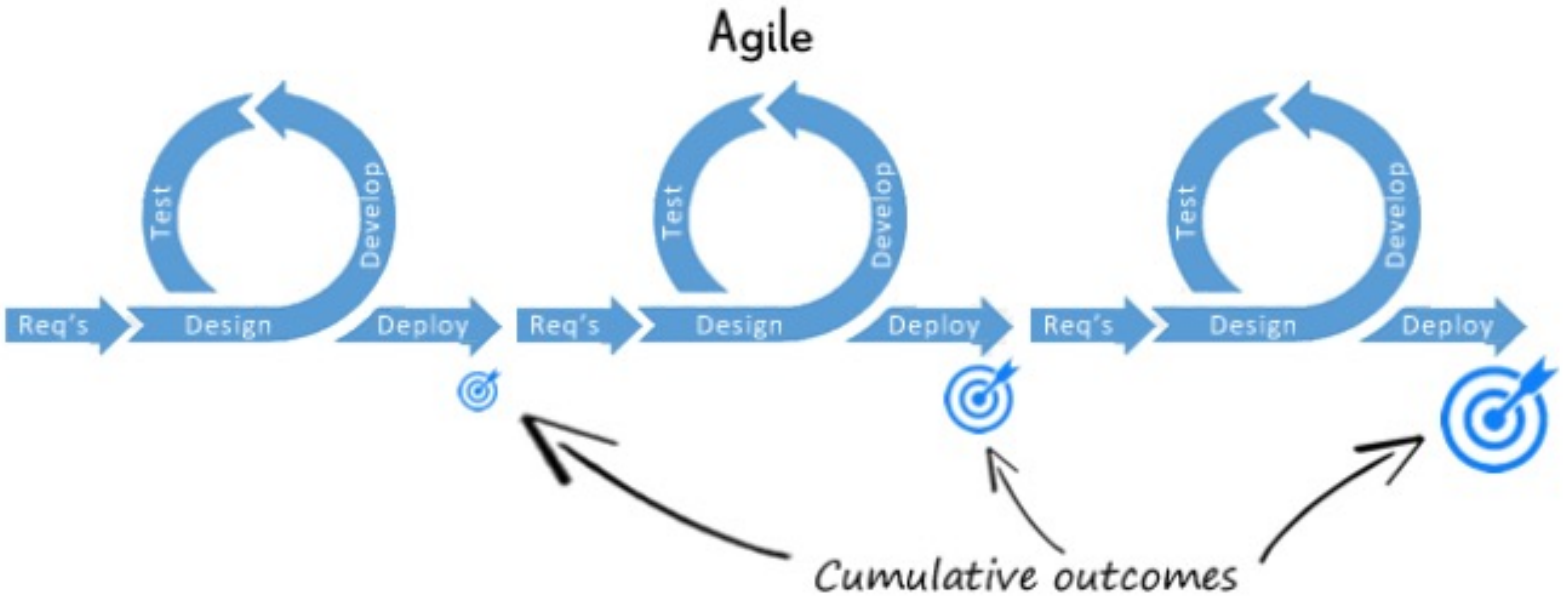


Recebe os pedidos...

A simple an analogy – software

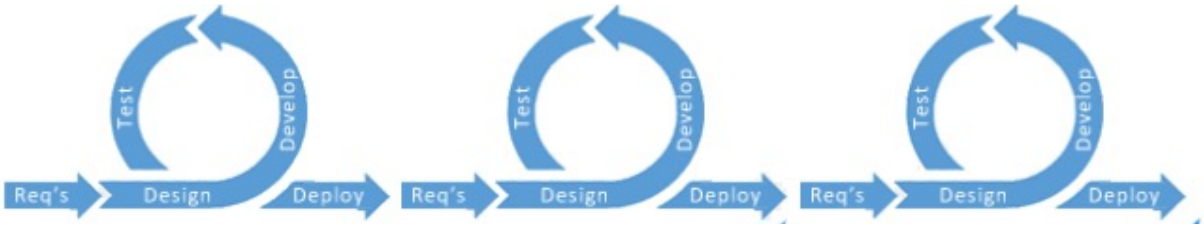
Topics covered

- Agile principles
- **Agile development techniques**
- Agile project management
- Scaling agile methods



Agile development

- Specification, design, implementation and evaluation phases are inter-leaved
  - The system is developed as a series of versions, involving stakeholders in the specification and evaluation
  - Frequent delivery of new versions for evaluation
- Extensive tool support (e.g., automated testing tools) used to support development
- Minimal documentation
  - focus on working code → O que eu gostava que estivesse aqui, para recomeçar daqui a 1 ano?

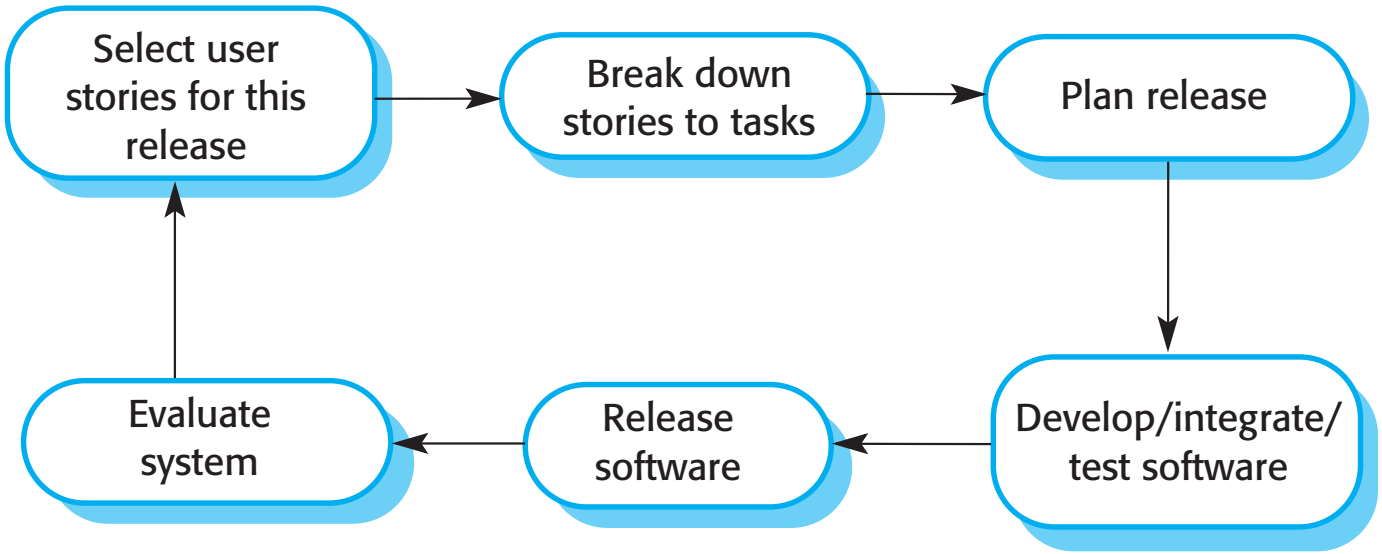


Extreme programming (XP)

Nunca pode ser em full time!

- *Extreme Programming* (XP) is the most widely used approach to agile software development.
- **It takes an 'extreme' approach to iterative development.**
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks; software diferente todos os dias
  - All tests must be run for every build and the build is only accepted if tests run successfully.
- Uses an object-oriented approach as its preferred development paradigm.
- Encompasses a set of rules and practices that occur within the context of four framework activities: **planning**, **design**, **coding**, and **testing**.

The XP release cycle



Extreme programming practices (I)

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices (II)

Principle or practice	Description
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop, and all the developers take responsibility for all the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium-term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

Influential XP practices

- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.
  - Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.

- Key practices
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming → o mesmo problema... código revisto pelo meu colega

User stories for requirements

Não só para o XP
também são úteis para o Agile

- In XP, a customer or user is part of the XP team and is **responsible for making decisions on requirements**.

Colocamos no perfil do cliente (aluno, professor, ...)
Mais fácil de perceber do que os requesitos

- User requirements are expressed as **user stories** or **scenarios**.

- These are written on cards and the development team break them down into **implementation tasks**.
  - These tasks are the basis of schedule and cost estimates.

- The customer chooses the **stories for inclusion** in the next release based on their priorities and the schedule estimates.

A ‘prescribing medication’ example

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either ‘current medication’, ‘new medication’ or ‘formulary’.

If you select ‘current medication’, you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

Porquê? → Falamos com o cliente...

If you choose, ‘new medication’, the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose ‘formulary’, you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click ‘OK’ or ‘Change’. If you click ‘OK’, your prescription will be recorded on the audit database. If you click ‘Change’, you reenter the ‘Prescribing medication’ process.

Examples of tasks

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

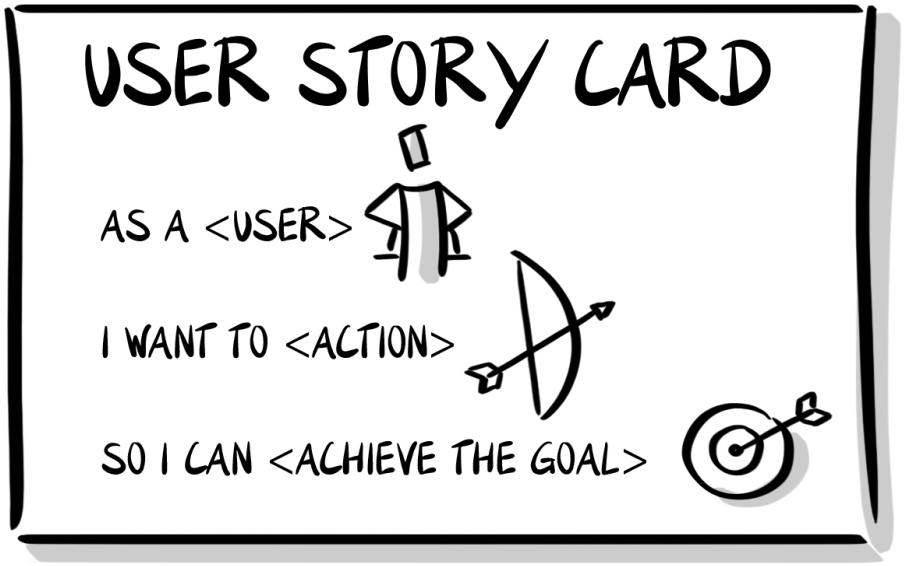
Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.
If within the range, enable the 'Confirm' button.

User Stories templates

1º Passo para cada iteração no Modelo Agile !

- Write them in the perspective of the customer

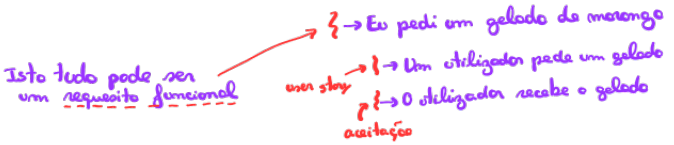
- Format
  - As a (role/user)
  - I want (functionality)
  - So that (benefit)



- These should include information that defines when they are finished (**acceptance criteria**)

User Stories templates

- As a (**user**), I (**want to**), (**so that**)



- **User:** User refers to the end user of the software.
- **Want to**: We are referring to the intent of end users, not the features they use. We need to specify what they're trying to achieve with the task, not to mention the UI of the app.
- **So that**: Describes the bigger picture. What's the end goal users are trying to achieve with the feature?

User Stories templates - example

As a student,

I want to submit my assignments online,

so that I can meet deadlines more efficiently.

- Acceptance criteria:
  - The system allows file uploads.
  - The submission is confirmed with a success message.
  - Assignments are timestamped upon submission.

Exercise #1

- Form groups of 4 (again)
  - These may not be the same groups of practical classes.

1. Use the same requirements from the previous class
2. Identify a set of personas relevant to the project
3. Write 4 user stories in a "report" format, considering
   1. Student ID numbers (nº mec.)
   2. Requirements
   3. User stories

User Stories organization

- Role-Goal-Benefit
  - It forces the customer to really think about who is going to benefit from a feature, what they're trying to achieve, and why they want to achieve that.
- Limitations, needs
  - Subset of situations that matter for this feature.
- Definition of Done
  - How to validate the feature and know that it is complete
- Engineering tasks
  - How this feature interacts with other features within the system or other subsystems.
- Effort estimate
  - It provides a concrete measurement about the value of a feature.

Refactoring

+Eficiente para leitura ou em tempo

- Conventional wisdom in software engineering is to design for change.
  - It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.

- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
  - Rather, it **proposes constant code improvement** (refactoring) to make changes easier when they have to be implemented.

Refactoring

- Programming team look for possible **software improvements** and make these improvements even where there is no immediate need for them.

- This improves the **understandability of the software** and so reduces the need for documentation.

  • Poucos comentários... O próprio código é autoexplicativo

- Changes are **easier to make** because the code is **well-structured and clear**.

- However, some changes requires architecture refactoring, and this is much more expensive.

  ↳ Arquitetura não poder mudar muito...

  ↳ Refactoring de arquitetura é difícil...

Examples of refactoring

- **Re-organization** of a class hierarchy to remove duplicate code.
- Tidying up and **renaming** attributes and methods to make them easier to understand.
- The **replacement of inline code** with calls to new methods that have been included in a program library.

Test-first development

- Testing is central to XP

- Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Run all component tests each time that a new release is built.

Test case description for dose checking

Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

Test automation

- Tests are written as **executable components** before the task is implemented. They should
  - be stand-alone,
  - simulate the submission of input to be tested,
  - check that the result meets the output specification.

- An **automated test framework** (e.g., Junit) makes it easy to write and execute tests.

- As testing is automated, there is always a set of tests that can be quickly and easily executed.
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

Problems with test-first development

- Programmers prefer programming to testing and sometimes they take **short cuts when writing tests**.
  - For example, they may write incomplete tests that do not check for all possible exceptions that may occur.
- Some tests are **very difficult to write incrementally**
  - For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.
- It is hard to judge the completeness of a set of tests.
  - Although you may have a lot of system tests, your test set **may not provide complete coverage.**

Pair programming

- Pair programming involves programmers working in pairs, developing code together.
  - programmers sit together at the same computer to develop the software.
  - It serves as an informal review process as each line of code is looked at by more than 1 person.

não à prova que eu revi...

→ somos os 2 responsáveis pelo código!

- This helps developing common ownership of code and spreads knowledge across the team.
  - It reduces the overall risks to a project when team members leave.

Pair programming

- In pair programming, Pairs are created dynamically so that all team members work with each other during the development process.
- It encourages refactoring as the whole team can benefit from improving the system code.

Não é bom na Universidade para aprender...

- Pair programming is not necessarily inefficient
  - There is some studies that suggests that a pair working together is more efficient than 2 programmers working separately.

Topics covered

- Agile principles
- Agile development techniques
- **Agile project management**
- Scaling agile methods

Agile project management

→ Role de Project Manager verifica !

- The principal responsibility of software project managers is to **manage the project**
  - so that the software is delivered on time and within the planned budget for the project
- The standard approach to project management is **plan-driven**
  - Defines what should be delivered, when it should be delivered and who will work on the project deliverables
- Agile project management requires a different approach
  - It is adapted to **incremental development** and the **practices used in agile methods**.

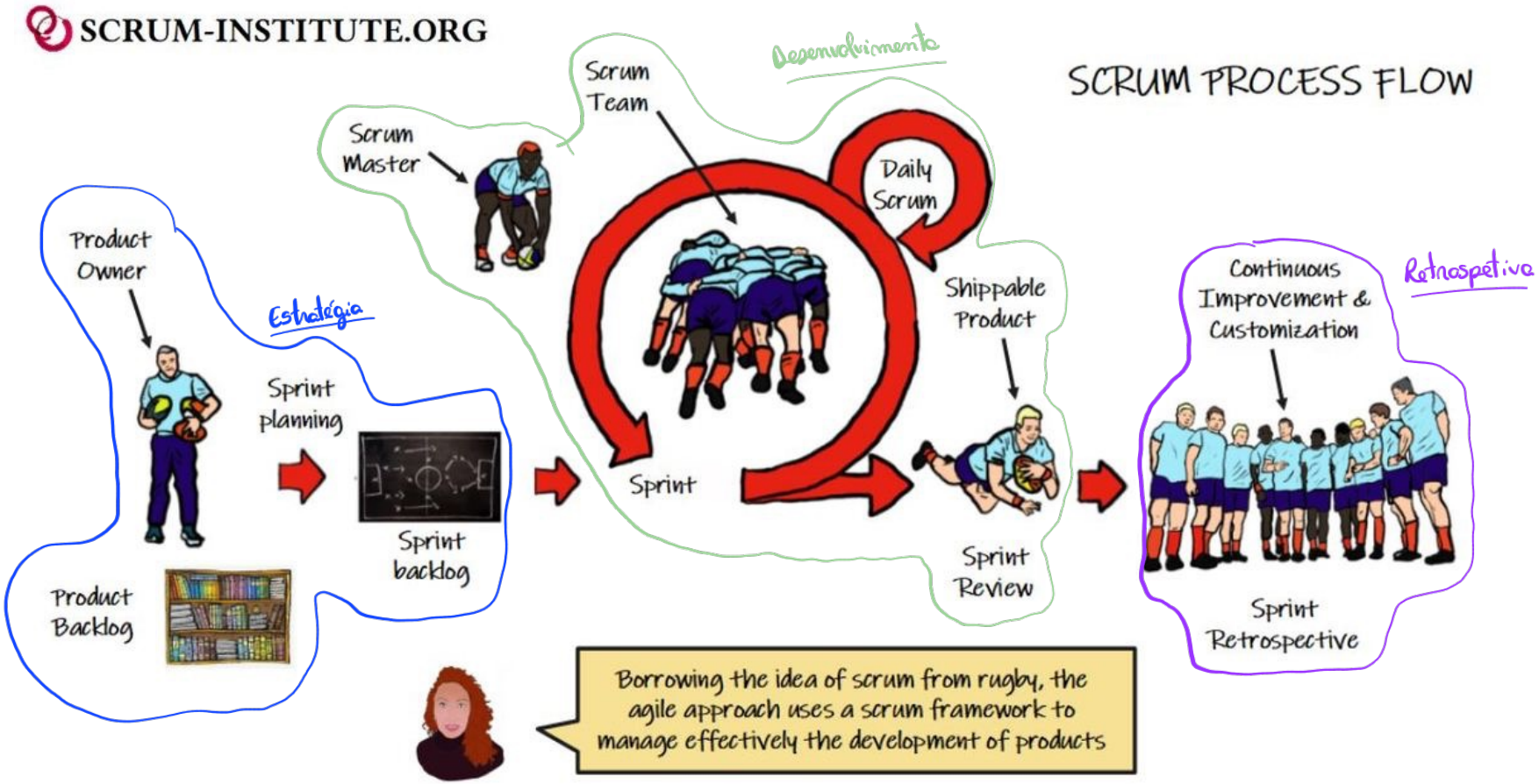
Scrum

- Scrum **is an agile method** that focuses on managing iterative development.

- There are three phases in Scrum.

  1. The initial phase is an **outline planning** phase where you establish the general objectives for the project and design the software architecture.
  2. This is followed by a series of **sprint cycles**, where each cycle develops an increment of the system.
  3. The **project closure** phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

Scrum


SCRUM-INSTITUTE.ORG
SCRUM PROCESS FLOW
Desenvolvimento
Scrum Team
Scrum Master
Daily Scrum
Product Owner
Estratégia
Sprint planning
Sprint backlog
Product Backlog
Sprint
Shippable Product
Sprint Review
Continuous Improvement & Customization
Retrospetiva
Sprint Retrospective
Borrowing the idea of scrum from rugby, the agile approach uses a scrum framework to manage effectively the development of products

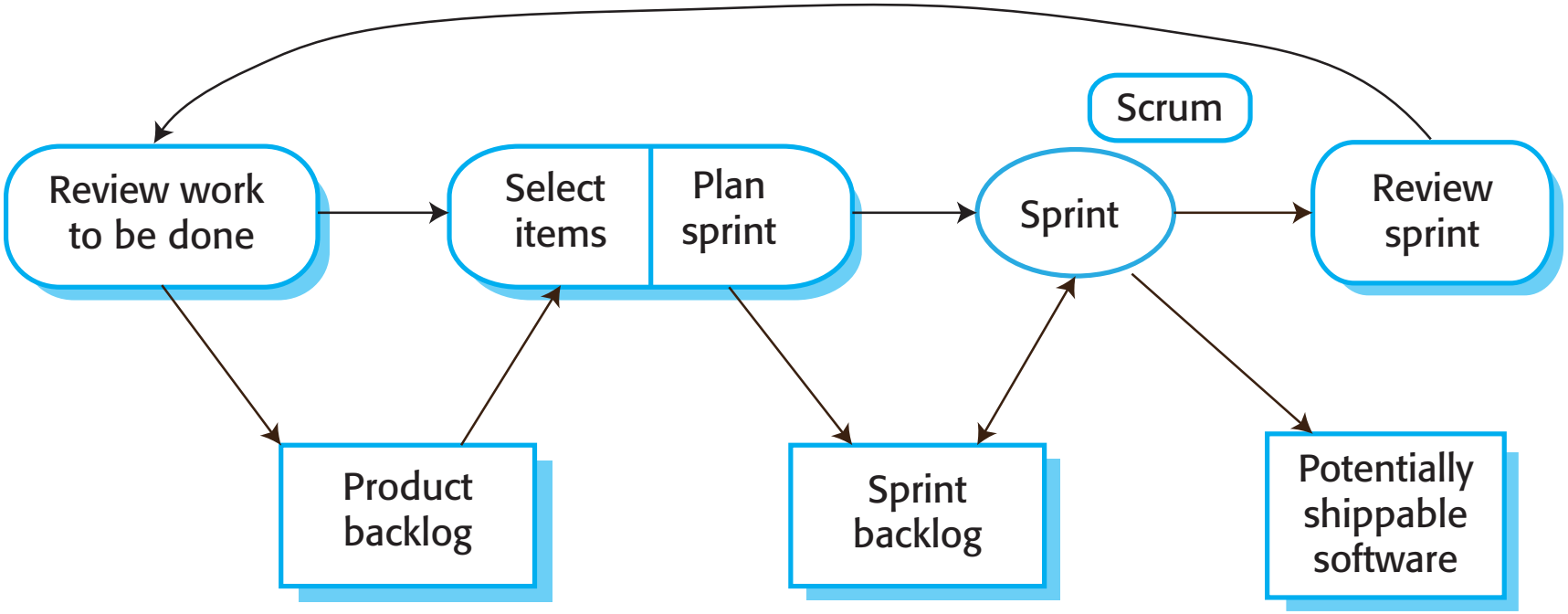
Scrum terminology (I)

Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Scrum terminology (II)

Scrum term	Definition
Scrum meetings	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum sprint cycle



Scrum sprint cycle

- The starting point for planning is the **product backlog**
  - It is the list of work to be done on the project.
- The **selection phase** involves all the team
  - who works with the customer to select the features and functionality from the product backlog to be developed during the sprint.
- During the **development stage** the team is isolated from the customer and the organization
  - All communications channelled through the 'Scrum master'
- At the **end of the sprint**
  - The work done is reviewed and presented to stakeholders.

Teamwork in Scrum

- The role of the **Scrum maste**r is to protect the development team from external distractions.
  - arranges daily meetings, ou semanais
  - tracks the backlog of work to be done,
  - records decisions,
  - measures progress against the backlog
  - communicates with customers and management outside of the team.
- The team attends short daily meetings (**Scrums**)
  - members share information, describe progress/problems since the last meeting, and what is planned for the day.
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

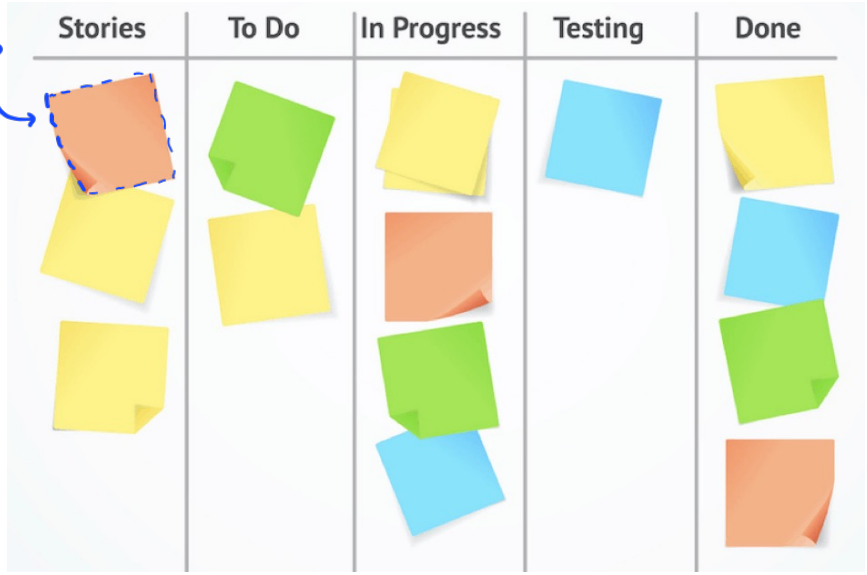
Scrum benefits

- The product is broken down into a set of manageable and understandable chunks.
- **Unstable requirements** do not hold up progress.
- The whole team have visibility of everything
  - Consequently, team communication is improved.
- Customers see on-time delivery of increments
  - Gain feedback on how the product works.
- Trust between customers and developers
  - Everyone expects the project to succeed

Kanban

Não é do Agile! "organizar tarefas"

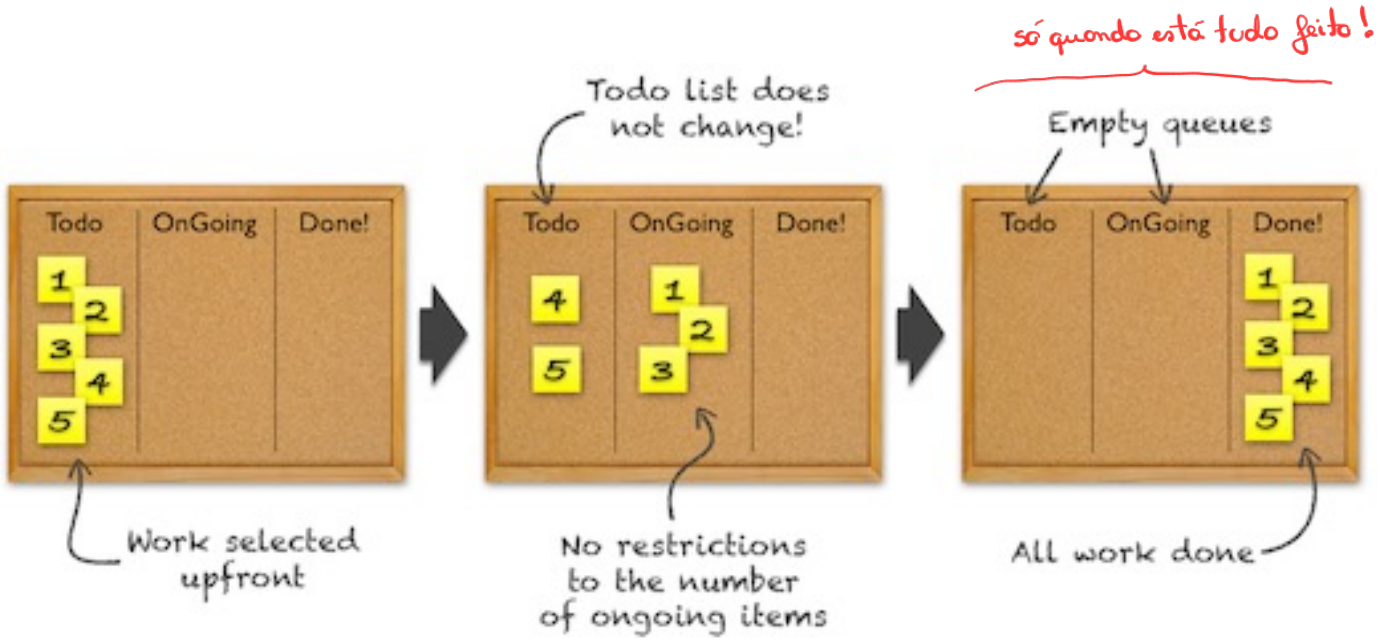
- **Kanban** is a lean agile system that can be used to enhance any software development lifecycle including Scrum, XP, or Waterfall.



- The name 'Kanban' originates from Japanese, it means "signboard", "billboard"
  - Used for Just in Time (JIT) manufacturing at Toyota manufacturing plants in Japan to limit the amount of inventory tied up in "work in progress" (WIP) on a manufacturing floor

SCRUM vs. Kanban

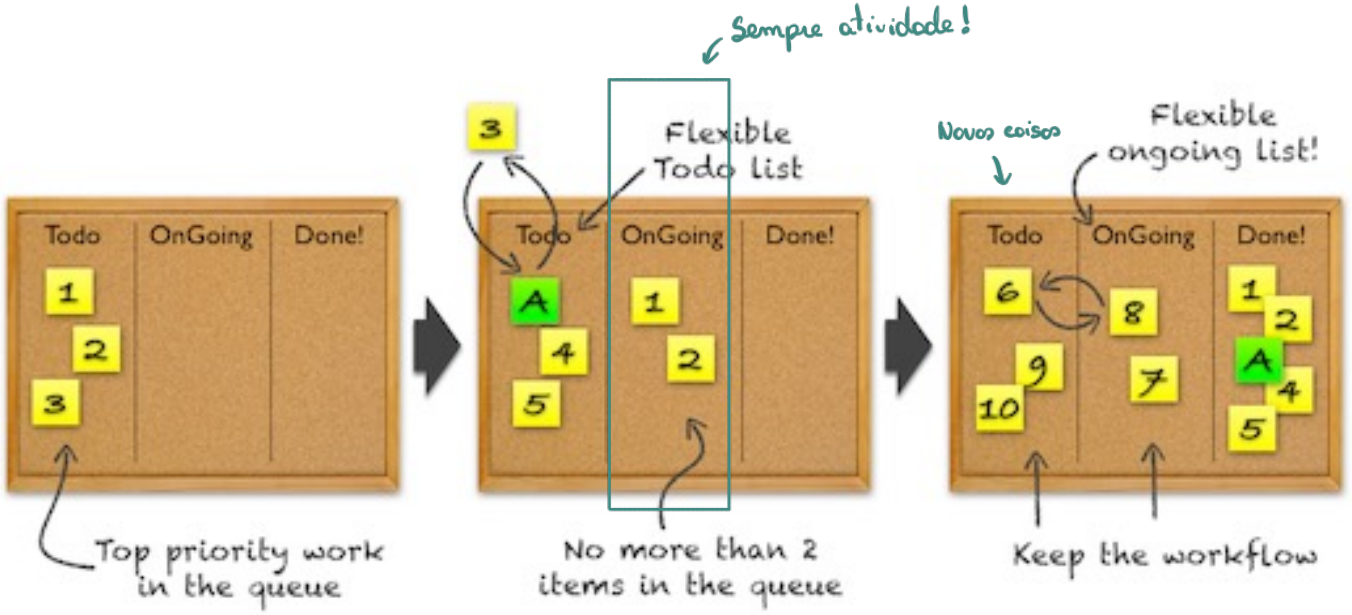
- In **Scrum**, you select the work you'll be doing for the next sprint beforehand.
  - You then lock the sprint, do all the work, and after a couple of weeks - the usual sprint duration - your queue is empty.



https://project-management.com/from-scrum-to-kanban/

SCRUM vs. Kanban

- In **Kanban**, all that's limited is the size of the queues, called the Work In Progress limit.
  - We can change the items in the queues at any time, and that there's no "sprint end". The work just keeps flowing.



https://project-management.com/from-scrum-to-kanban/

SCRUM vs. Kanban

↖ Para Software

	Scrum	Kanban
Cadence	Regular fixed length sprints (i.e., 2 weeks)	Continuous flow
Release methodology	At the end of each sprint	Continuous delivery
Roles	Product owner, scrum master, development team	No required roles
Key metrics	Velocity	Lead time, cycle time, WIP
Change philosophy	Teams should not make changes during the sprint.	Change can happen at any time

https://www.atlassian.com/agile/kanban/kanban-vs-scrum

Topics covered

- Agile principles
- Agile development techniques
- Agile project management
- **Scaling agile methods**

Scaling agile methods

- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.
- Scaling up agile methods involves changing these to cope with larger and longer projects
  - where there are multiple development teams, perhaps working in different locations.
- When scaling agile methods, it is important to maintain agile fundamentals:
  - Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

System issues

- How large is the system being developed?
  - Agile methods are most effective a relatively small co-located team who can communicate informally.
- What type of system is being developed?
  - Systems that require a **lot of analysis before implementation** need a (more) detailed design.
- What is the expected system lifetime?
  - **Long-lifetime systems require documentation** to communicate the intentions of the system developers to the support team.
- Is the system subject to external regulation?
  - If a system is regulated, you will probably be required to **produce detailed documentation** as part of the system safety case.

Documentação, os testes, user stories são MUITO importantes ...

RGPD

Large system development

- Large systems are usually collections of separate, communicating systems, where **separate teams** develop each system.
  - Frequently, these teams are working in different places, sometimes in different time zones.

- Large systems are '**brownfield systems**', that is they include and interact with several existing systems.
  - Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development.

Large system development

- Large systems and their development processes are often constrained by external **rules and regulations** limiting the way that they can be developed

- Large systems have a **long procurement** and development time
  - It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects

- Large systems usually have a diverse set of **stakeholders**
  - It is practically impossible to involve all these different stakeholders in the development process

Agile: Scaling up to large systems

Não podemos estar a adicionar features ...

- A completely incremental approach to requirements engineering is **impossible**

- There **cannot** be a single product owner

- **Not possible** to focus only on the code

Documentado em Issues com muita complexidade

- Cross-team communication mechanisms have to be **designed** and **used**

- Continuous integration is practically **impossible**
  - However, it is essential to maintain frequent system builds and regular releases of the system

Multi-team Scrum

- Role replication
  - Each team has a Product Owner for their work component and ScrumMaster.
- Product architects
  - Each team chooses a product architect, and these architects collaborate to design and evolve the overall system architecture.
- Release alignment
  - The dates of product releases from each team are aligned so that a demonstrable and complete system is produced.
- Scrum of Scrums
  - There is a daily Scrum of Scrums where representatives from each team meet to discuss progress and plan work to be done.

Summary

- Agile methods are incremental development methods

- Agile development practices include
  - User stories for system specification
  - Frequent releases of the software,
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team.

Summary

- Scrum is an agile method that provides a project management framework.
  - It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Many practical development methods are a mixture of plan-based and agile development.
- Scaling agile methods for large systems is difficult.
  - Large systems need up-front design, and some documentation and organizational practice, may conflict with the informality of agile approaches.

Resources & Credits

- Roger S. Pressman, Bruce Maxim, Software Engineering: A Practitioner's Approach, 7th Edition, McGraw-Hill Education, 2015
  chapter 5

- Ian Sommerville, Software Engineering, 10th Edition, Pearson, 2016
  chapter 3