Software Processes

Leonardo Mariani

University of Milano Bicocca

mariani@disco.unimib.it

What is Software Engineering?

A naive view:

$$\text{Problem Specification} \xrightarrow{\textit{coding}} \text{Final Program}$$

But ...

- Where did the *problem specification* come from?
- How do you know the problem specification corresponds to and satisfies the *user's needs*?
- How did you decide how to *structure* your program?
- How do you know the program actually *meets the specification*?
- How do you know your program will always *work correctly*?
- What do you do if the users' *needs change*?
- How do you *divide tasks up* if you have more than a one person in the developing team?
- How do you reuse existing software for solving similar problems?

What is Software Engineering?

"multi-person construction of multi-version software"

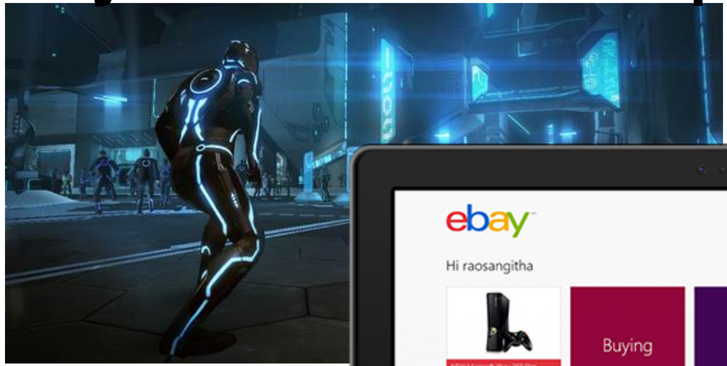
— Parnas

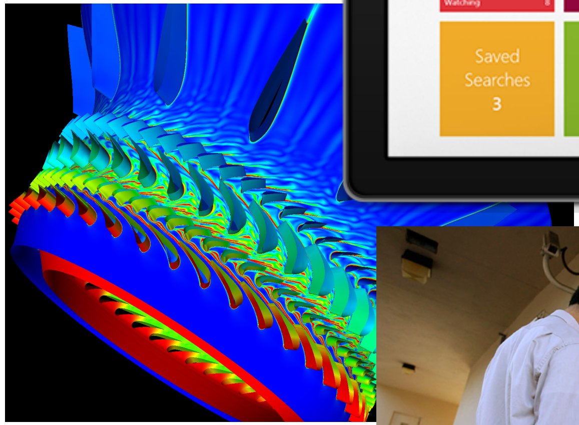
- Team-work
  - Scale issue + communication issues
- Successful software systems must ***evolve*** or ***perish***
  - *<u>Change is the norm</u>*, not the exception

Advanced School on PARALLEL COMPUTING

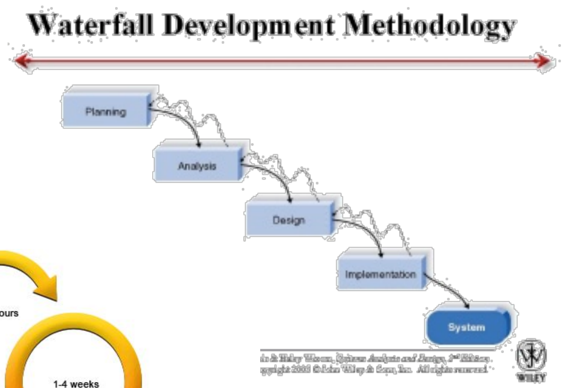
Software: the product of a process

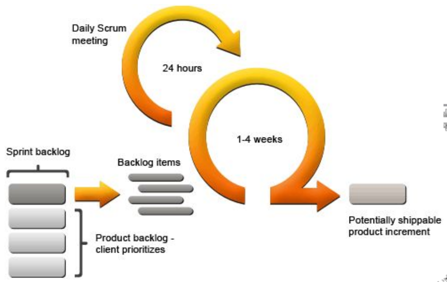
Many kinds of software products

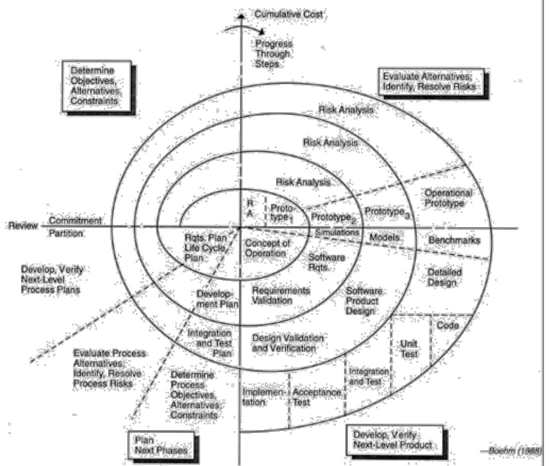




Many process models




Daily Scrum meeting
24 hours
1-4 weeks
Sprint backlog
Backlog items
Product backlog - client prioritizes
Potentially shippable product increment

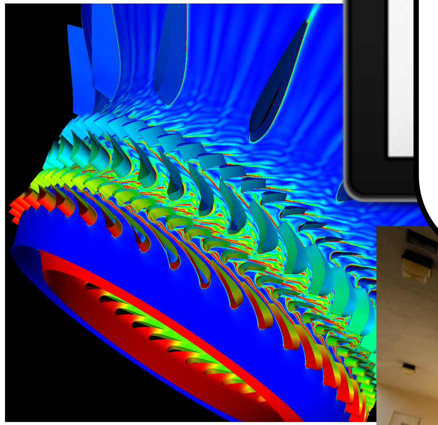


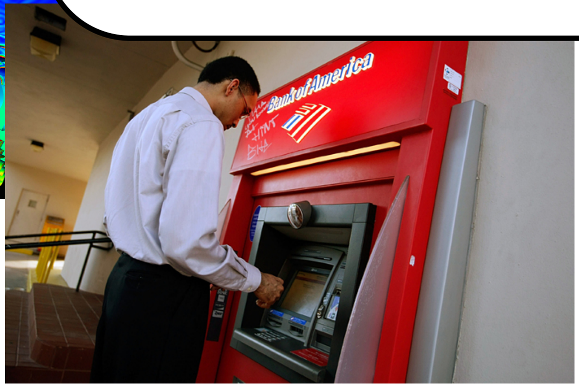
CINECA

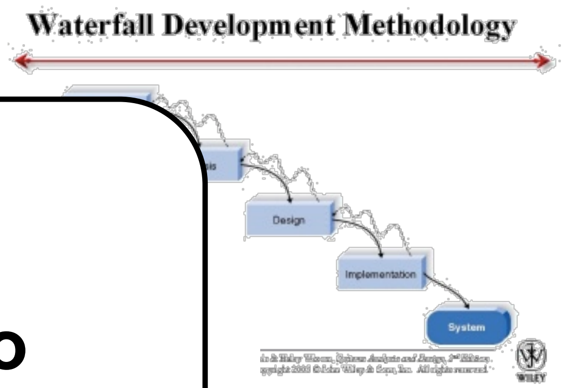
Software: the product of a process

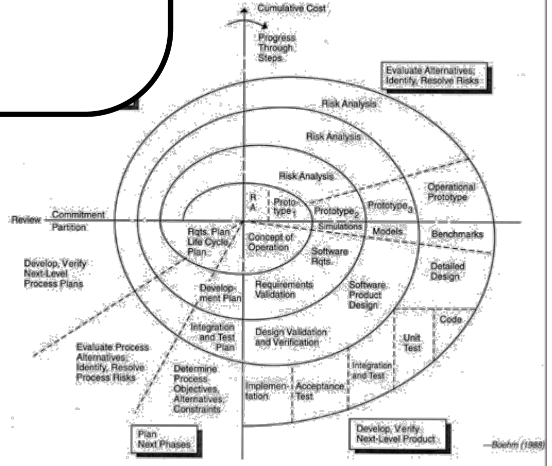
Many kinds of software products

Many process models







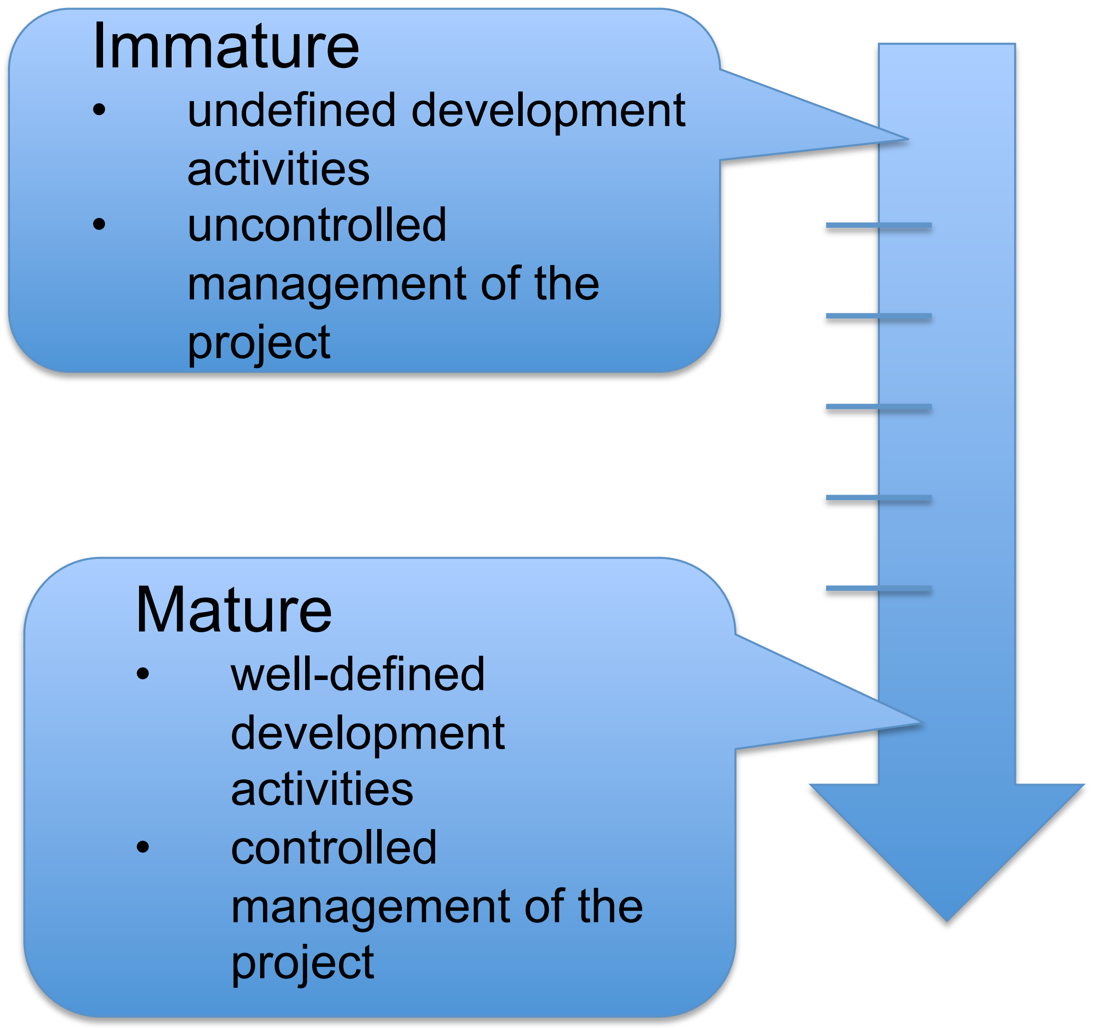


Study the process to improve the product

Better process → *Better product*

Process Maturity

Risk of Failure


Immature
• undefined development activities
• uncontrolled management of the project
Mature
• well-defined development activities
• controlled management of the project

The software development process

- **Software process**: set of **roles**, **activities**, and **artifacts** necessary to create a software product
- Possible roles: stakeholder, designer, developer, tester, maintainer, ecc.
- Possible artifacts: source code, executables, specifications, comments, test suite, etc.

Activities

Requirements Collection	Establish customer’s needs
Analysis	Model and specify the requirements (“what”)
Design	Model and specify a solution (“how”)
Implementation	Construct a solution in software
Testing	Validate the software against its requirements
Deployment	Making a software available for use
Maintenance	Repair defects and adapt the sw to new requirements

NB: these are ongoing activities, not sequential phases!

Agile Methods

Suitable for projects with <u>unknown</u>, <u>difficult to be discovered</u> or <u>continuously changing</u> **requirements**

Good compromise between <u>practicality</u> and <u>cost</u>

Agile development methods support development of <u>*complex systems*</u> with <u>*simple methodologies*</u>

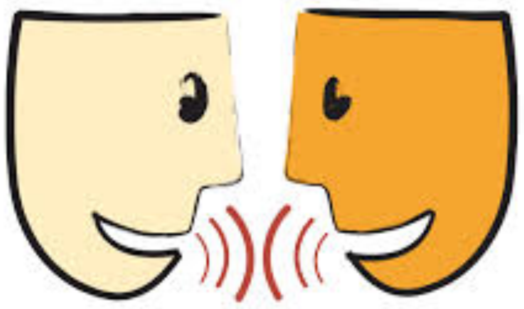
Agile Software Development Manifesto

- Published in 2001 by a consortium composed of consultants and practioners
  - Our highest priority is to satisfy the customer through ***early*** and ***continuous delivery*** of valuable software.
  - Welcome ***changing requirements***, even late in development. Agile processes harness change for the customer's competitive advantage.
  - ***Deliver*** working software ***frequently***, from a ***couple of weeks*** to a couple of months, with a preference to the shorter timescale.
  - Business people and developers must ***work together*** daily throughout the project.
  - Build projects around ***motivated individuals***. Give them the environment and support they need, and ***trust them*** to get the job done.
  - The most efficient and effective method of conveying information to and within a development team is ***face-to-face conversation***.
  - ***Working software*** is the primary measure of progress.
  - Agile processes promote ***sustainable development***. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
  - Continuous attention to ***technical excellence*** and ***good design*** enhances agility.
  - ***Simplicity***--the art of maximizing the amount of work not done--is essential.
  - The best architectures, requirements, and designs emerge from ***self-organizing teams***.
  - At regular intervals, the ***team*** reflects on how to become more effective, then ***tunes*** and adjusts its behavior accordingly.

Shared Aspects

people matter

less documents is possible

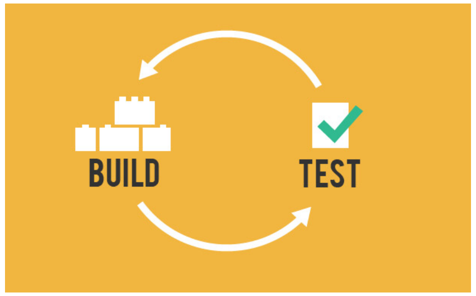


communication is a critical issue

NOPERFECT DESIGN

a **complete** and **detailed design** (before development) is **not necessary**

iterative, incremental and **continuous improvement** of **design quality**



continuous testing, for earlier defect detection

Overview of Agile Methods

- **Scrum** **(Schwaber and Beedle 2002)**
- **Extreme Programming** **(Beck, 1999)**
- Crystal Family **(Cockburn 2002)**
- Feature Driven Development **(Palmer and Felsing 2002)**
- Rational Unified Process **(Kruchten 1996)**
- Adaptative Software Development **(Highsmith 2000)**

Defining Your Own Agile Method

Scrum

Scrum

- Scrum includes ***few simple rules***

  "a simple process for management of complex processes"

- to correctly apply the rules is the difficult part
  - goal: to transform the lack of rules into ***agility***

- **iterative** process based on the empirical **control** of the current status of the project

- 3 fundamental principles

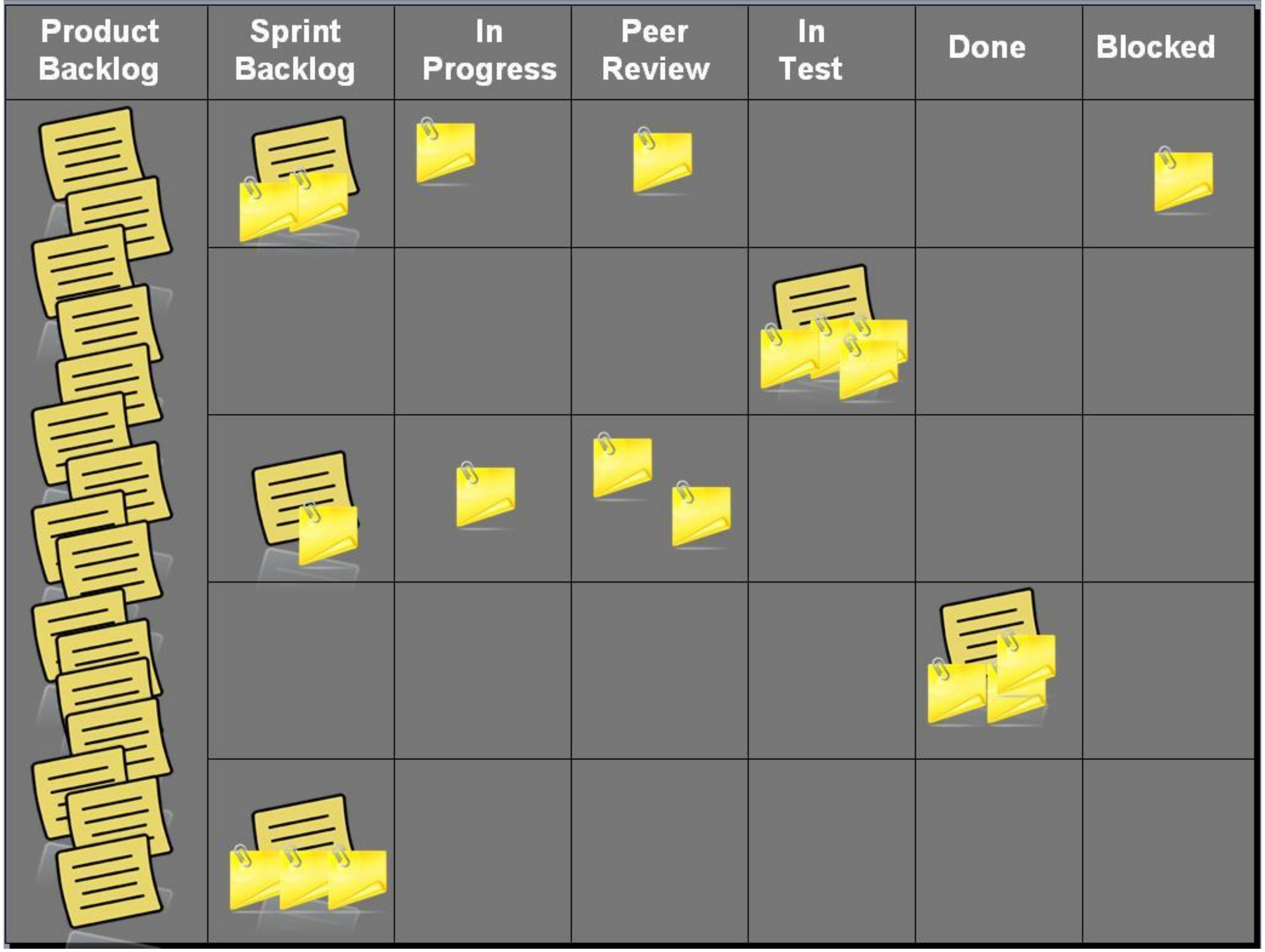
  visibility -> ***inspection*** -> ***adaptation***

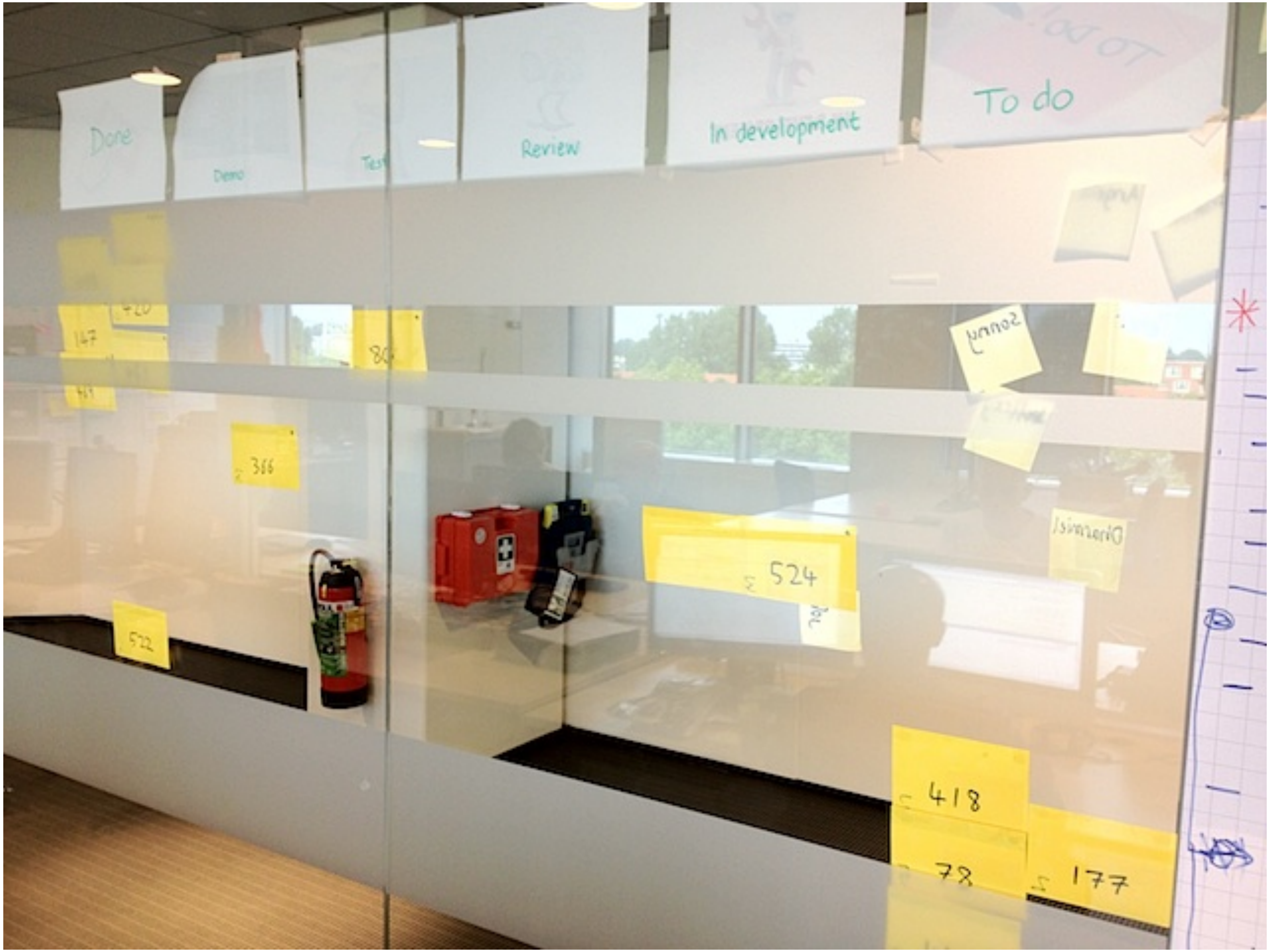
visibility -> inspection -> adaptation

- those **aspects of the process that affect the outcome must be visible** to those controlling the process
  - it must be clear which functionalities are completed, modified, faulty, …

- information must be clear
  - when a functionality can be labeled as “done”?

Example of a Scrum Task Board




Done
Demo
Test
Review
In development
To do
147
366
524
522
418
78
177

visibility -> **inspection** -> adaptation

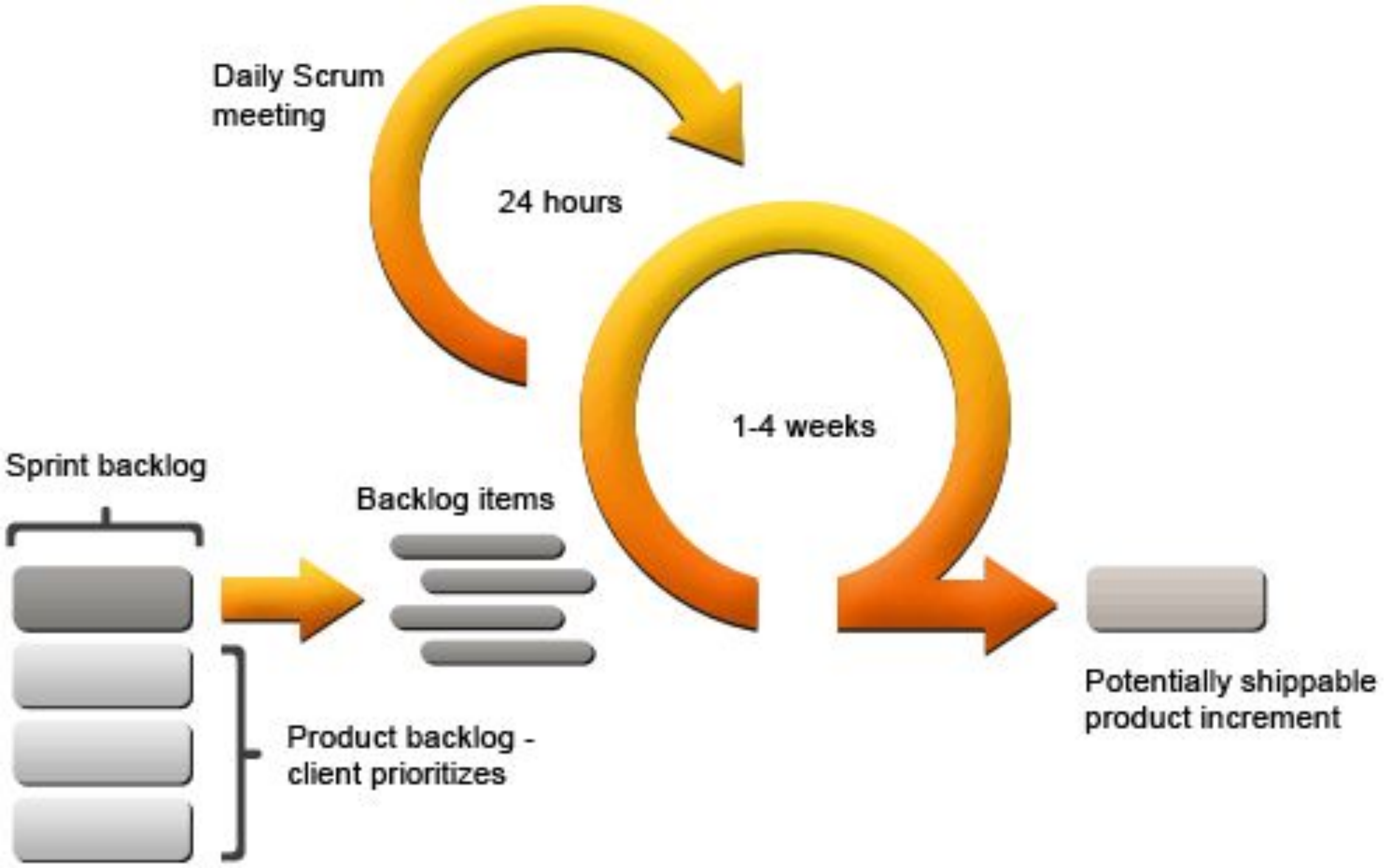
- all aspects of a development process **must be frequently inspected** to suddenly identify unacceptable variances

- frequency of inspection depends on the process
  - when artifacts are **available**?
  - when the feedback provided by inspection can be turned into **action items**?

visibility -> inspection -> **adaptation**

- if inspectors determine that one or more aspects are outside acceptable limits, both the **process** and **software** must be **adjusted**
- adaptation must be as quick as possible

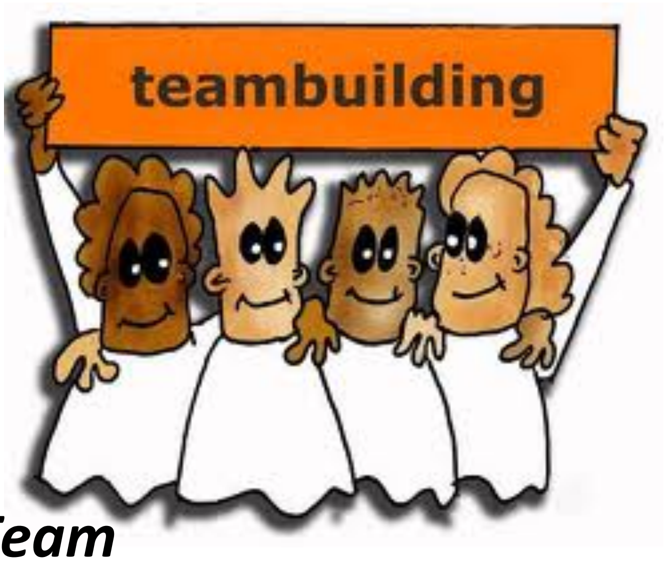
The Scrum Process


Daily Scrum meeting
24 hours
1-4 weeks
Sprint backlog
Backlog items
Product backlog - client prioritizes
Potentially shippable product increment

Roles

Product Owner

- represents (all) customers
- constantly collaborating with the team
- during a Spring, never interferes with the team

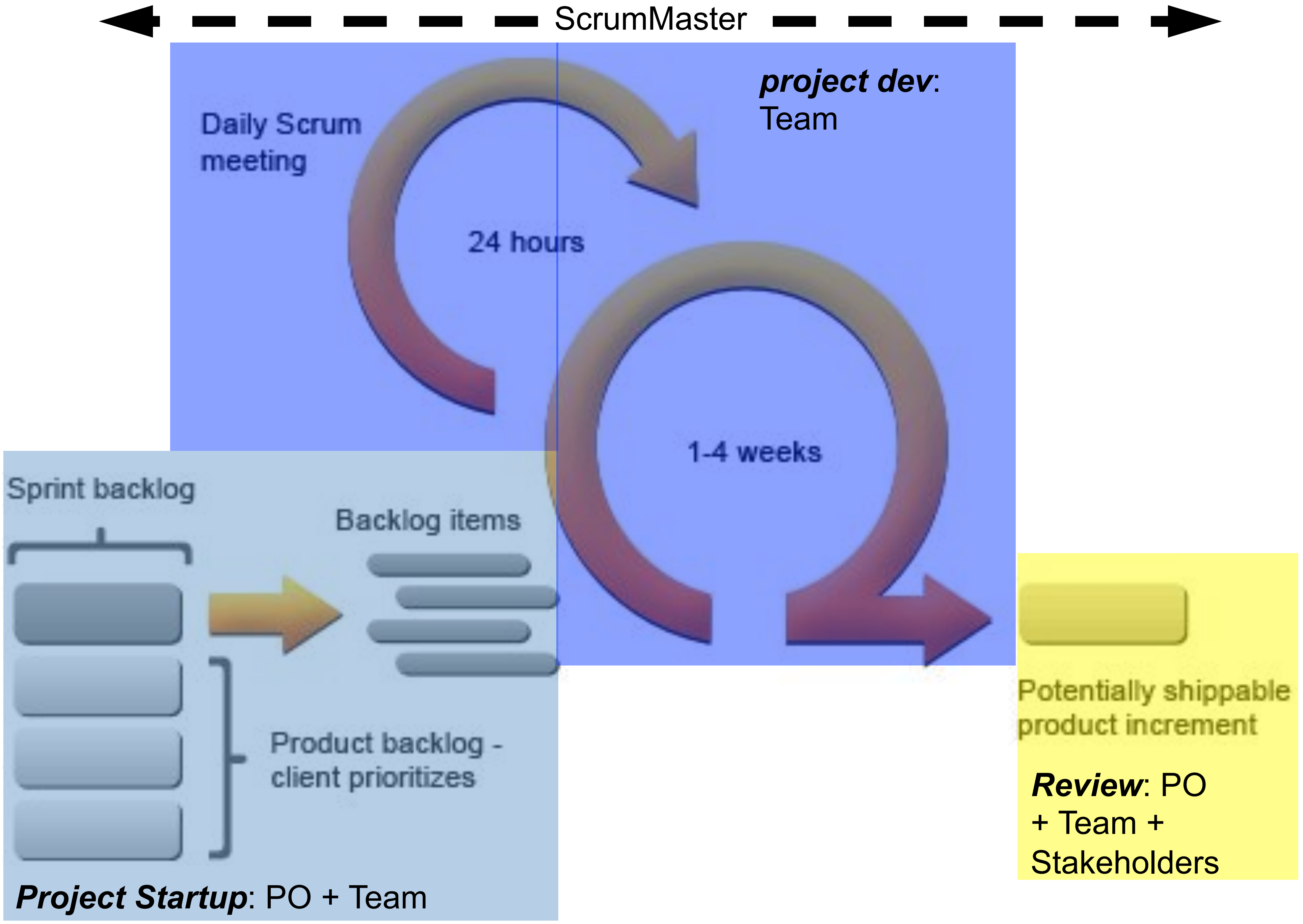


Team

- about 7 developers
- self-organizing
- cross-functional expertise

ScrumMaster

- is responsible for the development process,
- teaches Scrum,
- adapts Scrum to company's needs,
- overviews the behavior of all participants to the Scrum process


ScrumMaster
Daily Scrum meeting
24 hours
project dev: Team
1-4 weeks
Sprint backlog
Backlog items
Product backlog - client prioritizes
Project Startup: PO + Team
Potentially shippable product increment
Review: PO + Team + Stakeholders

Project Startup

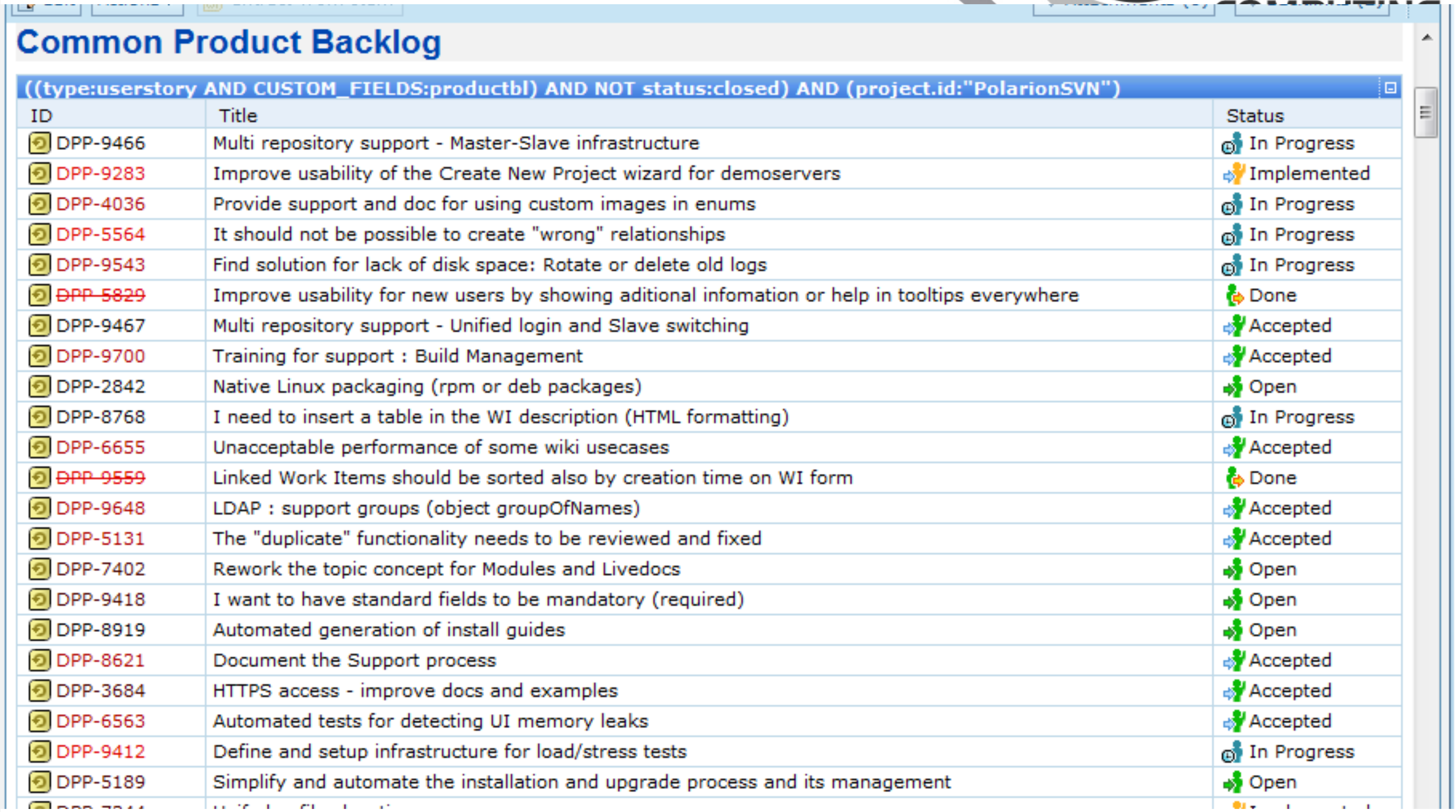
product backlog

sprint planning meeting

sprint backlog list

Product Backlog

- list of functional and non-functional requirements
  - ***prioritized*** and distributed across releases
  - ***continuously available*** to all project members
  - the product owner is responsible for the product backlog
  - ***NEVER COMPLETE!***
  - the life of the product backlog coincides with the life of the application



Common Product Backlog

((type:userstory AND CUSTOM_FIELDS:productbl) AND NOT status:closed) AND (project.id:"PolarionSVN")

ID	Title	Status
DPP-9466	Multi repository support - Master-Slave infrastructure	In Progress
DPP-9283	Improve usability of the Create New Project wizard for demoservers	Implemented
DPP-4036	Provide support and doc for using custom images in enums	In Progress
DPP-5564	It should not be possible to create "wrong" relationships	In Progress
DPP-9543	Find solution for lack of disk space: Rotate or delete old logs	In Progress
~~DPP-5829~~	Improve usability for new users by showing aditional infomation or help in tooltips everywhere	Done
DPP-9467	Multi repository support - Unified login and Slave switching	Accepted
DPP-9700	Training for support : Build Management	Accepted
DPP-2842	Native Linux packaging (rpm or deb packages)	Open
DPP-8768	I need to insert a table in the WI description (HTML formatting)	In Progress
DPP-6655	Unacceptable performance of some wiki usecases	Accepted
~~DPP-9559~~	Linked Work Items should be sorted also by creation time on WI form	Done
DPP-9648	LDAP : support groups (object groupOfNames)	Accepted
DPP-5131	The "duplicate" functionality needs to be reviewed and fixed	Accepted
DPP-7402	Rework the topic concept for Modules and Livedocs	Open
DPP-9418	I want to have standard fields to be mandatory (required)	Open
DPP-8919	Automated generation of install guides	Open
DPP-8621	Document the Support process	Accepted
DPP-3684	HTTPS access - improve docs and examples	Accepted
DPP-6563	Automated tests for detecting UI memory leaks	Accepted
DPP-9412	Define and setup infrastructure for load/stress tests	In Progress
DPP-5189	Simplify and automate the installation and upgrade process and its management	Open

Sprint Planning Meeting – part 1

- 4-hours segment
- Attendees: ScrumMaster, Product Owner, Team
  - business experts can be invited, they can only provide information and advices
- the **PO prepares** the Product Backlog before the meeting
  - in this task, the product owner can be substituted by the ScrumMaster
- The PO **presents** the highest priority product backlog entries
- The team **asks questions** about content, purpose, meaning, …
- Before the 4 hours elapse, the team **selects** the product functionality that can be committed by the end of next Sprint

Sprint Planning Meeting – part 2

- 4-hours segment (it starts immediately after the 1° segment)
- Attendees: ScrumMaster, Product Owner, Team
  - technology domain experts can be invited, they can only provide information and advices
  - team (self-)organizes the Sprint, Product Owner must be available to answer questions
  - decision are **ONLY UP TO THE TEAM**
- Hurry up! The Sprint just started!
- The team plans the Sprint
- Tasks are placed in the Sprint Backlog
- Tasks can evolve during the Sprint

Sprint Backlog List

- It defines the **tasks** that must be completed for turning the selected portion of the Product Backlog into an increment of potentially shippable product functionality
- A task should usually take from 4 to 16 hours to be completed
- Tasks longer than 16 hours usually represent tasks that have not been deeply analyzed yet
- **ONLY THE TEAM CAN CHANGE THE SPRINT BACKLOG**
- The sprint backlog should be **visible** to all participants to a Scrum process

Advanced School on PARALLEL COMPUTING

Project Development

Sprint Daily Meeting

CINECA

Sprint Daily Meeting

- face-to-face meeting
- Attendees: ScrumMaster, the Team
  - It is open to everyone, but only ScrumMaster and Team play an active role
- time-boxed to **15 minutes**
- always in the **same place** at the **same time** every **work day**
- it must be the **first thing** Team members do arriving at work
- all members MUST **be present**
  - absent must either attend by telephone, or
  - having another team member reporting on his/her status
- the meeting starts at the appointed time, regardless of who is present
  - any member who is late must pay, e.g., $1

Sprint Daily Meeting - Interactions

- ScrumMaster begins the meeting by starting to his/her left and proceeding counterclockwise
- Every team member answers to **3 questions**:
  - What have I done since the last Daily Scrum?
  - What am I going to do between now and the next Daily Scrum?
  - What is preventing me from doing my work as effectively as possible?
- No digress, only **shortly answer** to questions
- Only 1 person talks at time, other people listen, no interruption, **no discussion**
- After a team member reported, other team members can ask for **arranging a meeting** after the Daily Meeting
- Other attendees are not allowed to interact in ANY way and must be bound to the side of the room
- Other Attendees are **NOT ALLOWED to interact** with the team after the daily meeting **at ANY TIME**

Common Problems and Reactions

- *Implicit impediment*: point at the impediment
- *Side discussion*: ask people to listen when they are not speaking
- *Rambling on*: ask people to summarize more quickly
- *Sidetracked meeting*: ask people to have a meeting immediately afterwards for people who care about the topic
- *Observer who speaks*: remind them they are an observer
- *Late arrival*: charge them €1

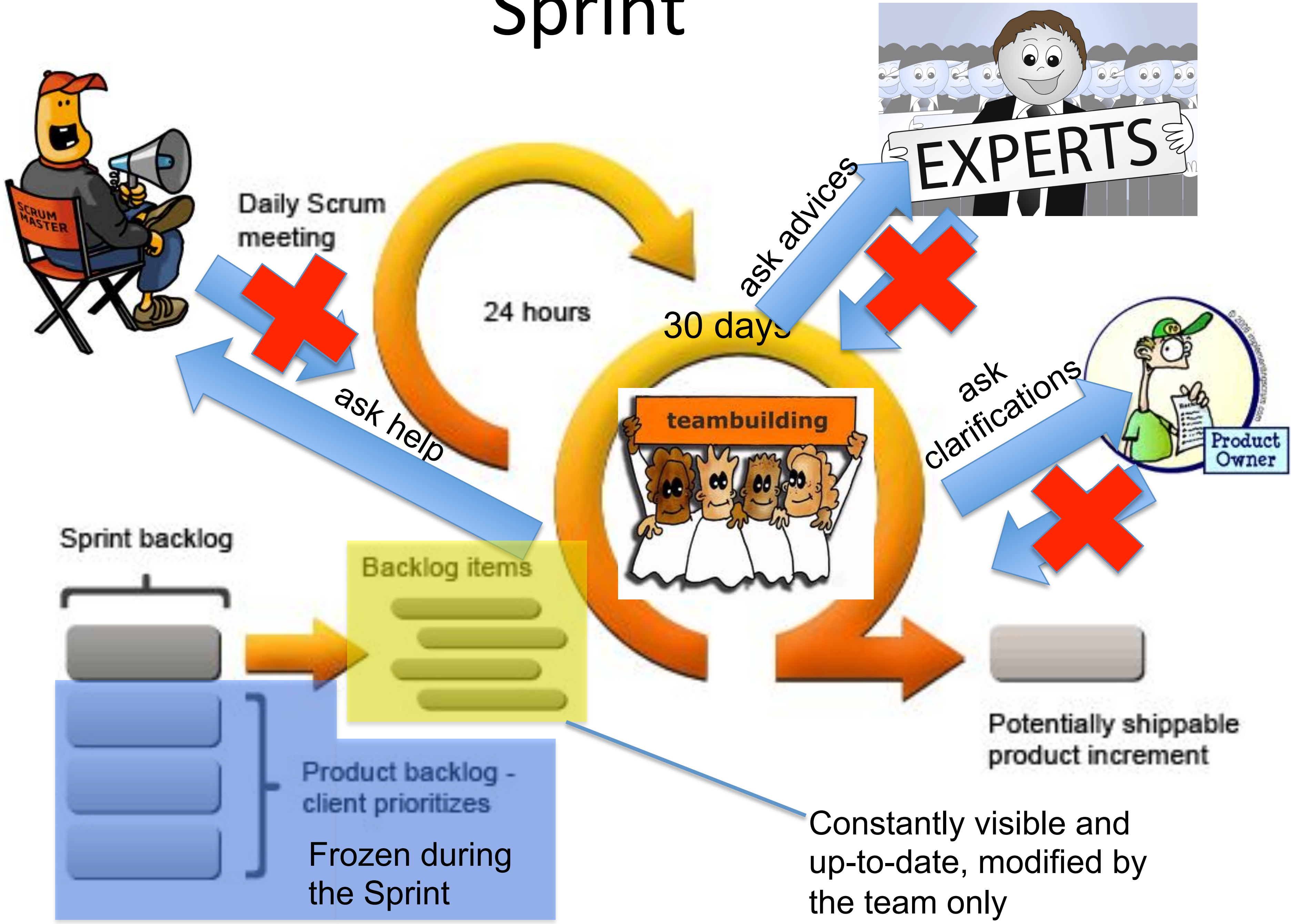
Exercise

- Simulation of a Scrum meeting (Scrum from Hell)
  - Imagine to be a member of a team developing an HPC system for weather forecasting; take a moment to decide what you've been working on and how you'll answer the three questions.
  - secret goal for some of you
  - if ScrumMaster addresses your behavior, do not persist in it

Debriefing

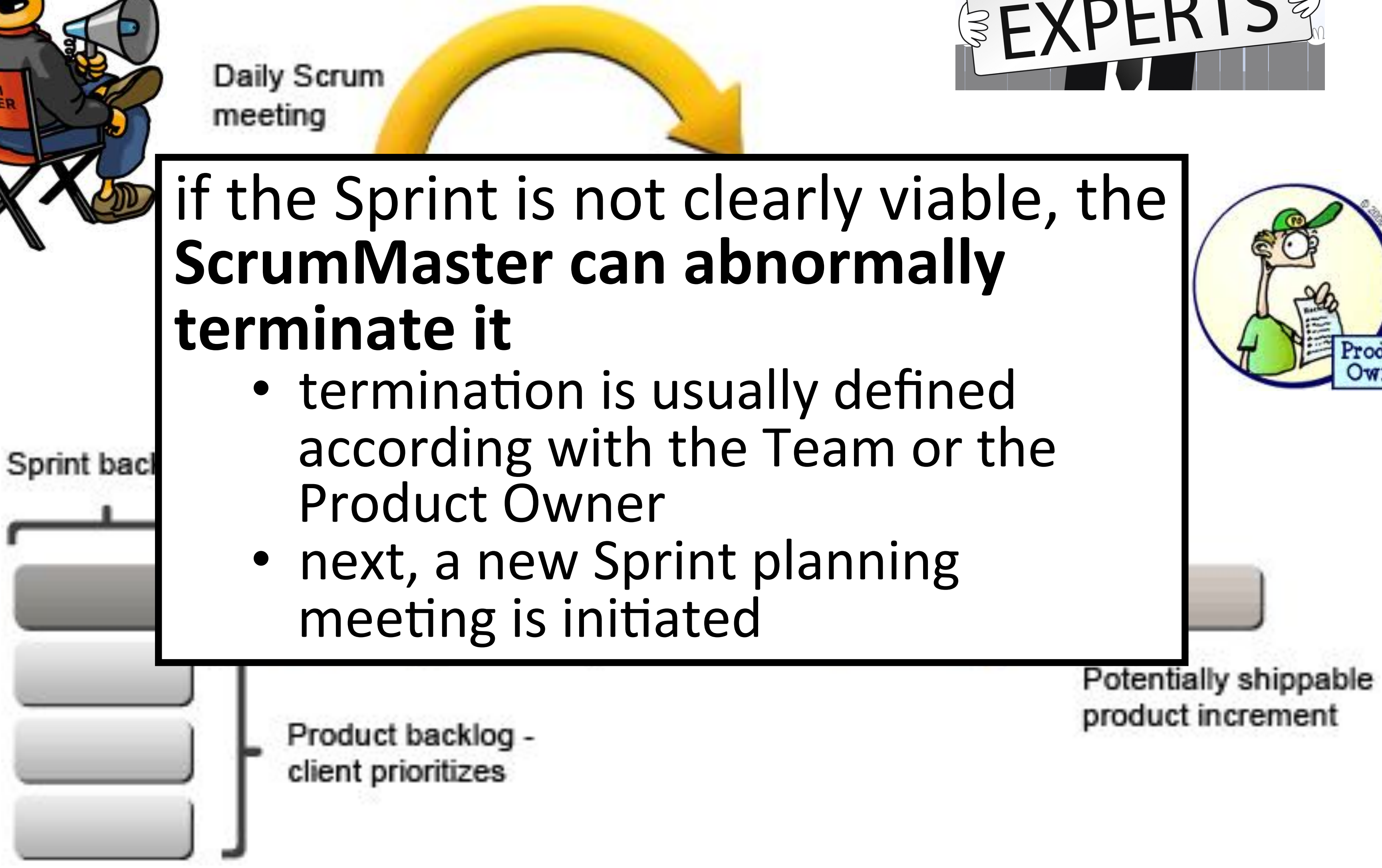
- What behaviors did you see?
- How was the meeting?

Sprint



Constantly visible and up-to-date, modified by the team only

Sprint


Daily Scrum meeting
Sprint bac
Product backlog - client prioritizes
Potentially shippable product increment

if the Sprint is not clearly viable, the **ScrumMaster can abnormally terminate it**

- termination is usually defined according with the Team or the Product Owner
- next, a new Sprint planning meeting is initiated

Review

Sprint Review Meeting

Sprint Review Meeting - setup

- time-boxed: 4 hours
- attendees: ScrumMaster, ProductOwner, Team, Stakeholders
- the team should **not spend more than 1 hour** to prepare this meeting
- goals:
  - to present product owner and stakeholders **functionality** that is **DONE**
  - functionality that is **not done CANNOT be presented**
  - anything that is **not a functionality CANNOT be presented**, unless to support the understanding of a functionality
  - functionality is demonstrated from team workstations (usually the quality assurance server)
- **ScrumMaster organizes the meeting**
  - logistics, inviting people, selecting people that will participate

Sprint Review Meeting - Interactions

- it starts with a **team member presenting**:
  - Sprint goal
  - Product backlog committed
  - Product backlog completed
- team members can discuss what went **well** and what well **wrong**
- most of the meeting is about
  - team members **demonstrating functionalities**
  - stakeholders and product owner **asking questions**
  - team members **noting changes** to do
- in particular, stakeholders can
  - make comments, observations, criticisms
  - identify **missing functionality**
  - noting functionality that behave differently from expected, **change functionality**
  - identify **new functionality**
- the meeting ends with a **polling** of all **stakeholders**, they report
  - their general impressions
  - any desired change
  - priority of changes
- product owner, the team and stakeholders discuss possible **re-arrangement of the product backlog**
- the ScrumMaster closes the review meeting by announcing place and date of next review meeting

Scrum Retrospective Meeting

- time-boxed: 3 hours
- PO (optional), Team and the ScrumMaster discuss what has gone **well** and **bad**, and accordingly **modify plans for next Sprint**
- Team members answer to **2 questions**:
  - What went well during last Sprint?
  - What could be improved in the next Sprint?
- ScrumMaster summarizes all answers in a form
- The team prioritizes items in the form
- The team decides which **items are turned into high-priority action items within next Sprint Backlog**, as non-functional requirements

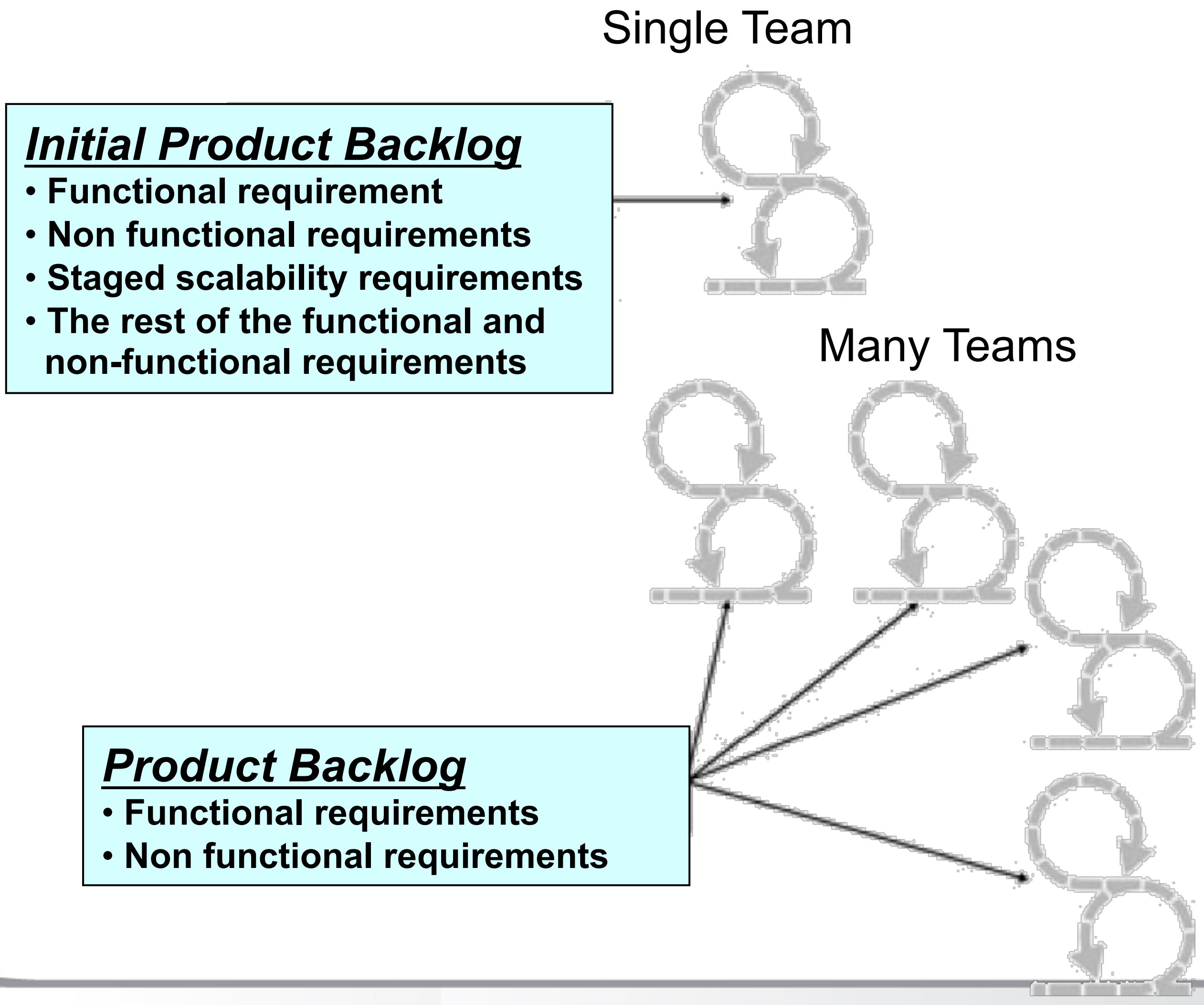
Coordination of Multiple Teams

Scalability Issues

- Many projects require **more than one** team
- How to create, manage and coordinate them?

- by
  1) **incrementally defining** and **activating** teams
  2) starting from the **infrastructure** and the **architecture**

Scalability Sprints



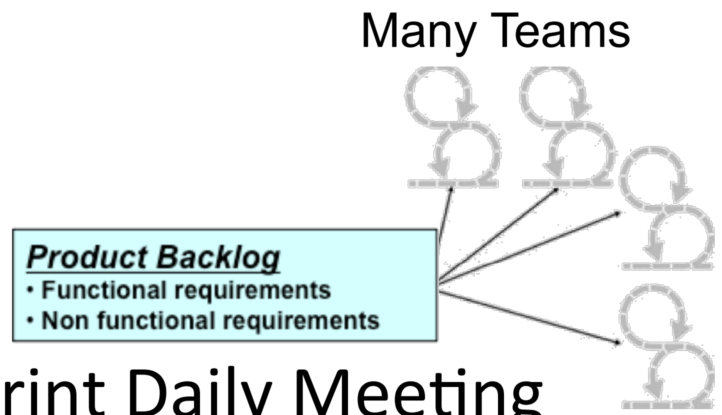
Initial Product Backlog

- **Functional requirement**
- **Non functional requirements**
- **Staged scalability requirements**
- **The rest of the functional and non-functional requirements**

Product Backlog

- **Functional requirements**
- **Non functional requirements**

Scrum of Scrums



- It is equivalent to the Team’s Sprint Daily Meeting
- **1 participant for each team**
- team members report their work, and eventually schedule further meeting for inter-team coordination

- There could be a scrum of scrums also among
  - POs, to coordinate requirements
  - ScrumMasters, to coordinate the process

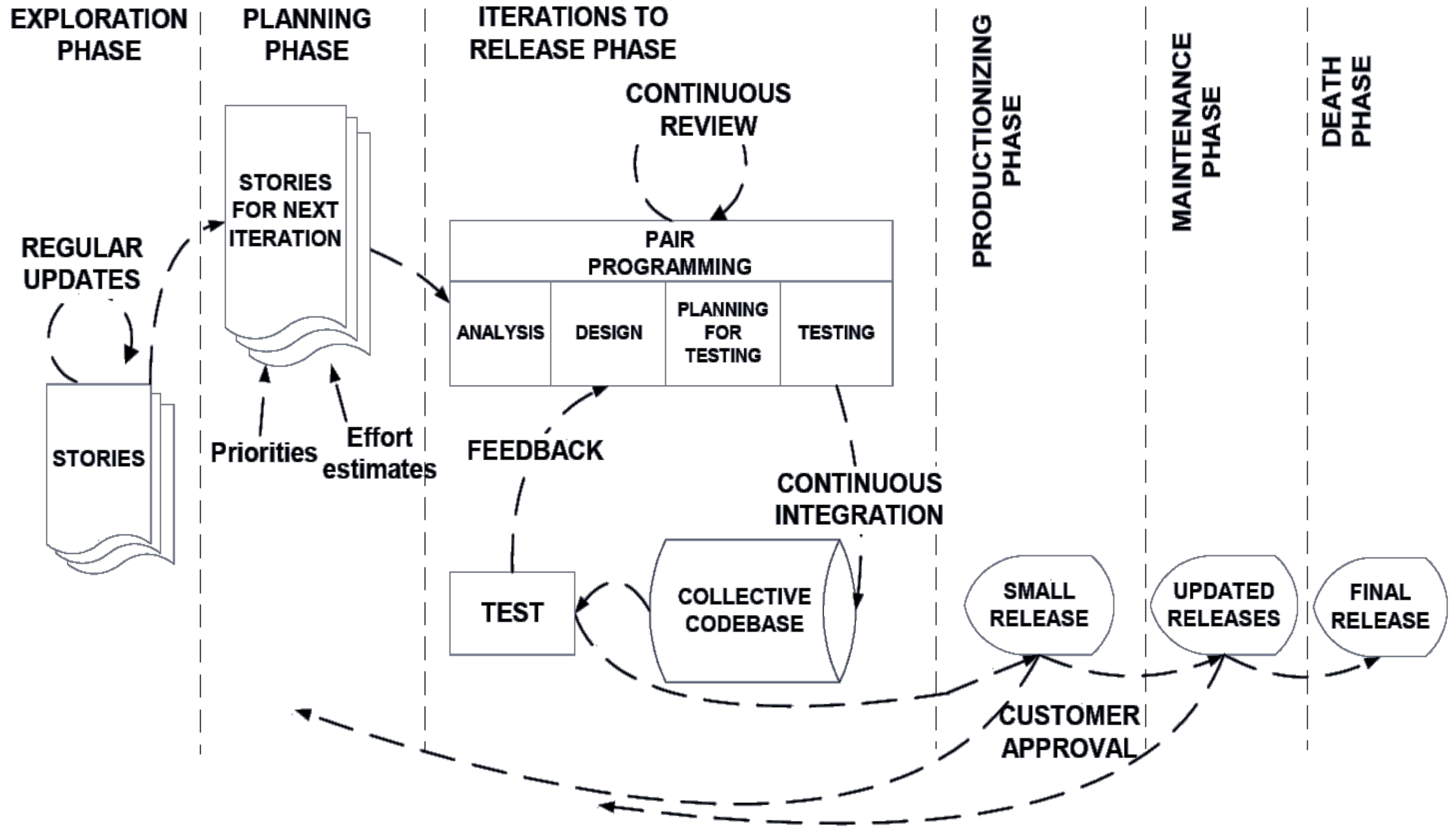
Advanced School on PARALLEL COMPUTING

Extreme Programming

XP Practices Only

CINECA

Extreme Programming



XP practices

- The Planning Game
- Short Releases
- Metaphor
- Simple Design
- Refactoring
- Test-First Design
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-Hour Week
- On-Site Customer
- Coding Standards
- Open workspace

Refactoring

- At any time during development:
  - if necessary, rework the code
  - small incremental changes (5 min)
  - **always execute test cases after changes!!**
- **Before** the addition of a new functionality
  - Is it possible to modify the system to favor the addition of the new functionality?
- **After** the addition of a new functionality
  - Is it possible to simplify the system, without modifying test cases?
- *NOTE: if building and testing are expensive, refactoring is limited*
- Check-in only when
  - all test cases have been passed
  - duplicated code has been removed
  - the code is readable (expressive)
  - the code is as simple as possible

Collective Ownership

- Each **developer can access to the whole code**
  - If a programmer needs a change… he/she accesses to the code and modifies it
  - If the **code** is **reworked** by multiple developers, its quality **improves**
  - If a member of a pair has extended knowledge of a part of the system, the other member can take advantage of this knowledge
  - A version control system is necessary to enforce this practice

Coding Standard

- **Coding conventions shared** between team members
  - they **spontaneously emerge** overtime (as a consequence of collective ownership)
    - Sometime enforced with static analysis tools
  - continuous **evolution**
  - increase **expressiveness** and **readability** of the code
    - automatic generation of code documentation (e.g., DOXYGEN)

Pair programming

- Two programmers work side-by-side at a computer: one types, the other reviews and inspects the newly typed code
- Features:
  - Fine grained code inspection
  - Could facilitate teamwork and concentration
  - Requires a constructive attitude
    - "egoless programming"
    - responsibility is up to programmers

Other Practices

- On-site customer
  - A business representative is part of the team (in most cases is not a real customer)
- 40-hour week
  - Developers must work in their best mental conditions

Continuous Integration

- Frequent Check-in
- Daily builds (end to end)
- Integration testing after each build
  - test over-night
  - often not only testing, but also static and dynamic analysis

Exercise

EXTREME HOUR

The Experience

- Demonstrate how to <u>develop</u> and <u>test</u> a product using principles from agile methods

The Product

- A better mousetrap
  - Our new product must dominate the corporate sector of the well established mousetrap market
  - Plan, schedule, develop and quality assure the initial release!
  - Timeframe: 45’

coding = drawing

45’ project

- 10’ Requirements
- 5’ Priority and initial commitment schedule
- 10’ iteration 1
- 5’ fix commitment schedule
- 10’ iteration 2
- 5’ Release!

Rules

- Not drawn = not delivered
- Not written on a napkin = no story/functional test
- Roles
  - Developers
  - Quality Assurance
  - Stakeholders
- QA can’t see what Devs draw until end of 10’
- Devs don’t know what QA and stakeholders write until end of 10’

Requirements

Stakeholders
- write requirements (quantify relevant qualities)
- Mark requirements as either
  - Must Have
  - Costly to Lose
  - Nice to Have

QA

Developers

Priority and Initial Commitment Schedule

Stakeholders

- Rank relative priorities within each pile
- (pass the pile to developers as done)

QA

Developers

- Assign Minute cost to requirements
  - Note risks
- Schedule requirements for next iteration

Iteration 1

Stakeholders

- Secretly think to new requirements! (the evil bastard)
- Write requirements

QA

- Write functional tests for each requirement (can't see what developers do)
- At the end of iteration "run" tests
- Requirements with bugs are incomplete

Developers

- 10 secs planning
- Draw the solutions
- If a draw is not understandable by another developer, it fails a unit test
- Refactor when possible

Fix Commitment Schedule

Stakeholders

- Prioritize and classify the requirements

QA

Developers

- Turn QA bugs into requirements
- Fit requirements into schedule, possibly replacing the existing ones

Iteration 2

Stakeholders

- Secretly think to new stories! (the evil bastard)…

QA

- Write functional tests for each requirement (can't see what developers do)
- At the end of iteration "runs" tests
- Stories with bugs are incomplete

Developers

- 10 secs planning
- Draw the solutions
- If a draw is not understandable by another developer, it fails a unit test
- Refactor when possible

Release!

Stakeholders

- Is the result marketable?

QA

Developers

Take Home

- Agile methods could be useful to develop software in a disciplined way, without introducing overhead
- Individual practices could be adopted regardless the process you use
- Don’t forget to customize the process to your needs!