

# Collaborative Software Development

---

Michel Beaudouin-Lafon

Based on slides by Cédric Fleury revised by Anastasia Bezerianos

Université Paris-Saclay  
mbl@lisn.fr

# Software development

Several users working on a same project

- Remote or collocated users

- Each one works on their own computer (asynchronous)

  - on different tasks

  - at different times

Collaboration is hard to organize

- Versioning, synchronization between users

- Task distribution, social aspects

# Outline

Collaborative software development

Version control

Software development methods

# Outline

Collaborative software development

Version control

Software development methods

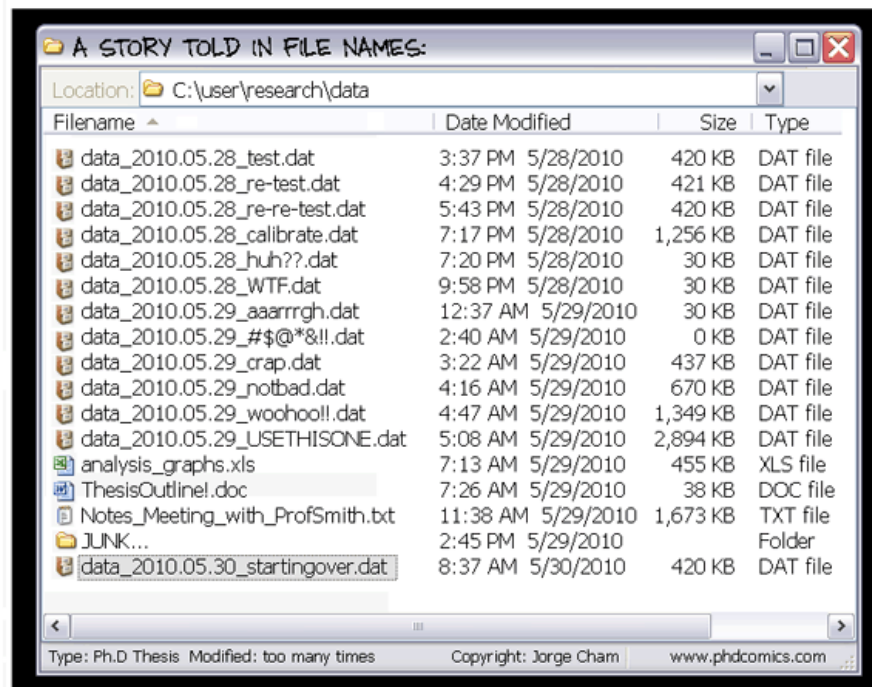
# Why version control?

We want to avoid:

Manually sharing files (USB key, email, Dropbox)

Deleting or overwriting the files of other users

Breaking the project by making a mistake



["Piled Higher and Deeper" by Jorge Cham: [www.phdcomics.com](http://www.phdcomics.com)]

# Version control software

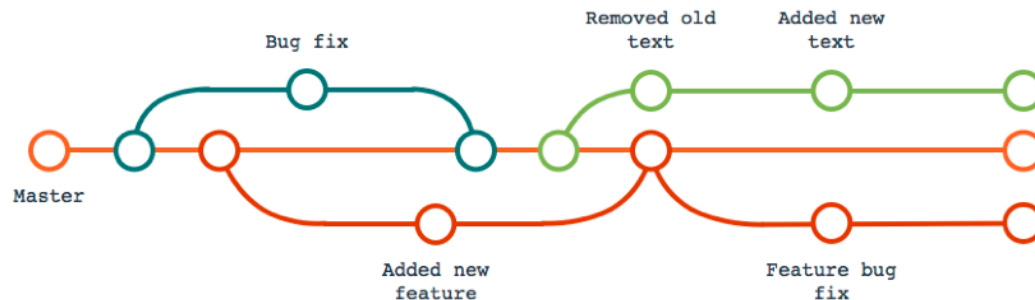


Save & restore different *versions* of the files

Synchronize users' versions

Keep track of modifications and their authors

Manage *branching* and *merging*



Not only for software development

Report, images, data from experiments

# Basic version control actions

**Create a repository**

**Create a local working copy**

**Add** a file to the repository

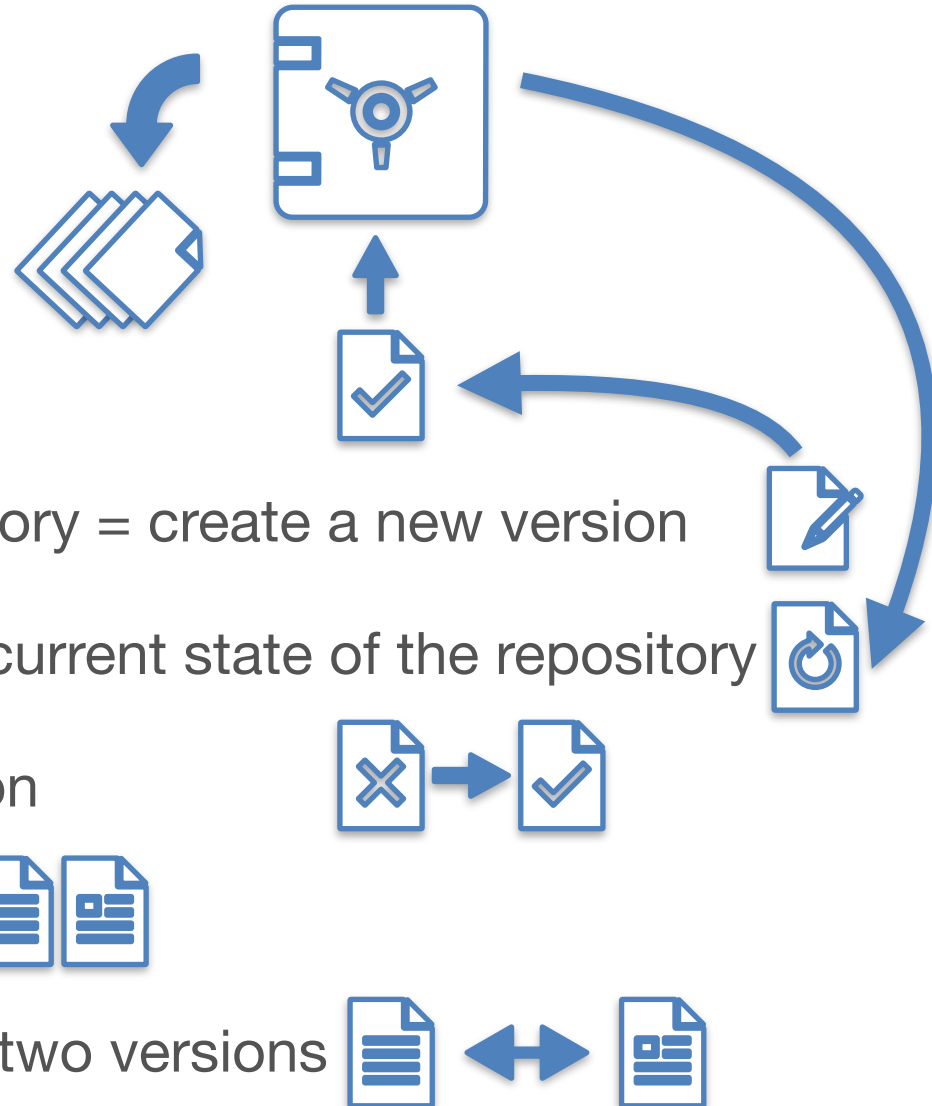
**Commit** a change to the repository = create a new version

**Update** the local copy with the current state of the repository

**Revert** a file to a previous version

**Show the history** of a file

**Show the differences** between two versions



# Version control software architectures

Centralized

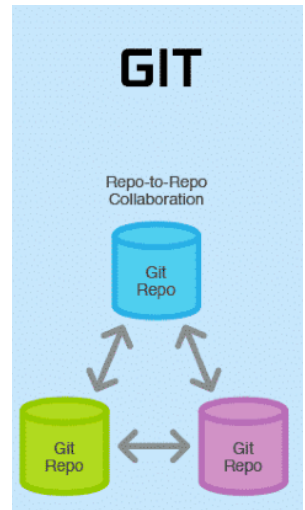
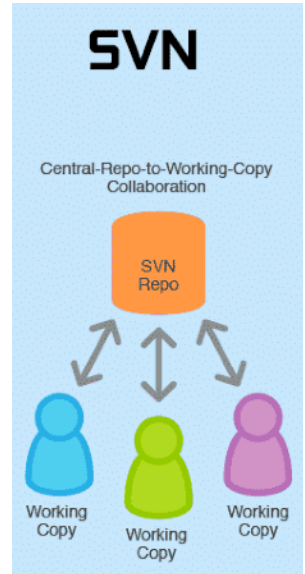
CVS, **SVN**, TFVC, ...

Decentralized (peer-to-peer):

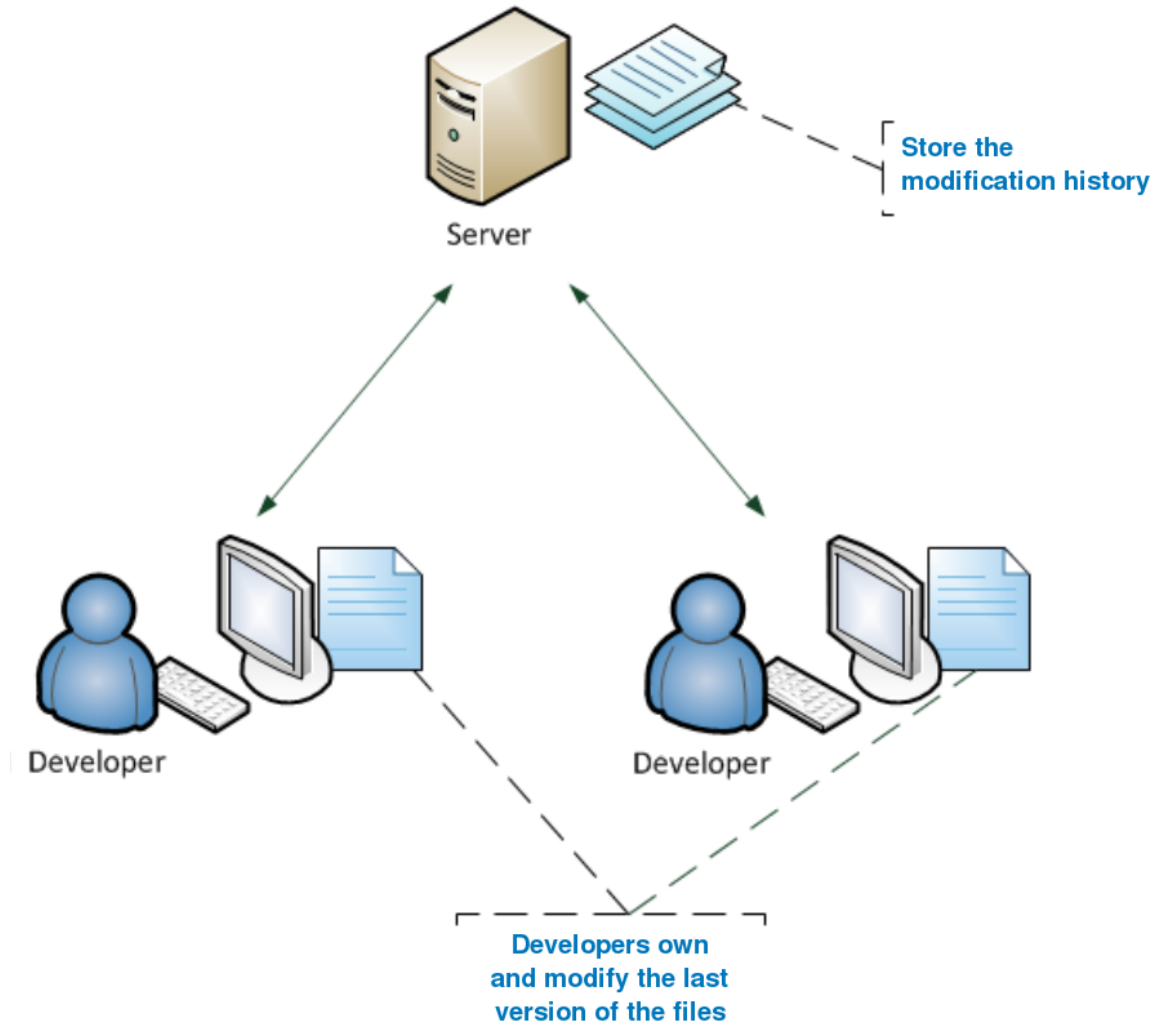
GNU Arch, Mercurial, Bazaar, **Git**,...

Decentralized can be used as a Hybrid Architecture

One peer can be a central server (github)



# Centralized architecture



# Drawbacks of centralized architecture

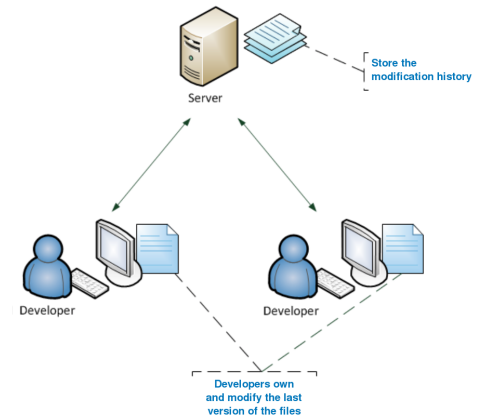
Single access point to the data

Single communication point between users

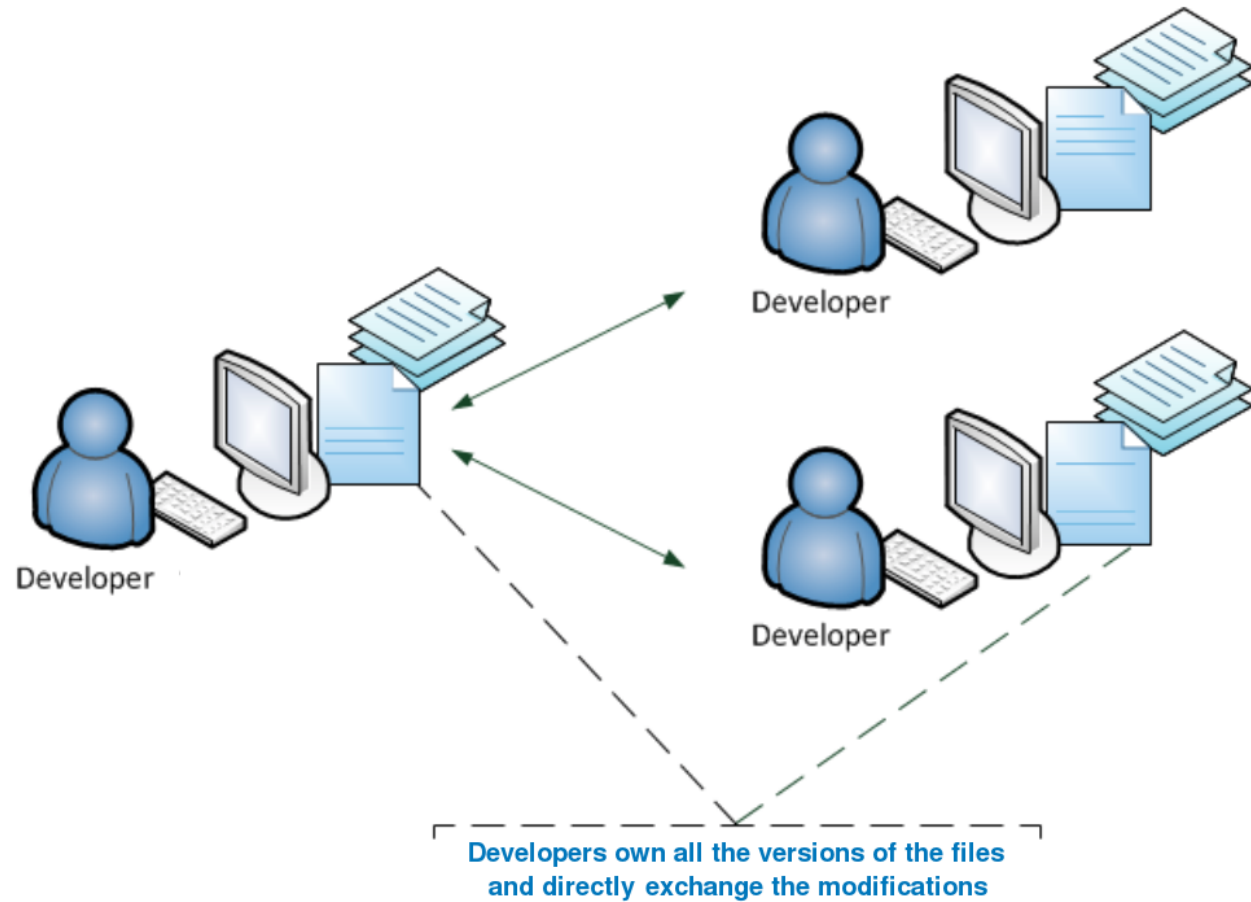
Single history / timeline of the files

Versioning and sharing are the same operation

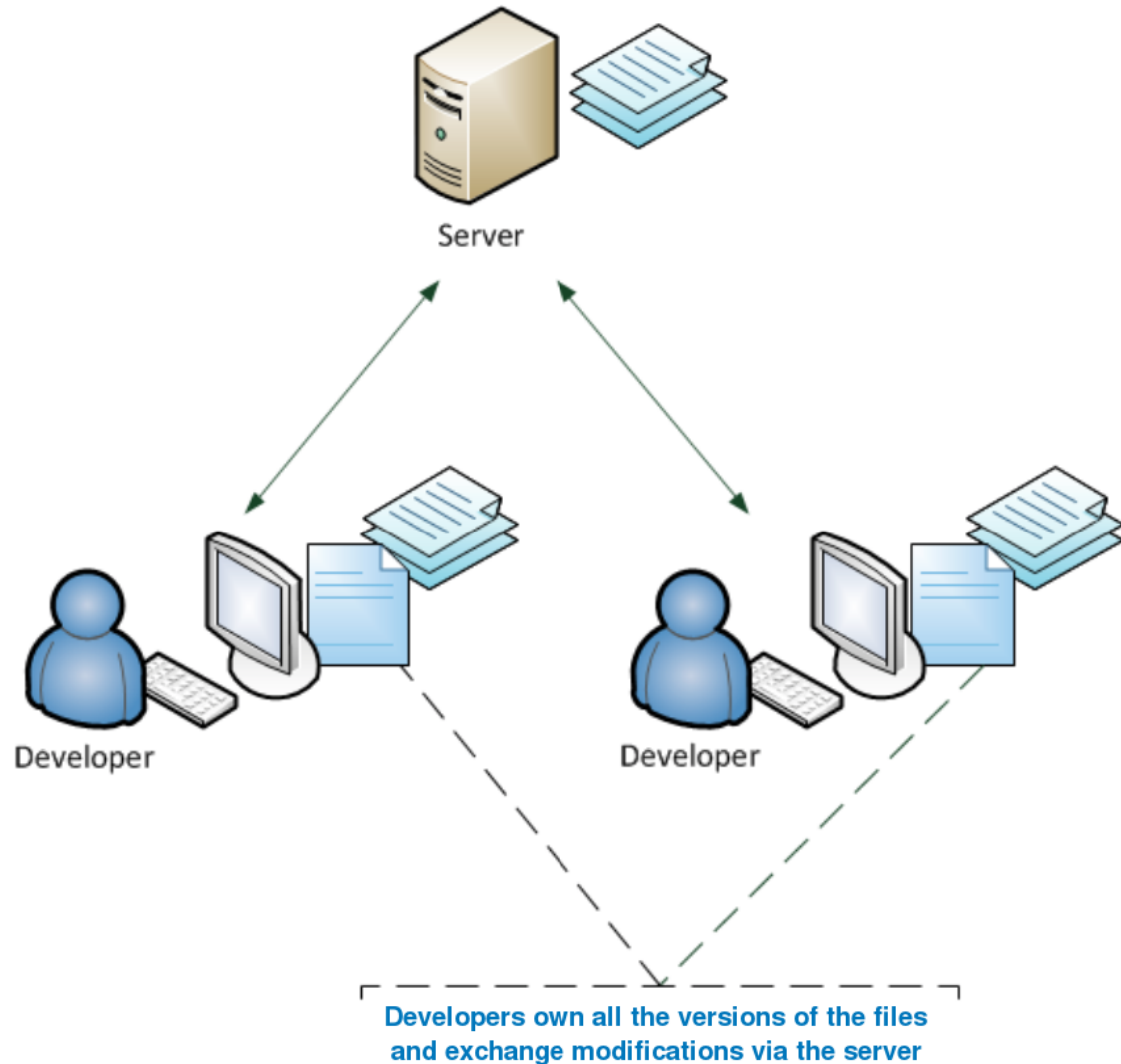
Need to have a stable state before “committing” a change



# Decentralized architecture



# Hybrid architecture



# Vocabulary (SVN)

Architecture

Repository

Working copy

Actions

Checkout

Add

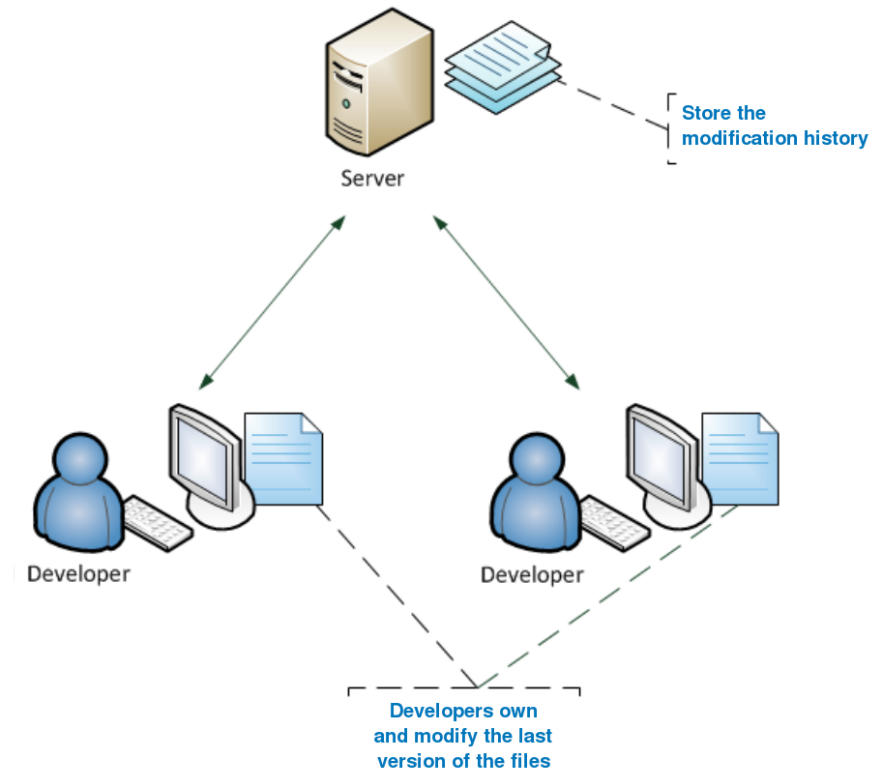
Commit

Update

Revert

Diff, log, status

## Centralized Architecture



# Vocabulary (git)

## Architecture

Remote and local repository

Working copy

## Actions

Clone

Add

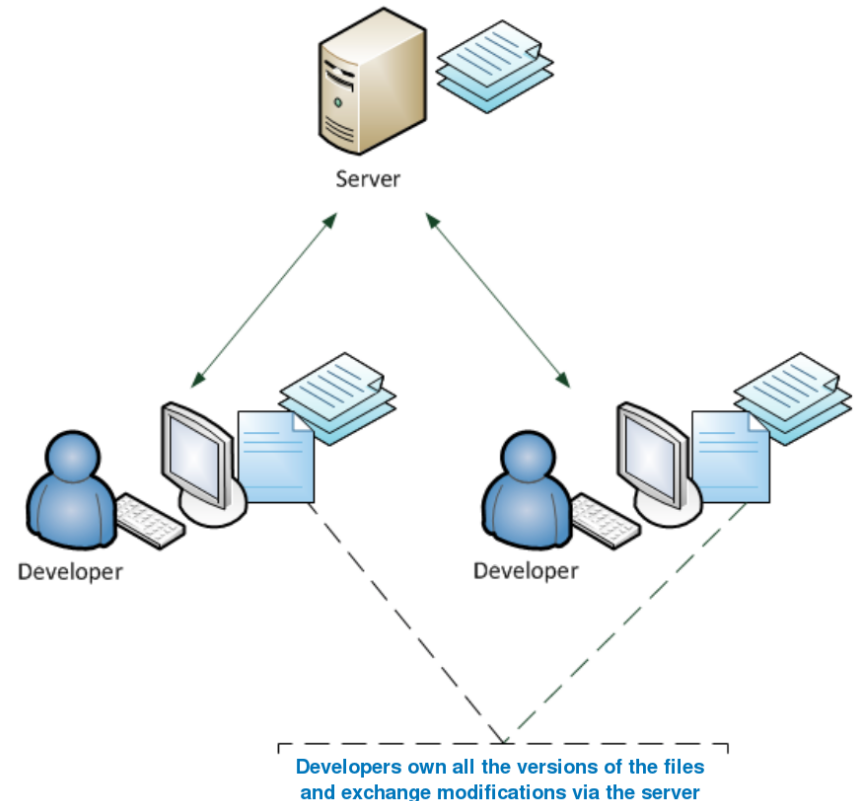
Commit

Push / Pull

Reset

Diff, log, status

## Hybrid Architecture



# Good practices

Work on the local copy

Before sending a modification

- Check if the code compiles locally

- Check for updates from the other users

  - Manage conflicts if there are any

- Check if the code compiles with the updates

Commit the change to the repository

- Triggers a continuous integration workflow

# Conflict management

Multiple users can make changes to the same (text) file

If the changes do not overlap

=> the system merges the changes

If the changes do overlap

=> a conflict appears, to be resolved “by hand”

By telling the system which version is correct

By merging the modifications of the users

# Conflict management

```
C:\workspace\test>svn update
Conflict discovered in 'test.txt'.
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: p
C      test.txt
Updated to revision 3.
Summary of conflicts:
  Text conflicts: 1
```

```
08/10/2010  11:44 AM          94 test.txt
08/10/2010  11:44 AM          26 test.txt.mine
08/10/2010  11:44 AM          27 test.txt.r2
08/10/2010  11:44 AM          31 test.txt.r3
```

**test.txt**

```
<<<<<< .mine
test User2 making conflict
=====
User1 am making a conflict test
>>>>>> .r3
```

# Three-way merge

The screenshot displays the SmartSynchronize 3.4.12 (3-Way Merge) interface. It shows three side-by-side code editors representing different versions of a Java file: `C:\temp\a\SdPrivateKeyCredential-master.java`, `C:\temp\a\SdPrivateKeyCredential-common.java`, and `C:\temp\a\SdPrivateKeyCredential-branch.java`. The interface includes a menu bar (File, Edit, View, Go To, Window, Help) and a toolbar with actions like Save All, Reload, Prev. Change, Next Change, Take Left, Take Right, Left + Merge, All, Merge + Right, and Merge Below. The code is color-coded to show changes: red for additions in the branch, green for additions in the common version, and blue for additions in the master version. Colored arrows (red, green, blue) connect the lines of code between the three editors, illustrating the flow of changes and the merge process. For example, the `getFile()` method in the branch is synchronized with the common version, while the `setPassphrase()` method in the branch is synchronized with the master version. The status bar at the bottom left shows 'Ready'.

# Collaboration workflows

## Centralized vs Decentralized

Integration mode

“Dictator” mode

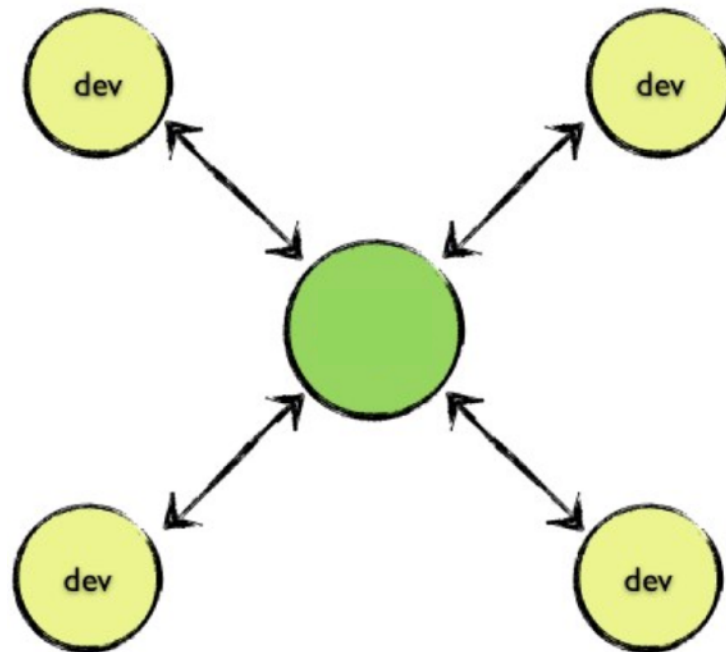
Multi-location teams

## Branch management

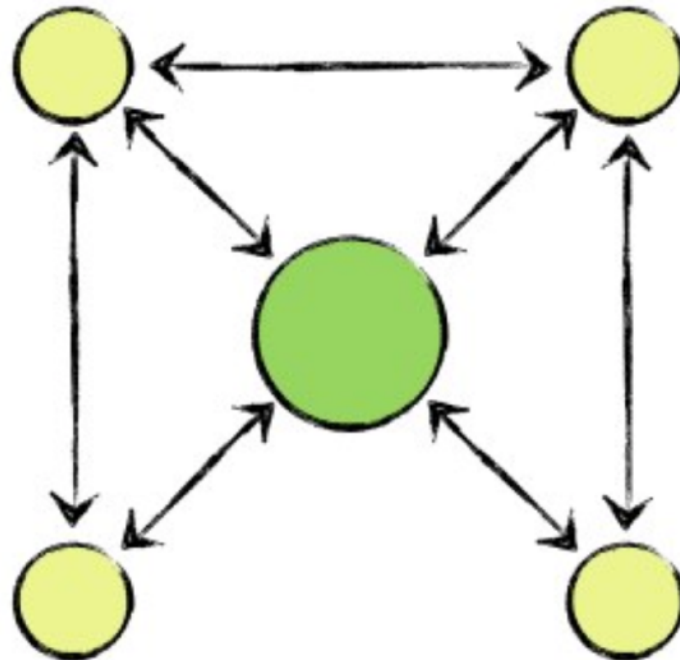
Merging and conflicts

## Continuous integration

# Collaboration scenario: centralized (SVN)



# Collaboration scenario: decentralized (Git)

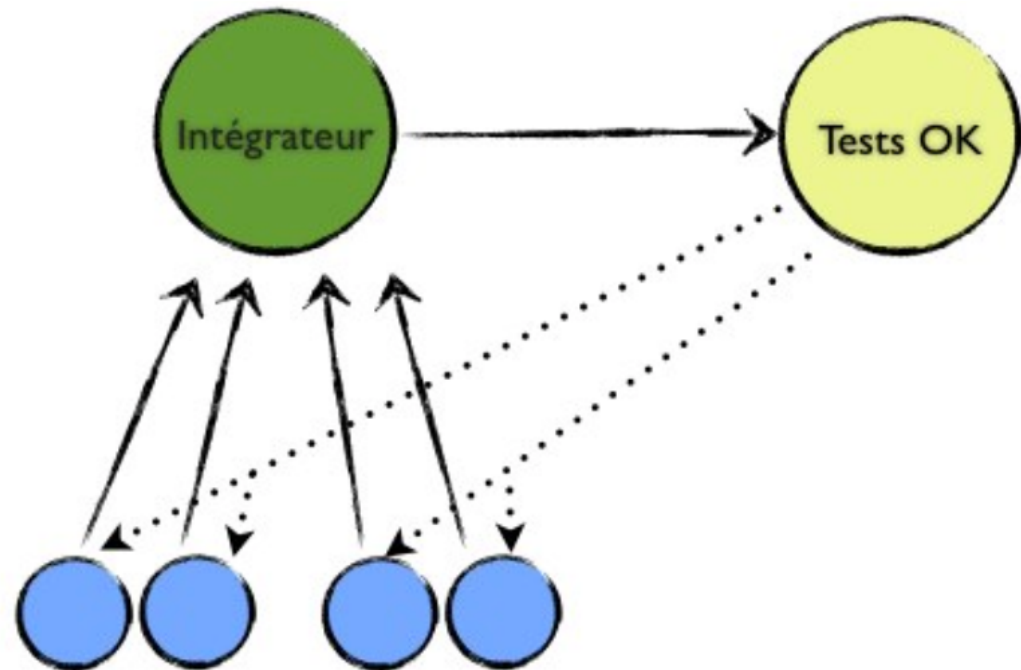


Inter-personal communications

# Collaboration scenario: decentralized (Git)

Integration mode

A repository is in charge of the test

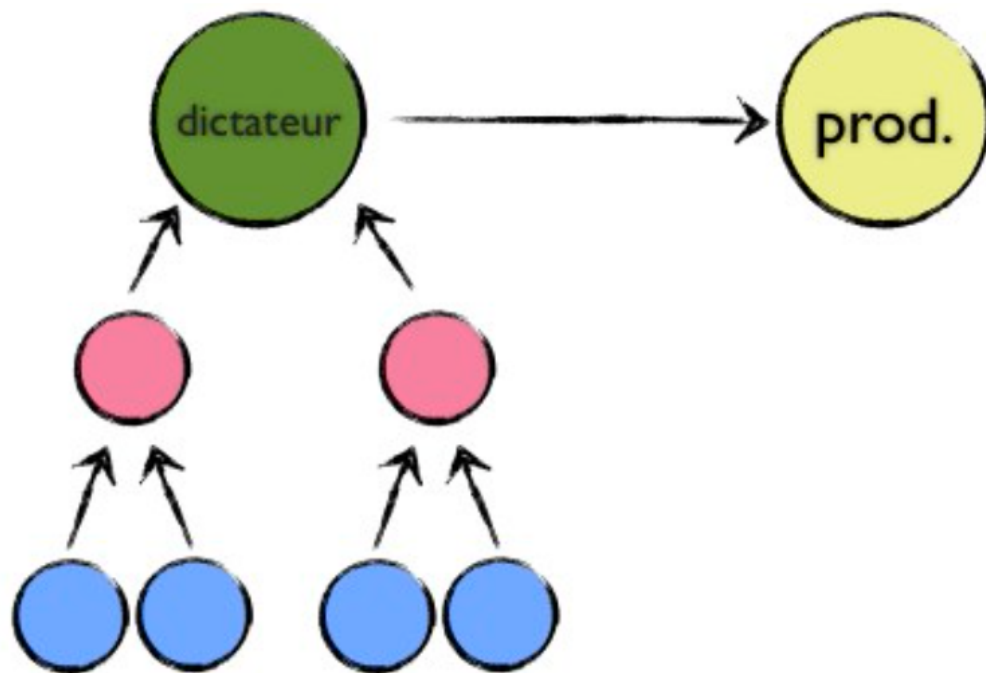


# Collaboration scenario: decentralized (Git)

## Dictator mode

Open-source projects

"Lieutenants" make a first check  
before sending to the "dictator"

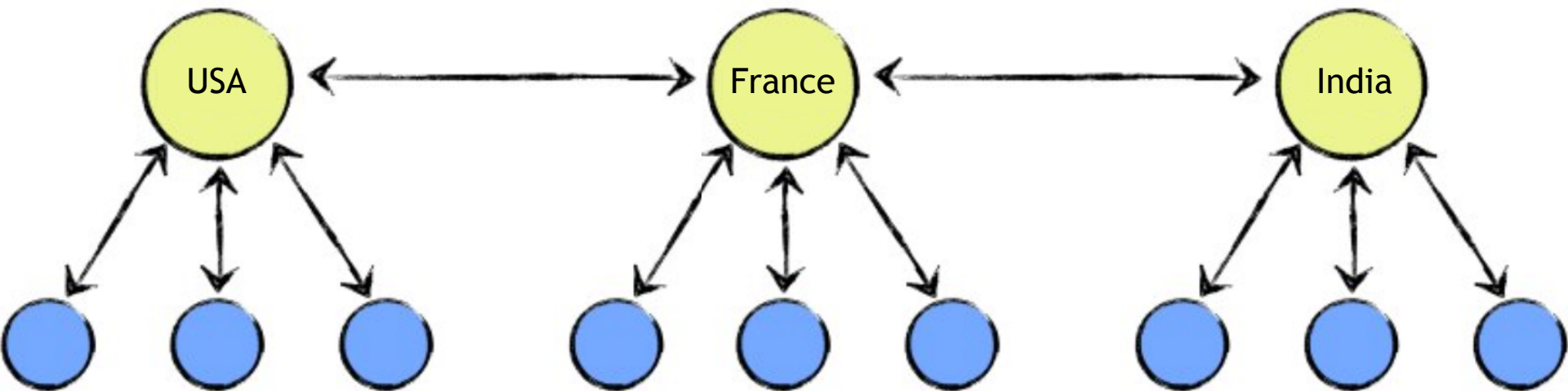


# Collaboration scenario: decentralized (Git)

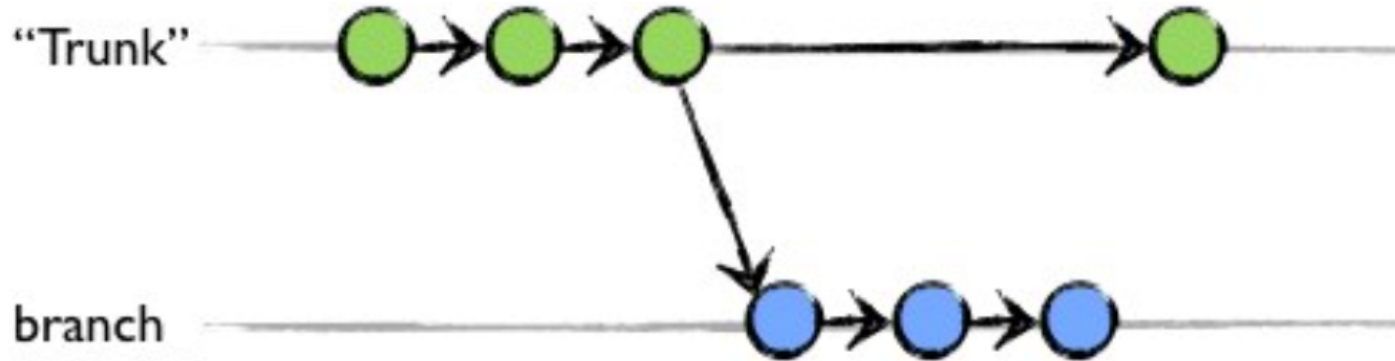
Multi-location teams

Each team can work independently

Regular integration of the work of each team



# Branch management

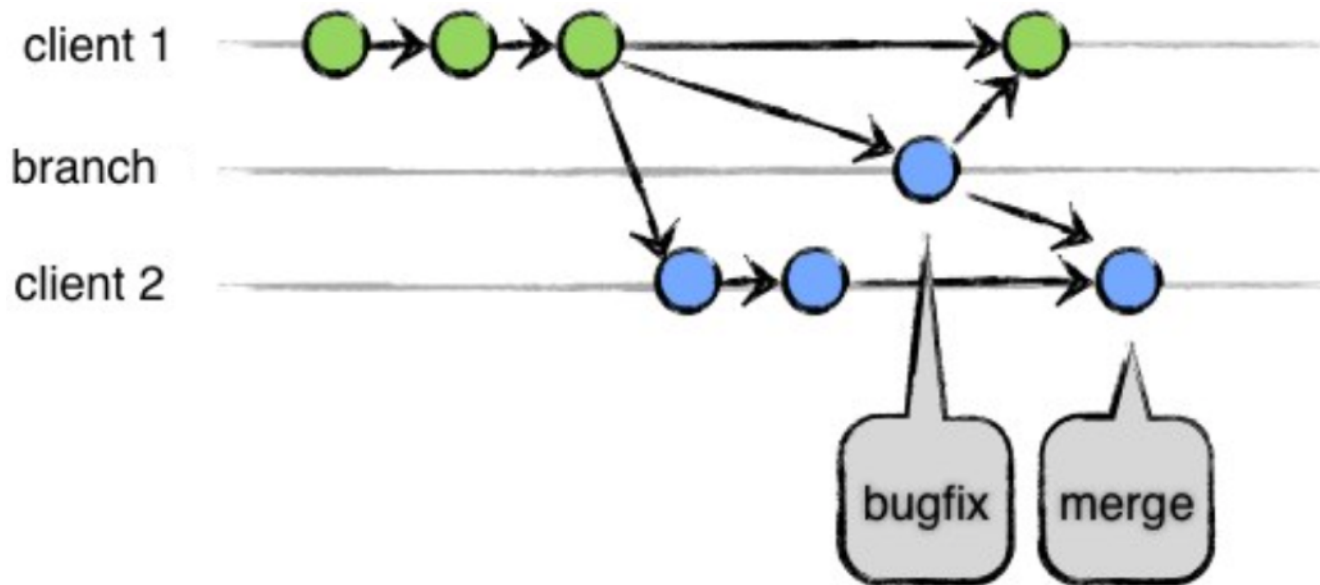


SVN makes a copy of the entire repository

Git makes a link to a particular state of the files

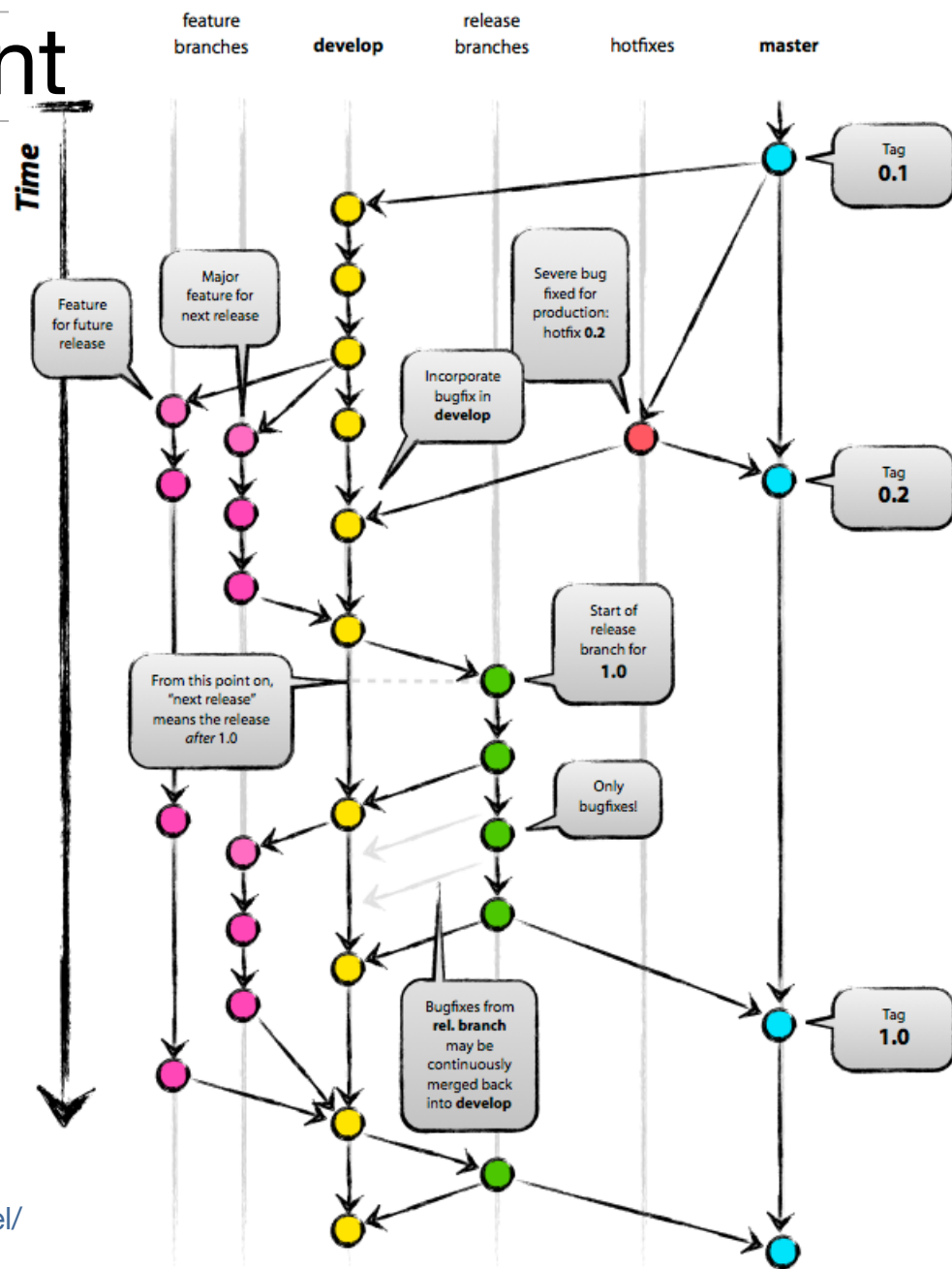
# Branch management

Merging branch (very complex to achieve with SVN)



# Branch management

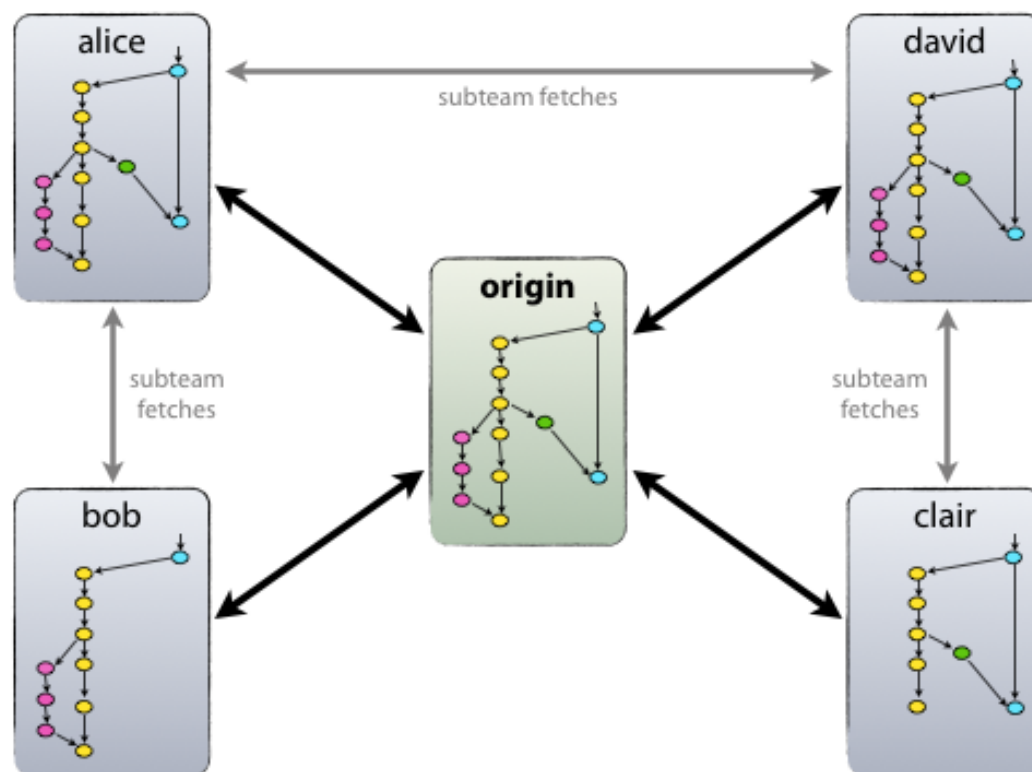
Classical organisation of a project into branches



# Branch management

Each user can work on a particular branch (or branches)

Branches can be synchronized between users



# Usability

git conceptual model is complex and can be counter-intuitive

See the article by Perez de Rosso & Jackson, "What's wrong with git?: a conceptual design"

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



# Continuous integration

Continuous merging & testing the work of several developers

Automatic deployment

System always running

## Goals

Test modifications from the beginning

Detect integration problems early

Always have a running system

Tests, demos, discussions with the customers

# Principles of continuous integration

Version control in a repository

Automatic and fast build

Auto-testing

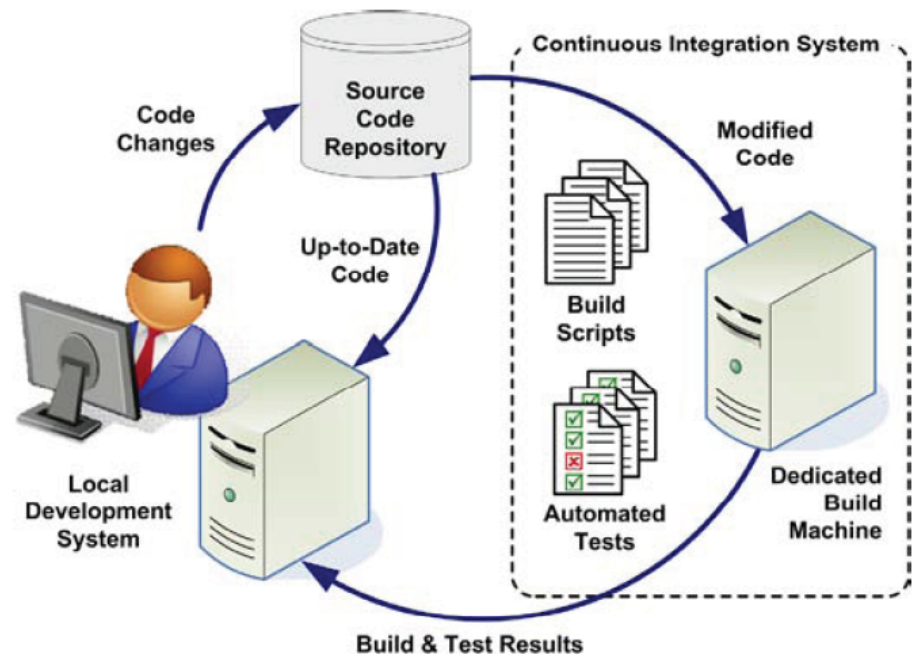
"Commit" every day

Deployment on an integration computer after each "commit"

Automatic deployment

Executable always available

Everybody knows the build state



# Web-based interfaces

Manage user access rights

Manage branches and access to branches

Review modifications and different versions

Track bugs

Run continuous integration scripts

Create wiki / web pages for projects

Social networking



# Outline

Collaborative software development

Version control

Software development methods

# Methods for software development

## **No methods: “Code and fix”**

Efficient for small projects

Difficult to add new features or to find bugs

## **Engineering / plan-driven methodologies**

Comes from civil or mechanical engineering

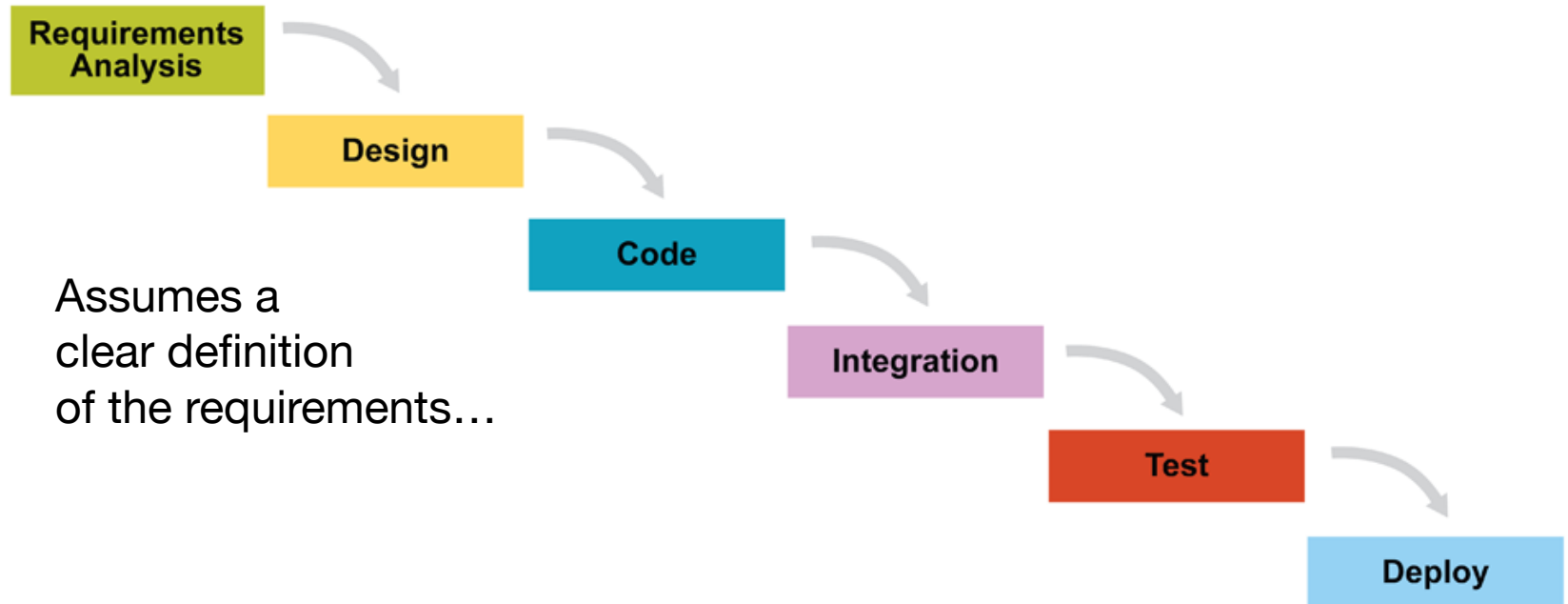
Drawing / construction plan / task distribution / construction

## **Agile methodologies**

Adaptive rather than predictive

People-oriented rather than process-oriented

# Classic methodology: the waterfall



Assumes a clear definition of the requirements...

... and a clear ending

# Predictability

Is software development end-result predictable?

Yes in some cases...

NASA programs (maybe)

Usually, requirements are unpredictable

(especially for software involving interaction with users)

Users / customers don't precisely know what they want

Hard to evaluate the cost of different options

Hard to estimate which features are useful

⇒ Requirements should be flexible

# Agile manifesto

Manifesto for Agile Software Development

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following plan

<http://agilemanifesto.org>

# Agile methods

Deal with unpredictable requirements

## **Iterative development**

Involve the customers at each iteration

Improve the team organization (self-adaptive process)

## **Effective team of developers (people first)**

Do not consider developers as replaceable parts

Analysts, coders, testers, managers

Developers are professionals who

make the technical decisions

evaluate the time required to perform the tasks

# Agile methods

## Examples

XP (Extreme Programming)

Test-driven development, **pair programming**

### **Scrum**

Crystal

Safety, efficiency, habitability (less disciplined than XP)

Open source process

Distributed contributors, parallelized debugging

Lean software development (Lean development @ Toyota)

Just in time, Jidoka ("automation with a human touch")

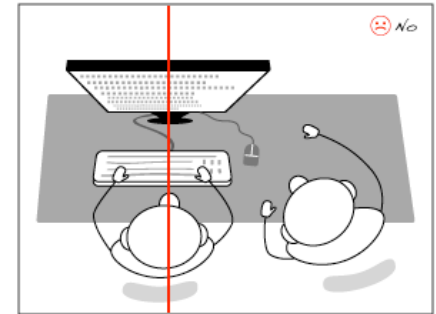
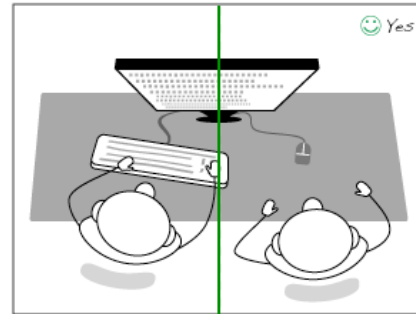
RUP (Rational Unified Process)

Use case driven, iterative, architecture centric

# Pair/peer programming

Two programmers

One computer



Roles

**Driver:** operates mouse and keyboard

Code: syntax, semantics, algorithm

**Navigator:** watches, learns, asks, talks, makes suggestions

Higher level of abstraction

Test, technical task, time since the last commit,

Quality of the overall design

# Pair/peer programming

## Advantages

### Code quality

Better designs, Fewer bugs

### Productivity

May be lower in short term, but higher in long term

### Spreading Knowledge

Pairs have to switch regularly

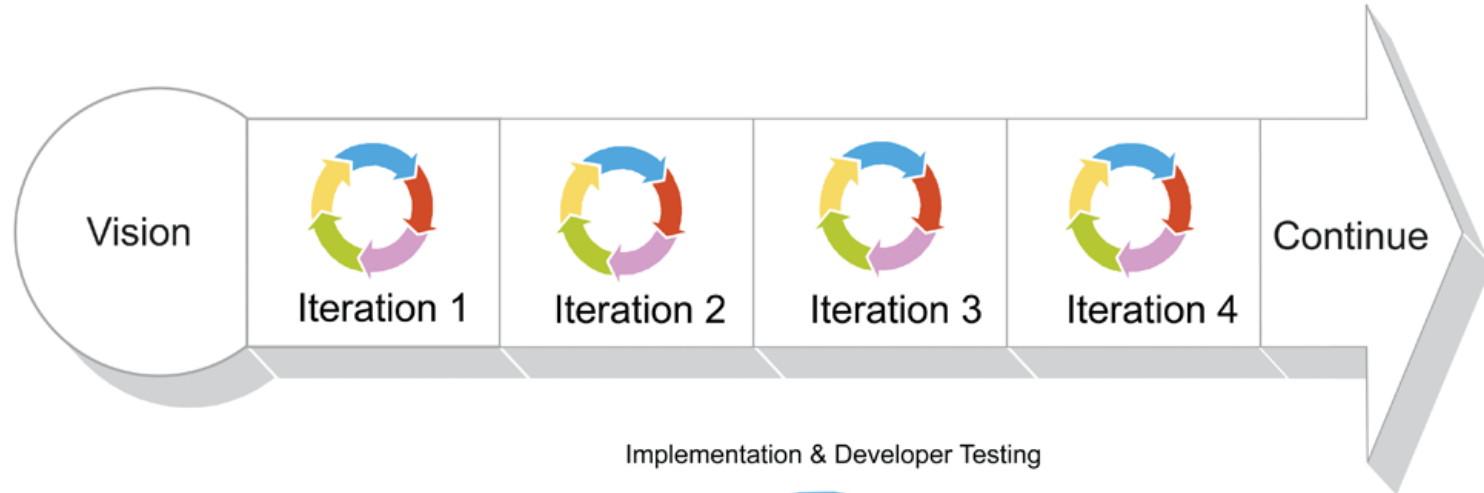
Technical and conceptual knowledge

### Social aspects

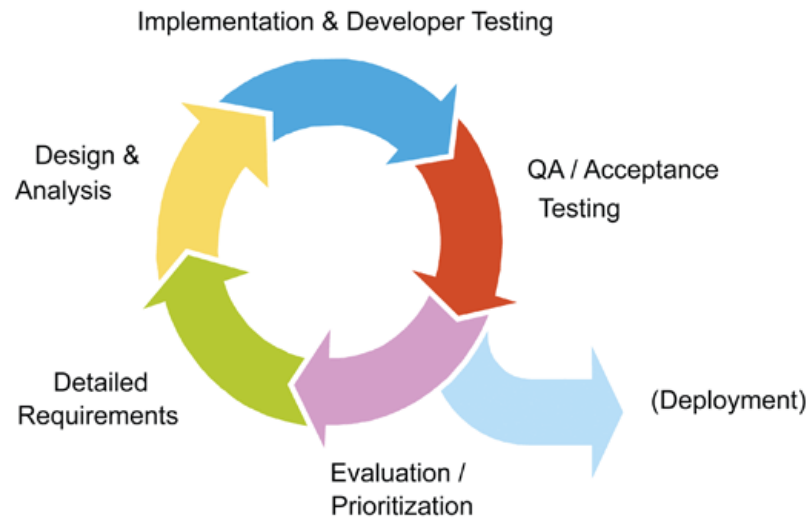
No loneliness, conviviality, better motivation

# Scrum

Iterations called Sprints (about 1 month)



## Iteration Detail



# Scrum: roles

## Product Owner (one person)

Responsible for products vision

Constantly re-prioritizes the Product Backlog

Accepts or rejects product increment



## Development team

Self-organized

Negotiates commitments with the Product Owner

Has autonomy regarding how to reach commitments

Intensely collaborative



## Master

Facilitates the Scrum process

Helps resolve issues

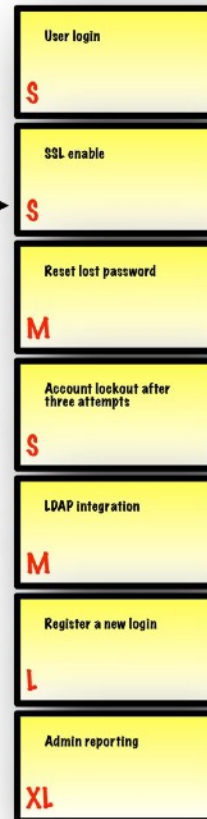
Shields the team from external inferences and distractions

Has no management authority



# Scrum: product backlog

top items  
are more  
granular



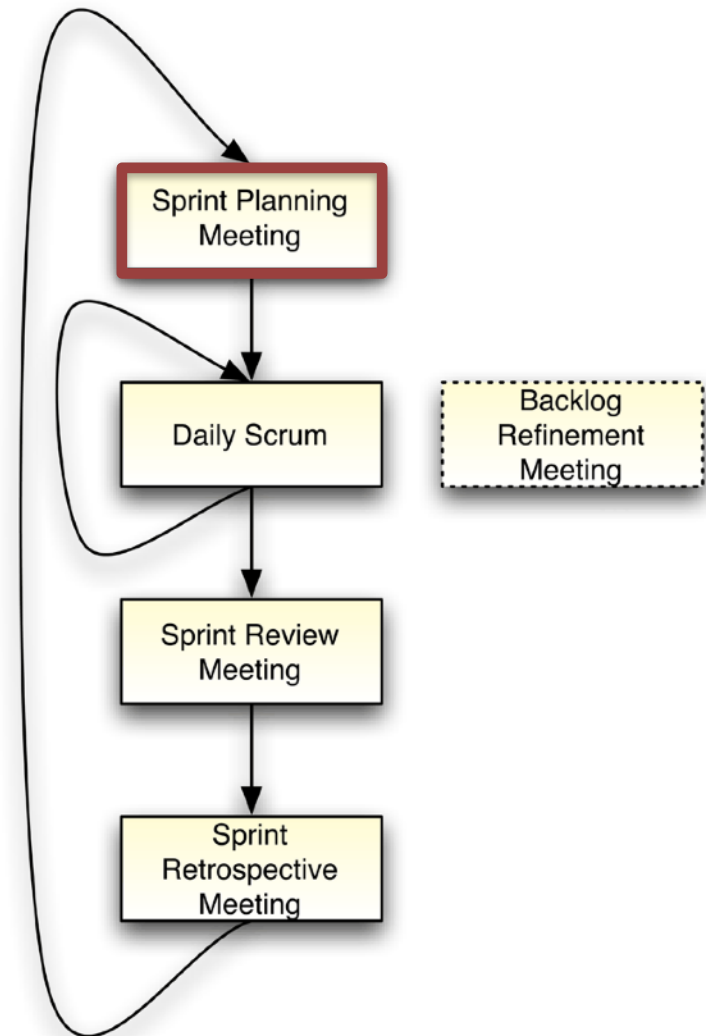
only one item  
at a time  
is top priority

# Sprint planning

## Planning Meeting

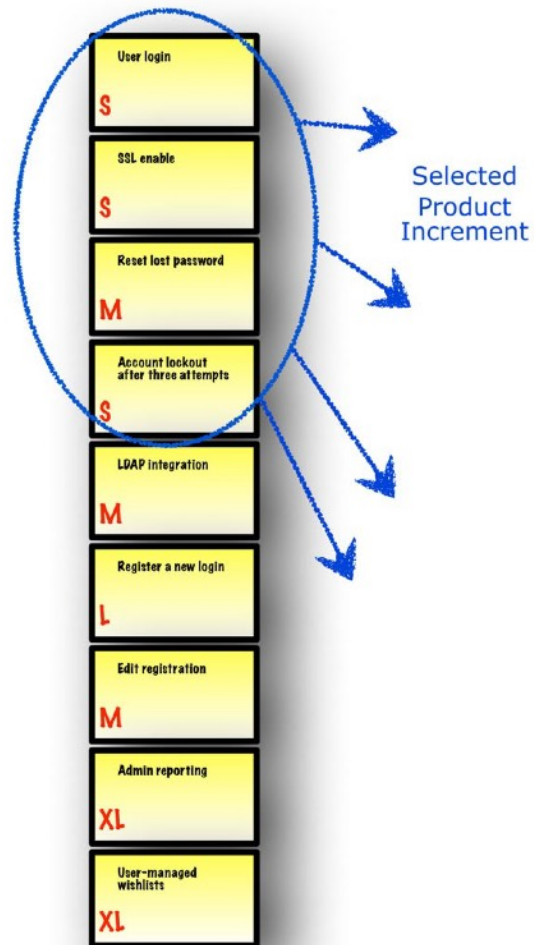
Negotiate which Product Backlog items will be processed

Break items into a list of sprint tasks

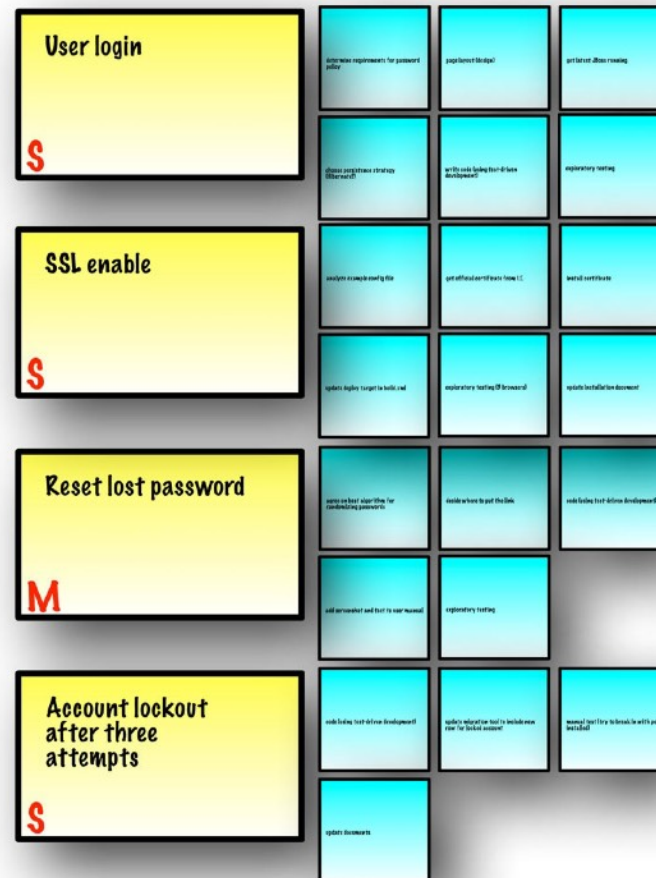


# Sprint backlog

## Product Backlog



## Sprint Backlog



# Daily scrum

## Planning Meeting

## Daily Meeting

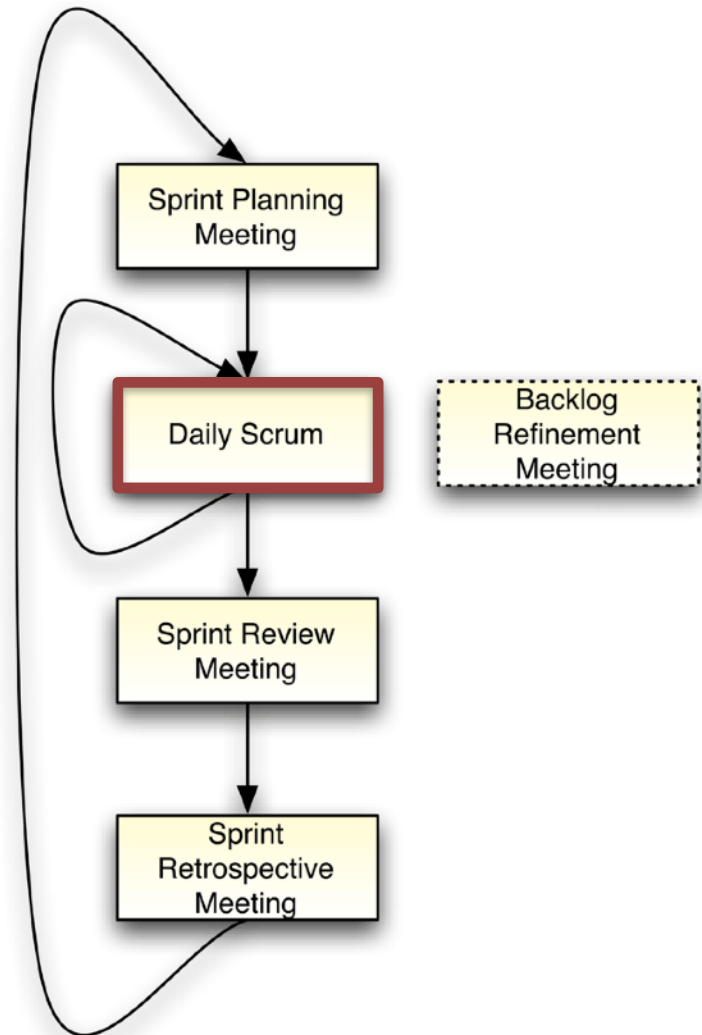
Same time and place

15 minutes, standing up






















Summarize work of previous day,  
work of today, issues

Maintain tasks list  
(not started, in progress, done),  
issues list and burn-down chart

Product Owner may attend



# Sprint backlog

Committed Backlog Items	Tasks Not Started	Tasks In Progress	Tasks Completed
	  		 
	  		
	     		
			

# Sprint review

**Planning Meeting**

**Daily Meeting**

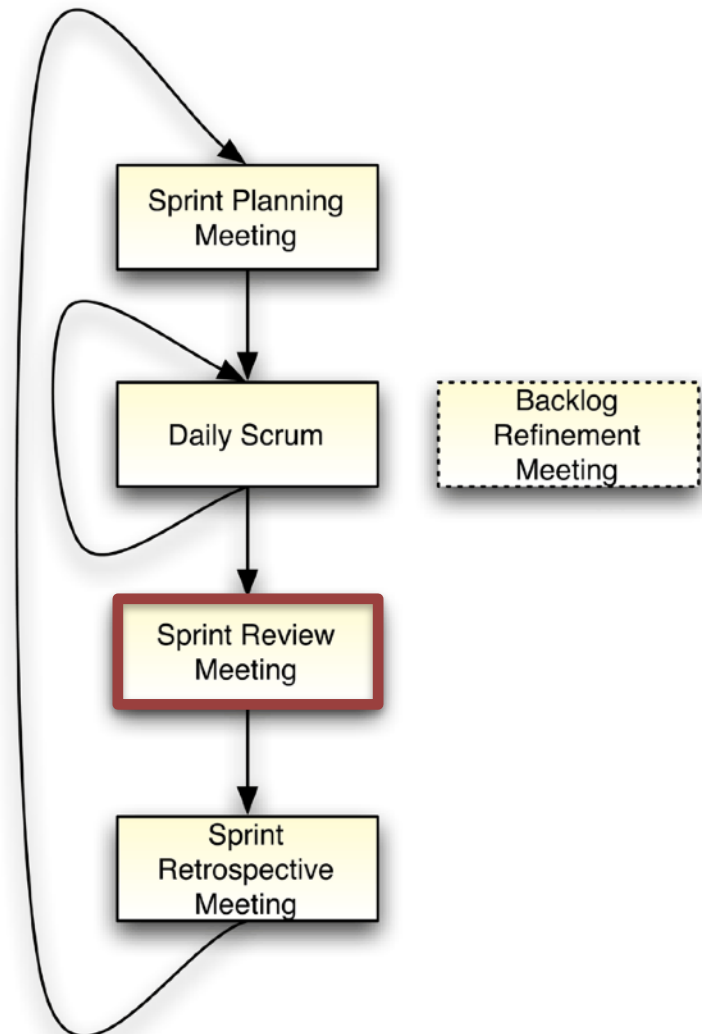
**Review Meeting**

Demonstrate the working product increment to the Product Owner

Product Owner declares which items are done

Unfinished items return to the Product Backlog

Master proposes new items for the Product Backlog



# Sprint retrospective

Planning Meeting

Daily Meeting

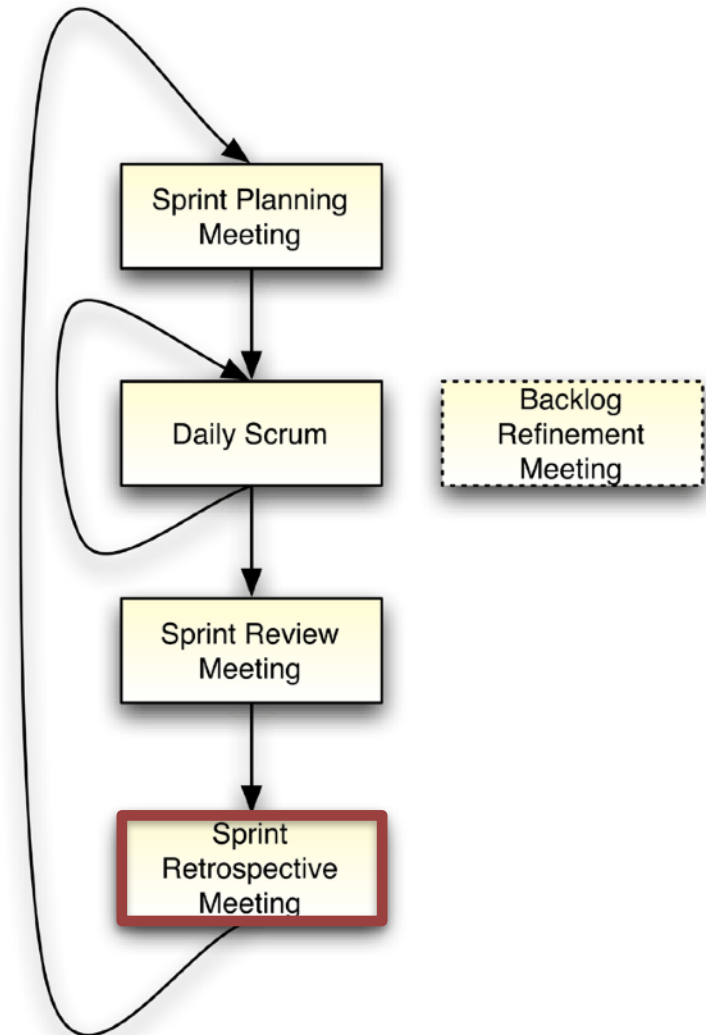
Review Meeting

**Retrospective Meeting**

Team reviews its own process

Team adapts it for future Sprints

Master has to manage  
the psychological aspects  
of the meetings



# Backlog refinement

Planning Meeting

Daily Meeting

Review Meeting

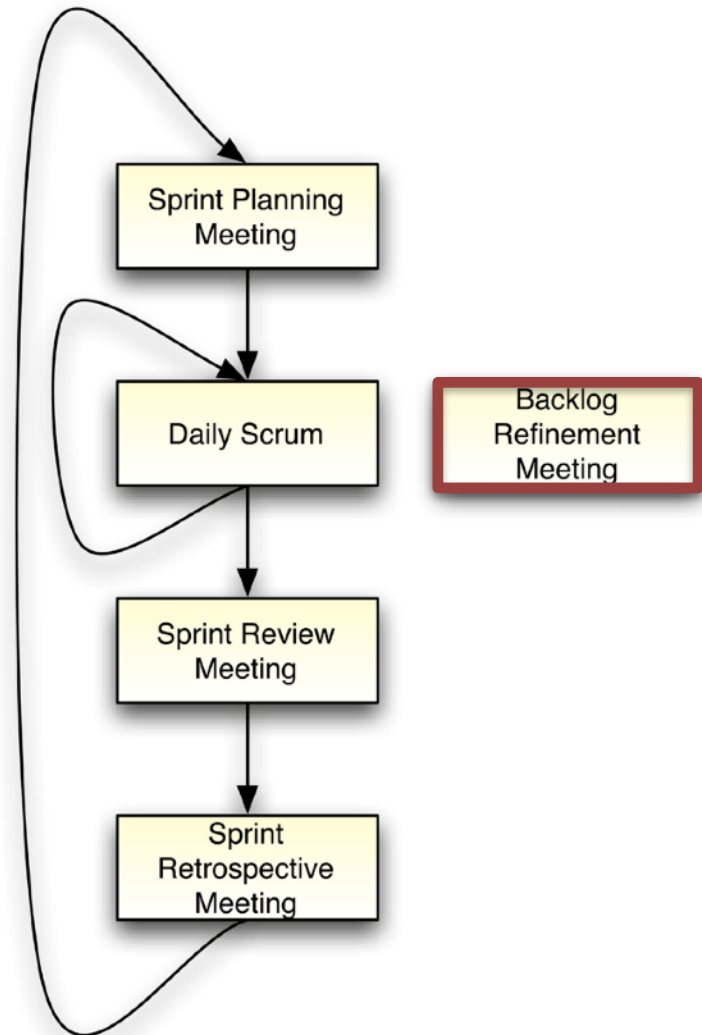
Retrospective Meeting

**Backlog Refinement Meeting**

Items are usually too large  
or poorly understood

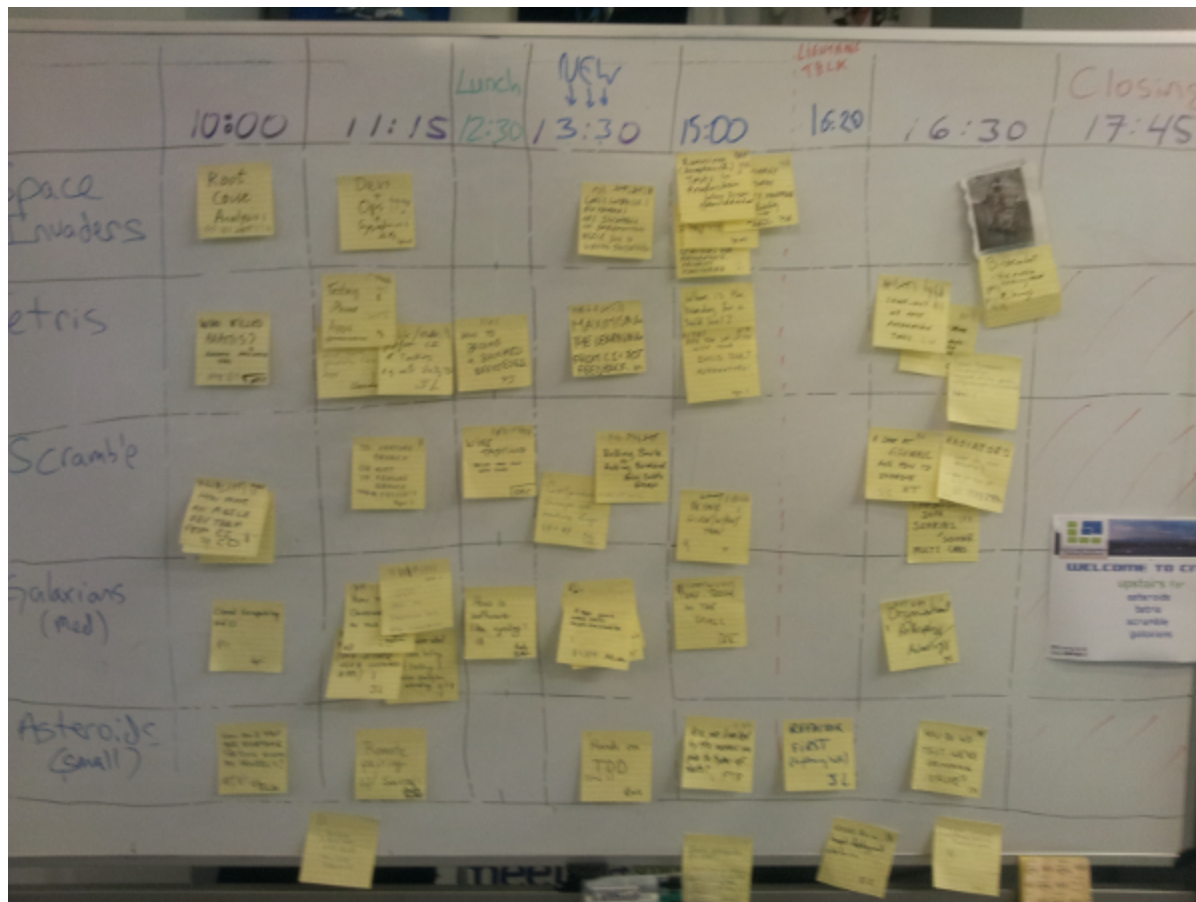
Refine items into smaller ones

Master can help



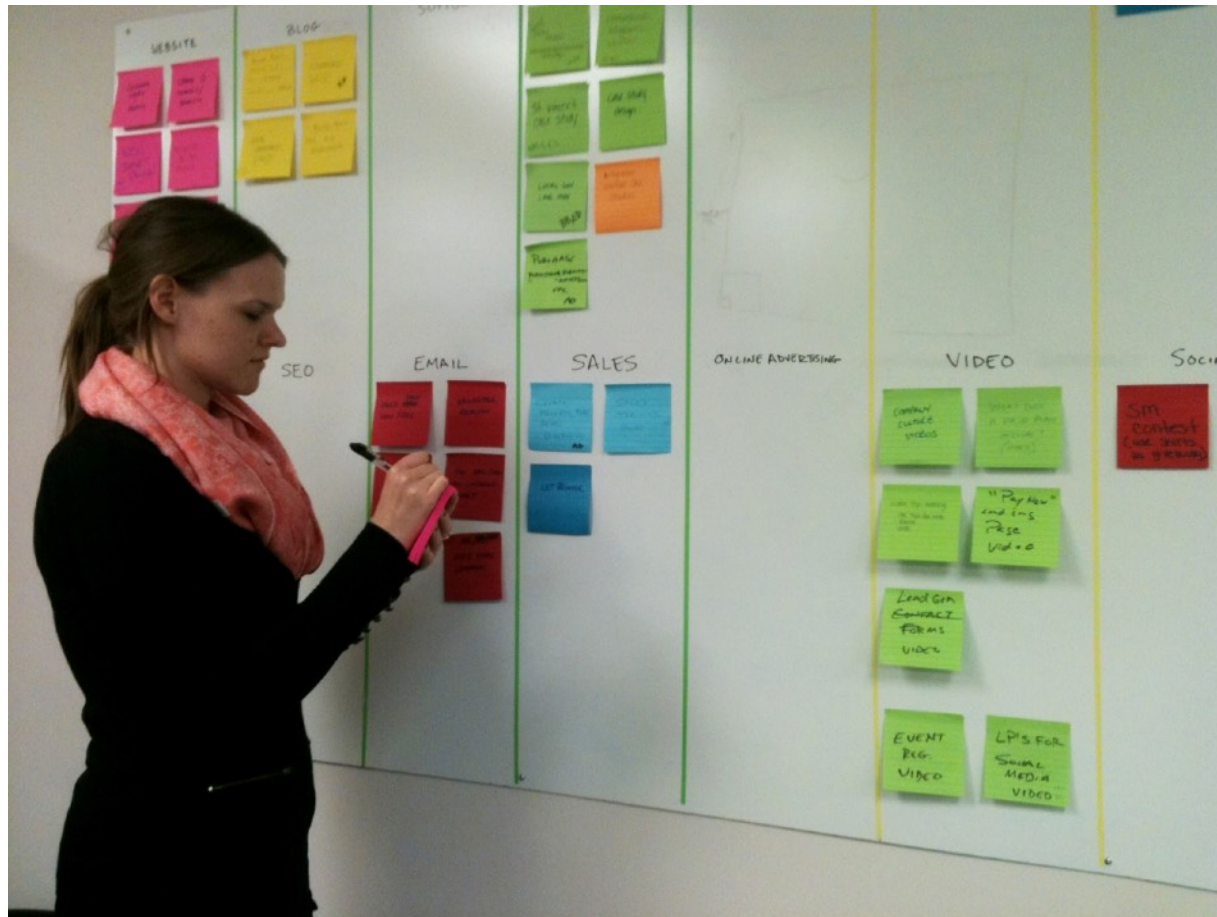
# Wallboard

## Feedback to the team

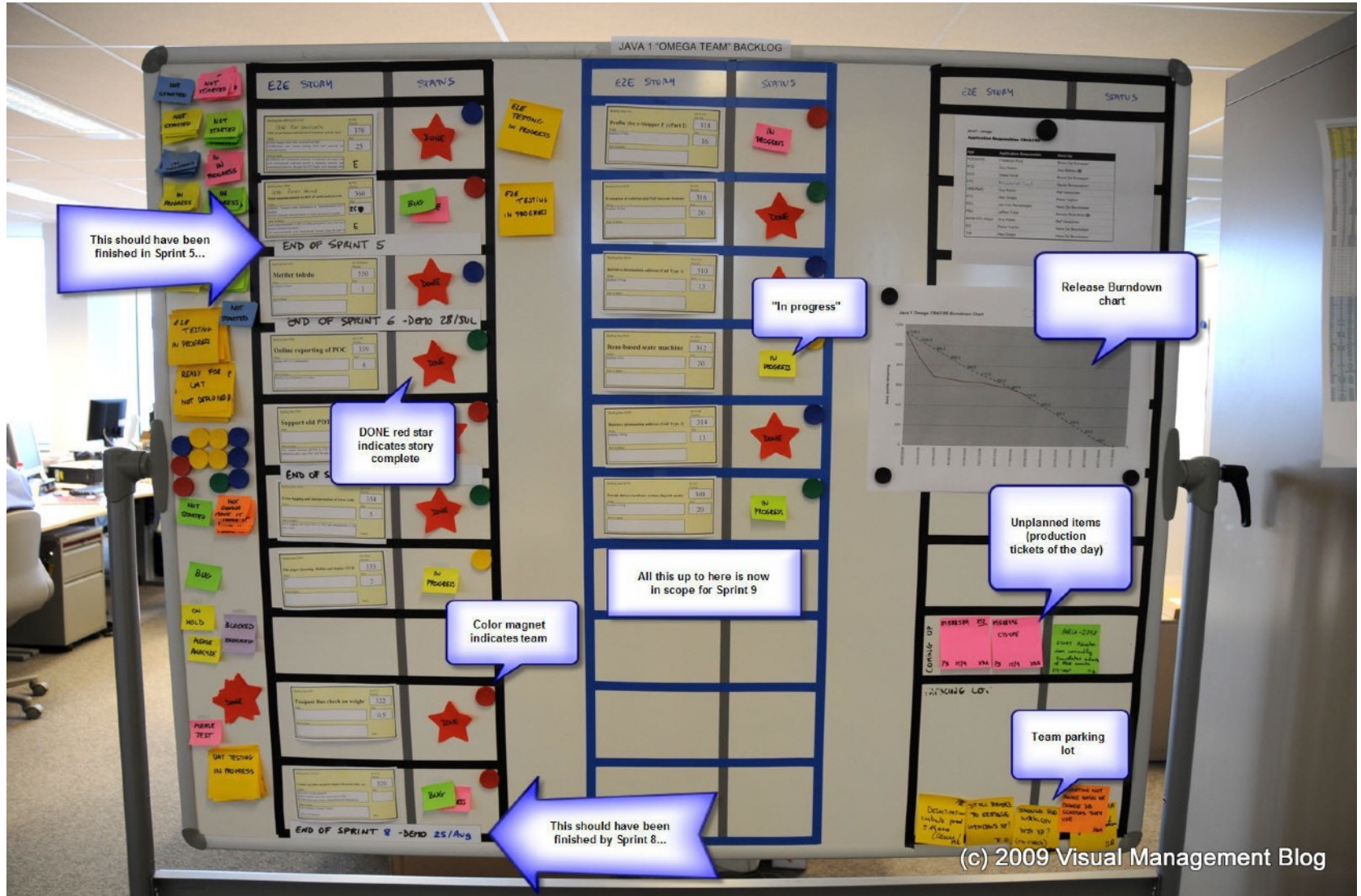


# Wallboard

Feedback to the team



# Wallboard

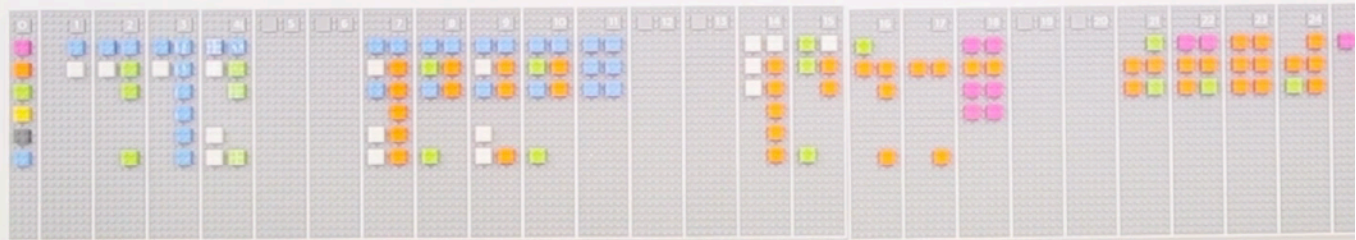


# LEGO Bit planner

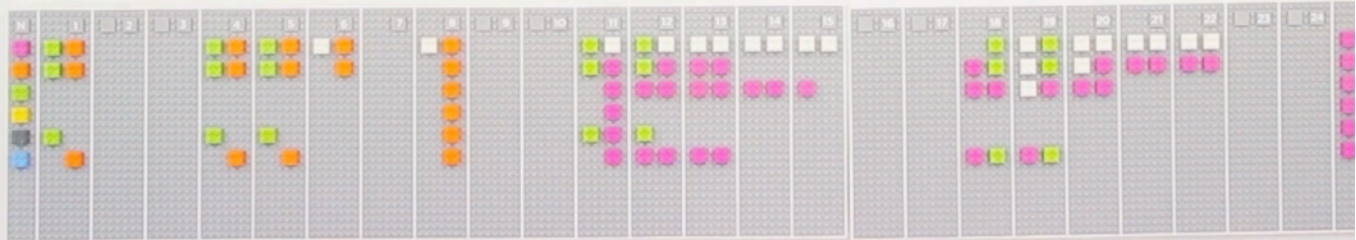
OCTOBER



NOVEMBER



DECEMBER



# Scrum software

The screenshot displays the JIRA Agile interface for a project named "Angry Nerds". At the top, the navigation bar includes "Dashboards", "Projects", "Issues", "Agile", and "Bonfire". The main header shows "Angry Nerds" with a search bar and filters for "Only My Issues" and "Recently Updated".

The central section features an "Angry Sprint" with a progress bar and a list of 8 issues:

- NERD-1: As a Front-Ender I would like to stop supporting IE6 so I can enjoy my life (3)
- NERD-2: As a Hacker I would like more Red Bull so I can work all night (8)
- NERD-3: As the Dev Manager I would like to look busy so I can keep my job
- NERD-4: As an Outsourcerer I want to get paid for working in my pyjamas (20)
- NERD-5: As an Agilista I want to play buzzword bingo so I kan ban
- NERD-6: As the Founder I want to have the last say so I can get my way (13)
- NERD-8: As a Bug I want to fly in the face of progress
- NERD-7: As a Bug I want to make like hard for the Angry Nerds

Below the current sprint are two "Upcoming Sprint" sections:

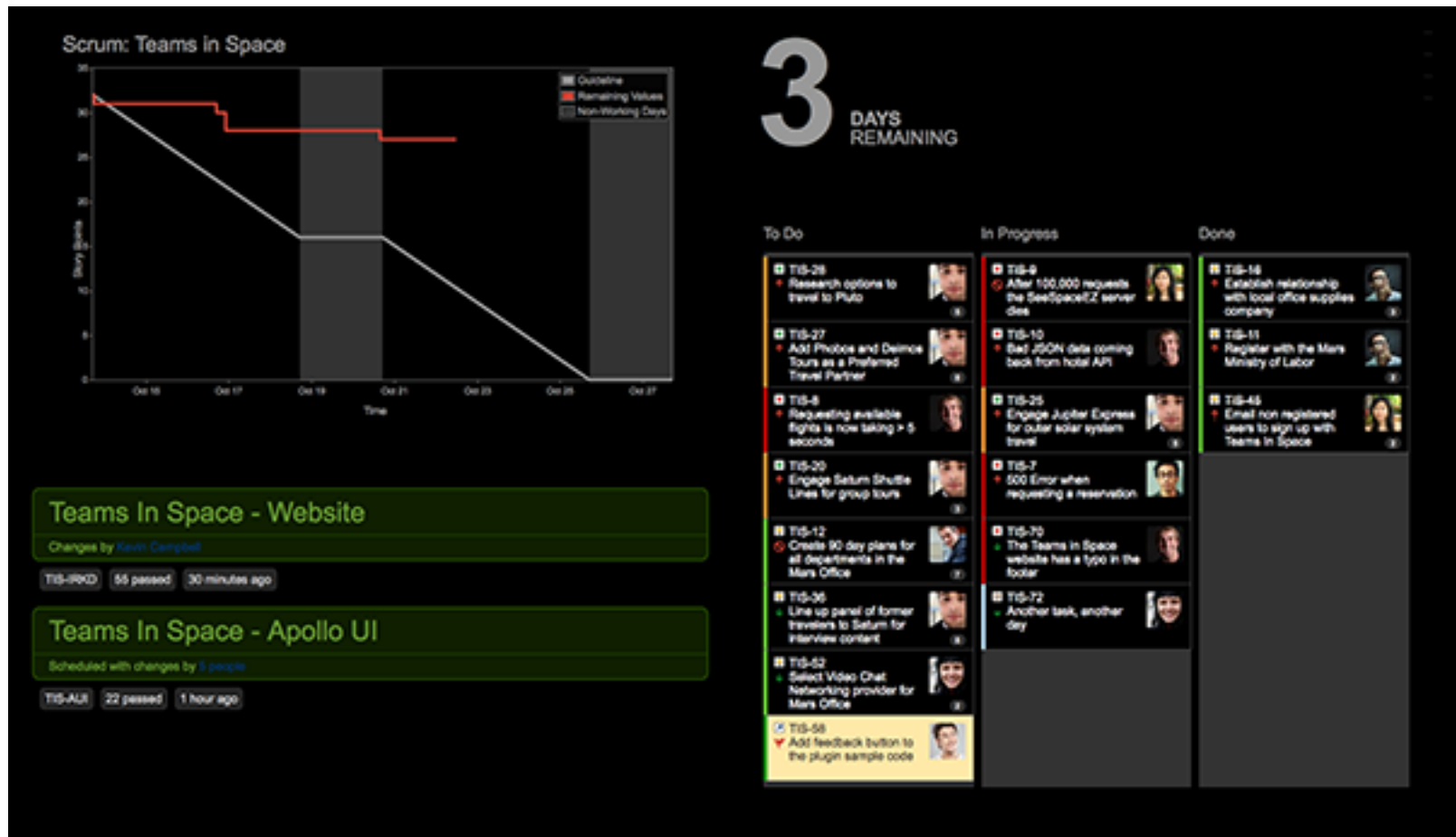
- Upcoming Sprint 1:** Contains 5 issues with a total estimate of 29. Issues include NERD-9 (5), NERD-12 (3), NERD-13 (13), and NERD-14 (8).
- Upcoming Sprint 2:** Contains 3 issues with a total estimate of 9. Issues include NERD-15 (3), NERD-16 (1), and NERD-17 (5).

On the right side, a detailed view of issue "NERD-4" is shown. The title is "As an Outsourcerer I want to get paid for working in my pyjamas" with an estimate of 20. The status is "Not Started" and 0/1 Completed. A table below shows session details:

Session	Status	Actions
Confirm they are in pyjamas	Created	Info, Settings

A "Create Session" button is located at the bottom of the issue detail panel.

# Scrum software



# Conclusion

Collaboration in software development

Is necessary for large projects

Is not obvious:

Technical, organizational and social aspects

Version control supports asynchronous collaboration

Explicit synchronization between versions

Branching: split work among developers

# Conclusion

## Continuous integration

Improves safety and efficiency

## Agile method

Organizes a development team

Proposes an adaptative process  
to unpredictable requirements

# References

## Version control

<http://nvie.com/posts/a-successful-git-branching-model/>

<http://www-igm.univ-mlv.fr/~dr/XPOSE2010/gestiondeversiondecentralisee/dvcs-svn.html>

<http://www.infres.enst.fr/~bellot/java/poly/git.pdf>

<http://fr.openclassrooms.com/informatique/cours/gerez-vos-codes-source-avec-git/qu-est-ce-qu-un-logiciel-de-gestion-de-versions>

## Continuous Integration

<http://martinfowler.com/articles/continuousIntegration.html>

## Agile Models

<http://agilemanifesto.org>

<http://martinfowler.com/articles/newMethodology.html>

## Pair Programming

[http://www.versionone.com/Agile101/Pair\\_Programming.asp](http://www.versionone.com/Agile101/Pair_Programming.asp)

## Scrum

<http://scrumreferencecard.com>