

*Groupware and Collaborative Interaction*  
**Collaborative Software Development**

*M2R Interaction / Université Paris-Sud / 2015 - 2016*

Cédric Fleury (cedric.fleury@lri.fr)

# Software development

Several users work on a same project

- Remote or collocated users

- Each one works on its own computer (asynchronous)

  - Work on different tasks

  - Work at different times

Collaboration is hard to organize

- Versioning, synchronization between users

- Tasks distribution, social aspects

## Collaborative software development

Version control

Continuous integration

Agile methods

# Outline

Collaborative software development

**Version control**

Continuous integration

Agile methods

# Version control

## Problematic

### We want to avoid:

Manually share the files (USB key, email, Dropbox)

Delete or overwrite the files of other users

Broke all the project by making a mistake

### We want to able to:

Edit the project at the same time

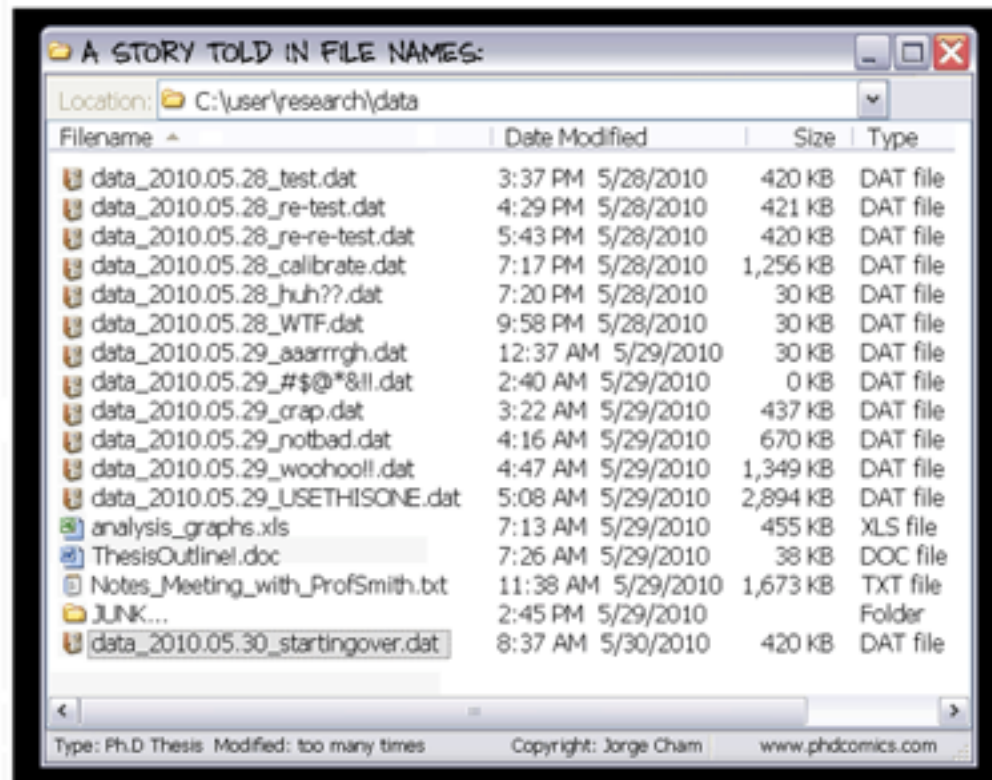
Keep an history of the modification

Keep the older version of the files + hierarchy

# Version control

Problematic

We want to avoid this:



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

# Version control

## 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

# Version control

## 2 kinds of architecture

### Centralized

CVS, SVN, ...

### Decentralized (peer-to-peer):

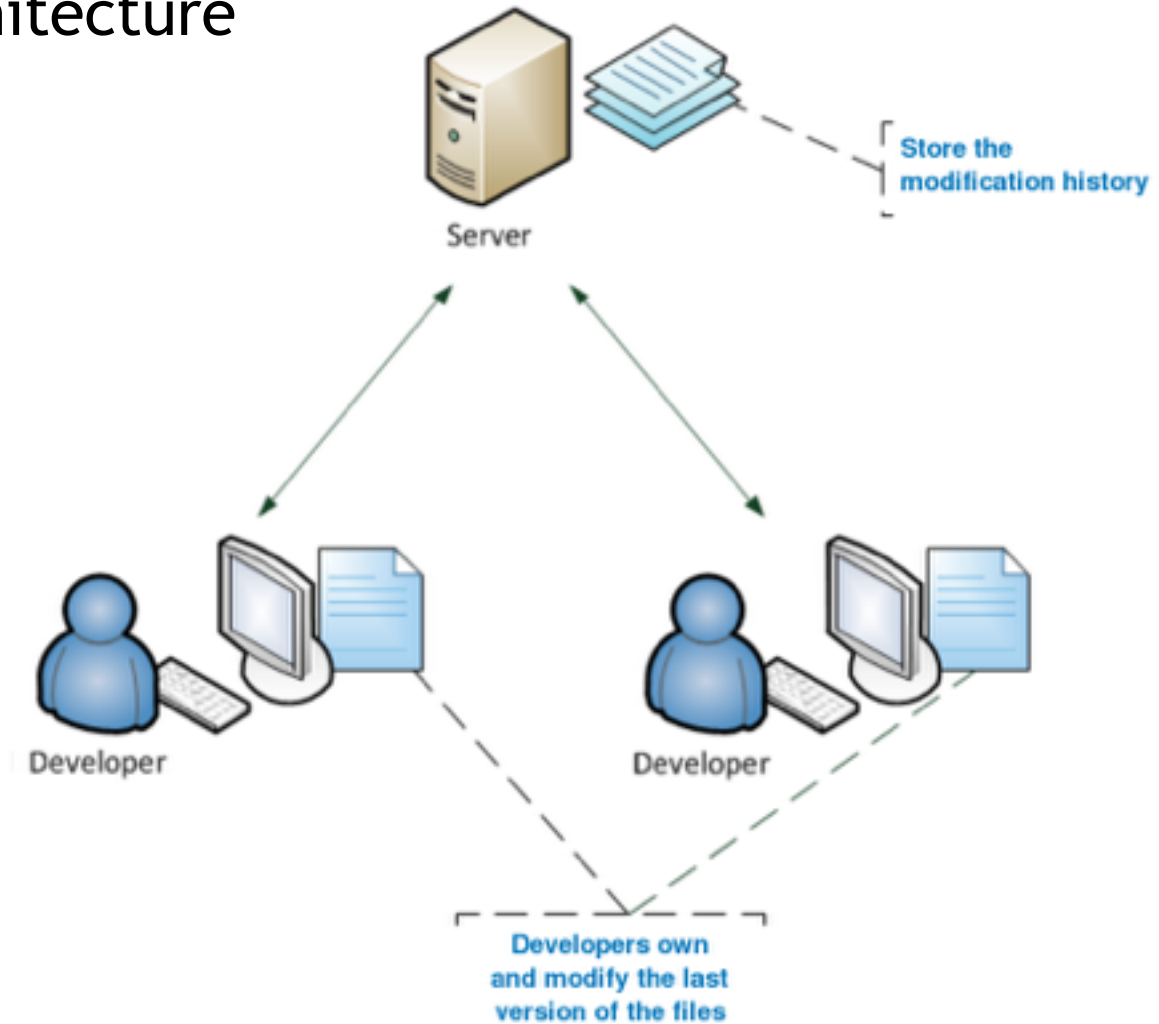
GNU Arch, Mercurial, Bazaar, Git,...

### Decentralized can be used as a Hybrid Architecture

One peer can be a central server

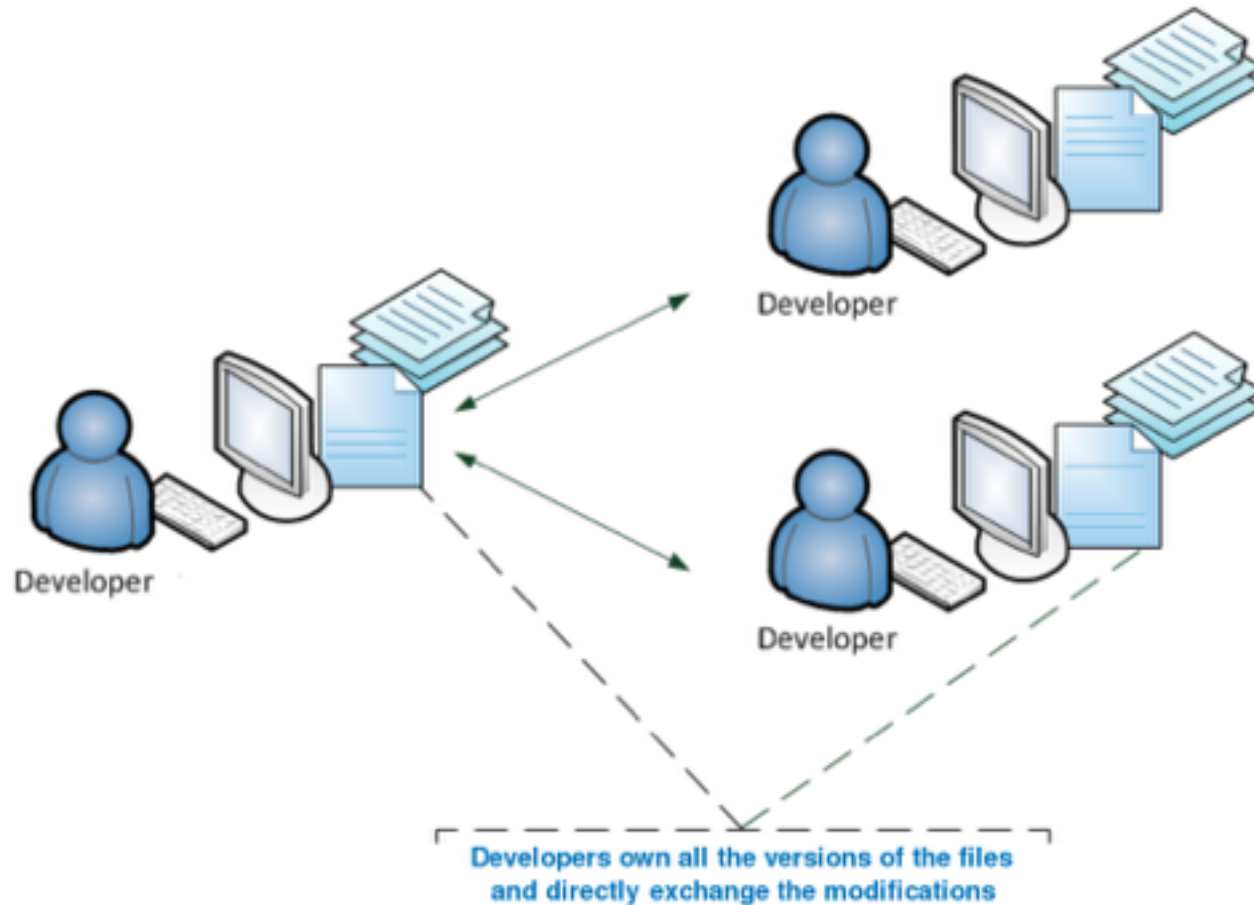
# Version control

## Centralized architecture



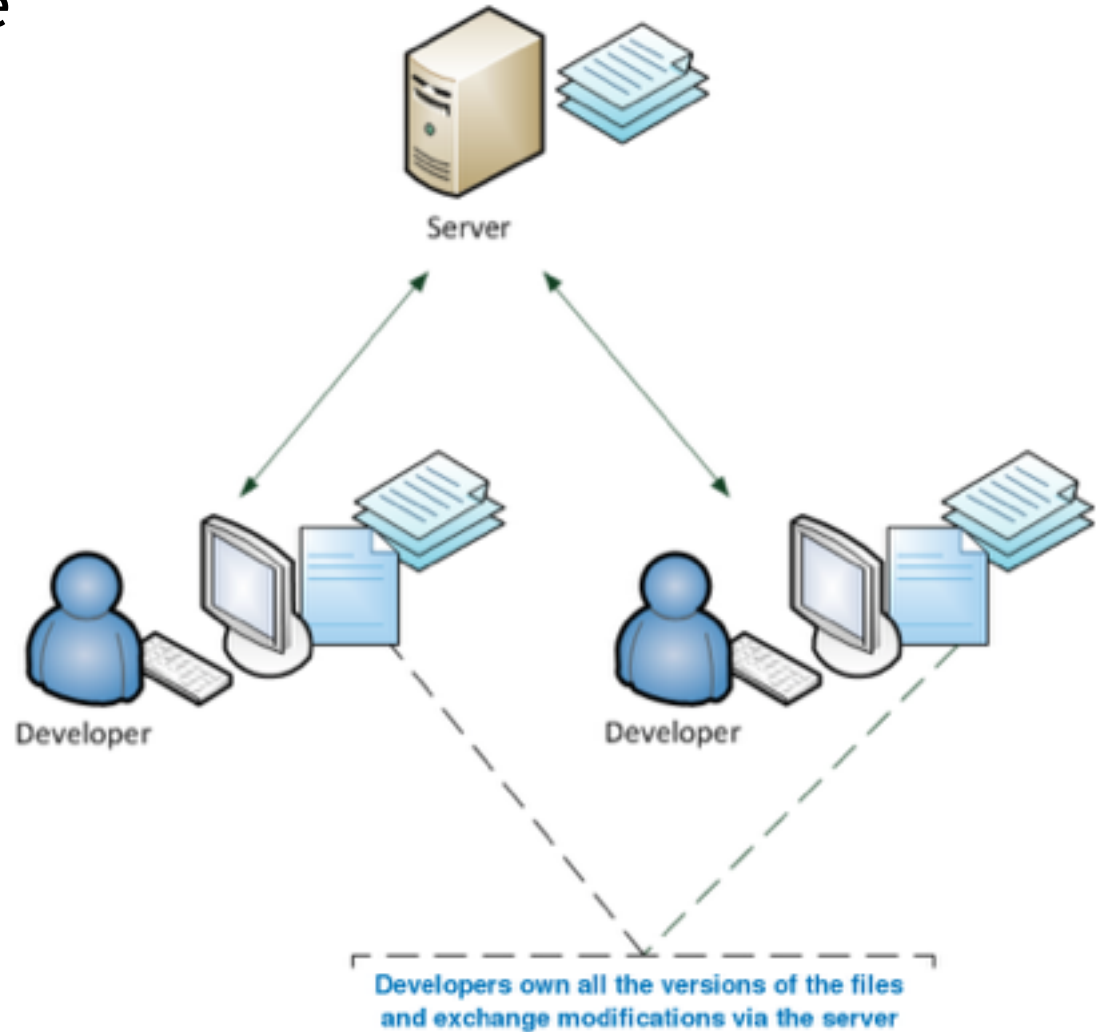
# Version control

## Decentralized architecture



# Version control

## Hybrid architecture



# Version control

## Vocabulary (SVN)

Architecture

Repository

Working copy

## Actions

Checkout

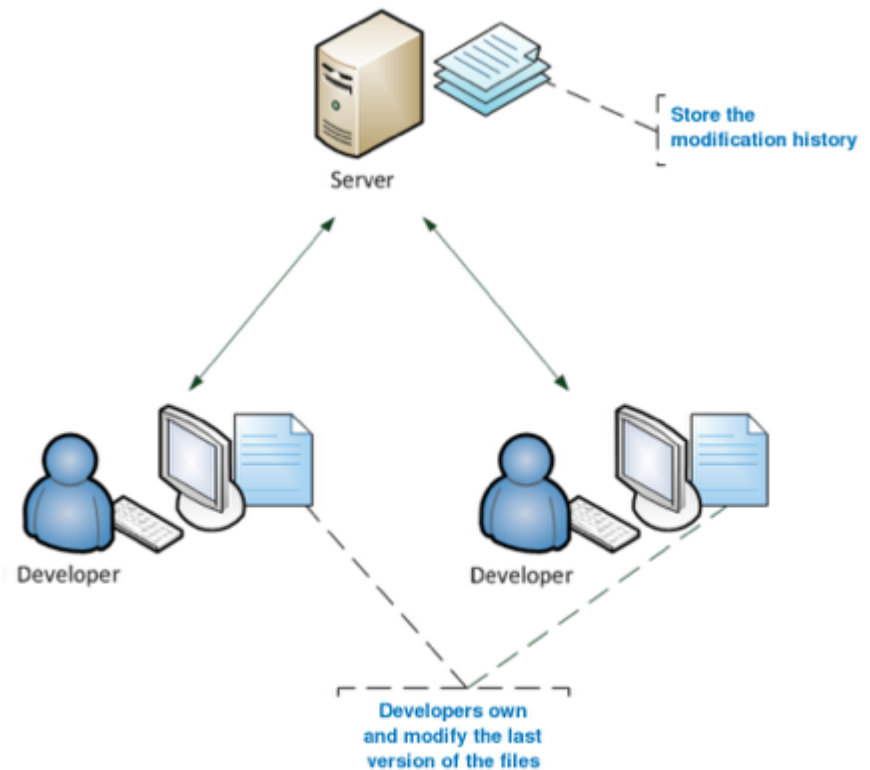
Update

Commit

Revert

Diff, log, status

## Centralized Architecture



# Version control

## Drawbacks of the centralized architecture

- Just one access point to the data

- Just one communication point between users

- Just one historic of the files

- Versioning and sharing are the same operation

  - Need to have a stable state before "committing"

# Version control

## Vocabulary (Git)

### Architecture

Remote and local repository

Working copy

### Actions

Clone

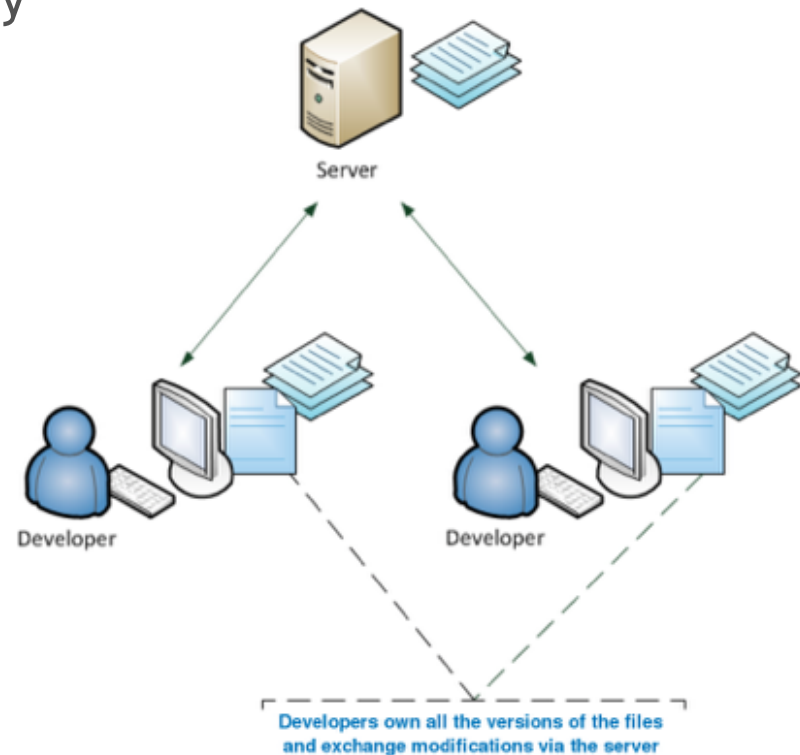
Pull, Push

Commit

Reset

Diff, log, status

## Hybrid Architecture



# Version control

## Good practices

Work on the local copy

Send the modification

Check if the code compiles locally

Check for updates from the other users

Manage conflicts if there are some

Check if the code compiles with the updates

Commit the code on the shared version (server)

# Version control

Users can modify the same file

But at different part/section of the files

If they modify the same part of a file

A conflict appends

Usually, it cannot be resolve automatically

Users have to fix the conflit

By telling to the system, which version is correct

By merging the modifications of the users

# Version control

## Conflicts management

```
C:\workspace\test>svn up
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
```

# Version control

## Conflicts management

```
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
```

# Version control

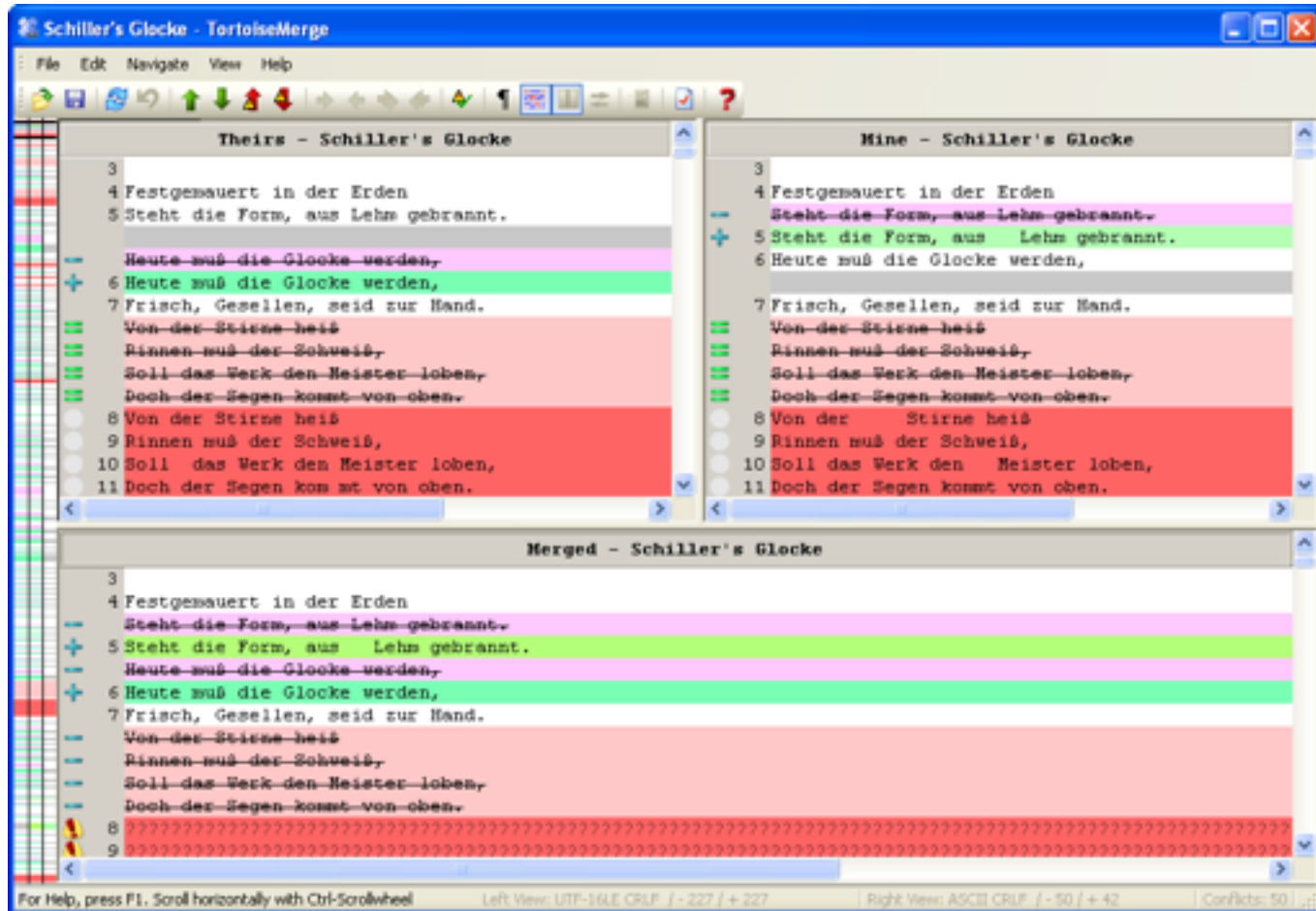
## Tools for conflict management (TortoiseMerge)

The image shows a screenshot of the TortoiseMerge application window. The window title is "SysImageList.cpp - TortoiseMerge". The interface is split into two panes. The left pane shows the original code with a conflict on line 58, where the string "doesn't matter" is highlighted in red. The right pane shows the merged code with a new line 57 containing the string "blablah" and a new method "Test()" starting at line 64. The status bar at the bottom indicates "Conflicts: 0".

```
56 SHGetFileInfo(  
57     T("doesn't matter"),  
58     FILE_ATTRIBUTE_DIRECTORY,  
59     &sfi, sizeof sfi,  
60     SHGFI_SYSICONINDEX | SHGFI_SMALLICON | SHGFI_USEFILEATTRIB  
61 )  
62  
63     return sfi.iIcon;  
64 )  
65  
66 int CSysImageList::GetDefaultIconIndex() const  
67 {  
68     SHFILEINFO sfi;  
69     // clear the struct  
70     ZeroMemory(&sfi, sizeof sfi);  
71  
72     SHGetFileInfo(  
73         T(""),  
74         FILE_ATTRIBUTE_NORMAL,  
75         &sfi, sizeof sfi,  
76         SHGFI_SYSICONINDEX | SHGFI_SMALLICON | SHGFI_USEFILEATTRIB  
77     )  
78  
79     return sfi.iIcon;  
80 }
```

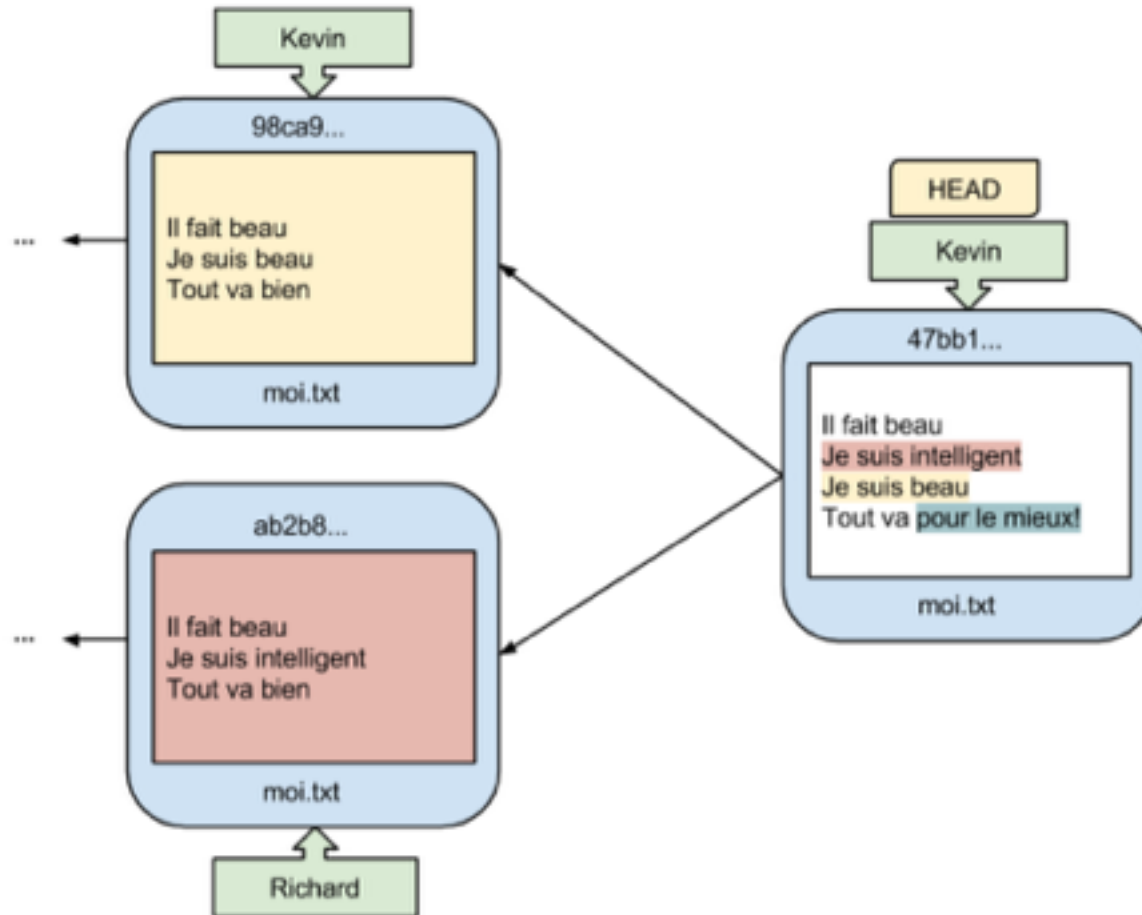
# Version control

## Tools for conflict management (TortoiseMerge)



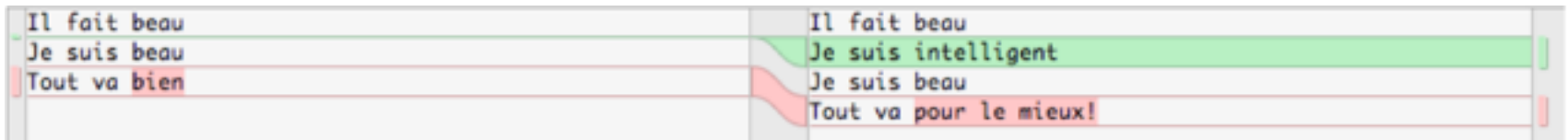
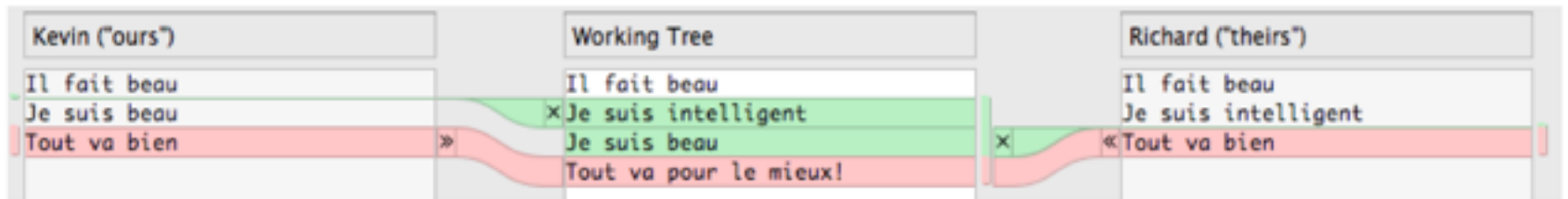
# Version control

## Tools for conflict management (SmartGit)



# Version control

## Tools for conflict management (SmartGit)



# Version control

## Conflicts management

### To avoid conflict:

Users are able to "lock" a file

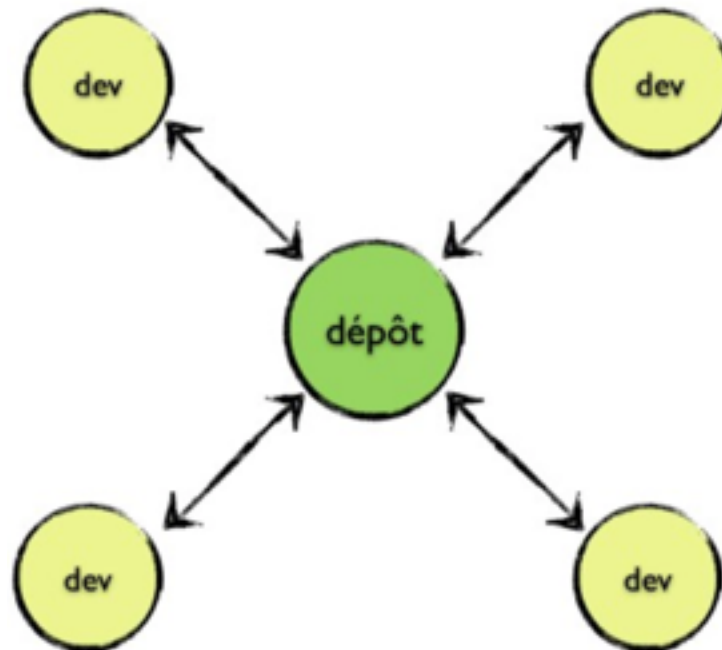
Only the user who locks the file can modify it

If another user try to lock a file while it is locked by another user, he receives an error message

Users have to manually unlock the file when they have finished to work on it.

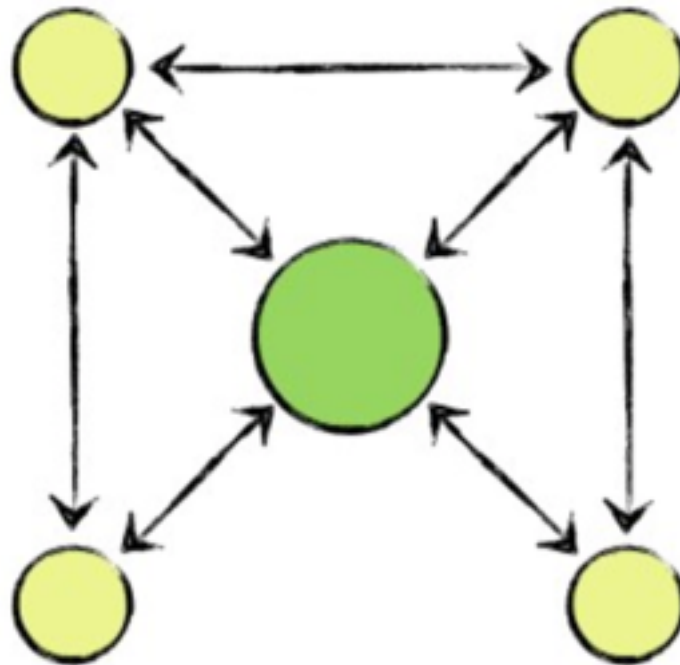
# Version control

Collaboration scenario : centralized (SVN)



# Version control

Collaboration scenario : decentralized (Git)



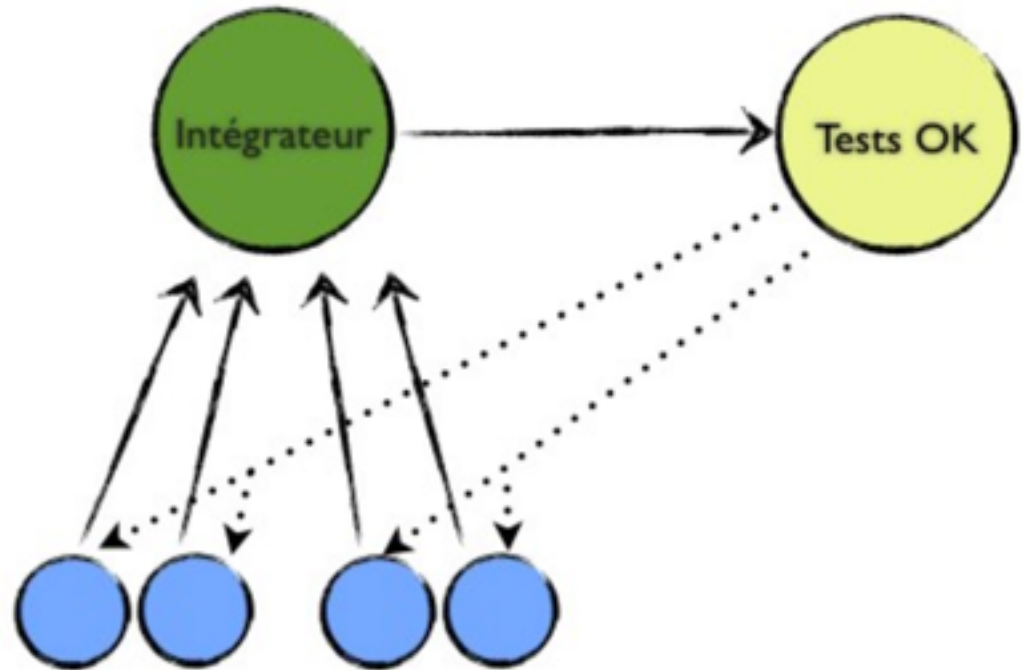
Inter-personal communications

# Version control

Collaboration scenario : decentralized (Git)

Integrator mode

A repository  
is in charge  
of the test



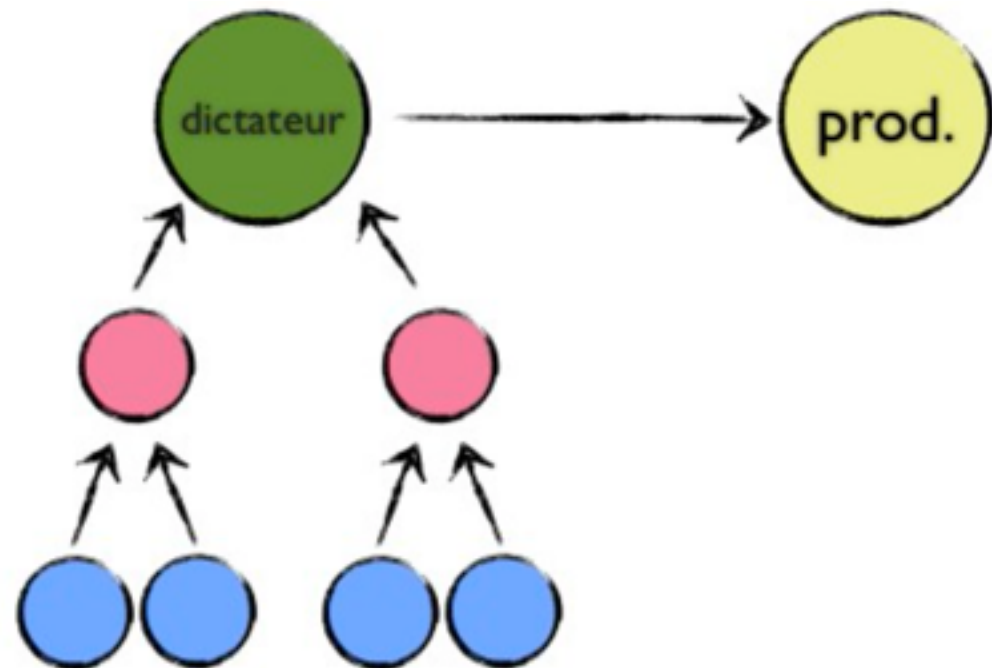
# Version control

## Collaboration scenario : decentralized (Git)

### Dictator mode

Open-source projects

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



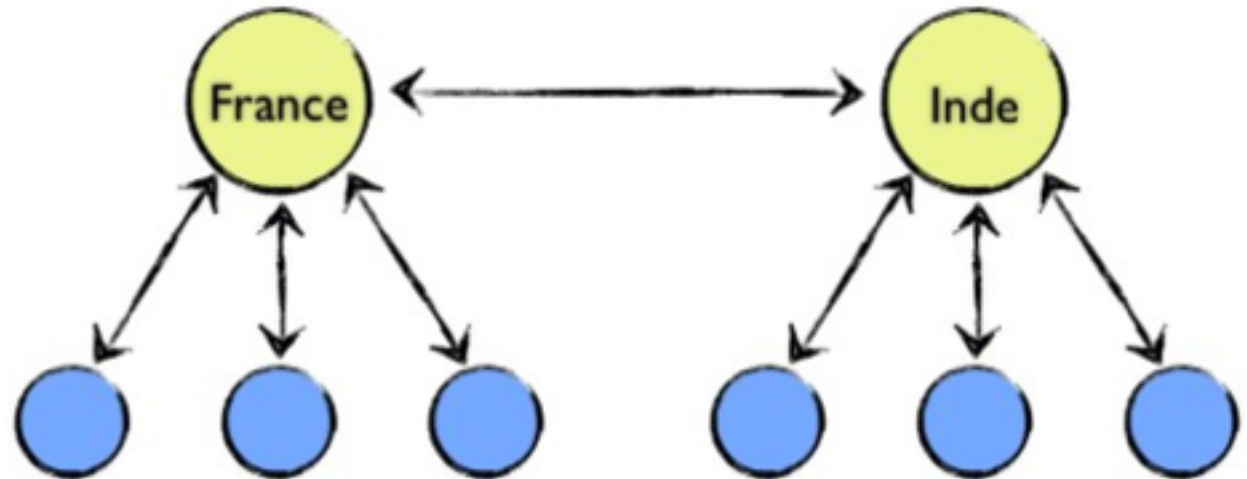
# Version control

## Collaboration scenario : decentralized (Git)

### Multi-location team

Each team can work independently

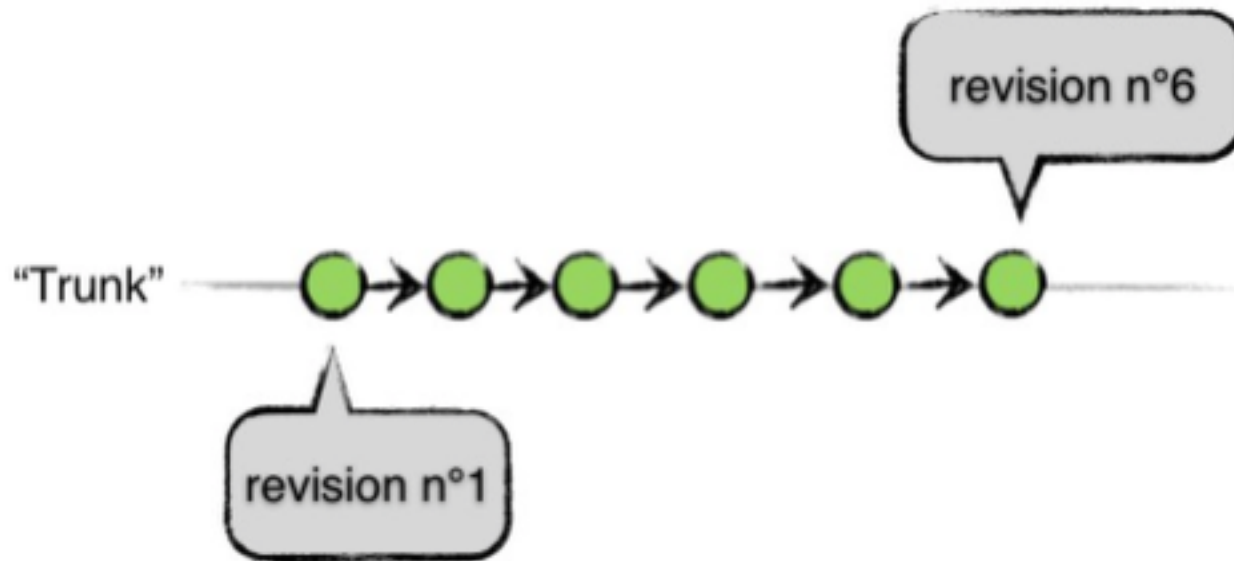
Regular integration of the work of each team can be done



# Version control

## Historic management

Computation of the historic is linear according to the "commit" order

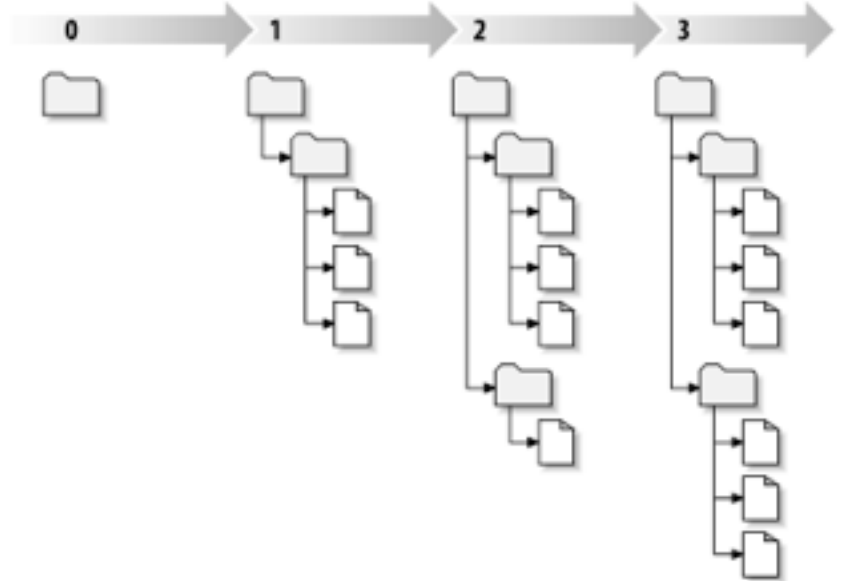


# Version control

## Historic management

SVN assigns a revision number to all the project

Git assigns a revision number per file

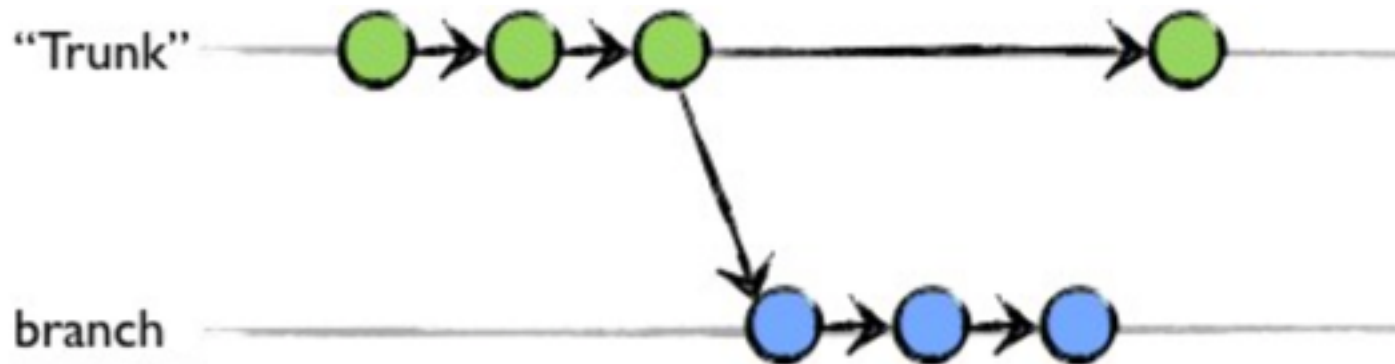


This difference has a strong impact on collaboration

Using branch for collaboration is easier with Git

# Version Control

## Branch management



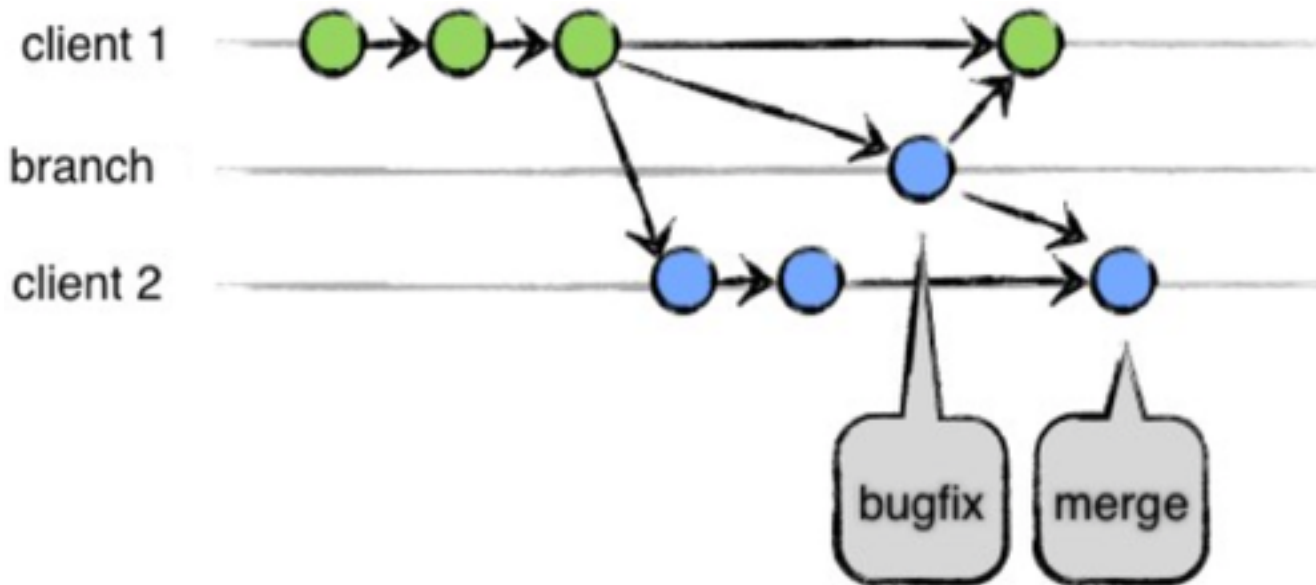
SVN make a copy of the all repository

Git make a link to a particular state of the files

# Version Control

## Branch management

Merging branch (very complex to achieve with SVN)

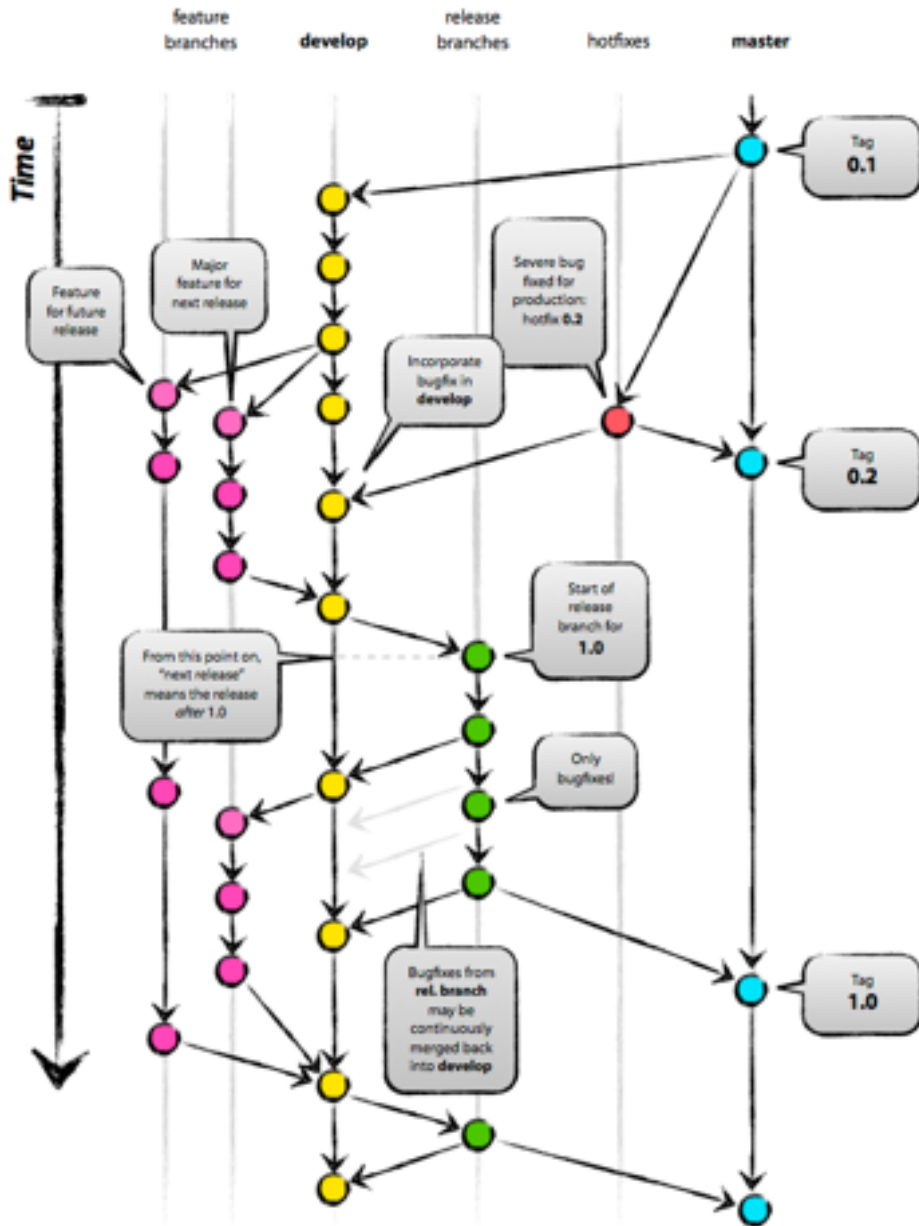


# Version Control

## Branch management

Classical organisation of a project into branches

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

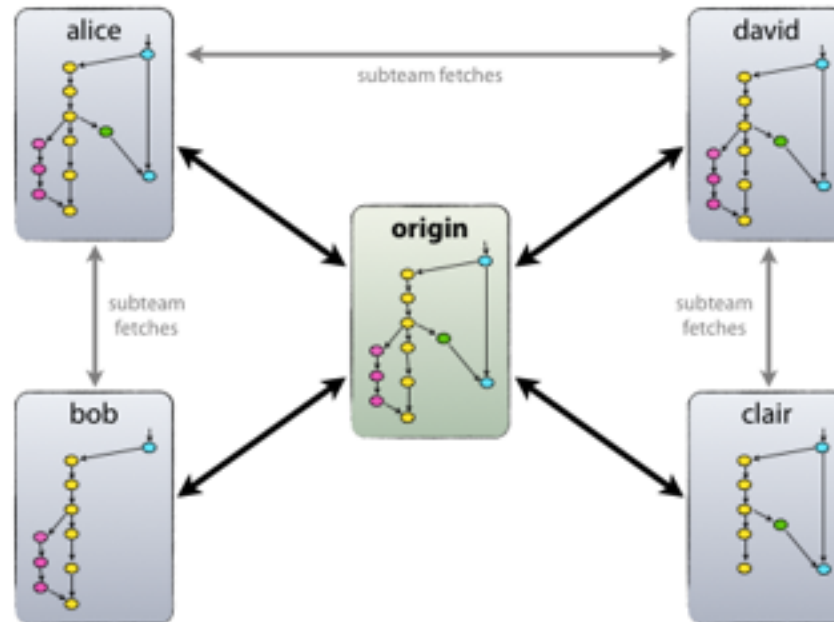


# Version Control

## Branch management

Each user can work on particular branches

Branches can be synchronized between users



Collaborative software development

Version control

**Continuous integration**

Agile methods

# Continuous integration

## Integration

Merging the work of several developers

## Goals

Test modifications from the beginning

Detect integration problems at an early stage

Avoid fastidious integration phases

Always have the system running

Tests, demos, discussion with the customers

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

# Continuous integration

## Principles

Version control on a repository

Automatic and fast build

Auto-testing

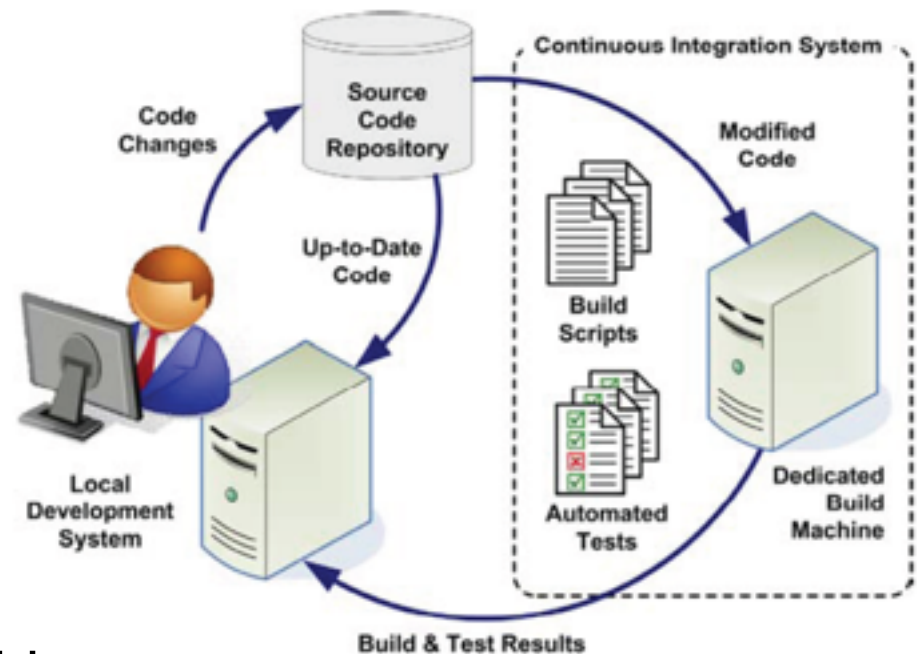
"Commit" every day

Deployment on an integration computer after each "commit"

Automatic deployment

Executable always available

Every body know the build state



# Continuous Integration

## Feedbacks for collaboration

Token on the desk of the person who builds

Make a sound when a build is valid

Web page of the integration server

Bubble light

Wallboard



# Outline

Collaborative software development

Version control

Continuous integration

**Agile methods**

# Software development

## Methods for software development

No methods: "Code and fix"

Efficient for small project

Difficult to add new features or to find bugs

Engineering / plan-driven methodologies

Come from civil or mechanical engineering

Drawing / construction plan / task distribution / construction

Agile methodologies

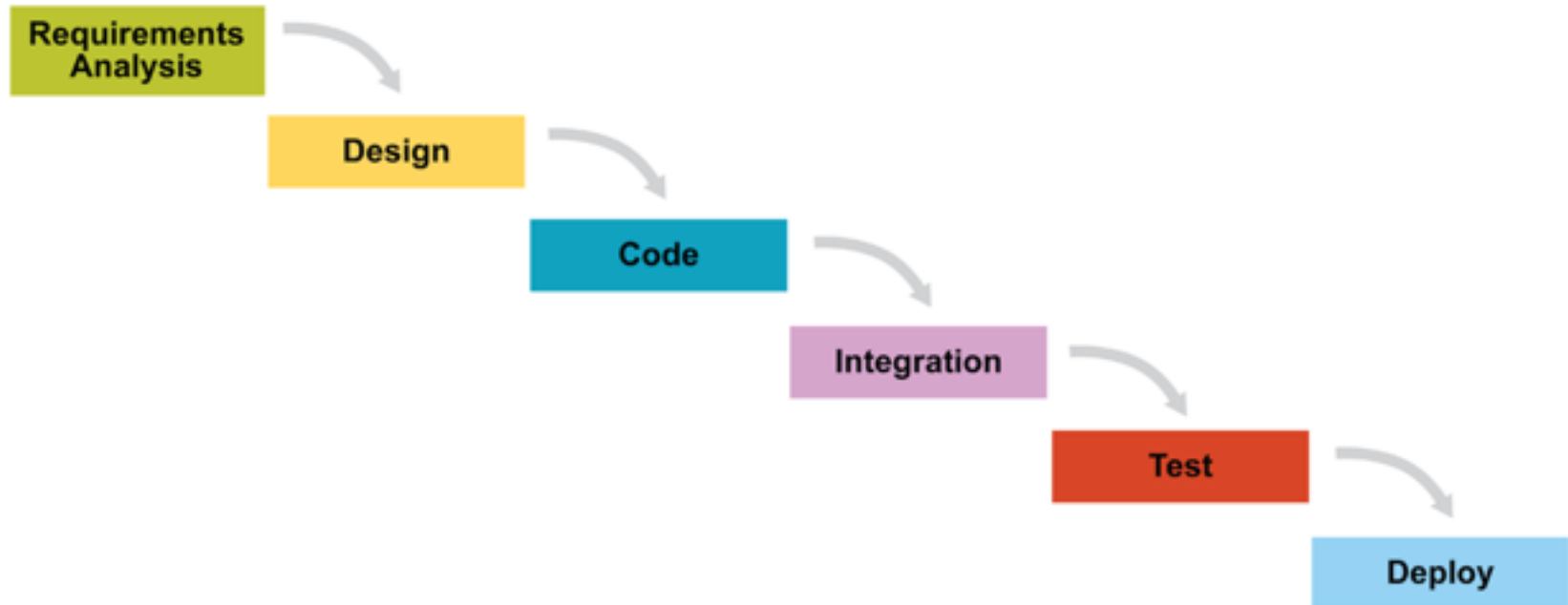
Adaptive rather than predictive

People-oriented rather than process-oriented

# Software development

## Engineering methodologies

### Example: Waterfall



# Software development

## Engineering methodologies

### Separation of design and construction

#### Design

Unpredictable

Require creative people

#### Construction

Predictable

Require people with lower skill

### Example: civil engineering

construction is bigger in cost and time than design

# Software development

Design and Construction for software?

UML = *Design*, coding = *Construction*?

Source code = *Design*, compilation = *Construction*

Construction is quick and cheap

Source code requires creative and talented people

Creative processes are unpredictable

Are the engineering methodologies well adapted ?

[Jack Reeves, C++ Journal, 1992]

<http://www.bleeding-edge.com/Publications/C++Journal/Cpjour2.htm>

# Software development

Is software development predictable?

Yes in some cases...

NASA programs

Usually, requirements are unpredictable

(especially for software involving interactions with users)

Customers don't precisely know what they want

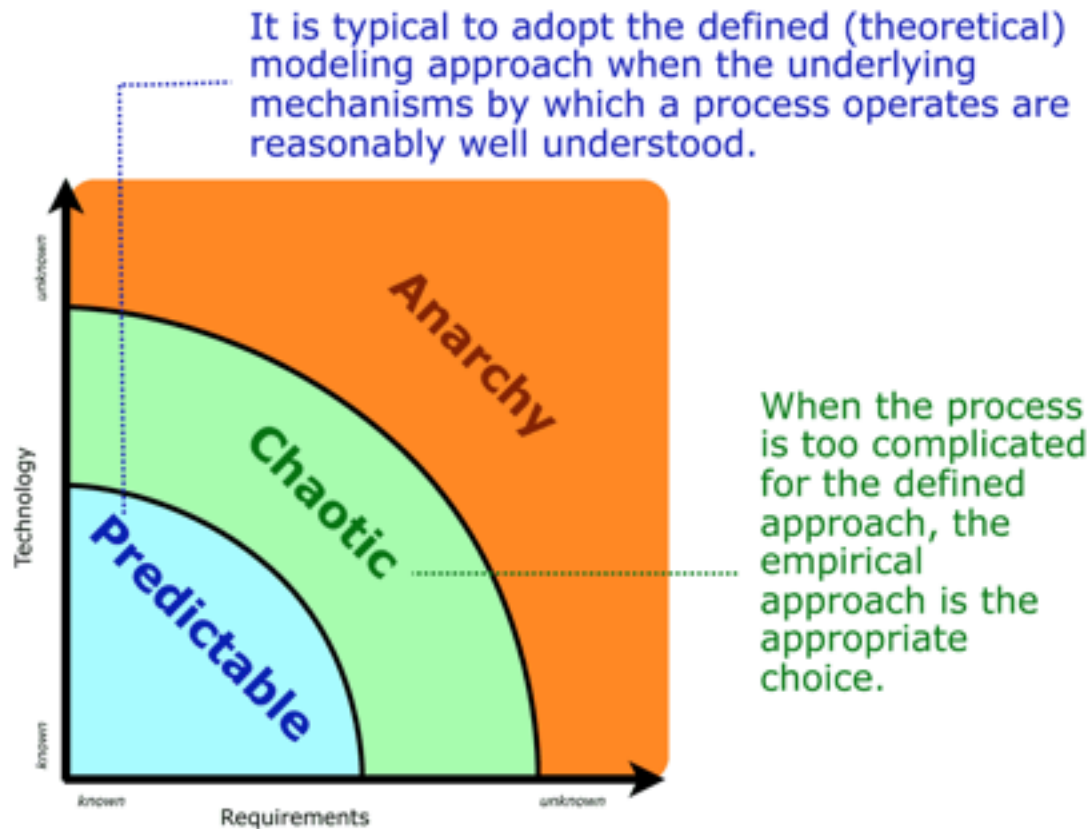
Hard to evaluate the cost of different options

Hard to estimate which features are useful

⇒ Requirements should be flexible in these cases

# Software development

Is software development predictable?



# 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 are replaceable parts

Analysts, coders, testers, managers

Developers are responsible professionals

Make the technical decisions

Evaluate the time required to perform the tasks

# Agile methods

## Manifesto for Agile Software Development

**Individuals and interactions** over processes and tools

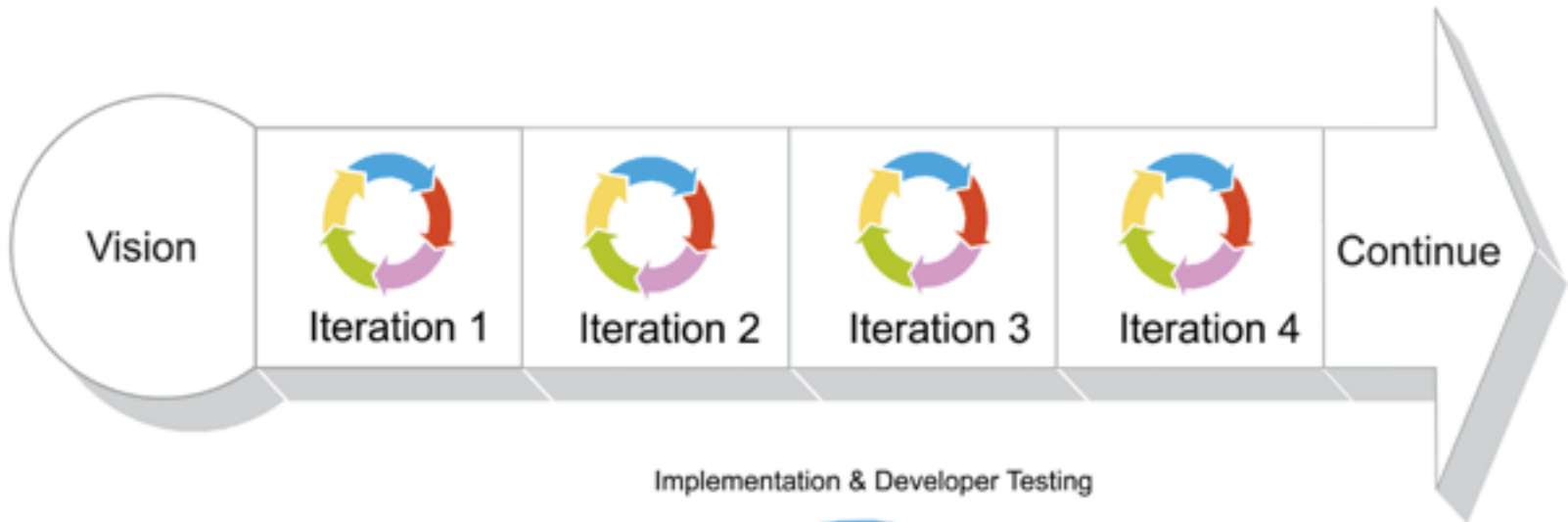
**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

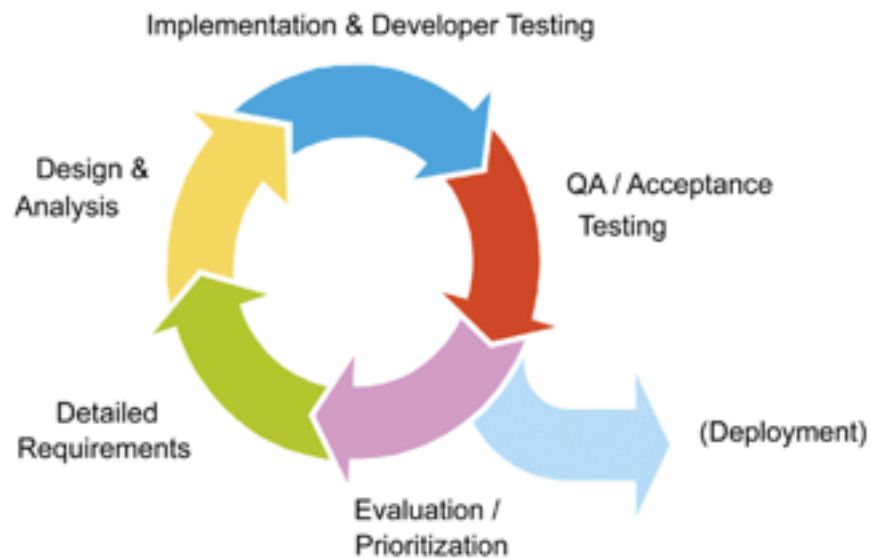
**Responding to change** over following plan

<http://agilemanifesto.org>

# Agile methods



## Iteration Detail



# Agile methods

## Examples

### XP (Extreme Programming)

Test driven development, pair programming

### Scrum

### Crystal

Safety, efficiency, habitability (less discipline than XP)

### Open source process

Distributed contributors, parallelized debugging

### Lean development (Lean @ Toyota)

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

### RUP (Rational Unified Process)

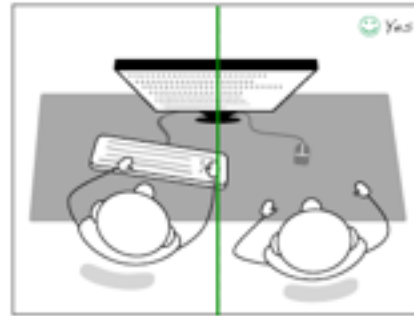
Use case driven, iterative, architecture centric

# Pair programming

Two programmers

One computer

Roles



One "drives": operating mouse and keyboard

Code: syntax, semantics, algorithm

One "navigates": watches, learns, asks, talks, makes suggestions

Higher level of abstraction

Test, technical task, time since the last commit,

Quality of the overall design

# Pair programming

## Advantages

### Code quality

- Better designs

- Fewer bugs

### Spreading Knowledge

- Pairs have to switch off regularly

- Technical and conceptual knowledge

### Social aspects

- No loneliness, conviviality, better motivation

# Pair programming

## Productivity

(it depends on how you measure productivity : lines of code VS running and tested features)

Short-term productivity might decrease slightly (about 15%)

Long-term productivity goes up

Because the code is better

Even better if you consider staff turnover

Truck number in XP

Close as possible to the team size

# Pair programming

## Pairing strategies

In XP, all production code is written by pairs

In non-XP agile teams, usually pairing is not used at all

A trade-off can be found

For some particular tasks

Mentoring new hires

Extremely high-risk tasks

Start of a new project when the design is new

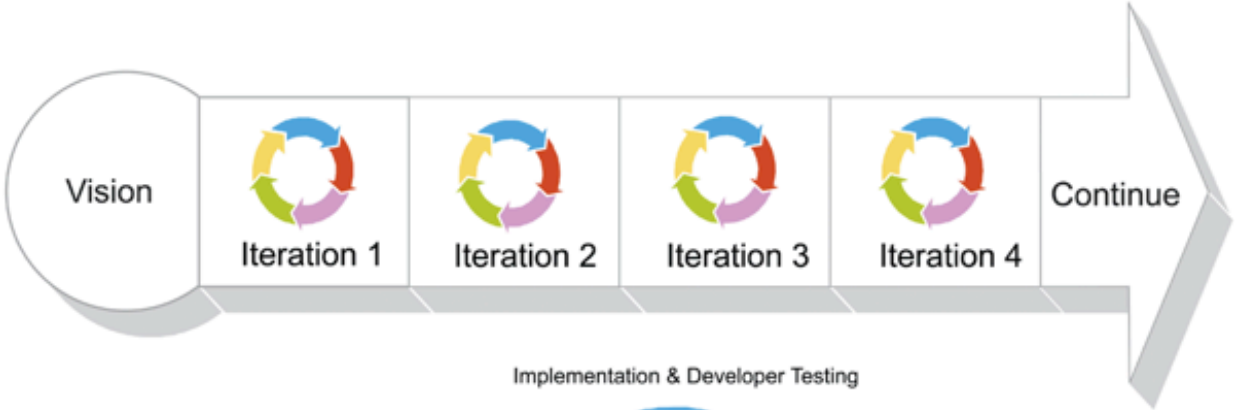
When adopting a new technology

On a rotating monthly or weekly basis

Developers who prefers to pair

# Scrum

Iterations called Sprint (about 1 month)



## Iteration Detail



# Scrum

## Roles

### Product Owner

Single person

Responsible for products vision

Constantly re-prioritizes the Product Backlog

Accepts or rejects product increment



### Development team

Self-organized

Negotiate commitments with the Product Owner

Has autonomy regarding how to reach commitments

Intensely collaborative



### Master

Facilitates the Scrum process

Helps resolve issues

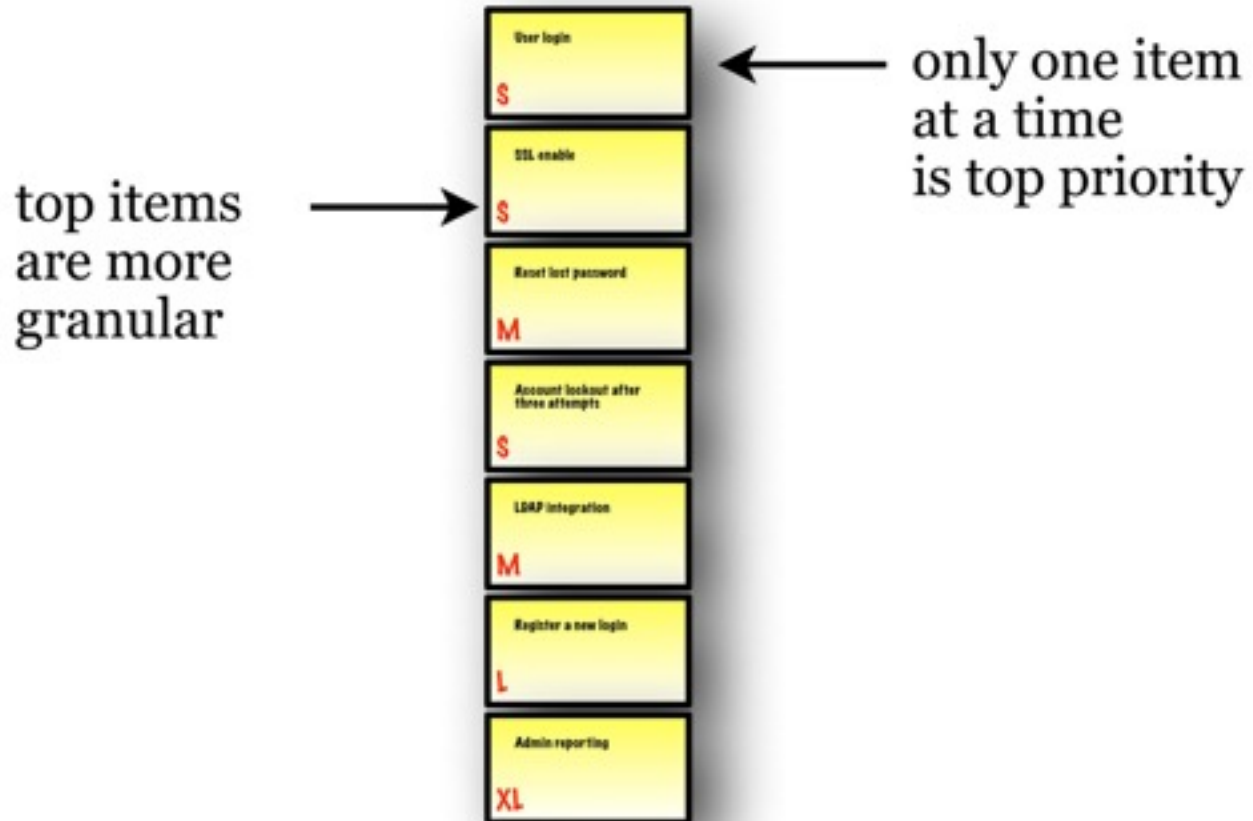
Shields the team from external interferences and distractions

Has no management authority



# Scrum

## Product Backlog



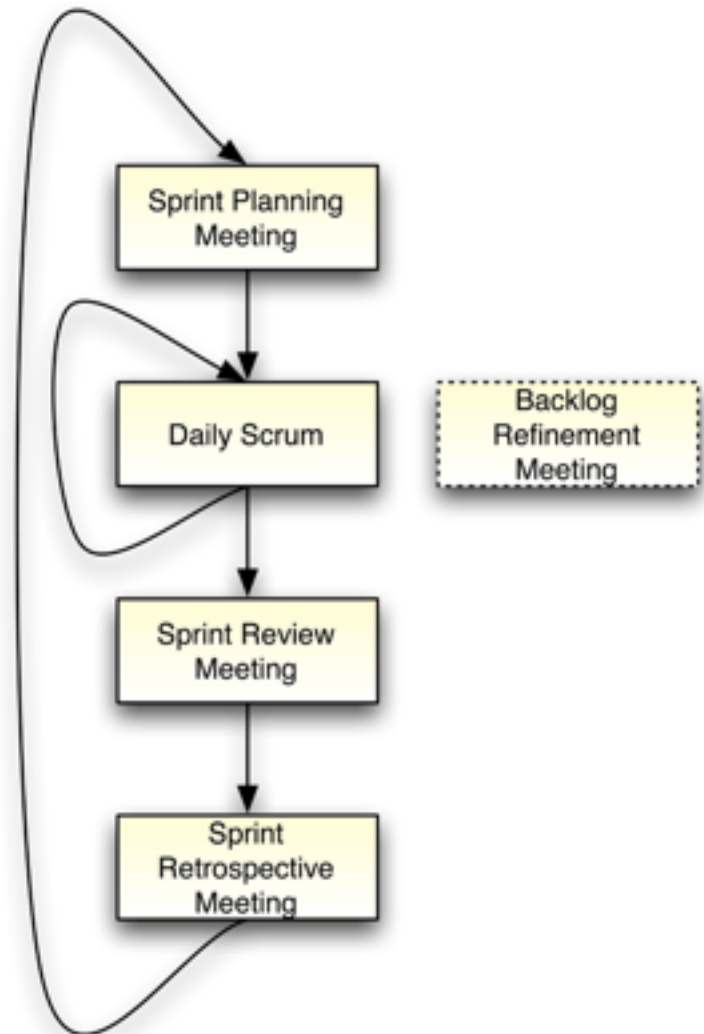
# Scrum

## Sprint

### Planning Meeting

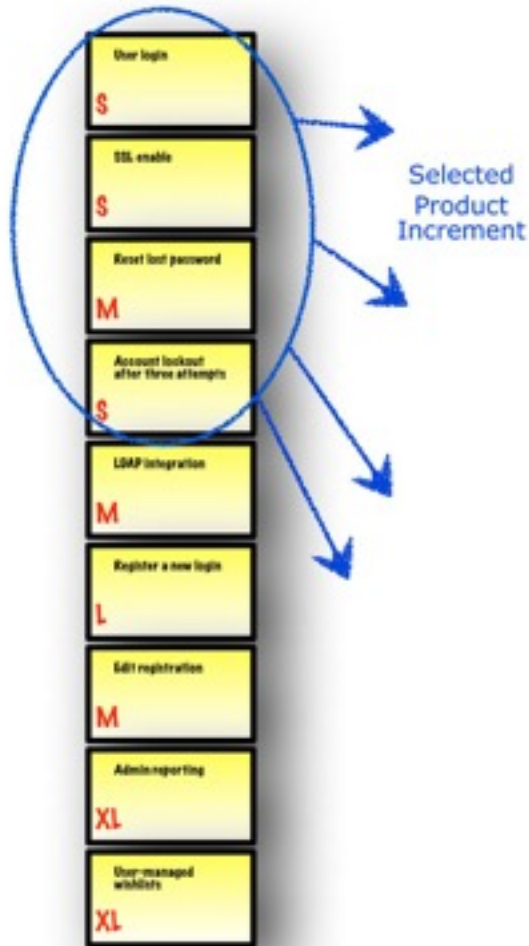
Negotiate which Product Backlog items will be processed

Break items into a list of sprint tasks

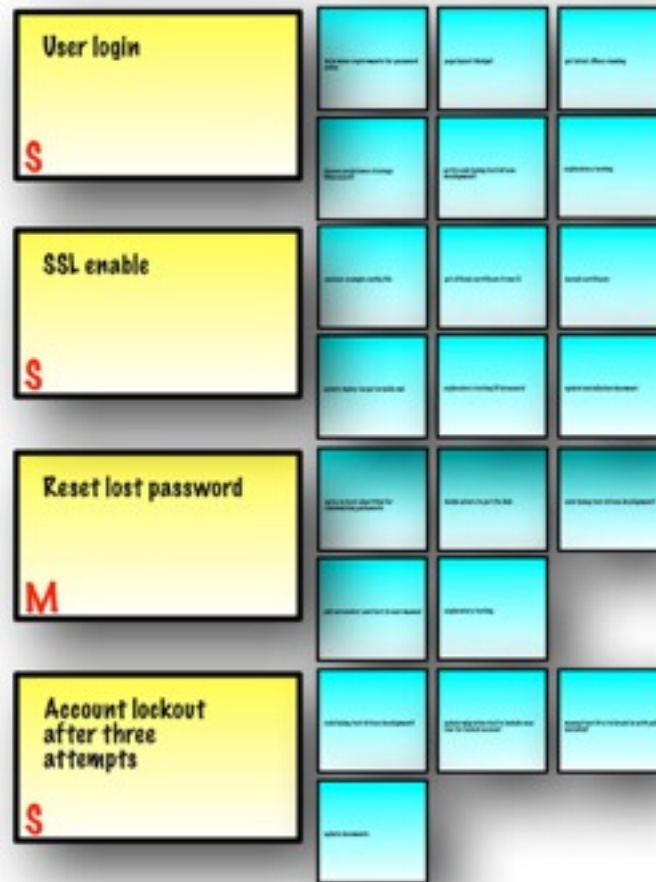


# Scrum

## Product Backlog



## Sprint Backlog



# Scrum

## Sprint

### Planning Meeting

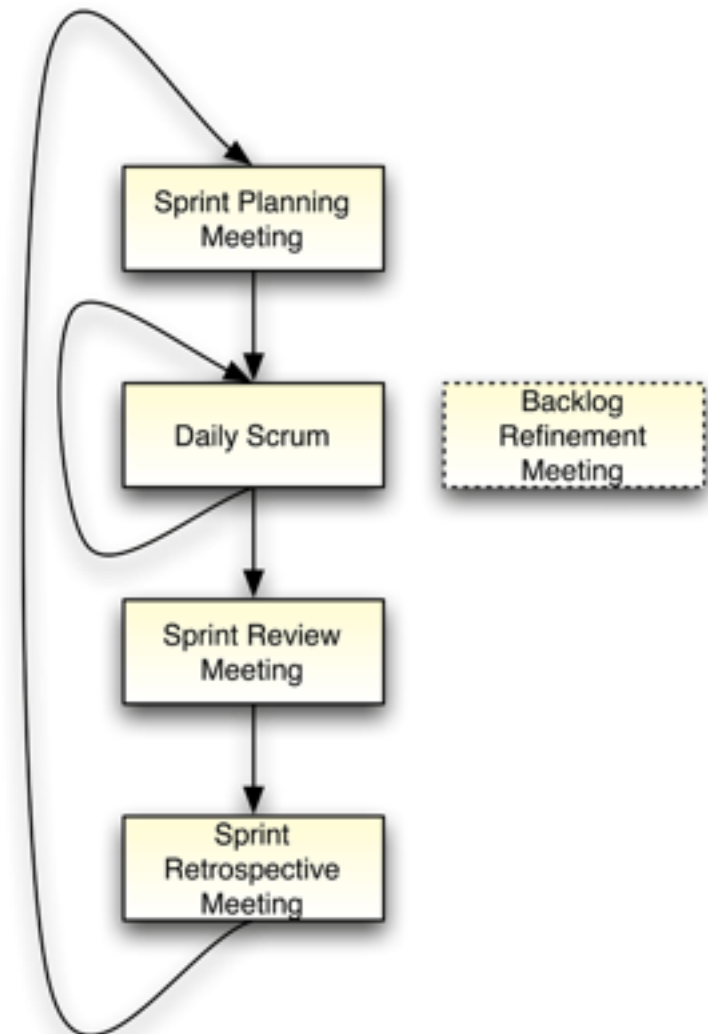
### Daily Meeting

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
	  		 
	  		
	     		
			

# Scrum

## Sprint

Planning Meeting

Daily Meeting

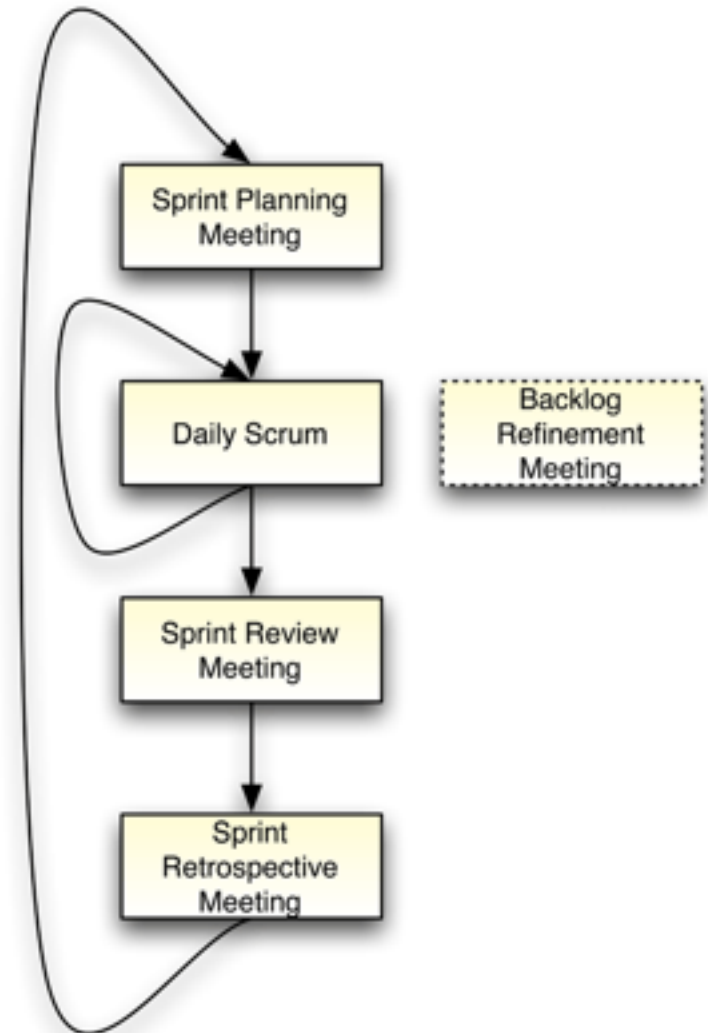
Review Meeting

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



# Scrum

## Sprint

Planning Meeting

Daily Meeting

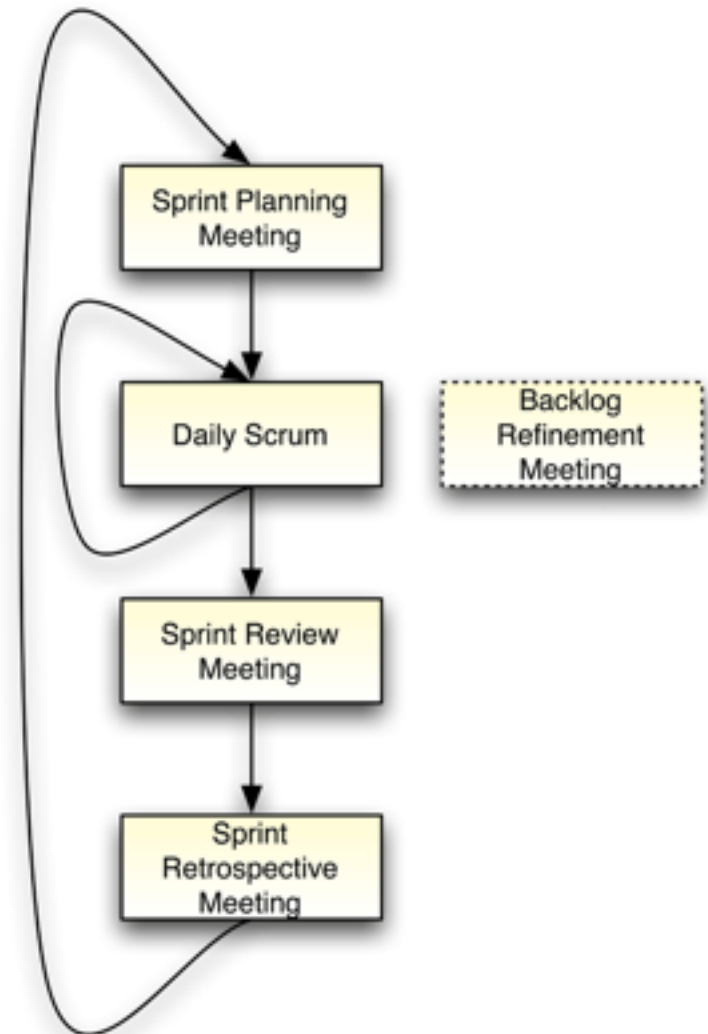
Review Meeting

Retrospective Meeting

Team reviews its own process

Team takes to adapt it for futur Sprints

Master have to manage the psychological safety of the meetings



# Scrum

## Sprint

Planning Meeting

Daily Meeting

Review Meeting

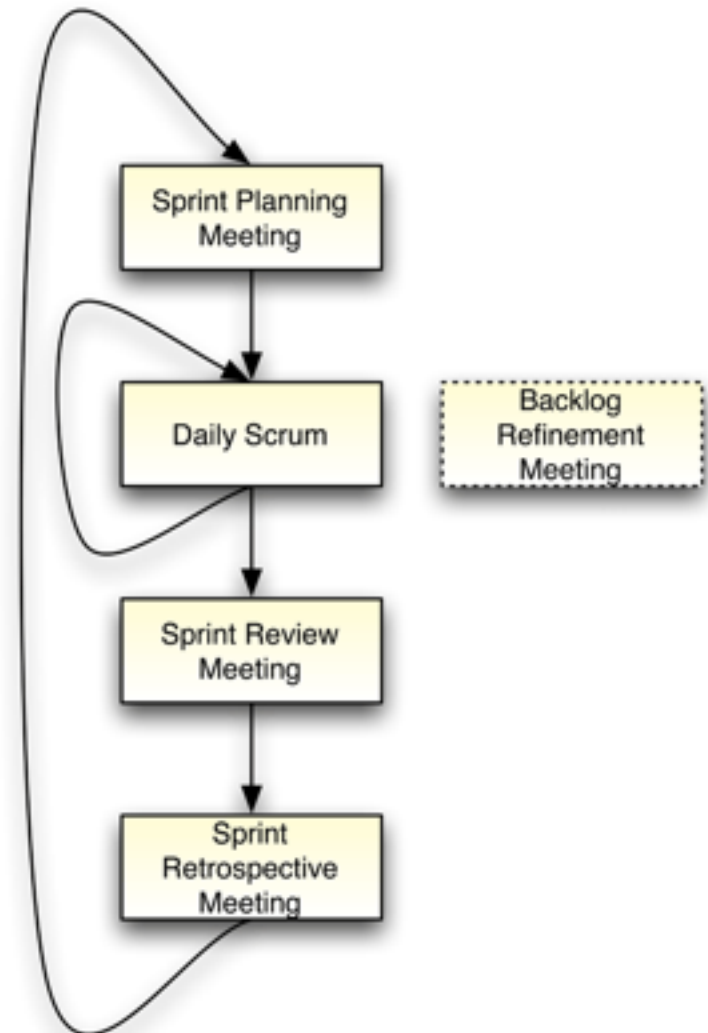
Retrospective Meeting

Backlog Refinement Meeting

Items are usually too large or poorly understood

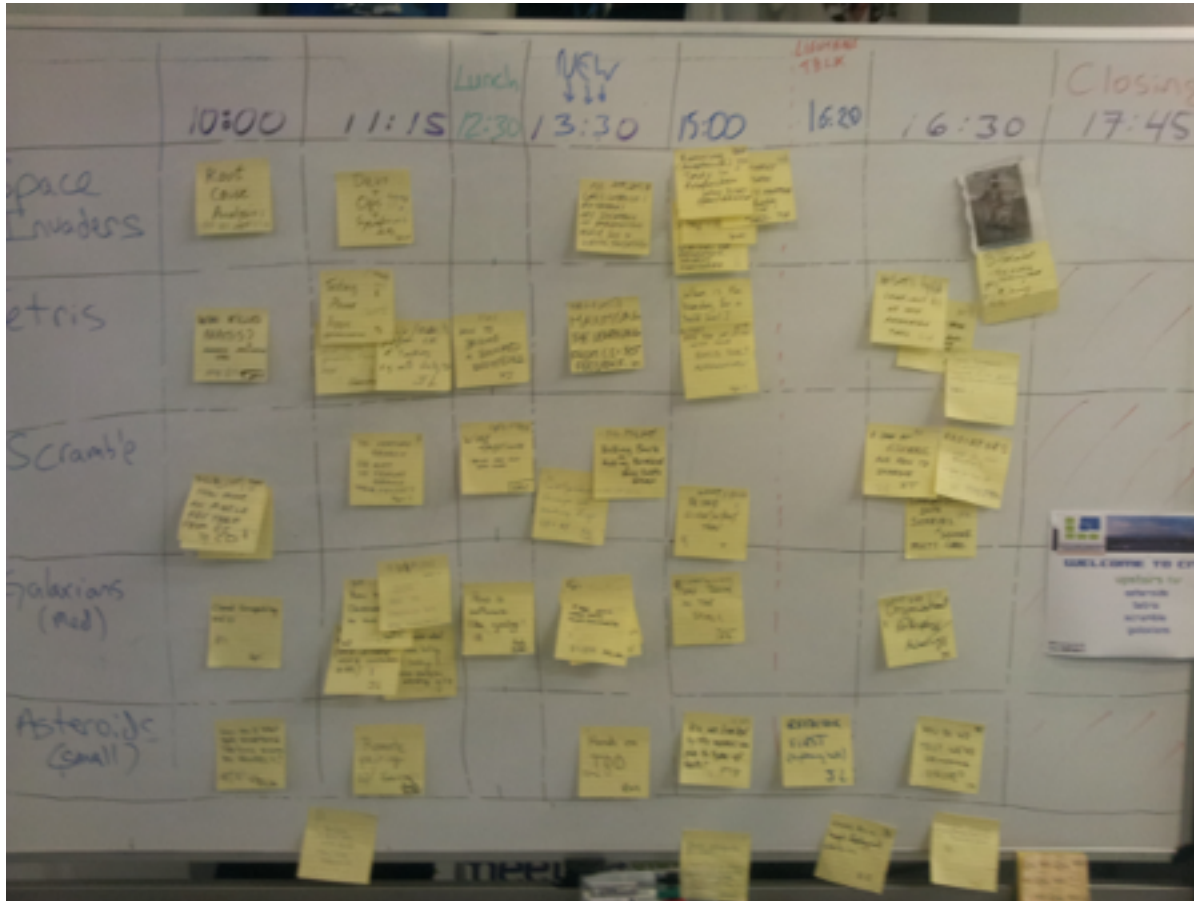
Refine these items into smaller one

Master can help



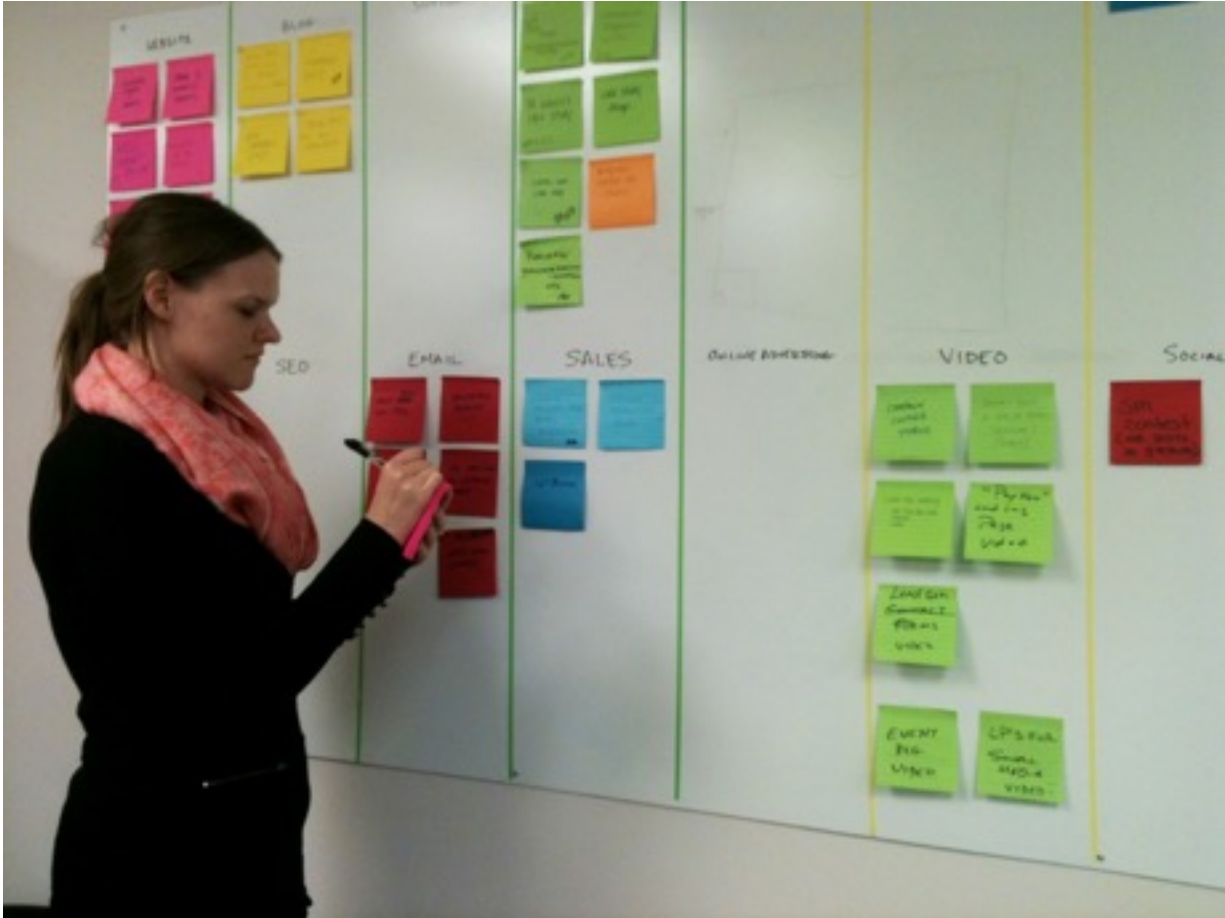
# Scrum

## Feedbacks to the team: wallboard



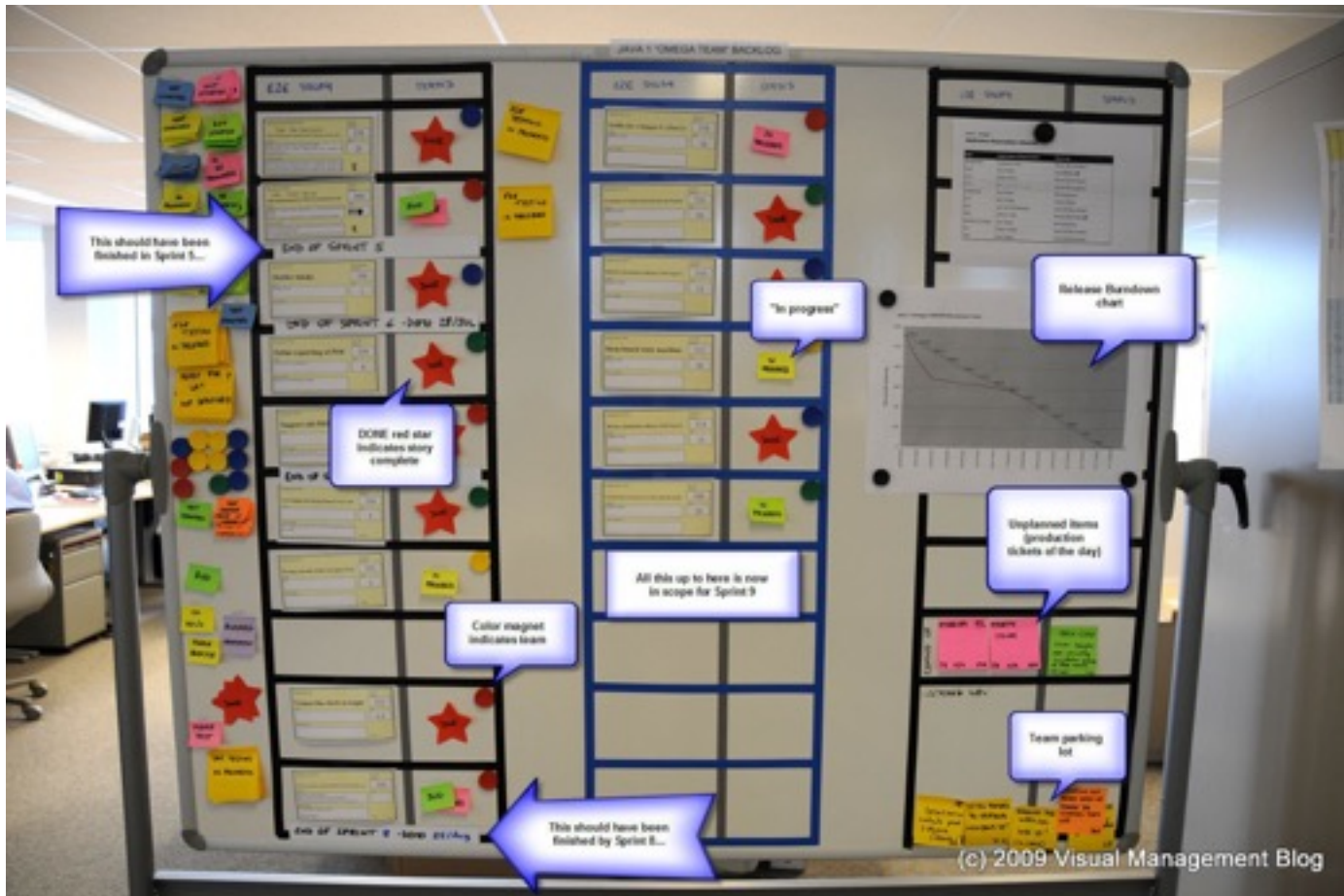
# Scrum

## Feedbacks to the team: wallboard



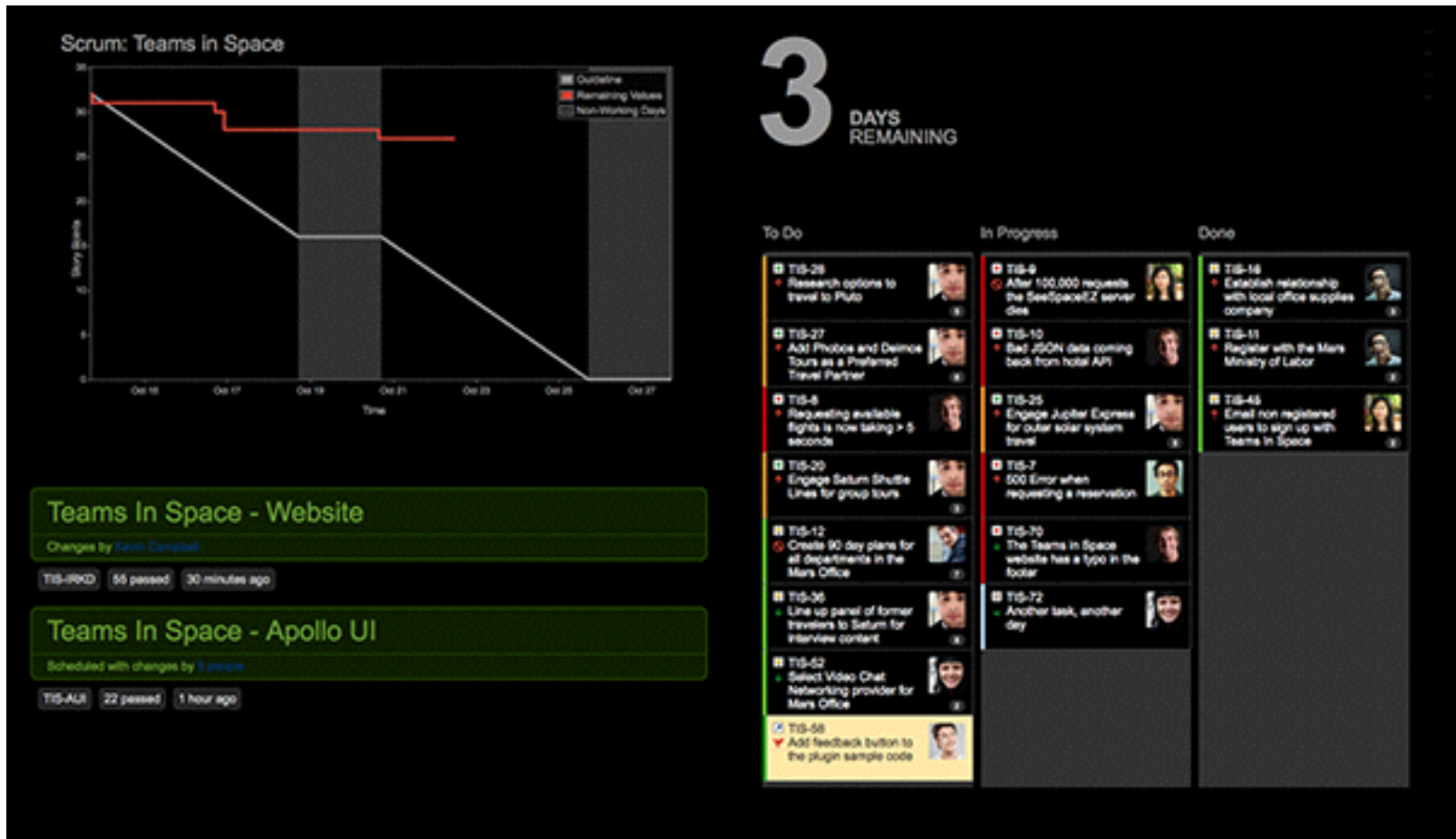
# Scrum

## Feedbacks to the team: wallboard



# Scrum

## Feedbacks to the team: wallboard



# Scrum

## Software to manage Scrum projects

The screenshot displays the JIRA Agile interface for a project named 'Angry Nerds'. The main view is a Scrum board showing a current sprint and two upcoming sprints. The current sprint, 'Angry Sprint', is in progress and has 8 days left. It contains 8 issues, with the most prominent one being 'NERD-4: As an Outsourcer I want to get paid for working in my pyjamas' with an estimate of 20. Below the current sprint are two upcoming sprints: 'Upcoming Sprint 1' with 5 issues and an estimate of 29, and 'Upcoming Sprint 2' with 3 issues and an estimate of 9. On the right side, a detailed view of the 'NERD-4' issue is shown, including its status 'Not Started', a session 'Confon they are in pyjamas' with a 'Created' status, and a 'Create Session' button.

<http://youtu.be/KdyV9okLRlc>

# Conclusion

## Collaboration in software development

- Is necessary for big projects

- Is not obvious

  - Technical, organizational and social aspects

## Version control

- Synchronization, versioning

- Branching: split work between users

# Conclusion

## Continuous integration

Improve safety and efficiency

## Agile method

Organize the team

Propose 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>