

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

Continuous integration

Software development methods

Outline

Collaborative software development

Version control

Continuous integration

Software development methods

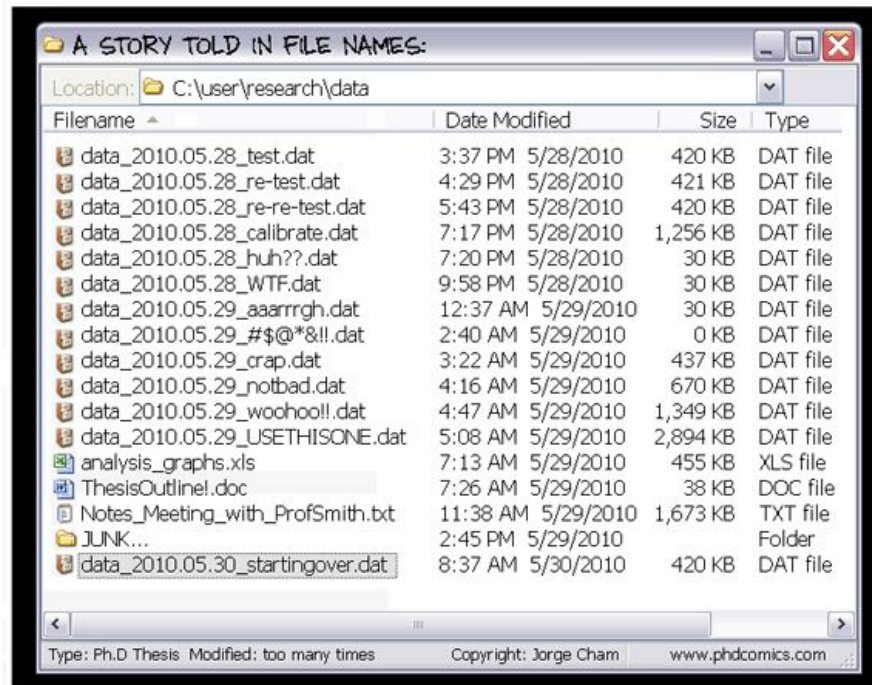
Why version control?

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]

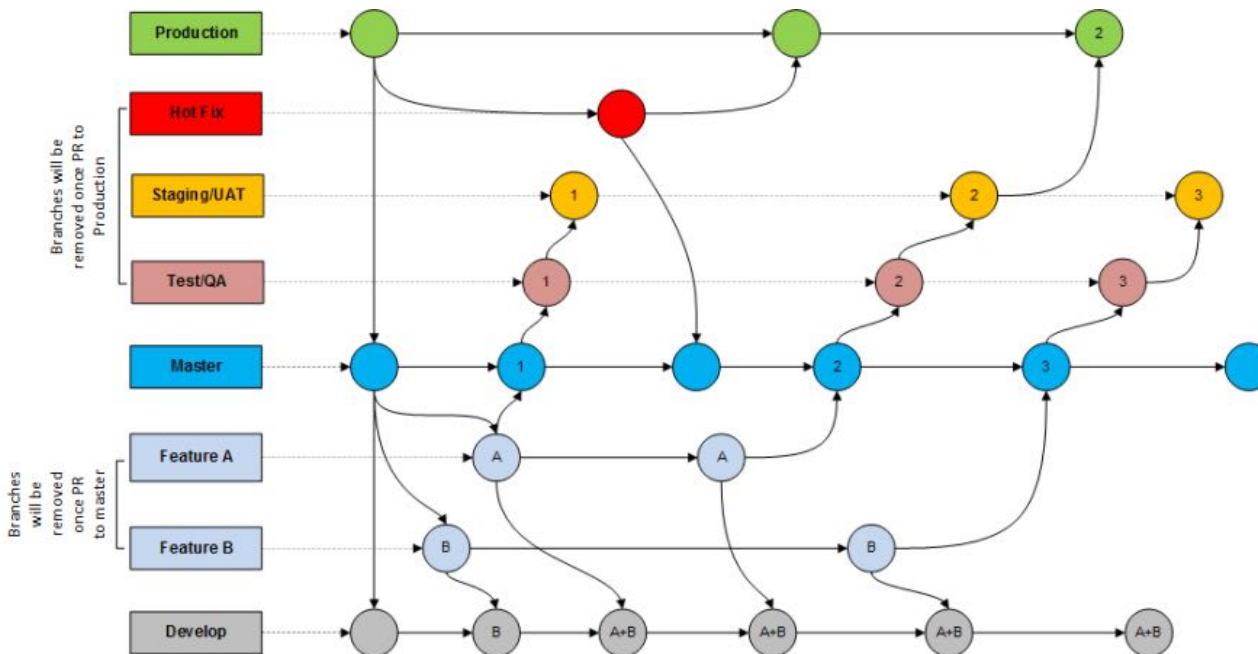
Why version control?

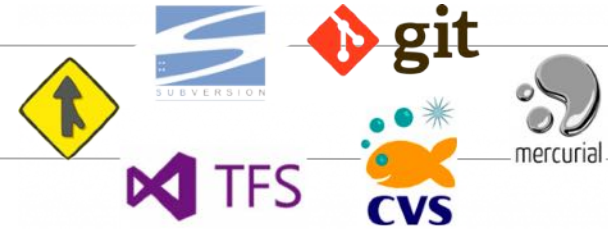
We want to be able to:

Edit the project at the same time, and merge our work

Keep an history of the modifications, restore old ones

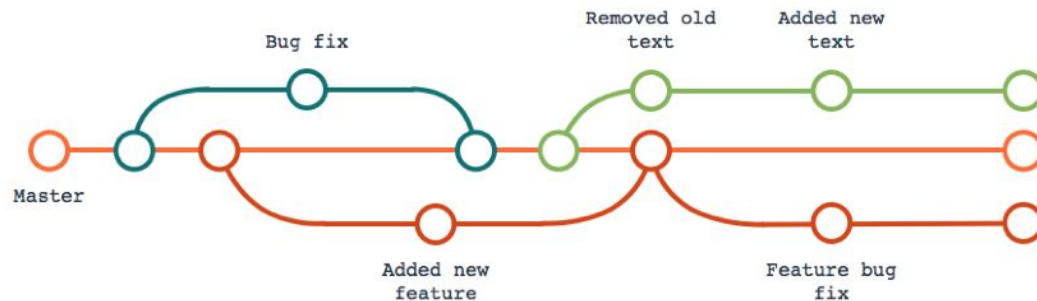
Keep older versions of the files





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

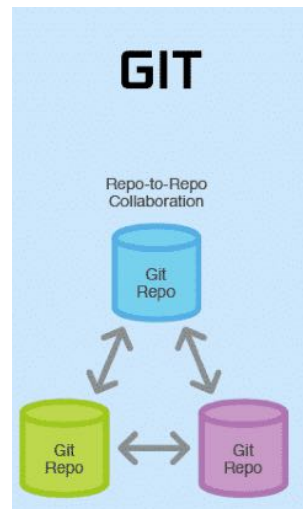
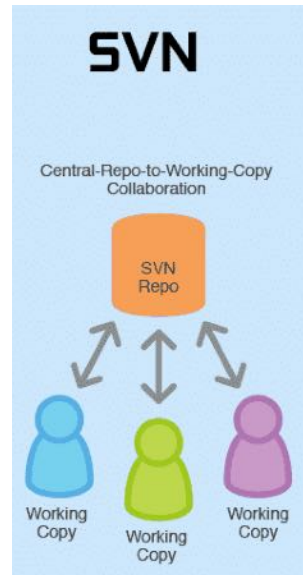
CVS, **SVN**, TFVC, ...

Decentralized (peer-to-peer):

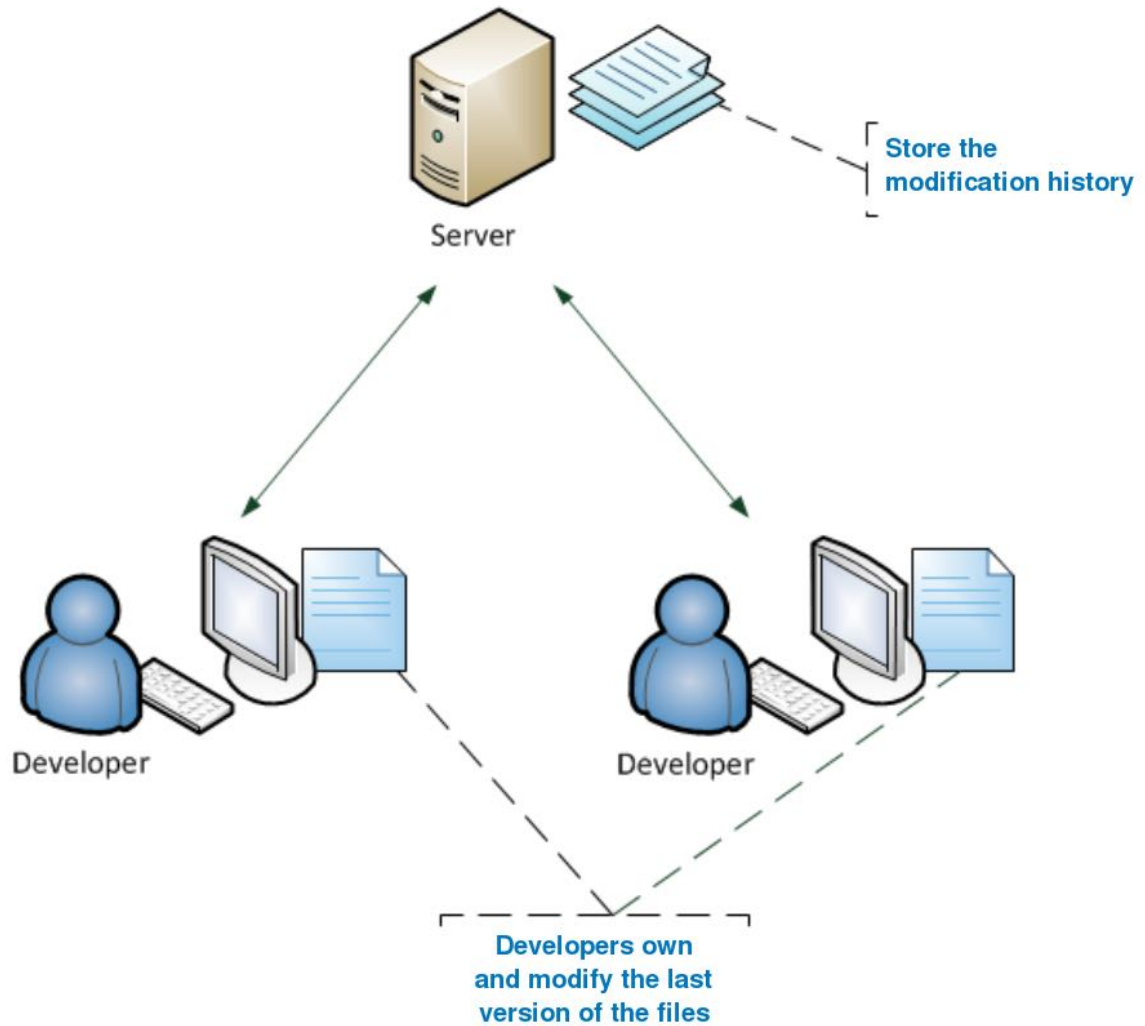
GNU Arch, Mercurial, Bazaar, **Git**,...

Decentralized can be used as a Hybrid Architecture

One peer can be a central server (github)



Centralized architecture



Drawbacks of centralized architecture

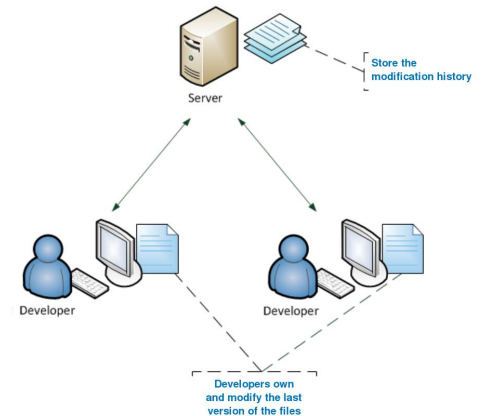
Single access point to the data

Single communication point between users

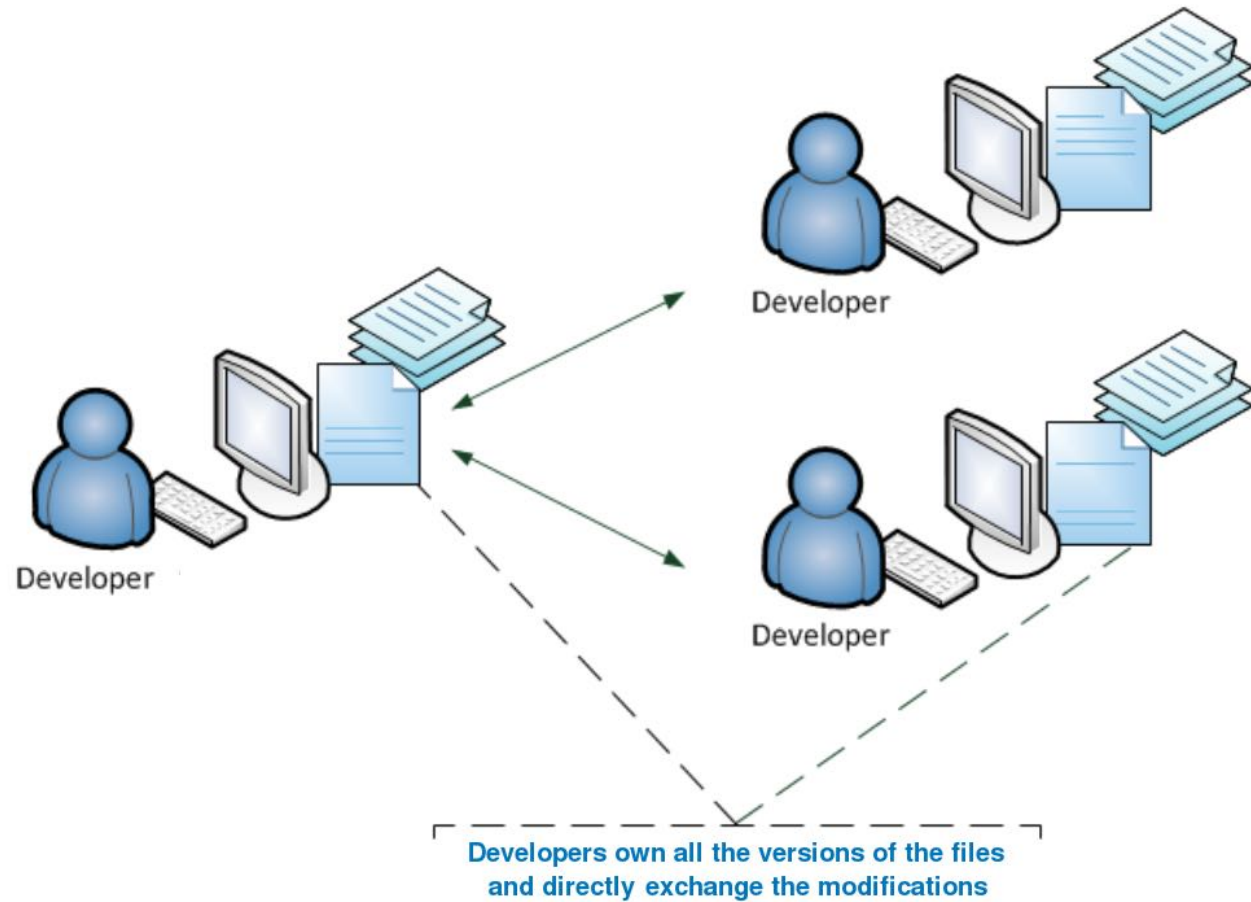
Single history / timeline of the files

Versioning and sharing are the same operation

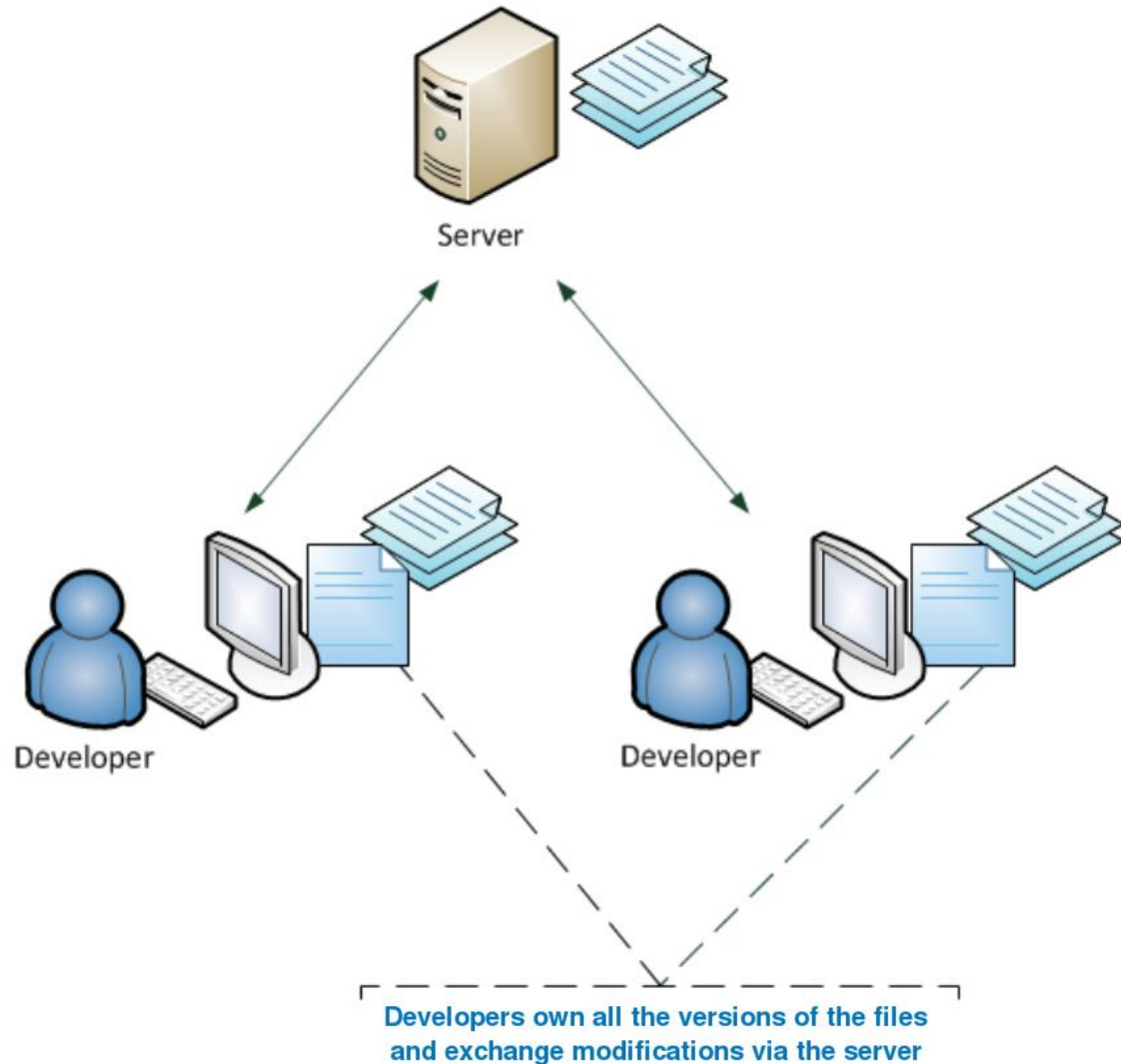
Need to have a stable state before “committing” a change



Decentralized architecture



Hybrid architecture



Vocabulary (SVN)

Architecture

Repository

Working copy

Actions

Checkout

Add

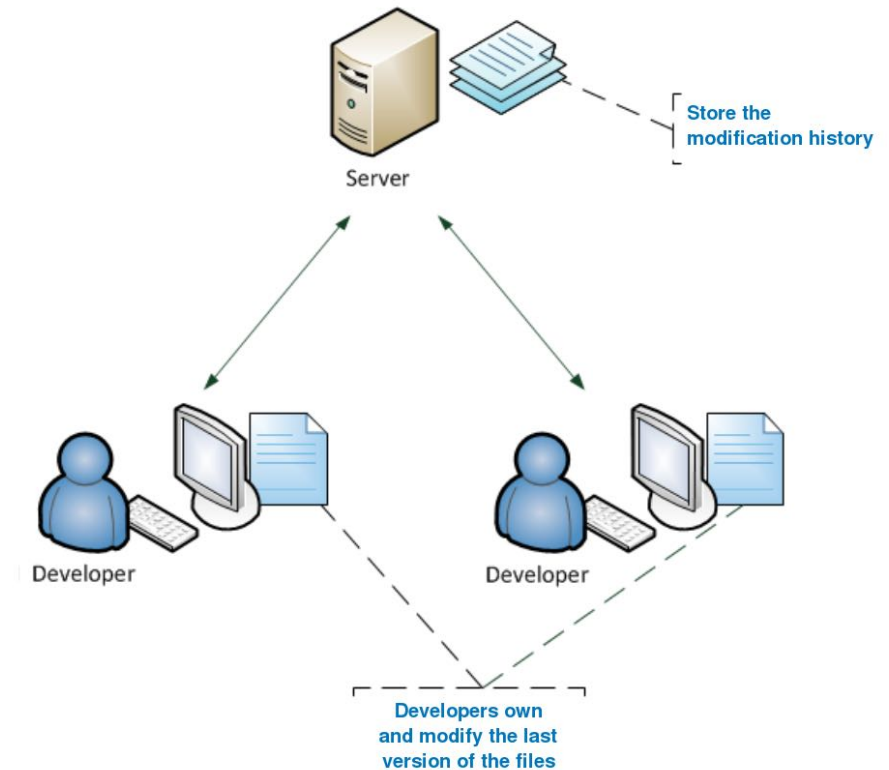
Commit

Update

Revert

Diff, log, status

Centralized Architecture



Vocabulary (git)

Architecture

Remote and local repository

Working copy

Actions

Clone

Add

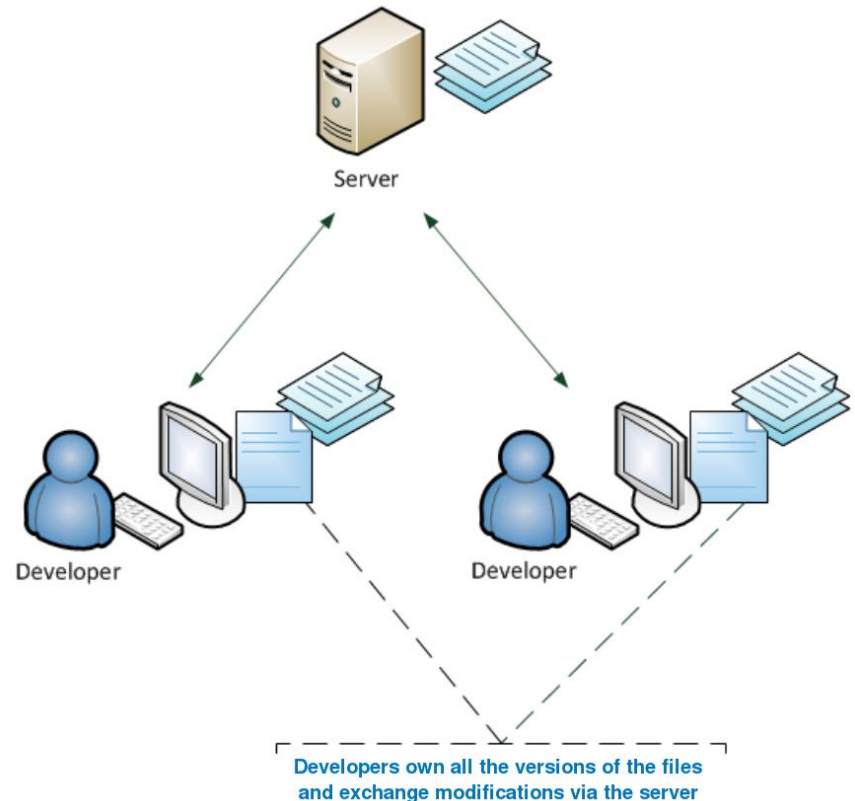
Commit

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 some

- Check if the code compiles with the updates

Commit the change to the repository

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 to 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
```

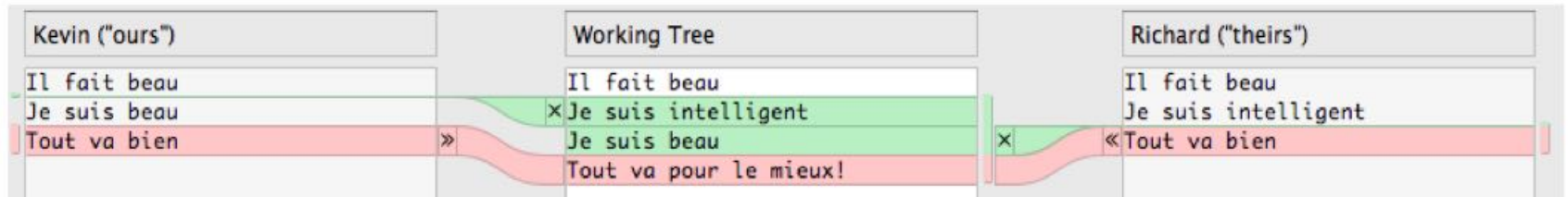
```
08/10/2010  11:44 AM          94 test.txt
08/10/2010  11:44 AM          26 test.txt.mine
08/10/2010  11:44 AM          27 test.txt.r2
08/10/2010  11:44 AM          31 test.txt.r3
```

test.txt

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

Version control

Tools for conflict management (SmartGit)



Three-way merge

SmartSynchronize 3.4.12 (3-Way Merge)

File Edit View Go To Window Help

Save All Reload Prev. Change Next Change Take Left Take Right Left + Merge All Merge + Right Merge Below

C:\temp\a\SdPrivateKeyCredential-master.java

```

34 private SdPrivateKeyCredential(ScPasswo
35 super(passwordSet);
36
37 this.file = file;
38
39 }
40
41 // Accessing =====
42
43 public File getFile() {
44 return file;
45 }
46
47 public void setFile(File file) {
48 this.file = file;
49 }
50
51 @NotNull
52 public ScPassword getScPassword() {
53 passphrase = maybeCreateInstance(pa
54 return passphrase;
55 }
56
57 public boolean isValid() {
58 return file != null;
59 }
60
61 public boolean matches(File file) {
62 return QCompareUtils.areEqual(file,
63 }
64
65 // Description =====
66
67 @Override
68 public void toDescription(@NotNull QDes
69 QDescriptionUtils.putFile(writer, "
70 putPassword(writer, "passphrase", p
71 }
72
73 @Override
74 public void fromDescription(@NotNull QD
75 file = QDescriptionUtils.getFile(re
76 passphrase = getPassword(reader, "p
77 }
78 }
79
                
```

C:\temp\a\SdPrivateKeyCredential-common.java

```

42
43 public File getFile() {
44 return file;
45 }
46
47 public void setFile(File file) {
48 this.file = file;
49 }
50
51 public boolean isPassphraseKnown() {
52 passphrase = notNull(passphrase);
53 return passphrase.isKnown();
54 }
55
56 public boolean isPassphraseStored() {
57 passphrase = notNull(passphrase);
58 return passphrase.isStored();
59 }
60
61 public boolean isValid() {
62 return file != null;
63 }
64
65 @Nullable
66 public String getPassphrase(@Nullable S
67 passphrase = notNull(passphrase);
68 return passphrase.getPhrase(userInt
69 }
70
71 public void setPassphrase(String passph
72 final ScPassword scPassword = notNU
73 scPassword.setPassphrase(passphrase
74 this.passphrase = scPassword;
75 }
76
77 public boolean matches(File file) {
78 return QCompareUtils.areEqual(file,
79 }
80
81 // Description =====
82
83 @Override
84 public void toDescription(@NotNull QDes
85 QDescriptionUtils.putFile(writer, "
86 putPassword(writer, "passphrase", p
87 }
                
```

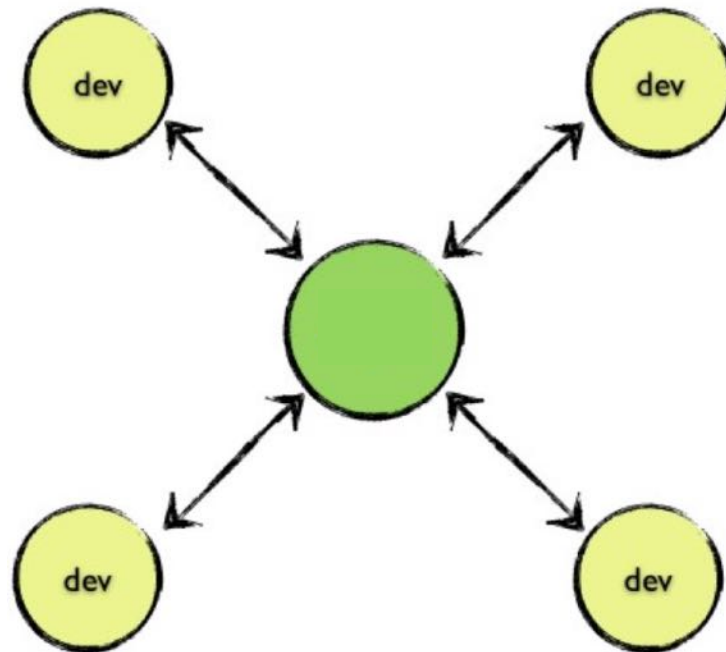
C:\temp\a\SdPrivateKeyCredential-branch.java

```

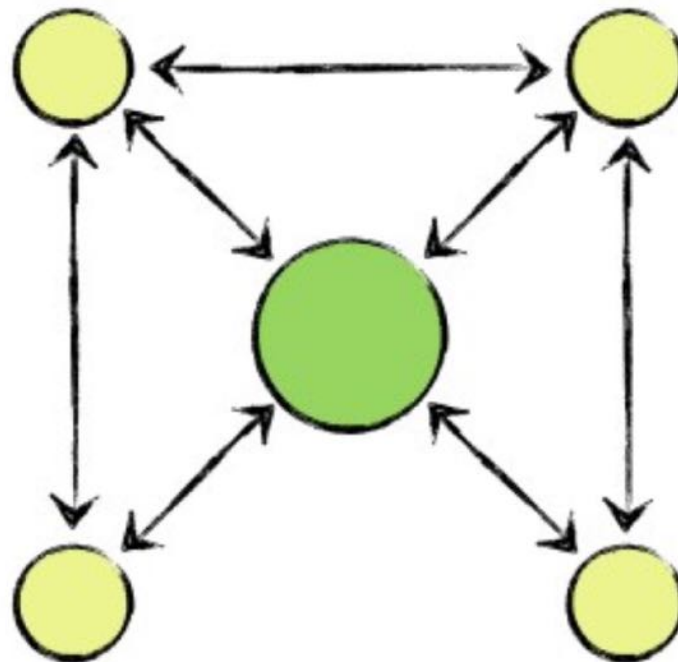
42 public synchronized File getFile() {
43 return file;
44 }
45
46
47 public synchronized void setFile(File f
48 this.file = file;
49 }
50
51 public synchronized boolean isPassphas
52 passphrase = notNull(passphrase);
53 return passphrase.isKnown();
54 }
55
56 public synchronized boolean isPassphas
57 passphrase = notNull(passphrase);
58 return passphrase.isStored();
59 }
60
61 public synchronized boolean isValid() {
62 return file != null;
63 }
64
65 @Nullable
66 public String getPassphrase(@Nullable S
67 synchronized (this) {
68 passphrase = notNull(passphrase
69 }
70 return passphrase.getPhrase(userInt
71 }
72 public void setPassphrase(String passph
73 synchronized (this) {
74 this.passphrase = notNull(this.
75 }
76 }
77 this.passphrase.setPassphrase(passph
78 }
79
80 public synchronized boolean matches(Fil
81 return QCompareUtils.areEqual(file,
82 }
83
84 // Description =====
85
86 @Override
87 public synchronized void toDescription(
                
```

Ready

Collaboration scenario: centralized (SVN)



Collaboration scenario: decentralized (Git)

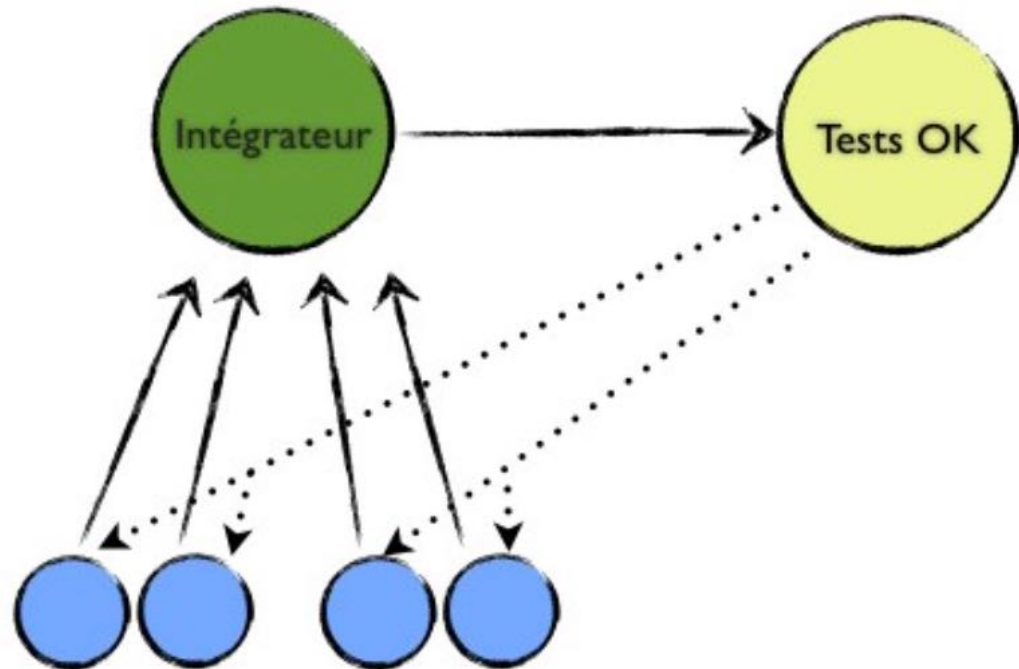


Inter-personal communications

Collaboration scenario: decentralized (Git)

Integration mode

A repository is in charge of the test

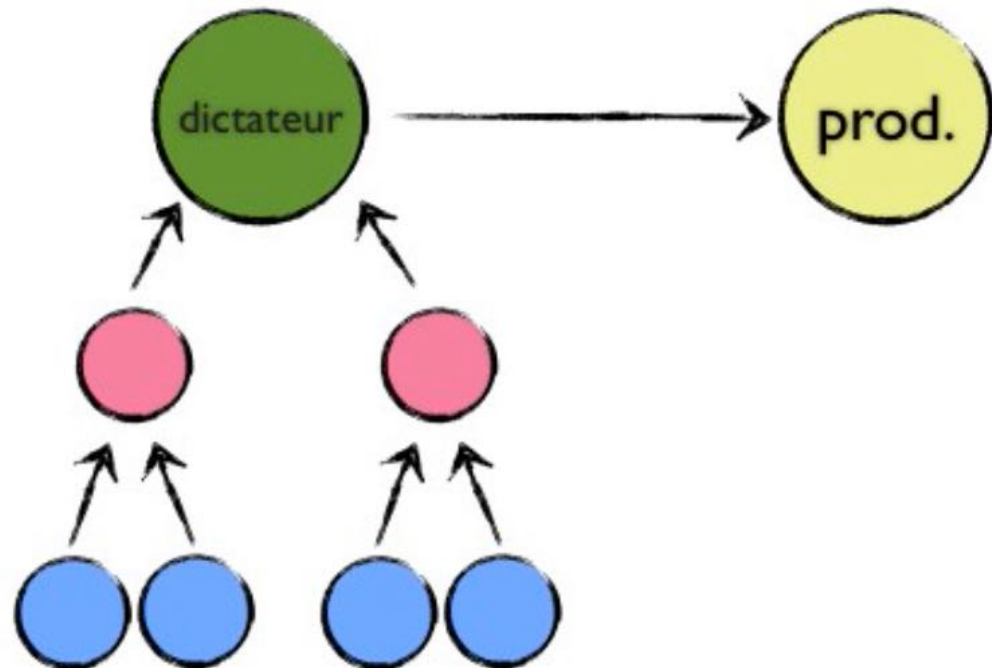


Collaboration scenario: decentralized (Git)

Dictator mode

Open-source projects

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

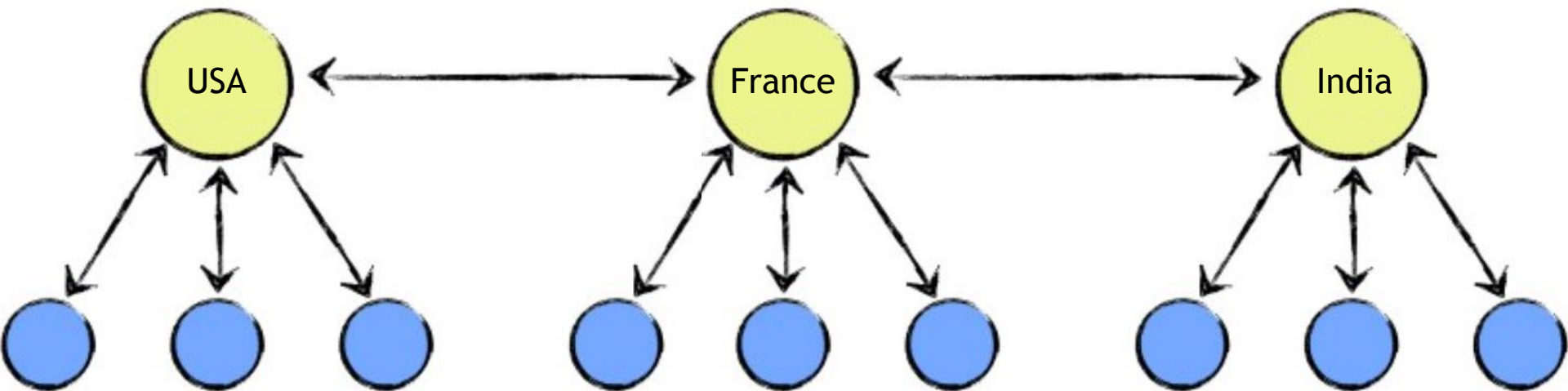


Collaboration scenario: decentralized (Git)

Multi-location teams

Each team can work independently

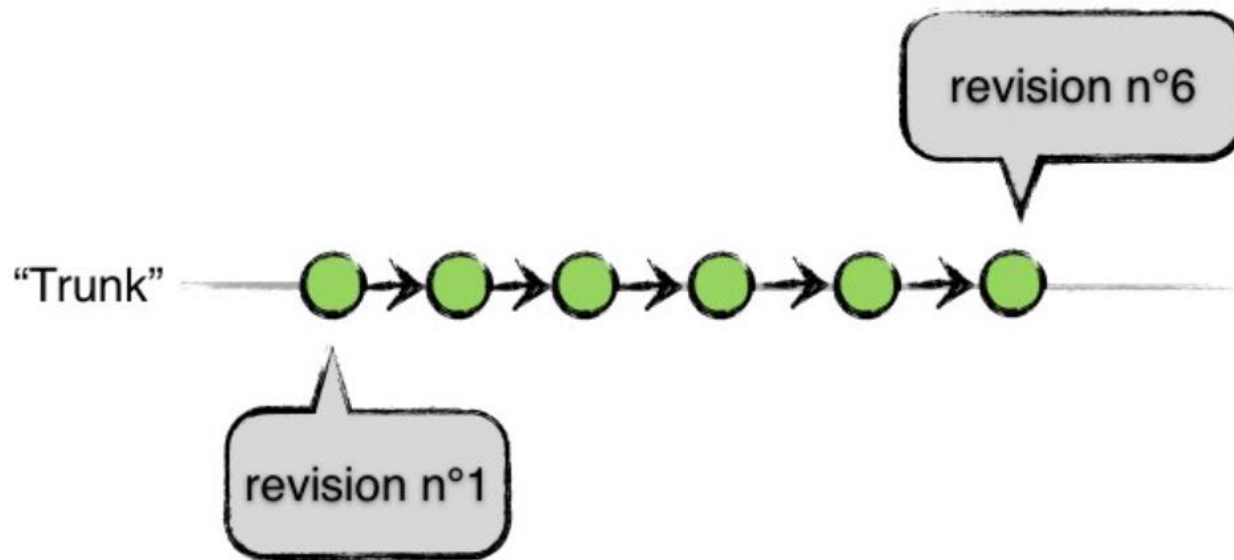
Regular integration of the work of each team



History management

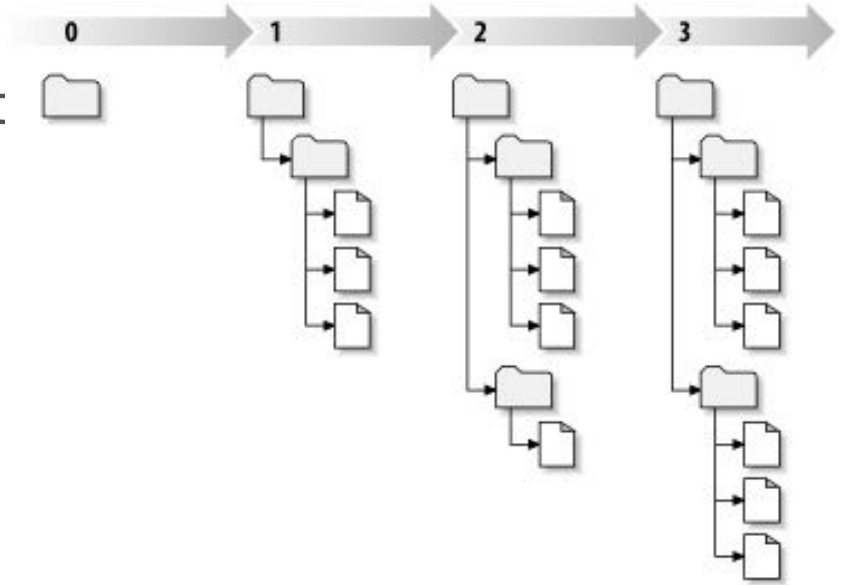
The repository stores differences between text files

Computation of the history is linear
when considering the order of “commits”



History management

SVN assigns a revision number to the entire project

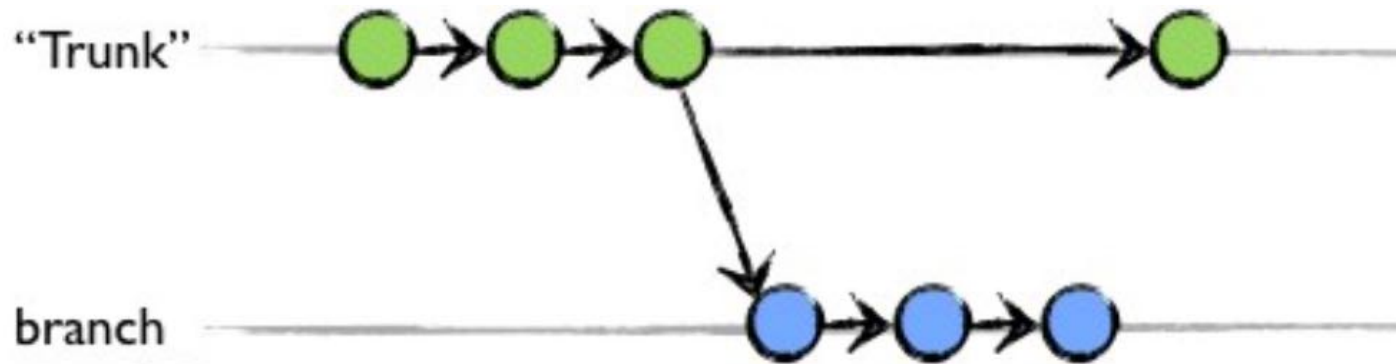


Git assigns a revision number per file

This difference impacts collaboration

Using branch for collaboration is easier with Git

Branch management

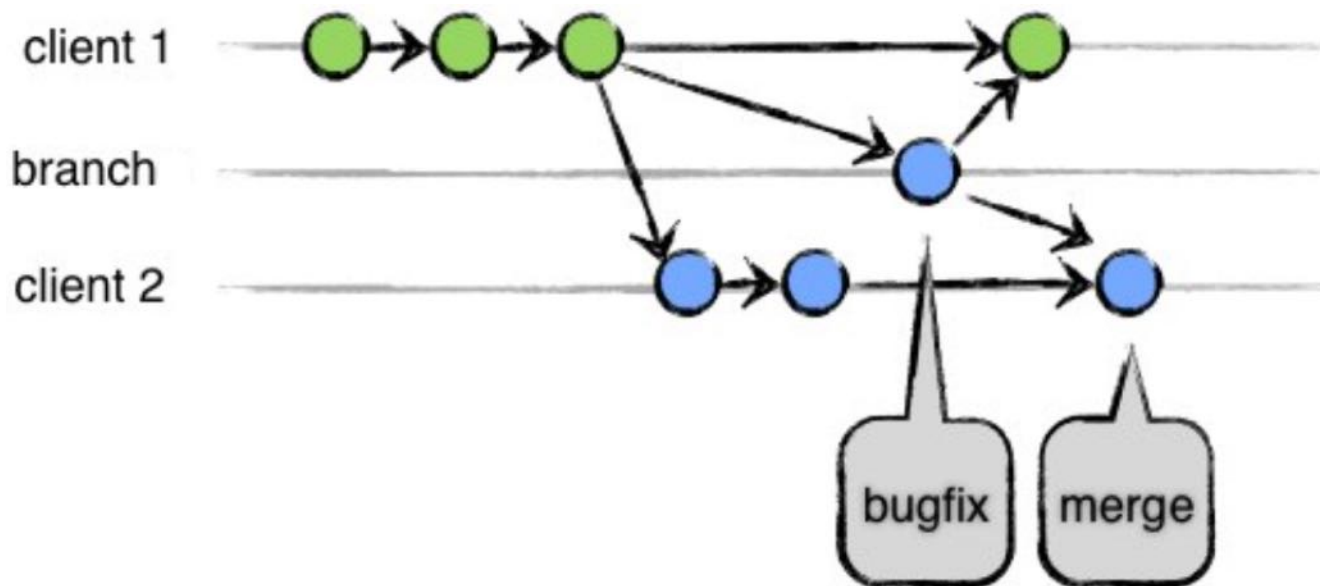


SVN makes a copy of the entire repository

Git makes a link to a particular state of the files

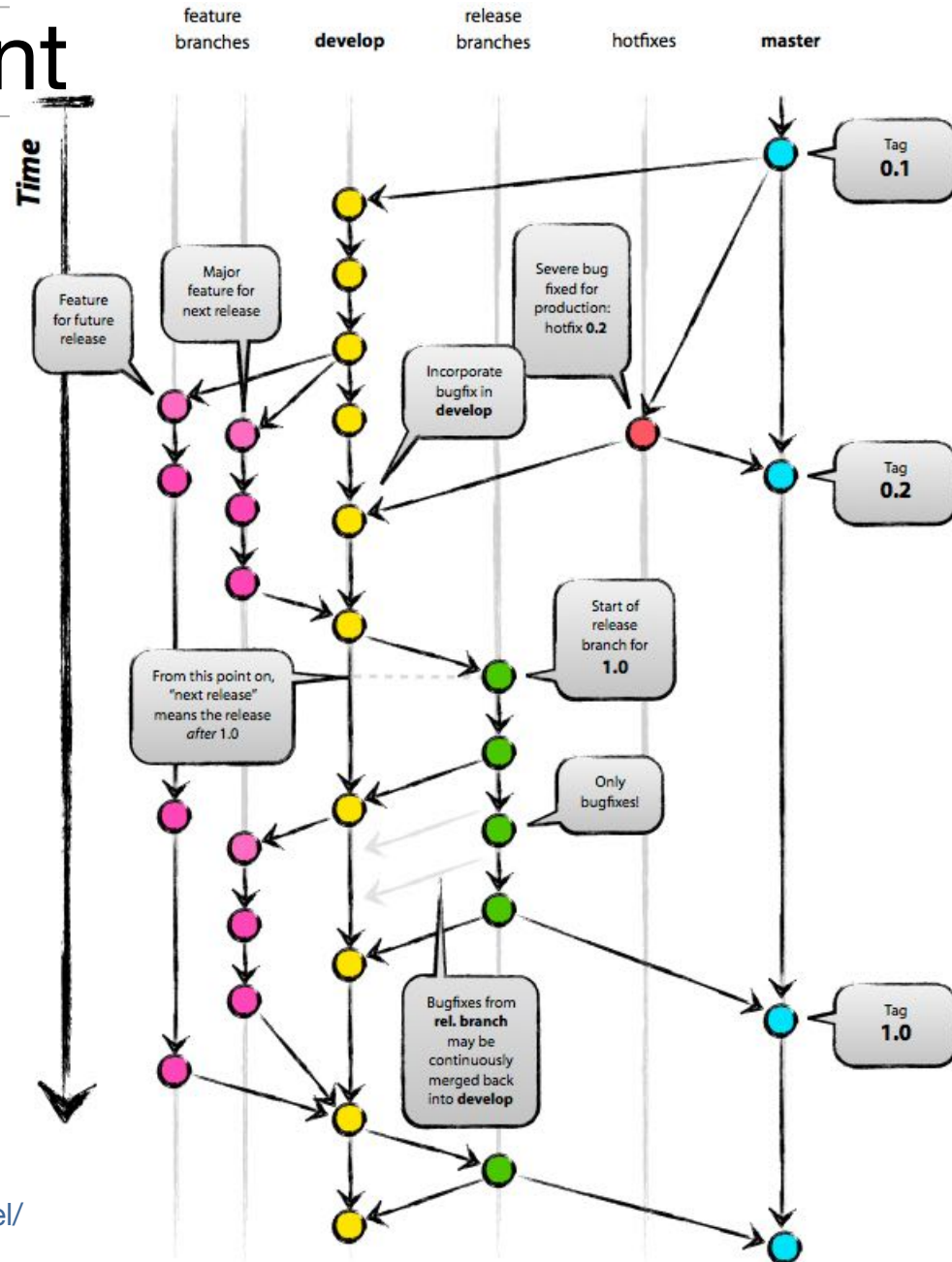
Branch management

Merging branch (very complex to achieve with SVN)



Branch management

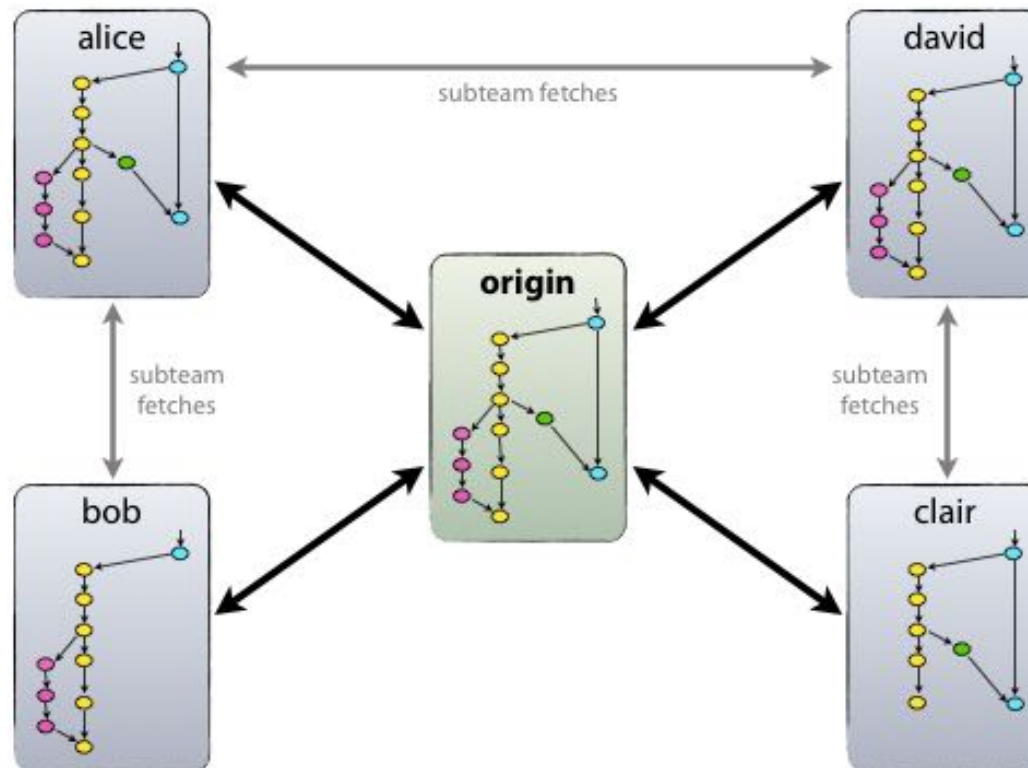
Classical organisation of a project into branches



Branch management

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

Branches can be synchronized between users



Web-based interfaces

Manage user access rights

Manage branches and access to branches

Review modifications and different versions

Track bugs

Create wiki / web pages for projects

Add social network functionalities



GitHub

Outline

Collaborative software development

Version control

Continuous integration

Software development methods

Continuous integration

Integration

Continuous merging & testing the work of several developers

Automatic deployment

System always running

Goals

Test modifications from the beginning

Detect integration problems at an early stage

Always have the system running

Tests, demos, discussion with the customers

Principles of continuous integration

Version control in a repository

Automatic and fast build

Auto-testing

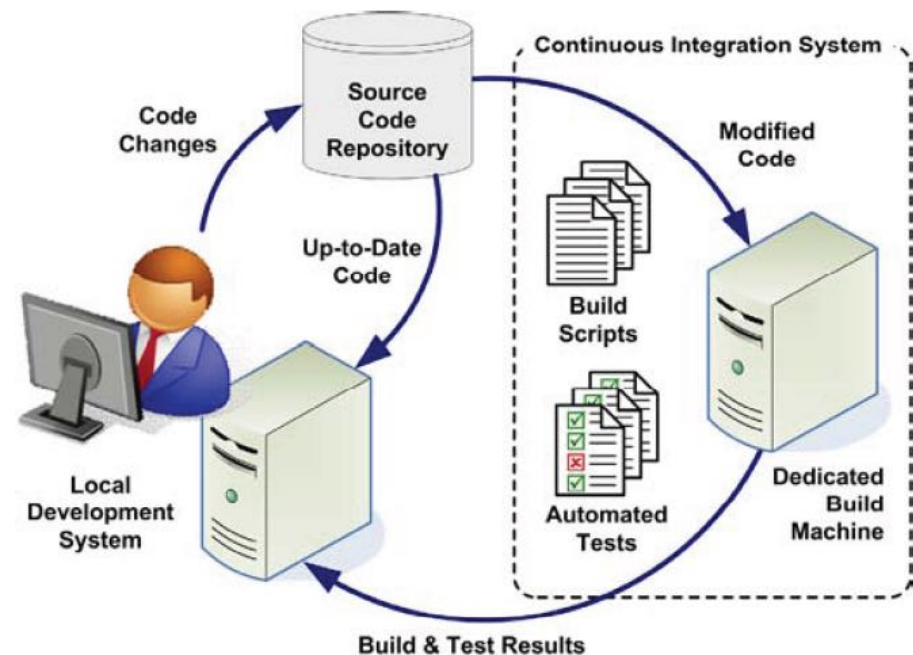
"Commit" every day

Deployment on an integration computer after each "commit"

Automatic deployment

Executable always available

Everybody knows the build state



Feedback for collaboration (awareness)

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

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

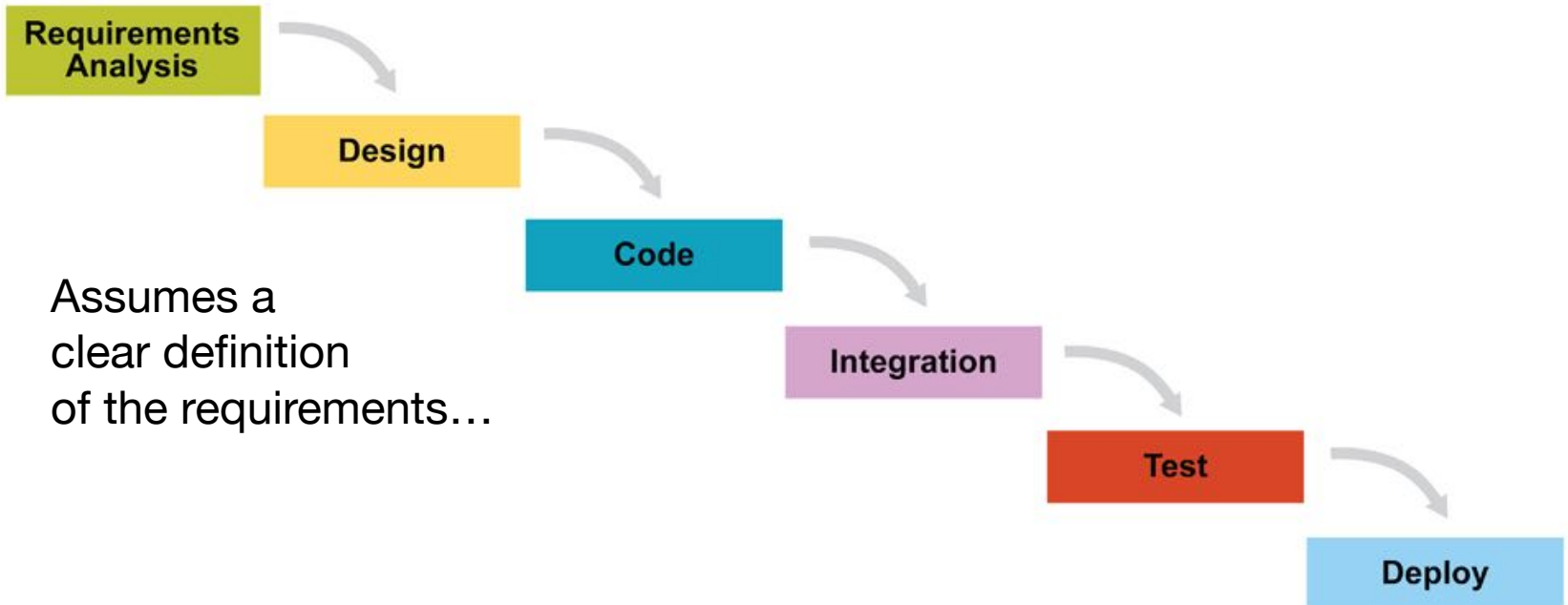
Drawing / construction plan / task distribution / construction

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

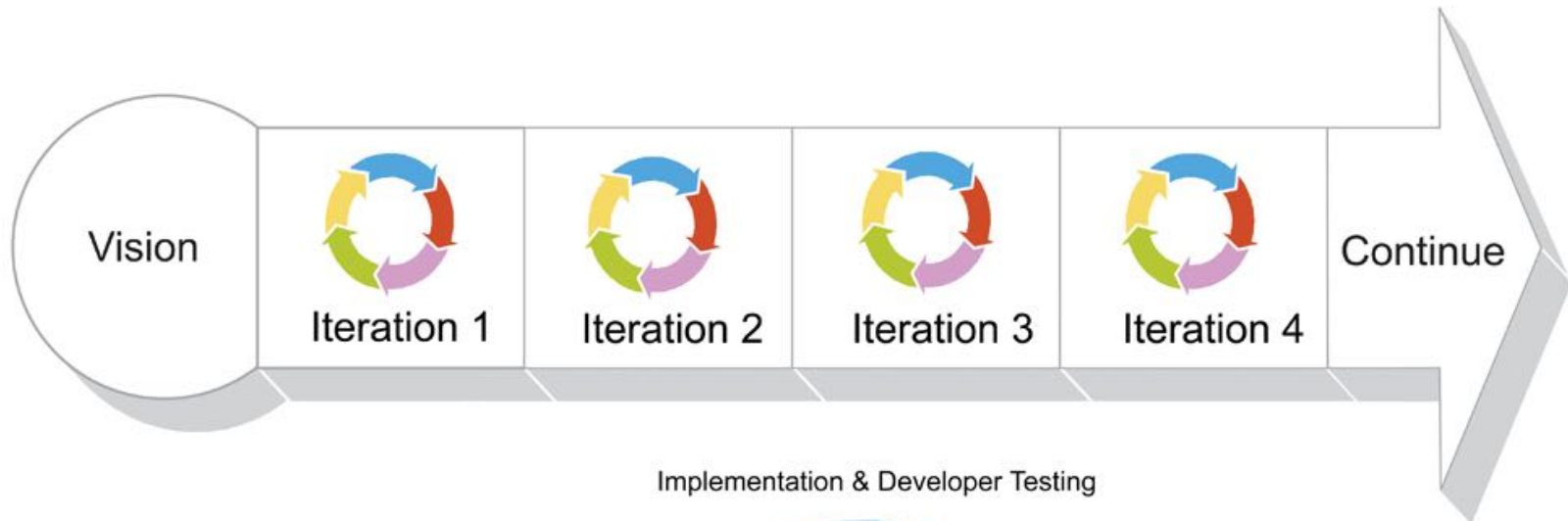
Analysts, coders, testers, managers

Developers are professionals who

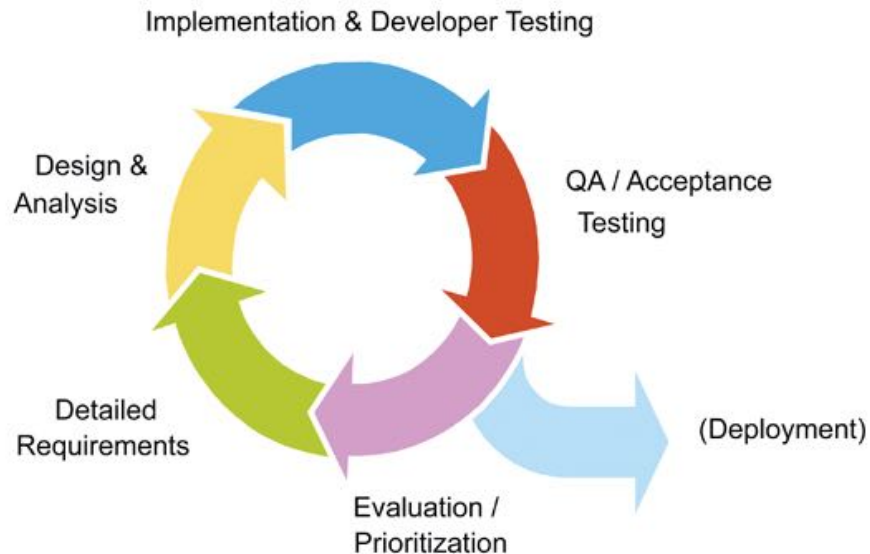
make the technical decisions

evaluate the time required to perform the tasks

Agile methods



Iteration Detail



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")

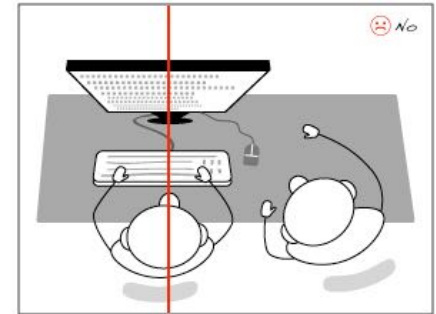
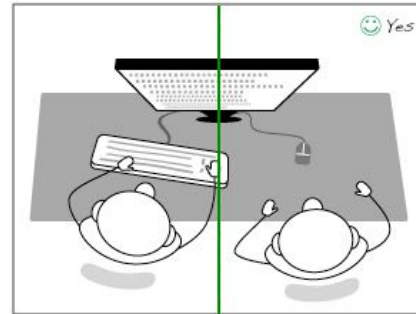
RUP (Rational Unified Process)

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

Pair/peer 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 tasks

Mentoring new hires

Extremely high-risk tasks

Start of a new project when the design is new

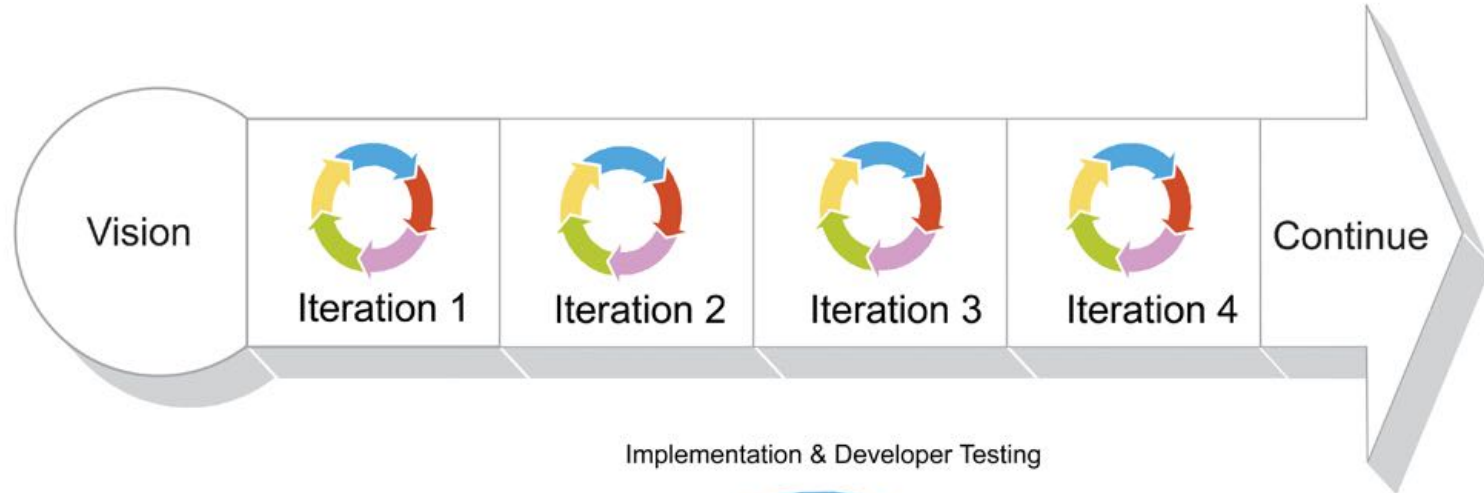
When adopting a new technology

On a rotating monthly or weekly basis

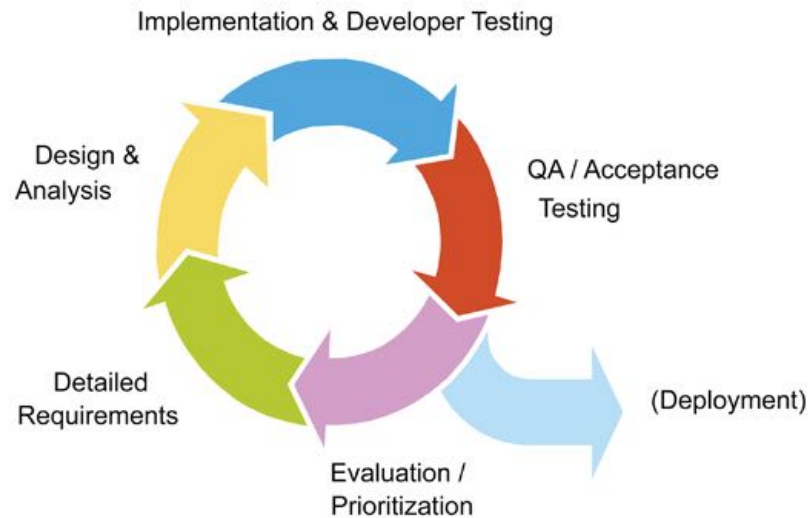
Developers who prefer to work in pairs

Scrum

Iterations called Sprint (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

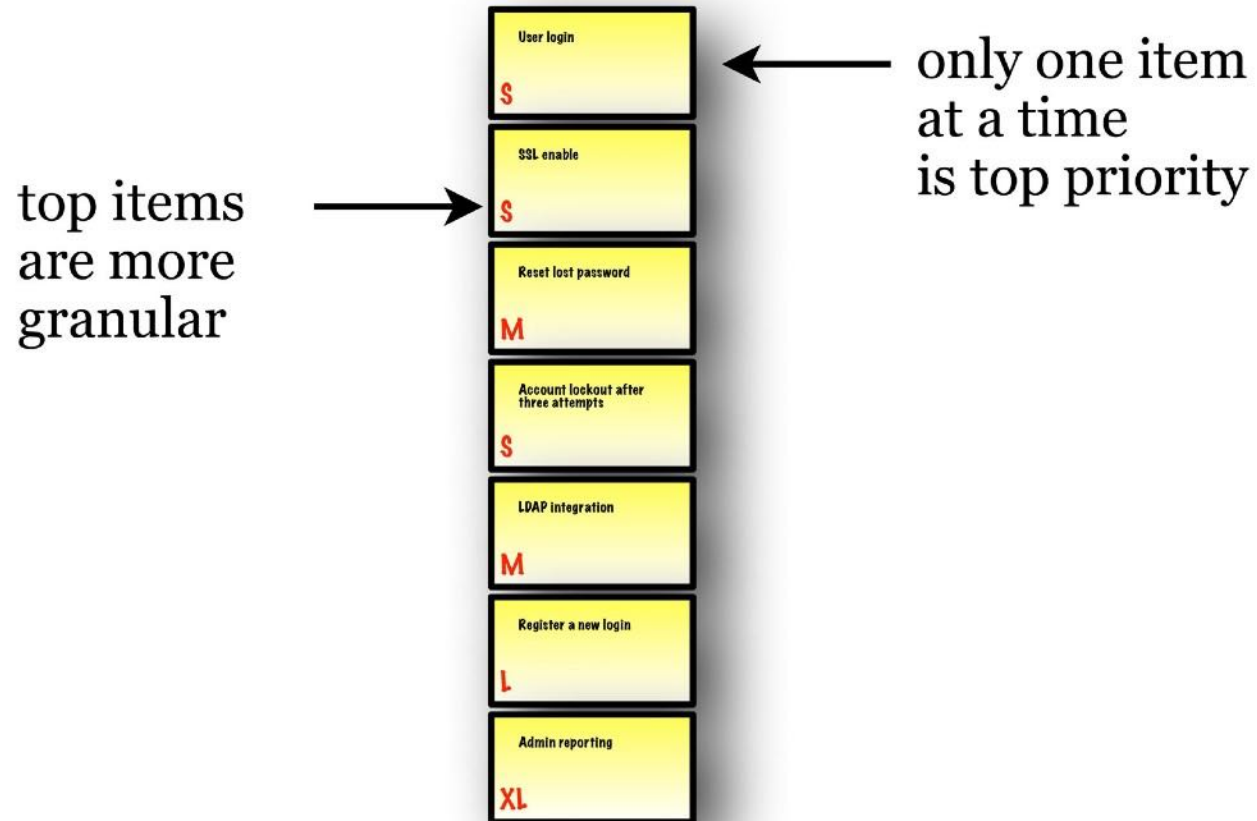
Helps resolve issues

Shields the team from external interferences and distractions

Has no management authority



Scrum: product backlog

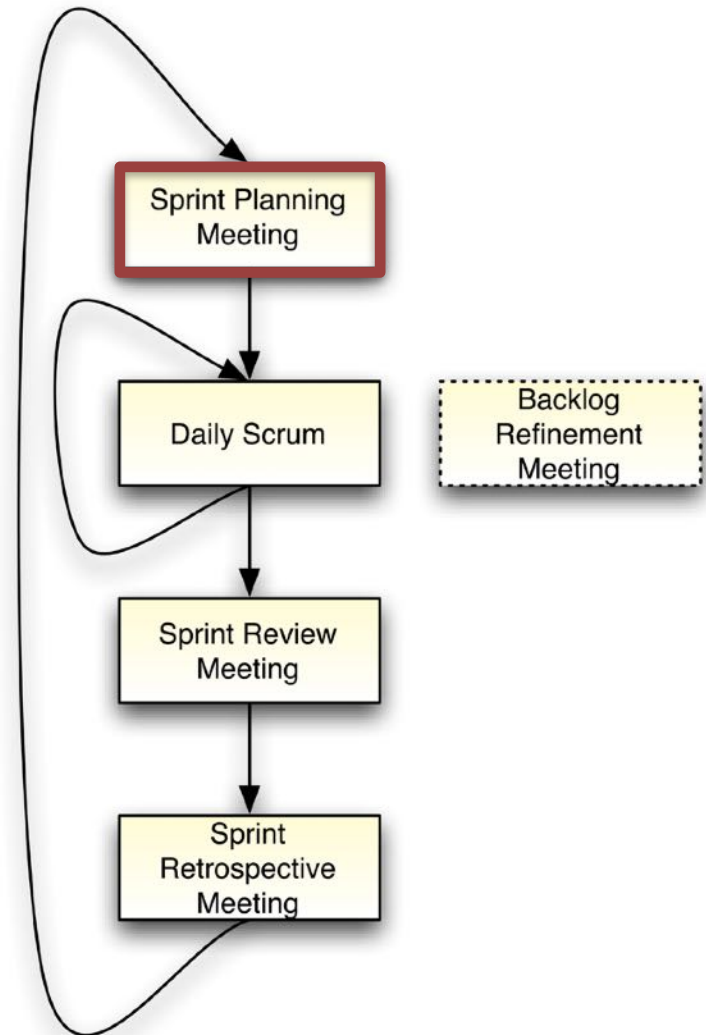


Sprint planning

Planning Meeting

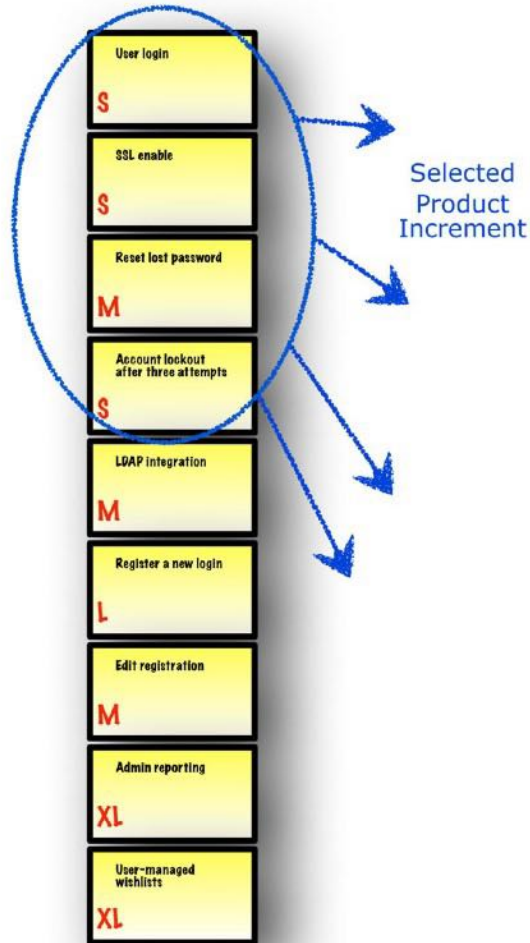
Negotiate which Product Backlog items will be processed

Break items into a list of sprint tasks

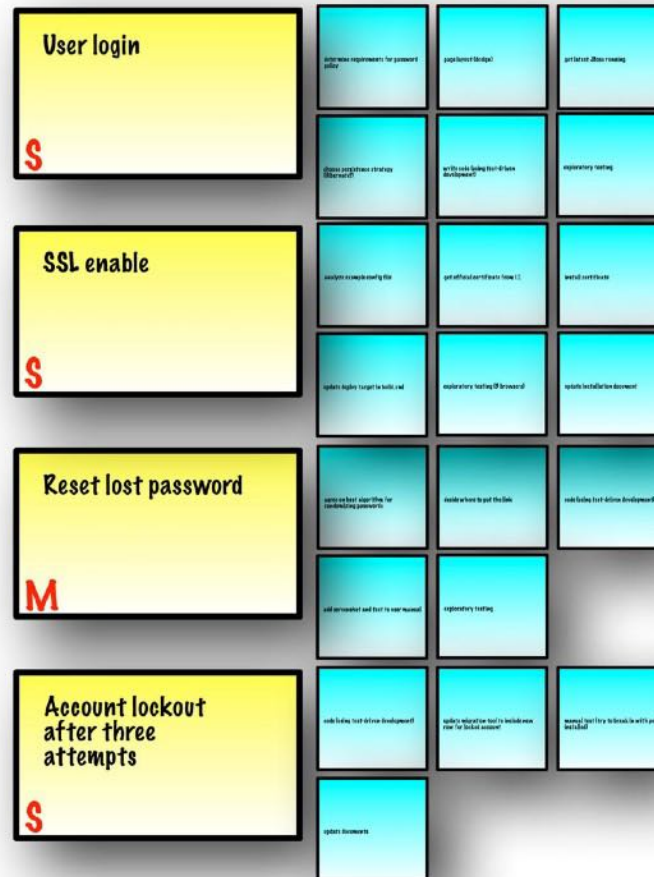


Sprint backlog

Product Backlog



Sprint Backlog



Daily scrum

Planning Meeting

Daily Meeting

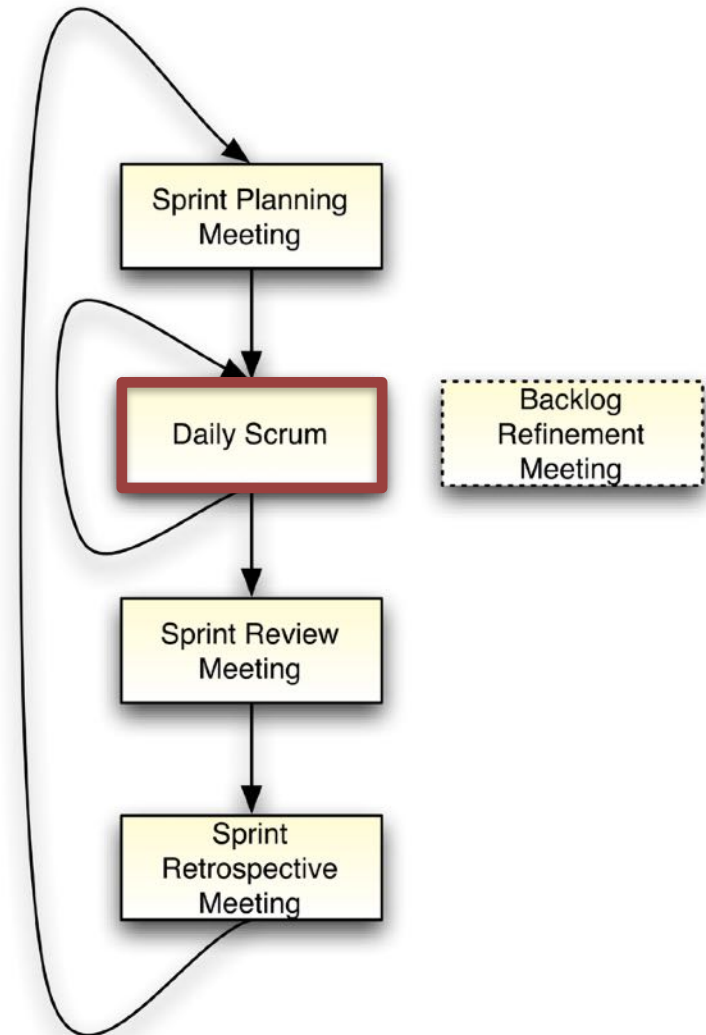
Same time and place

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

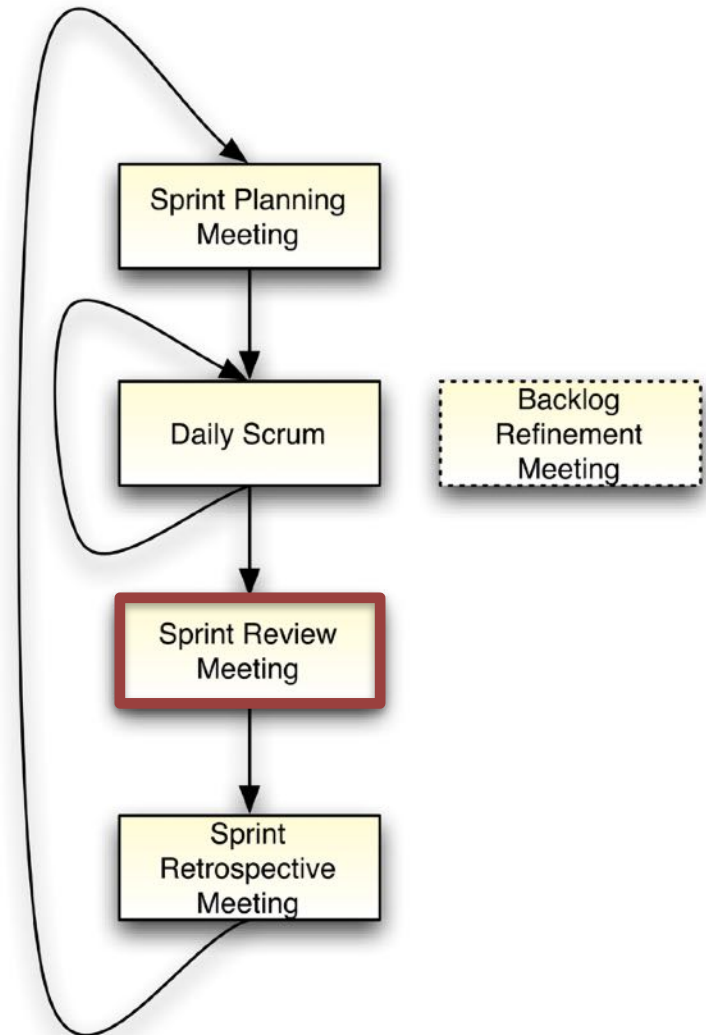
Review Meeting

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

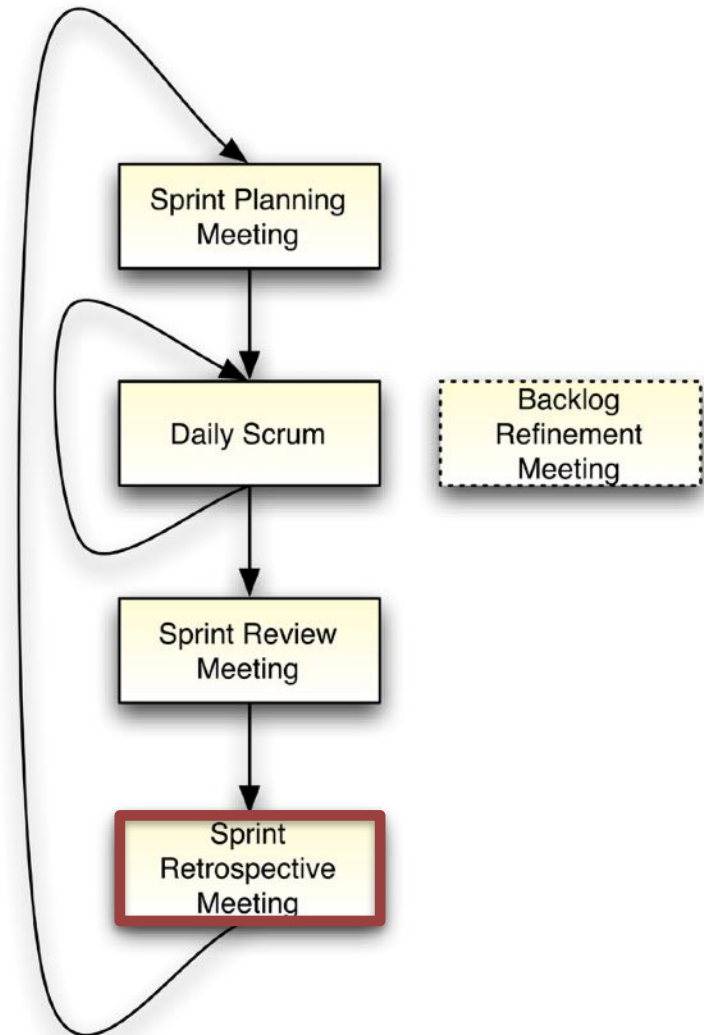
Review Meeting

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

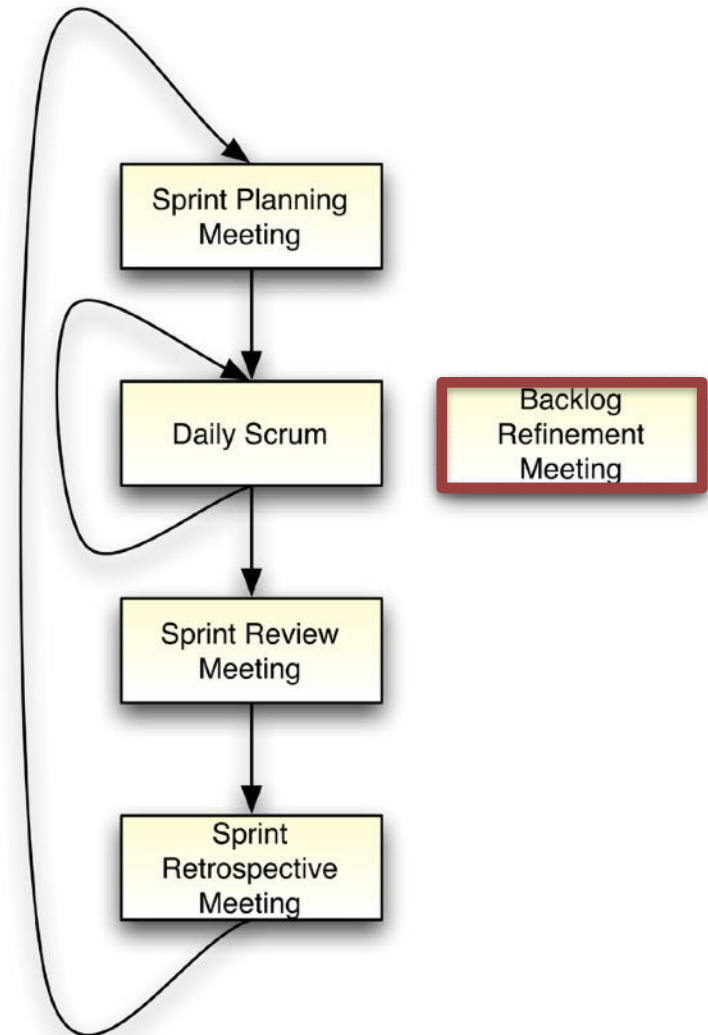
Retrospective Meeting

Backlog Refinement Meeting

Items are usually too large
or poorly understood

Refine items into smaller ones

Master can help



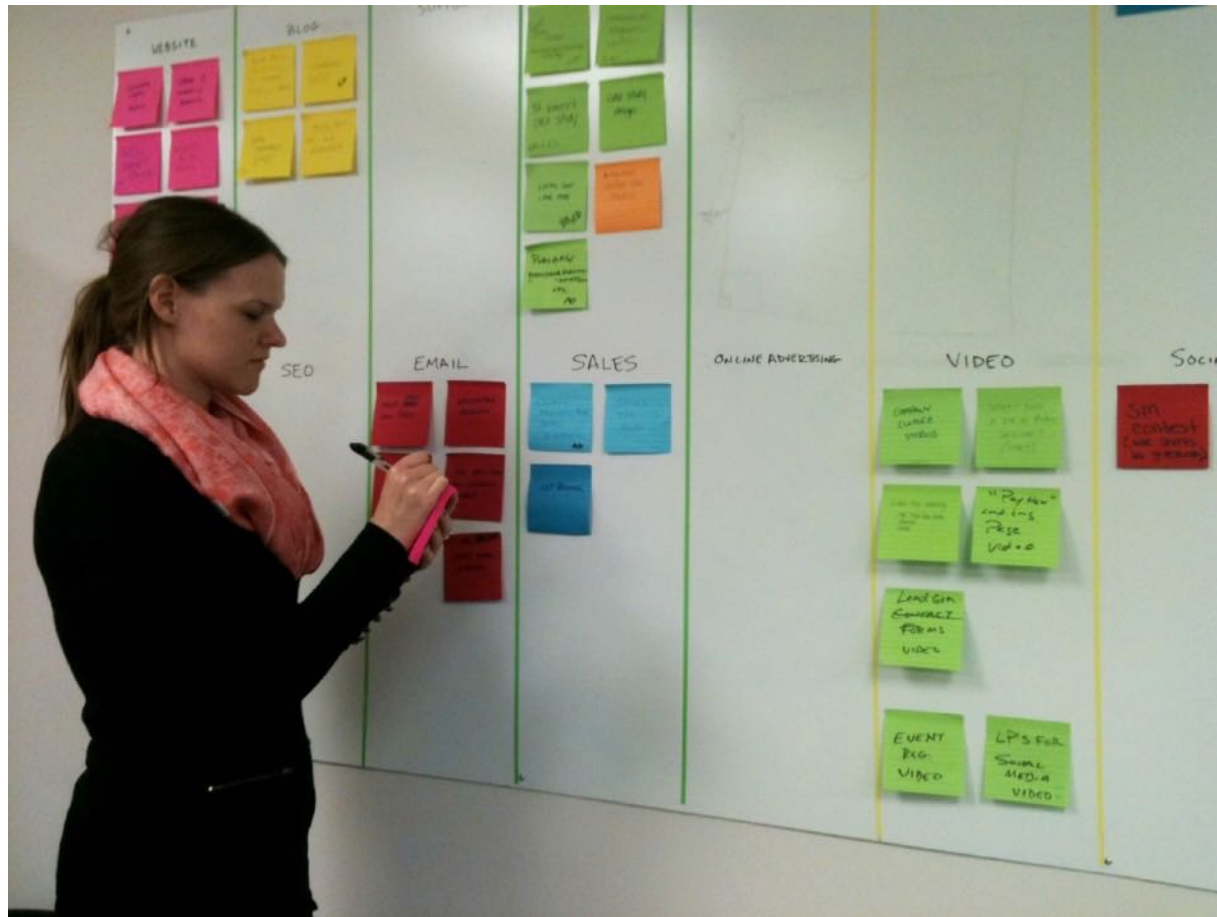
Wallboard

Feedback to the team

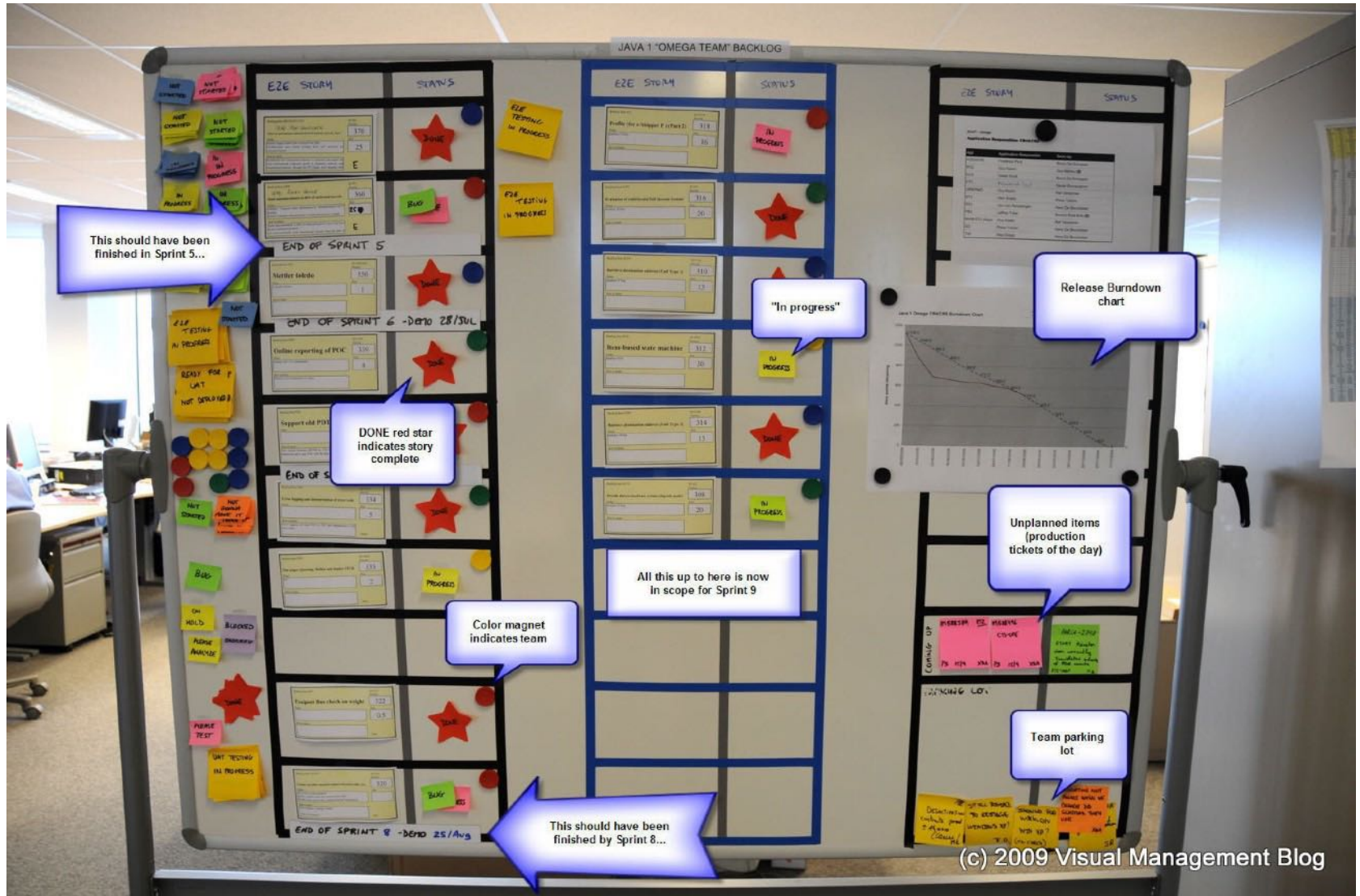


Wallboard

Feedback to the team



Wallboard



(c) 2009 Visual Management Blog

LEGO Bit planner

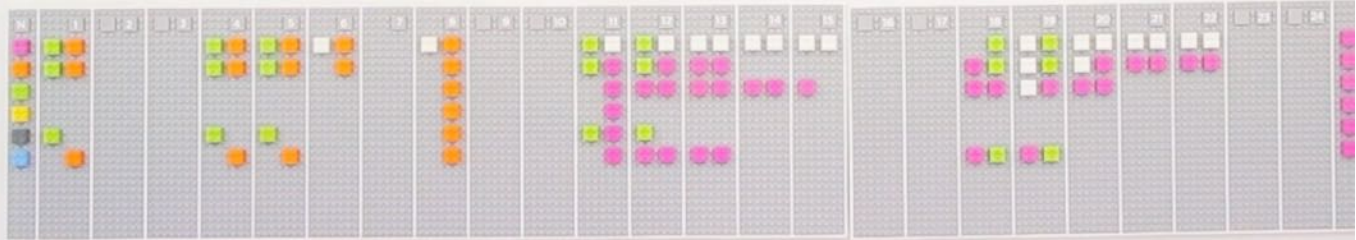
OCTOBER



NOVEMBER



DECEMBER



Scrum software

The screenshot displays the JIRA Agile interface for a project named "Angry Nerds". The top navigation bar includes the JIRA logo, a feedback link, user information (admin), and an Administration link. Below the navigation bar, there are tabs for Dashboards, Projects, Issues, Agile, and Bonfire. A "Create Issue" button and a search bar are also present.

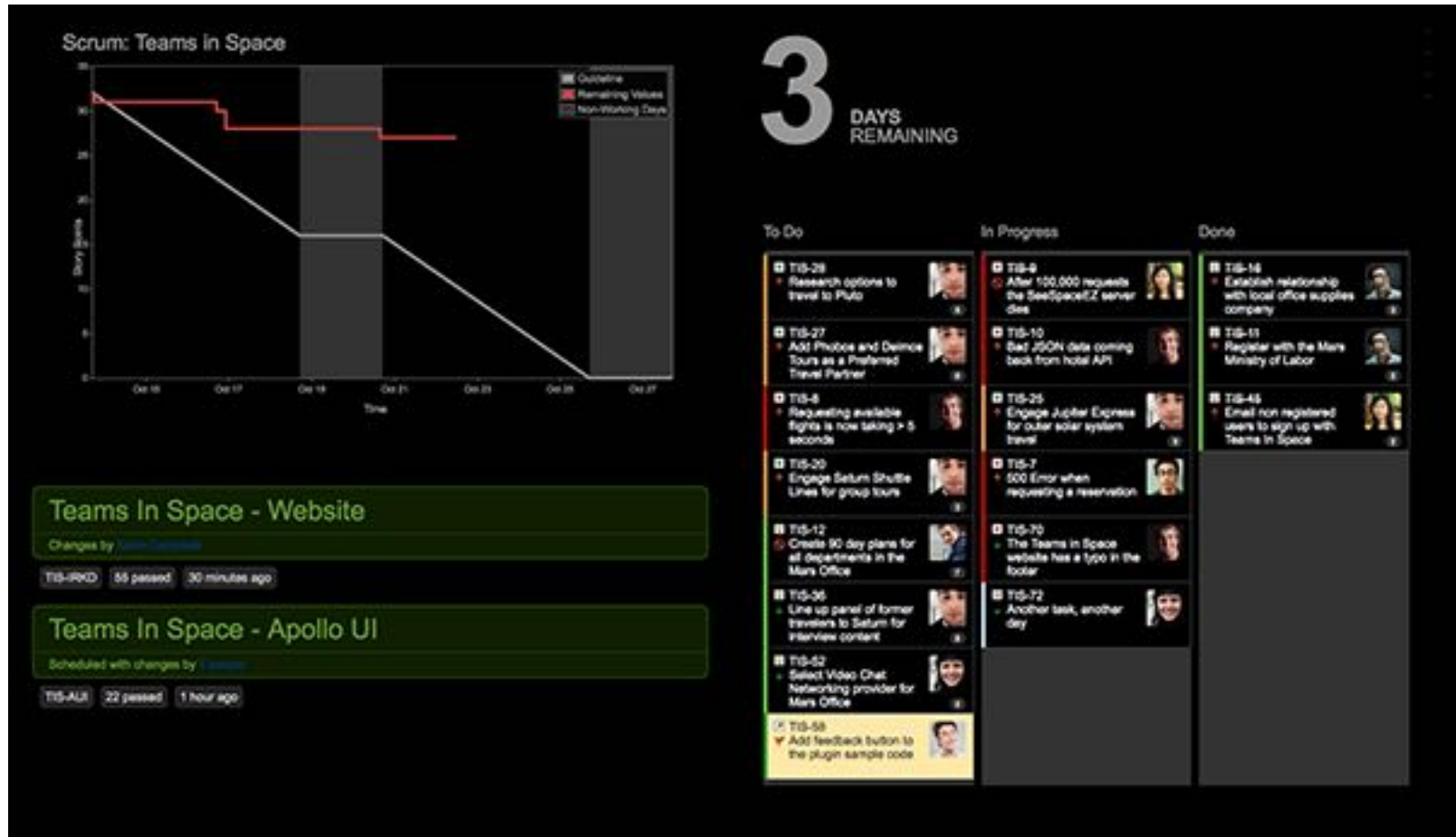
The main content area is divided into several sections:

- Angry Nerds -**: The project name, with a search bar and filters for "Only My Issues" and "Recently Updated".
- Angry Sprint**: A section showing the current sprint's progress (8 issues, 8 days left) and a list of issues. The issues are:
 - 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 can 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
- Upcoming Sprint 1**: A section showing the next sprint's issues:
 - NERD-9: As a Bug I want to be like The Beatles so I can be fab
 - NERD-12: As a Front-Ender I would like to stop supporting IE7 so I can enjoy my life (5)
 - NERD-13: As a Hacker I would like more pizza and beer (3)
 - NERD-20: As a Hacker I would like to hack the mainframe and enter the Matrix (13)
 - NERD-14: As a Dev Manager I want make everyone log time so I can make nice charts (8)
- Upcoming Sprint 2**: A section showing the following sprint's issues:
 - NERD-15: As an Outsourcerer I want to work more hours and get less done (3)
 - NERD-16: As an Agilista I want to be lean to make the green, quicker (1)
 - NERD-17: As an Agilista I want to iterate on the rate so I can rate the iteration (5)

The right-hand side of the interface shows a detailed view of issue NERD-4:

- Issue Title:** "As an Outsourcerer I want to get paid for working in my pyjamas"
- Estimate:** 20
- Status:** Not Started (0/1 Completed)
- Session:** Confirm they are in pyjamas (Created)
- Actions:** Create Session

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

Scrum

<http://scrumreferencecard.com>