Groupware and Collaborative Interaction

Collaborative Software Development

Anastasia.Bezerianos@lri.fr

some slides based on lecture by Cédric Fleury

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

Outline

Collaborative software development

- Version control
- Continuous integration
- Software development methods

Outline

Collaborative software development

Version control

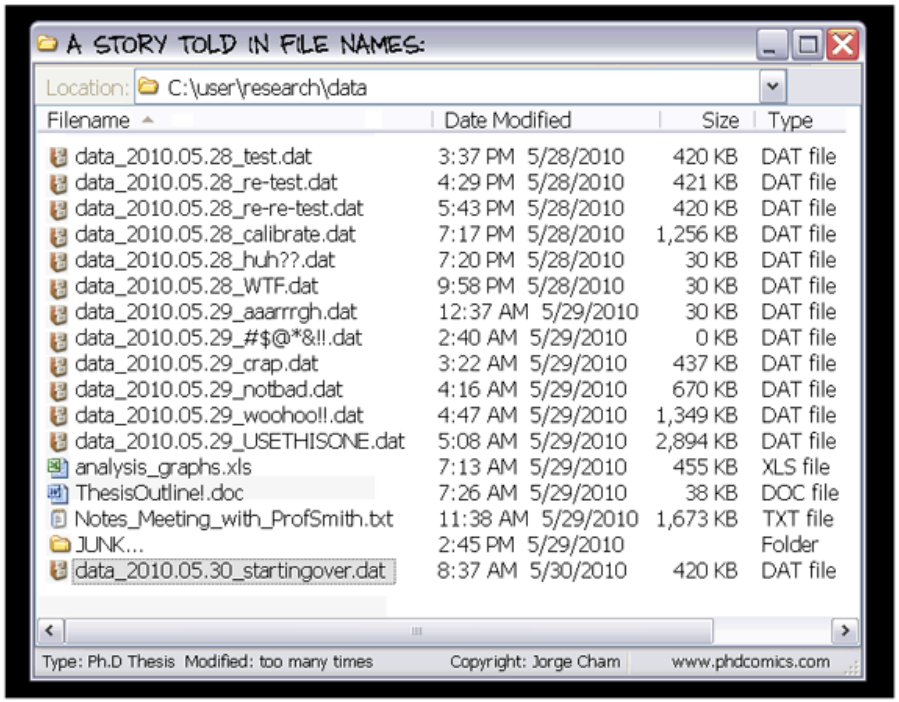
Continuous integration

Software development methods

Version control

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



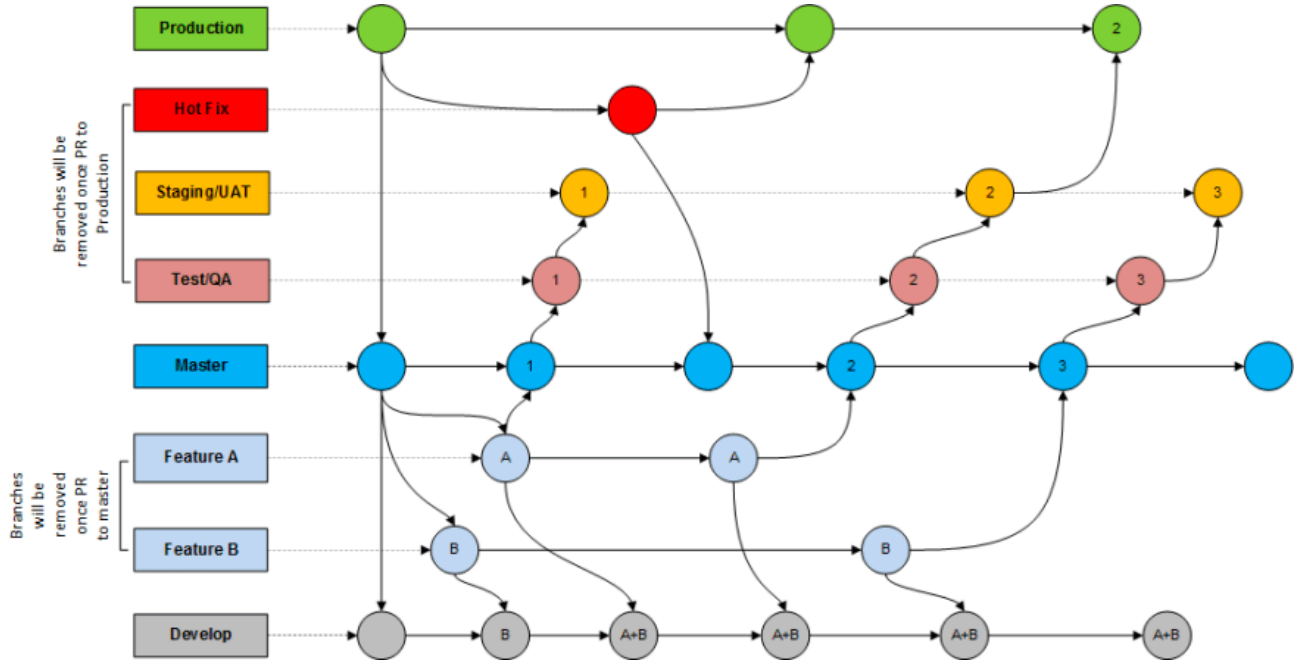
5

[“Piled Higher and Deeper” by Jorge Cham: www.phdcomics.com]

Version control

We want to able to:

- Edit the project at the same time, and merge our work
- Keep an history of the modifications, restore old ones
- Keep the older version of the files + hierarchy



GitFlow and GitFlowChart

Version control

Version Control Software architectures

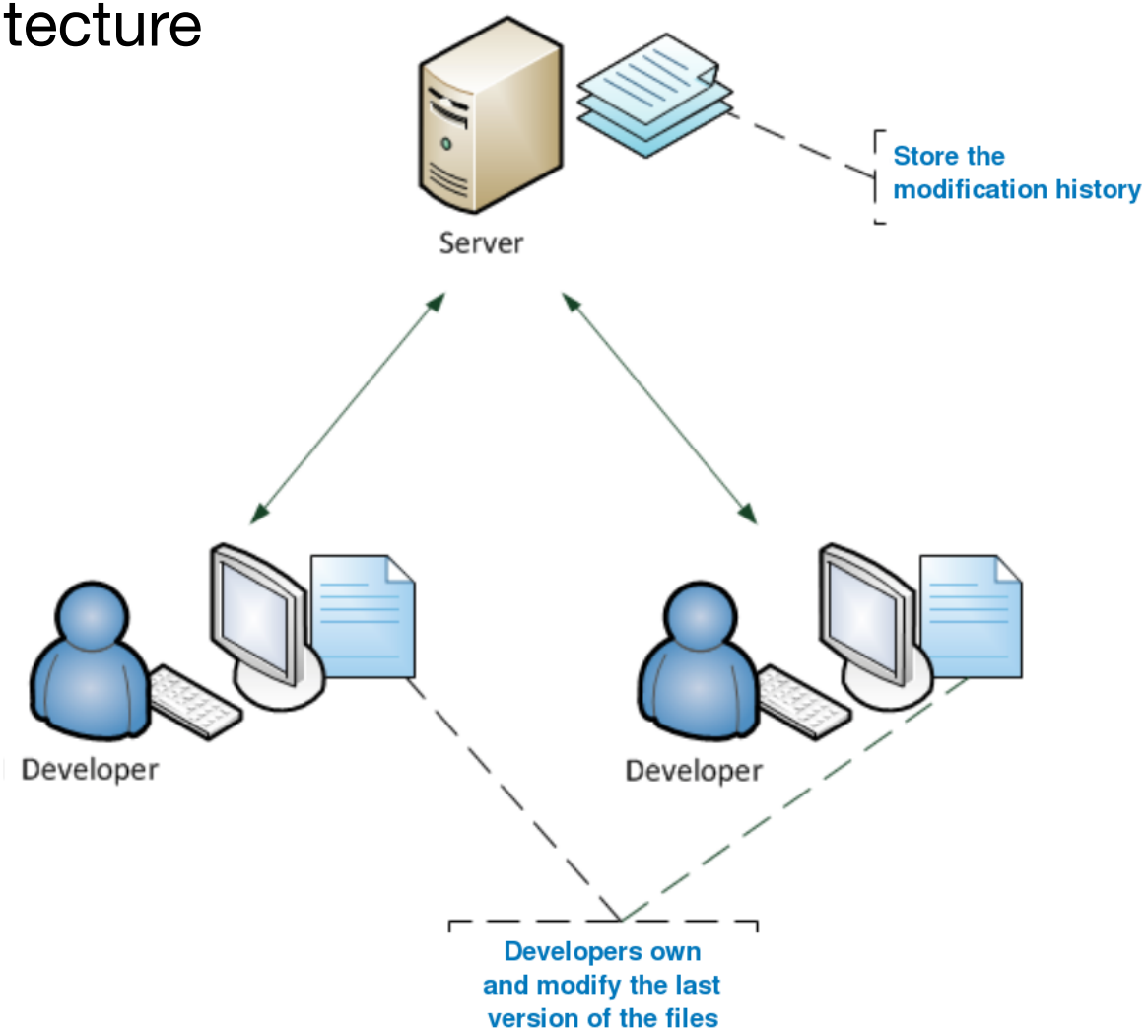
- Centralized
  - CVS, SVN, TFVC, …
- Decentralized (peer-to-peer):
  - GNU Arch, Mercurial, Bazaar, Git,…
- also Decentralized can be used as a Hybrid Architecture
  - One peer can be a central server

Not only for software development

- Reports, images, data from experiments

Version control

Centralized architecture

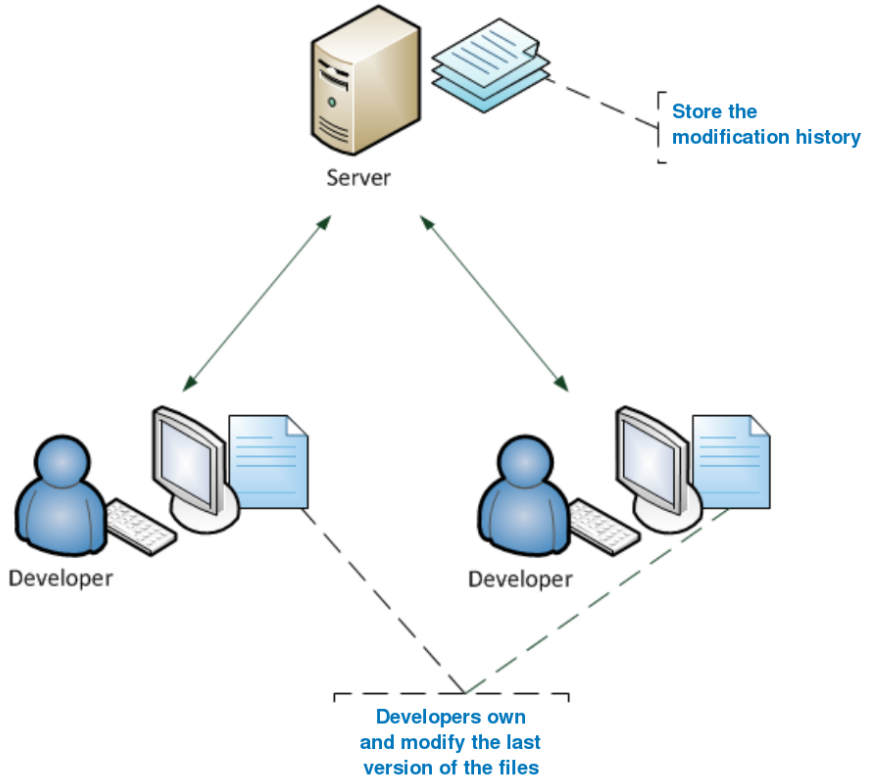


Version control

Vocabulary (SVN)

- Architecture
  - Repository
  - Working copy
- Actions
  - Checkout
  - Update
  - Commit
  - Revert
  - Diff, log, status

Centralized Architecture



9

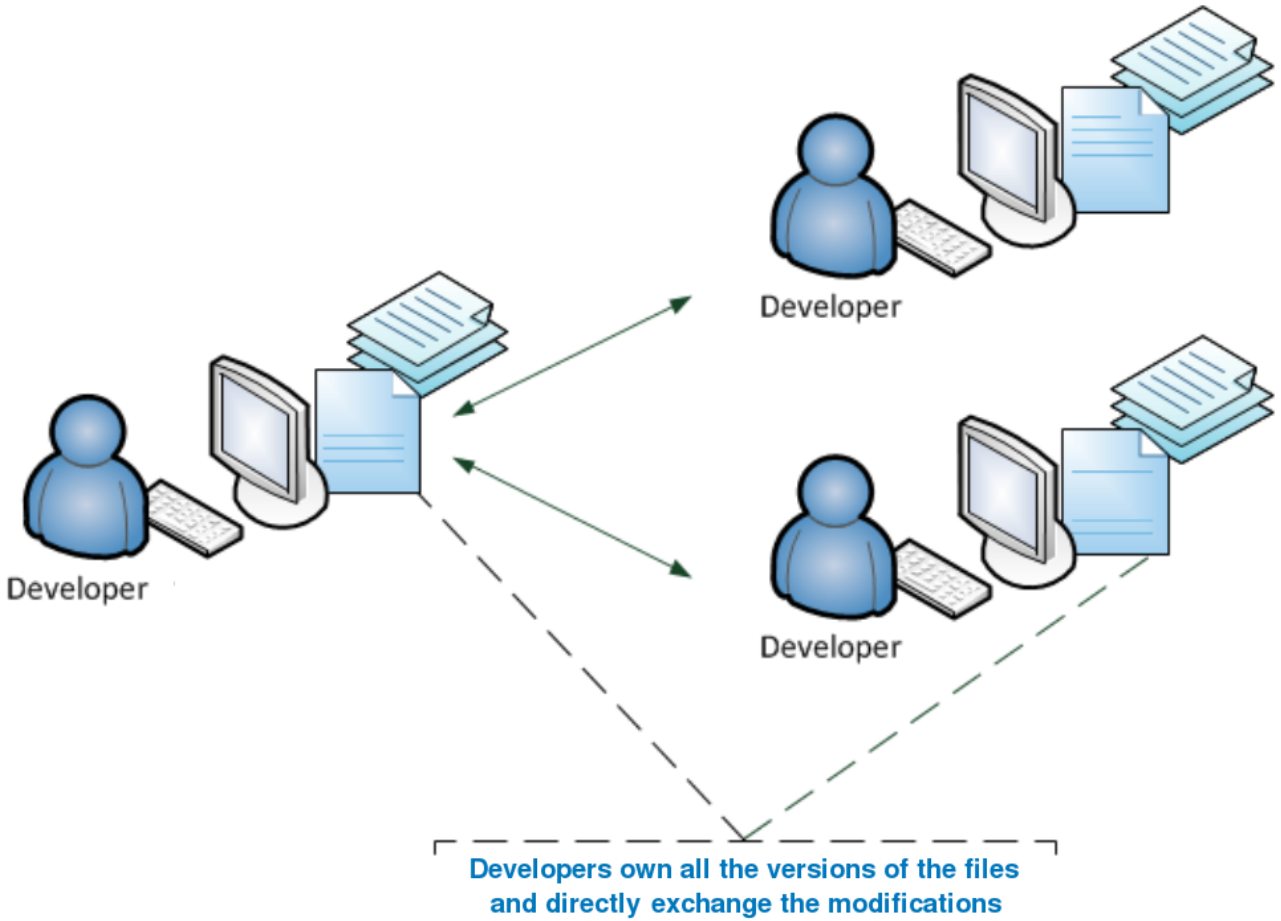
Version control

Drawbacks of the centralized architecture

- Just one access point to the data
- Just one communication point between users
- Just one history / timeline of the files
- Versioning and sharing are the same operation
  - Need to have a stable state before "committing"

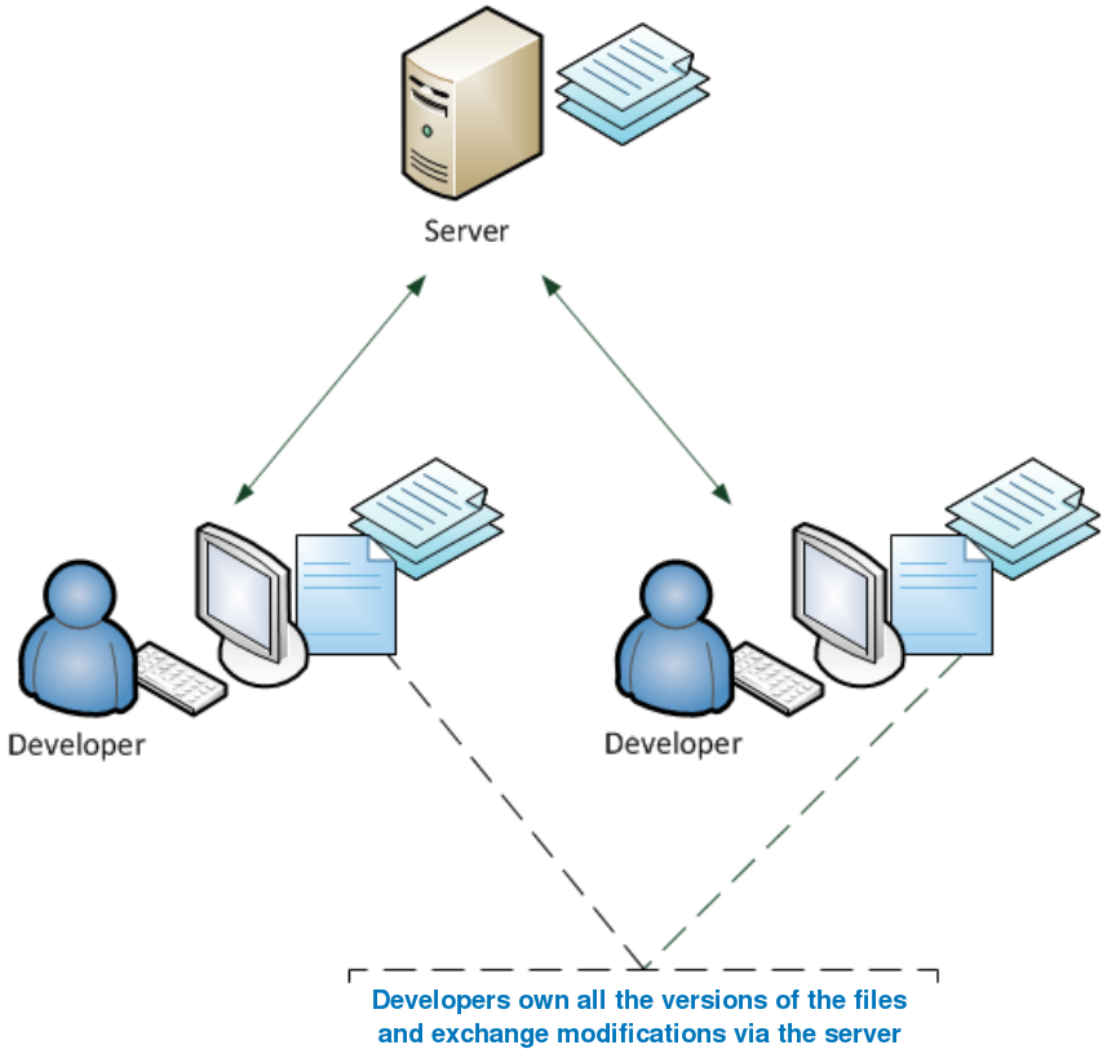
Version control

Decentralized architecture



Version control

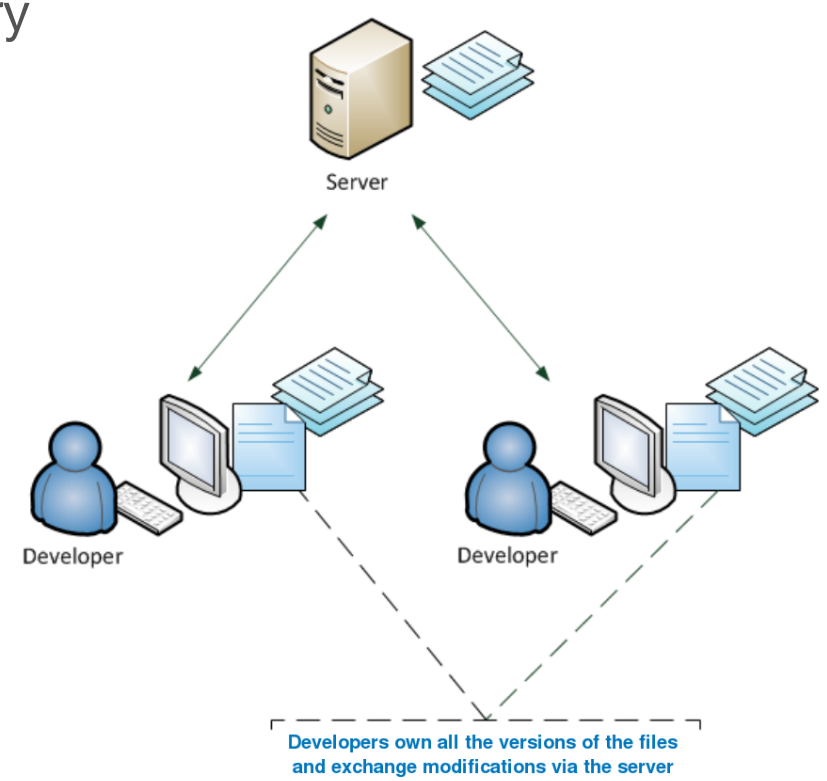
Hybrid architecture



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 (in all architectures)
  - 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 appears, version control appends both versions
  - Usually, it cannot be resolve automatically
  - Users have to fix the conflict
    - 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

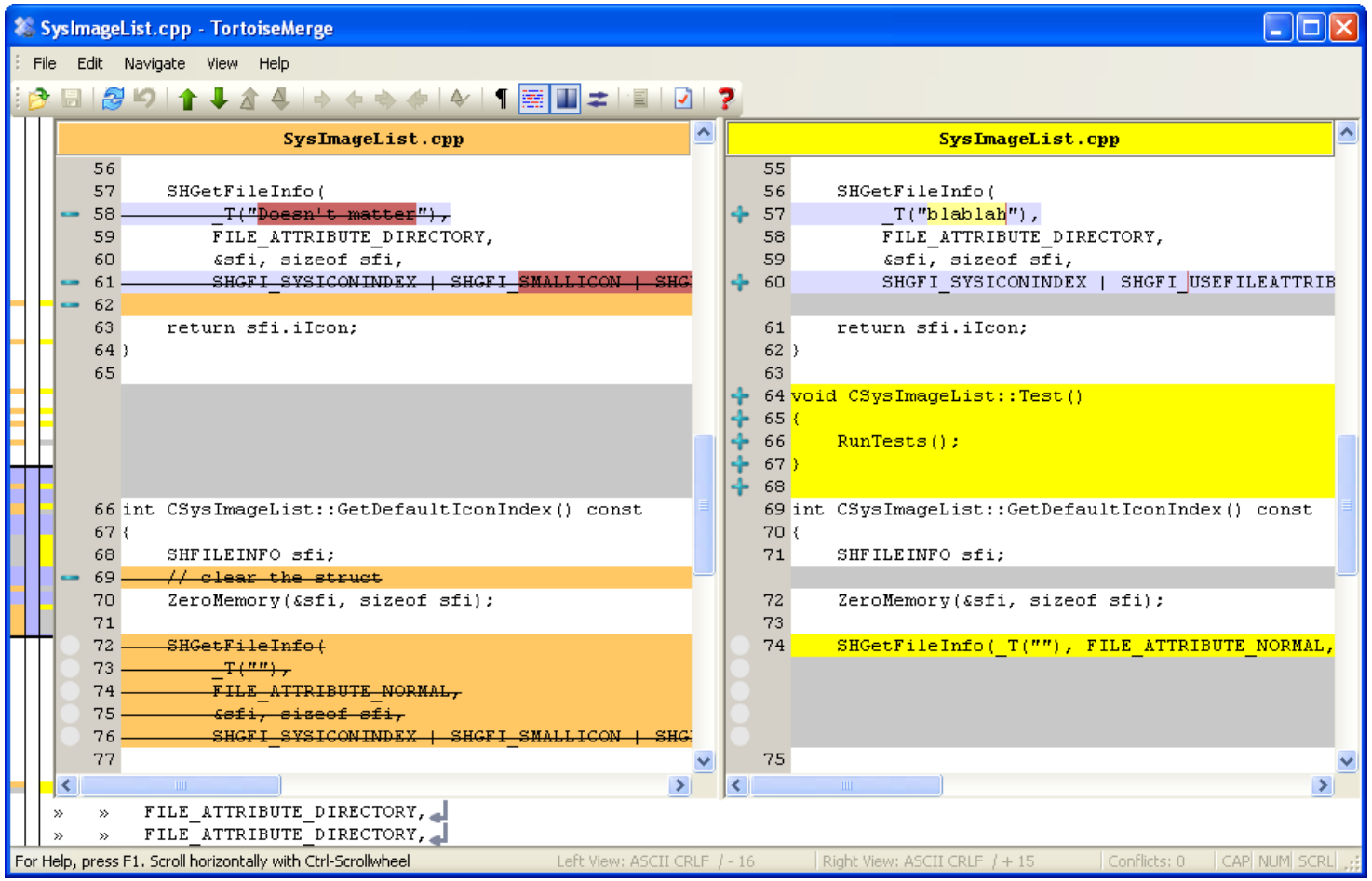
```
08/10/2010  11:44 AM                94 test.txt
08/10/2010  11:44 AM                26 test.txt.mine
08/10/2010  11:44 AM                27 test.txt.r2
08/10/2010  11:44 AM                31 test.txt.r3
```

test.txt

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

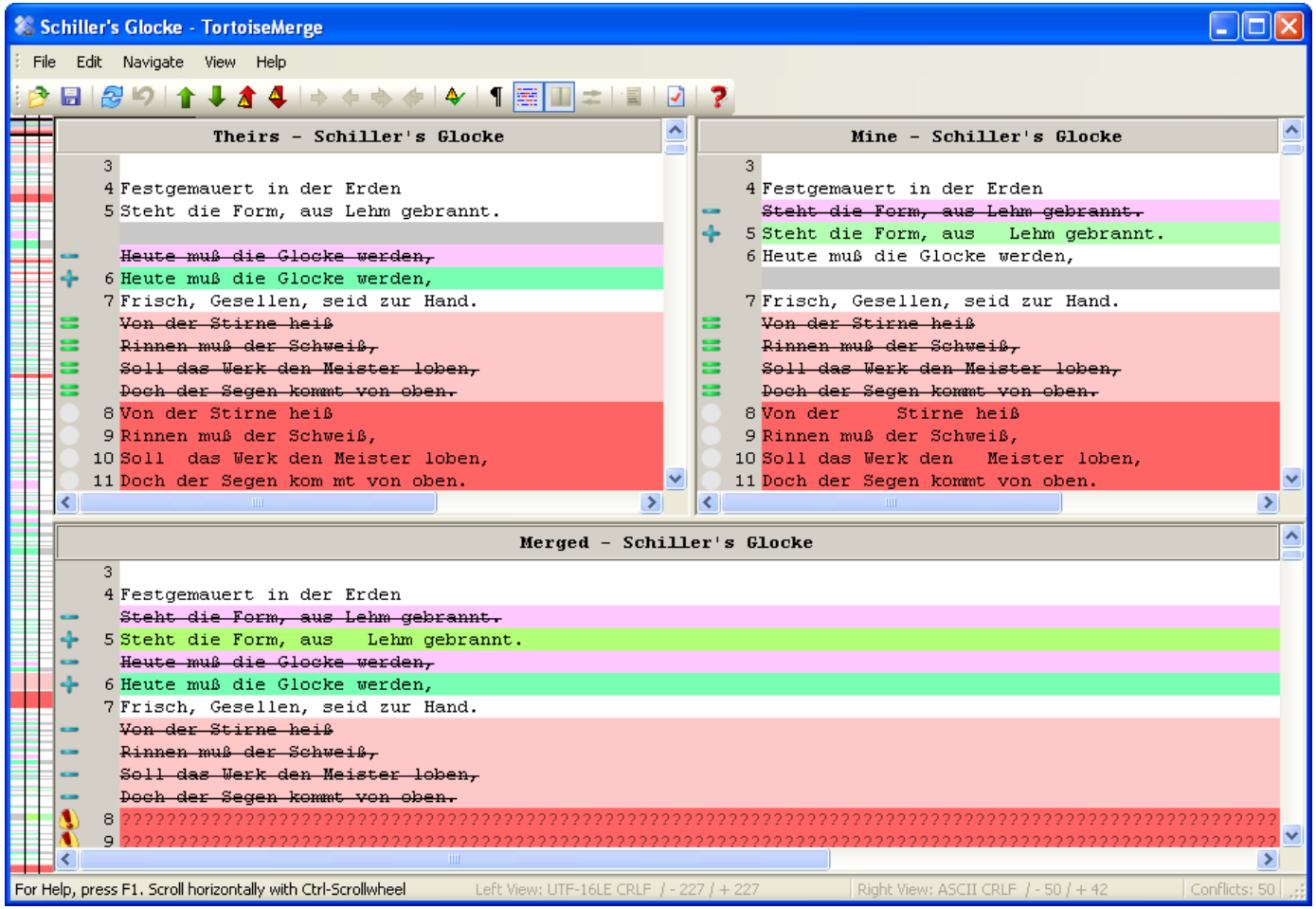
Version control

Tools for conflict management (TortoiseMerge)



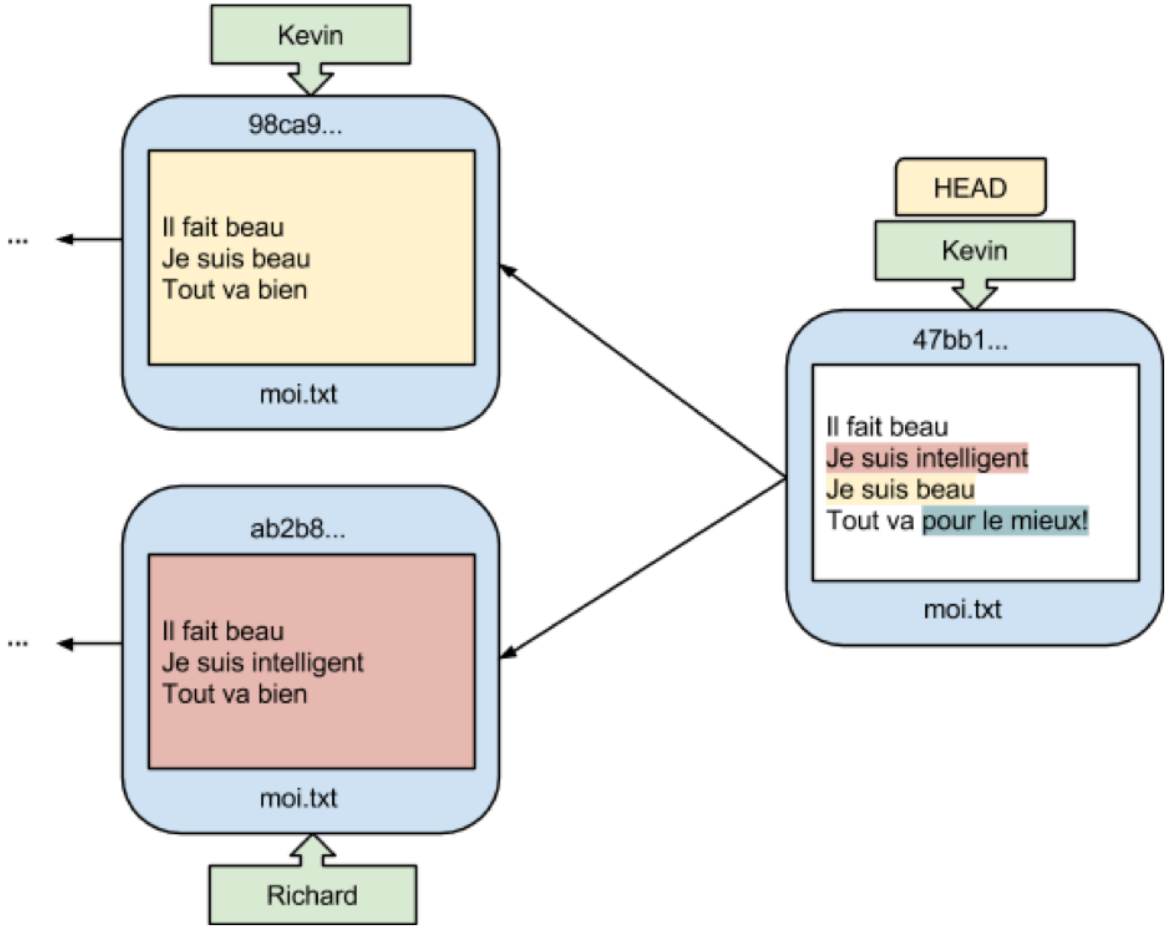
Version control

Tools for conflict management (TortoiseMerge)



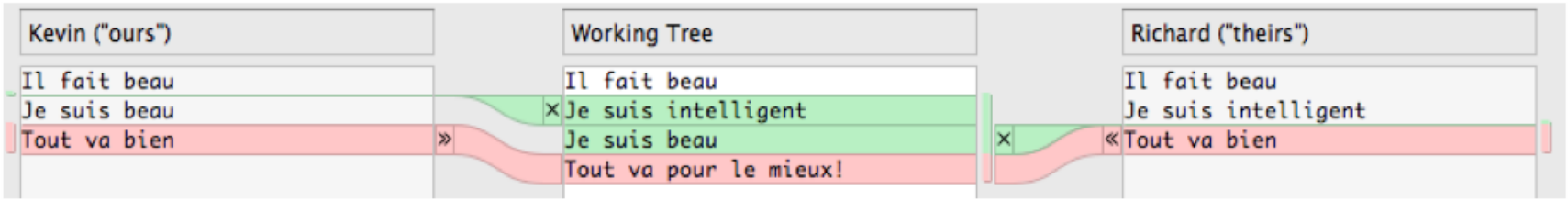
Version control

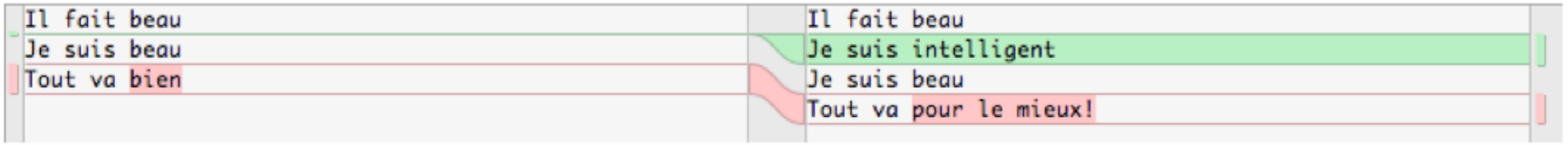
Tools for conflict management (SmartGit)



Version control

Tools for conflict management (SmartGit)





Version control

Diffamation

Using Text Animated Transitions to Support Navigation in Document Histories

Fanny Chevalier
Microsoft-INRIA
chevalie@lri.fr

Pierre Dragicevic
INRIA
dragice@lri.fr

Anastasia Bezerianos
École Centrale Paris
anastasia.bezerianos@ecp.fr

Jean-Daniel Fekete
INRIA
fekete@inria.fr

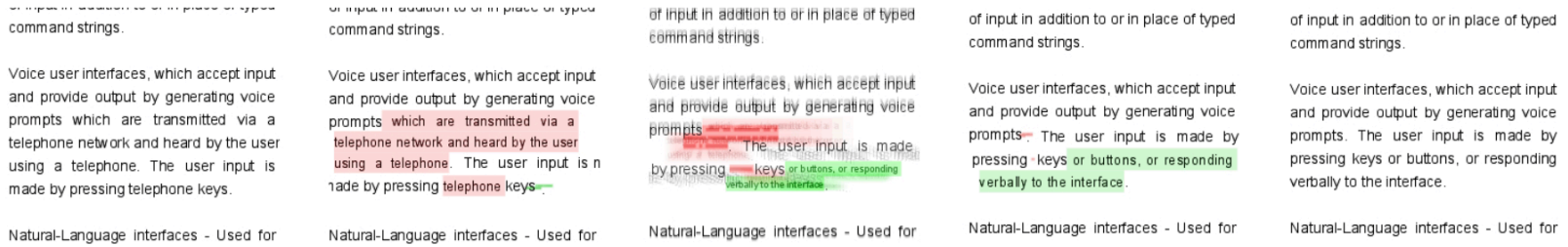


Figure 1. Detail of an animated transition between two revisions of the Wikipedia article *User interfaces*.

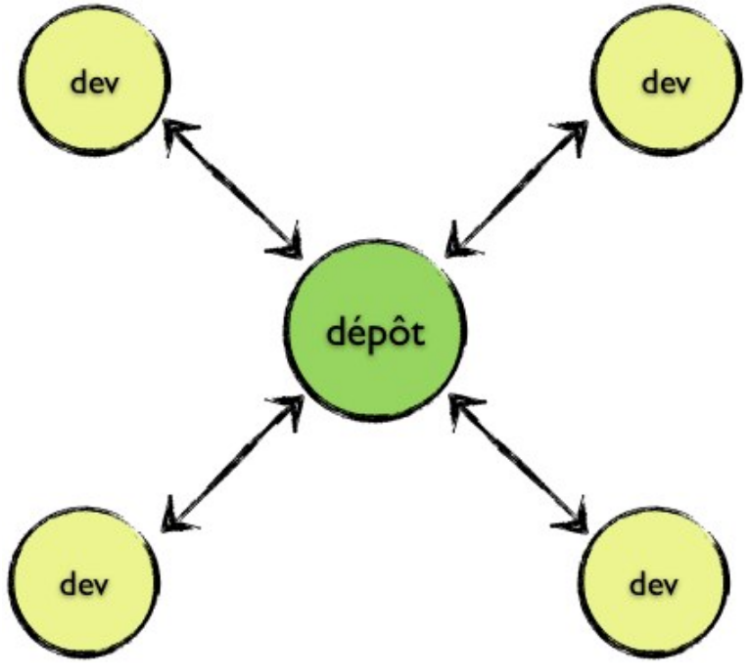
ABSTRACT
This article examines the benefits of using text animated transitions for navigating in the revision history of textual documents. We propose an animation technique for smoothly

others [20]. Supporting change awareness is not only essential for writing articles, but also for programming code where changes can profoundly impact the quality of a program. Millions of people, such as programmers, researchers

22

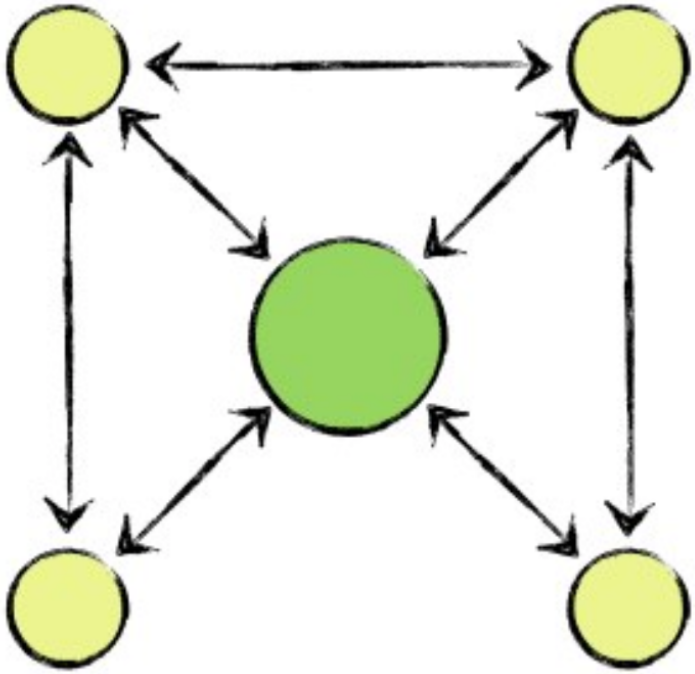
Version control

Collaboration scenario : centralized (SVN)



Version control

Collaboration scenario : decentralized (Git)



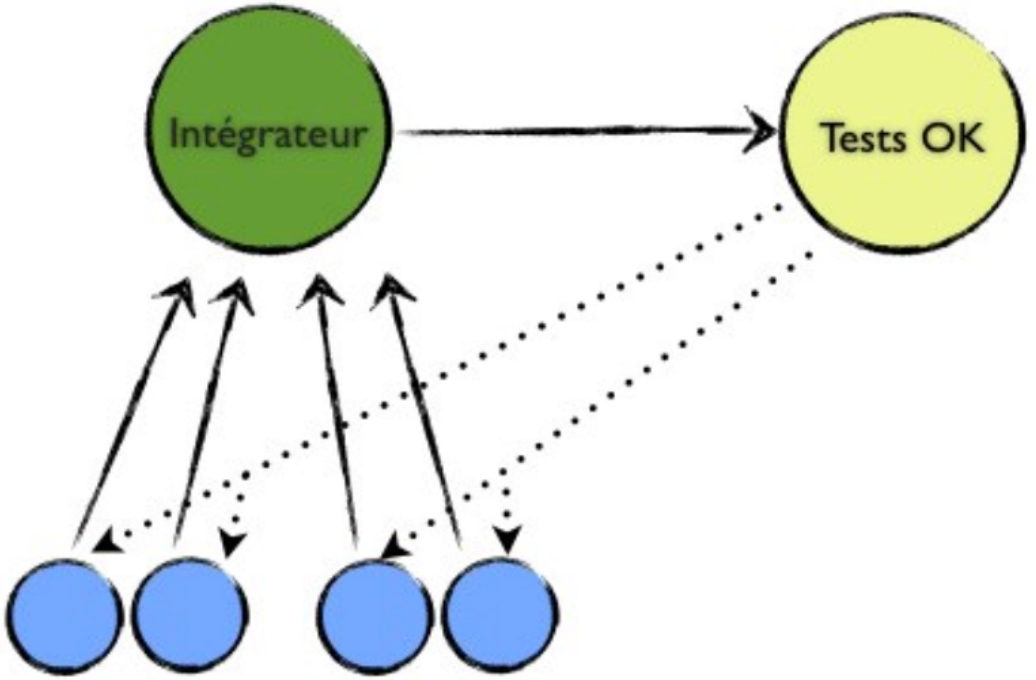
Inter-personal communications

Version control

Collaboration scenario : decentralized (Git)

Integrator mode

A repository
is in charge
of the test



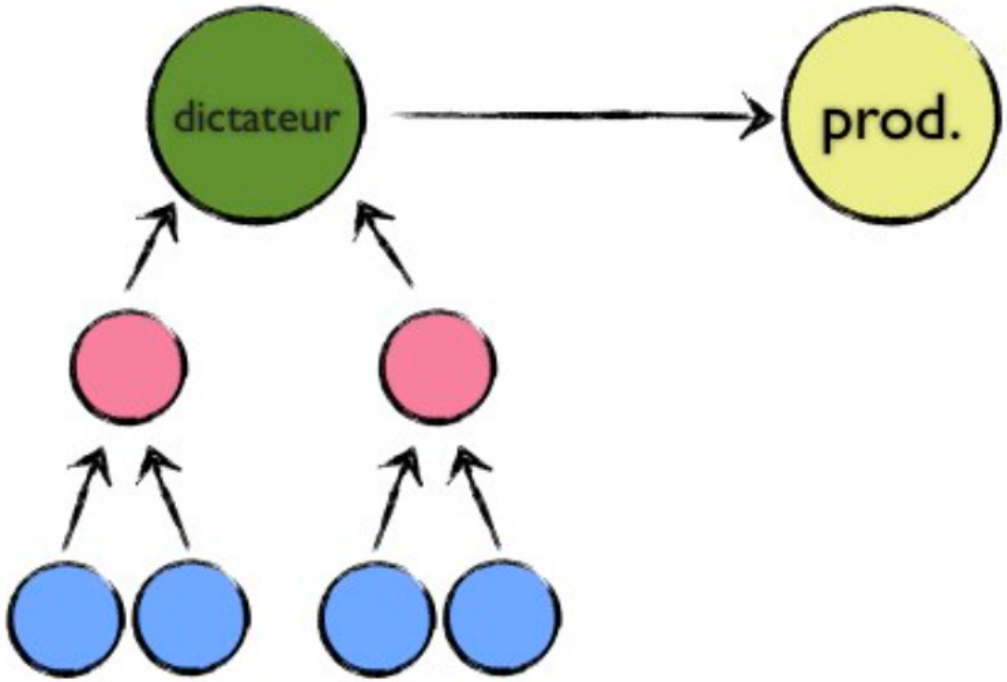
Version control

Collaboration scenario : decentralized (Git)

Dictator mode

Open-source projects

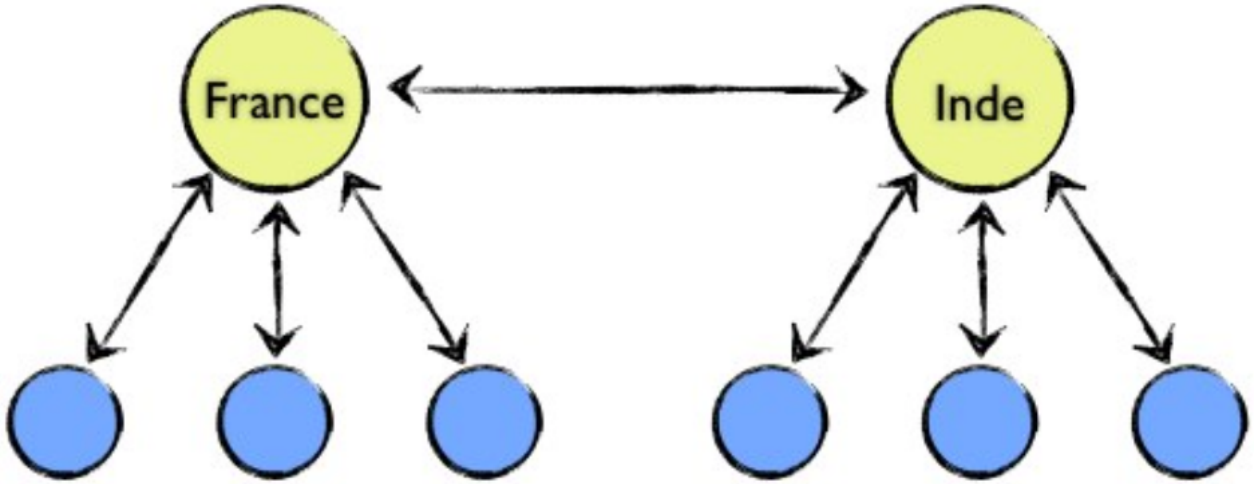
"Lieutenants" make a first check before sending to the "dictator"



Version control

Collaboration scenario : decentralized (Git)

- Multi-location teams
  - Each team can work independently
  - Regular integration of the work of each team

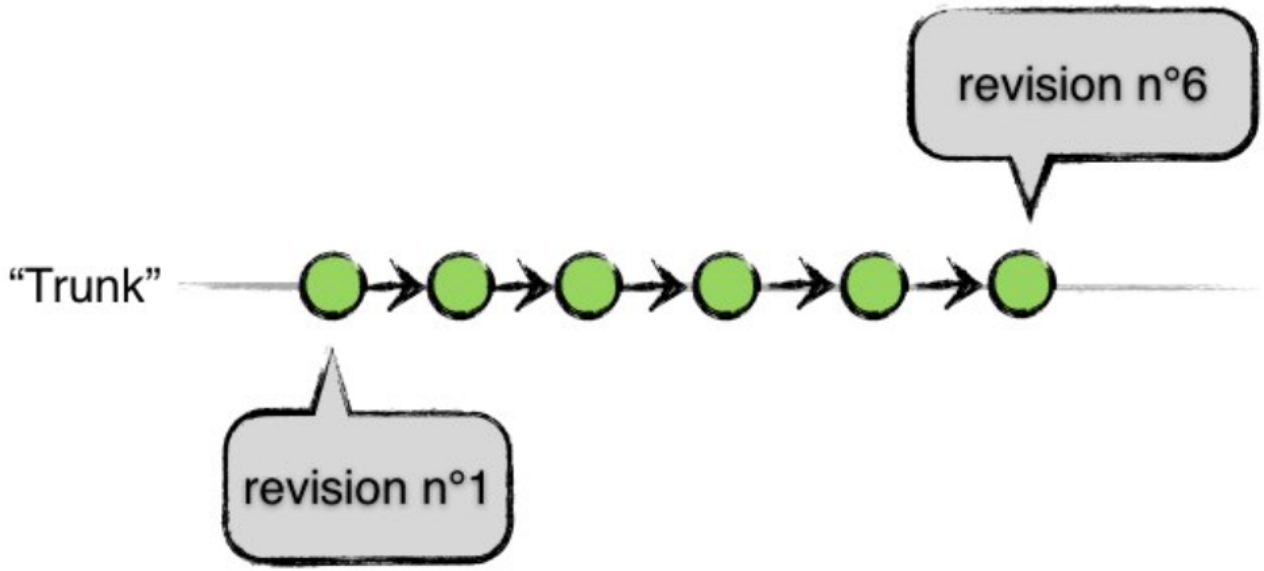


27

Version control

History management

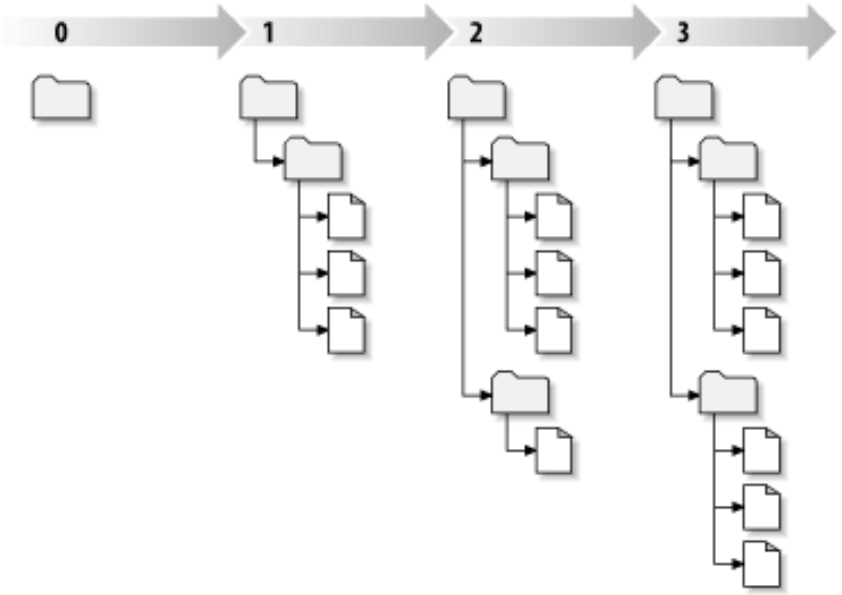
Computation of the history is linear if you consider the order of “commits"



Version control

History management

SVN assigns a revision number to the entire project



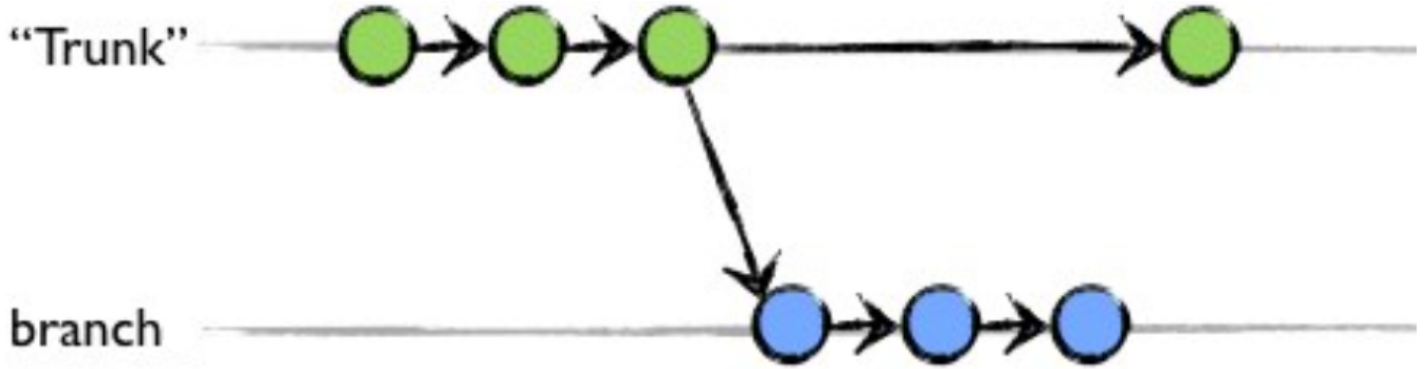
Git assigns a revision number per file

This difference impacts collaboration

Using branch for collaboration is easier with Git

Version Control

Branch management



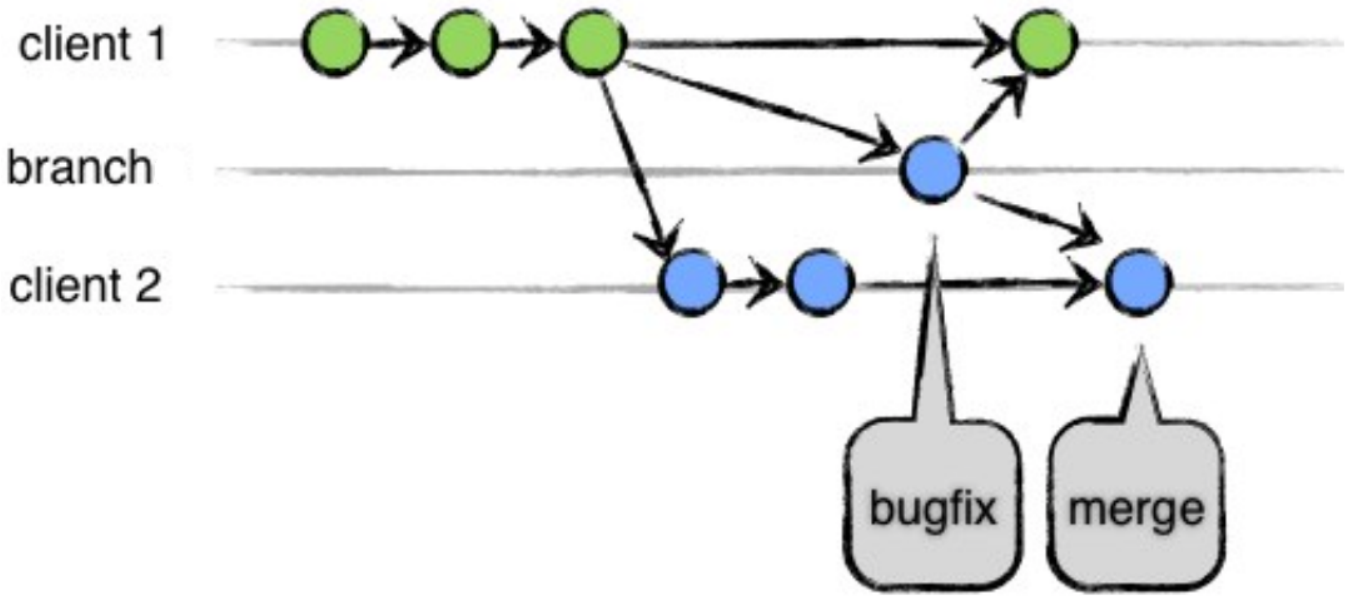
SVN makes a copy of the entire repository

Git makes a link to a particular state of the files

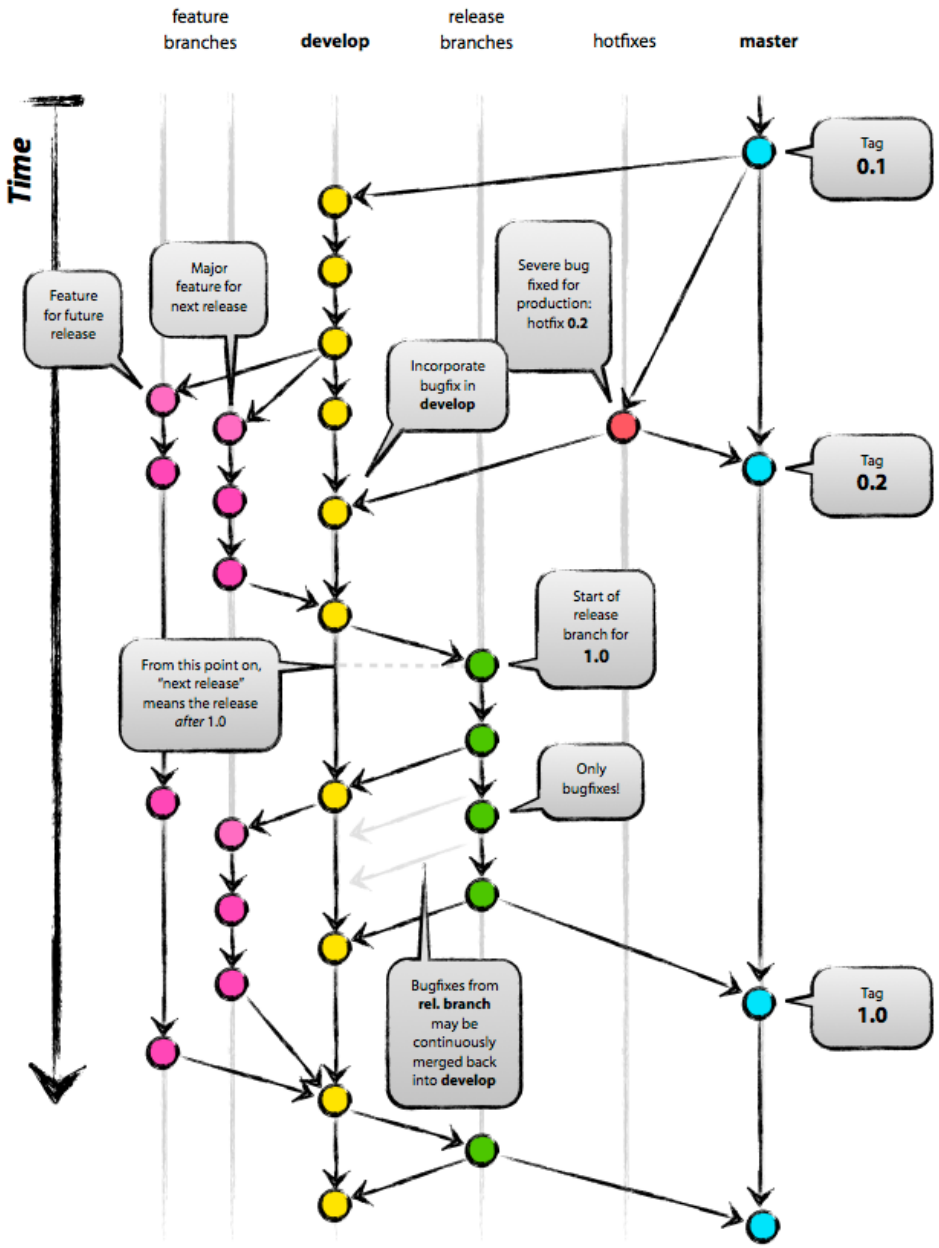
Version Control

Branch management

Merging branch (very complex to achieve with SVN)



Version Control



Branch management

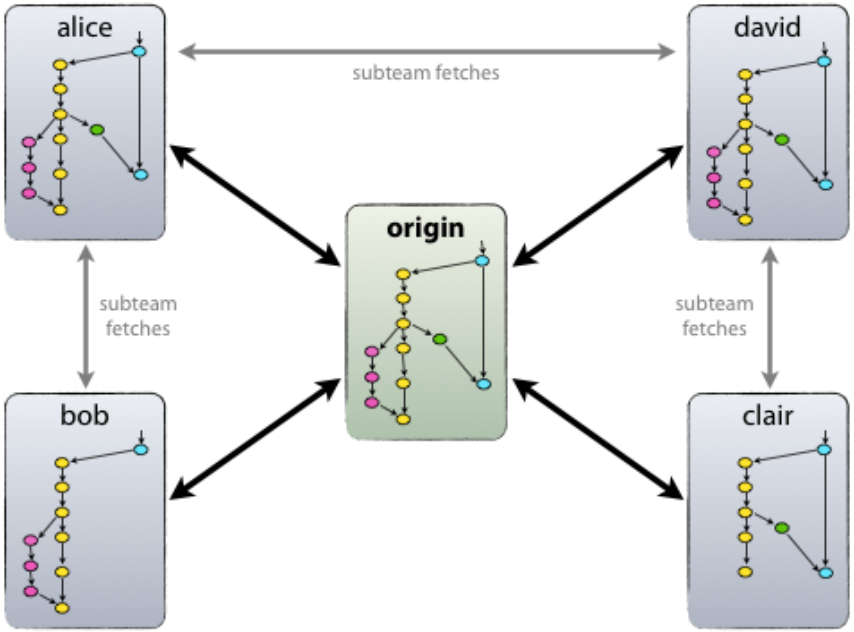
Classical organisation of a project into branches

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

Version Control

Branch management

- Each user can work on a particular branches (or branches)
- Branches can be synchronized between users



Version Control

Web 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

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 form the beginning
- Detect integration problems at an early stage
- Always have the system running
  - Tests, demos, discussion with the customers

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

Continuous integration

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

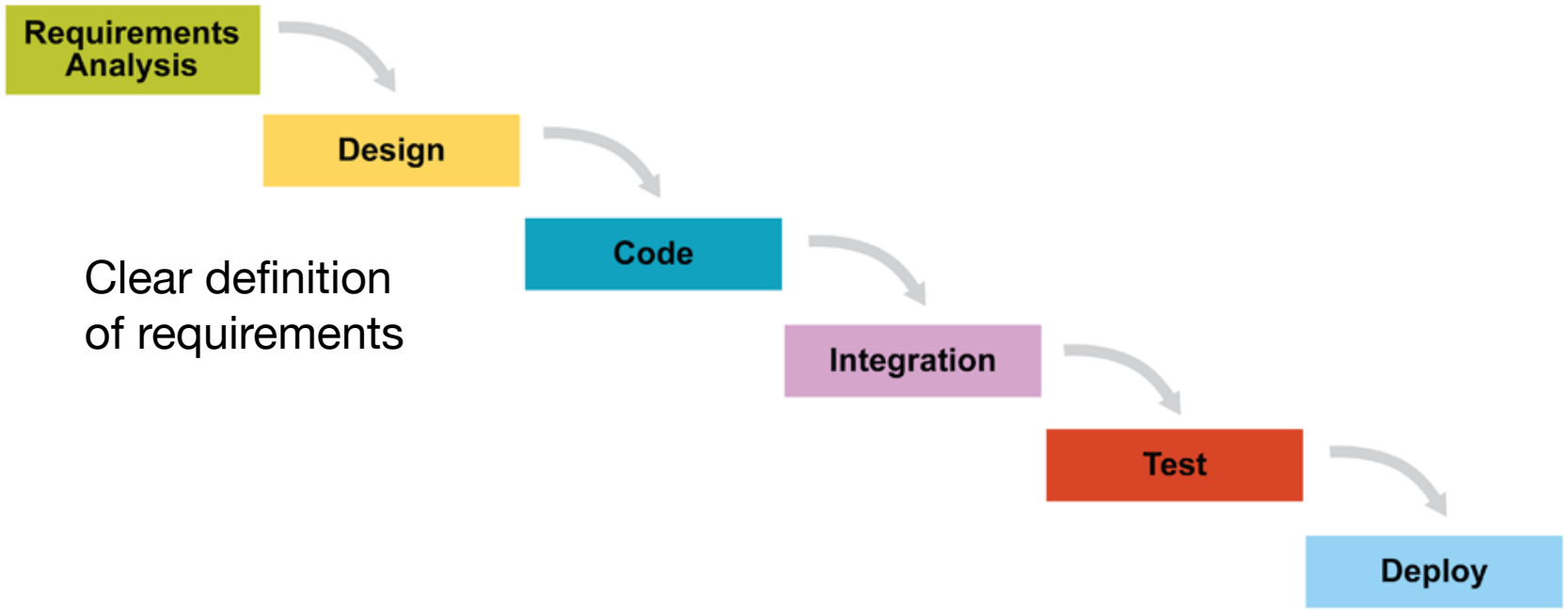
Software development methods

38

Software development

Historic Engineering methodologies

Example: Waterfall



39

Software development

Is software development end-result predictable?

Yes in some cases…

NASA programs

Usually, requirements are unpredictable
(especially for software involving interactions 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

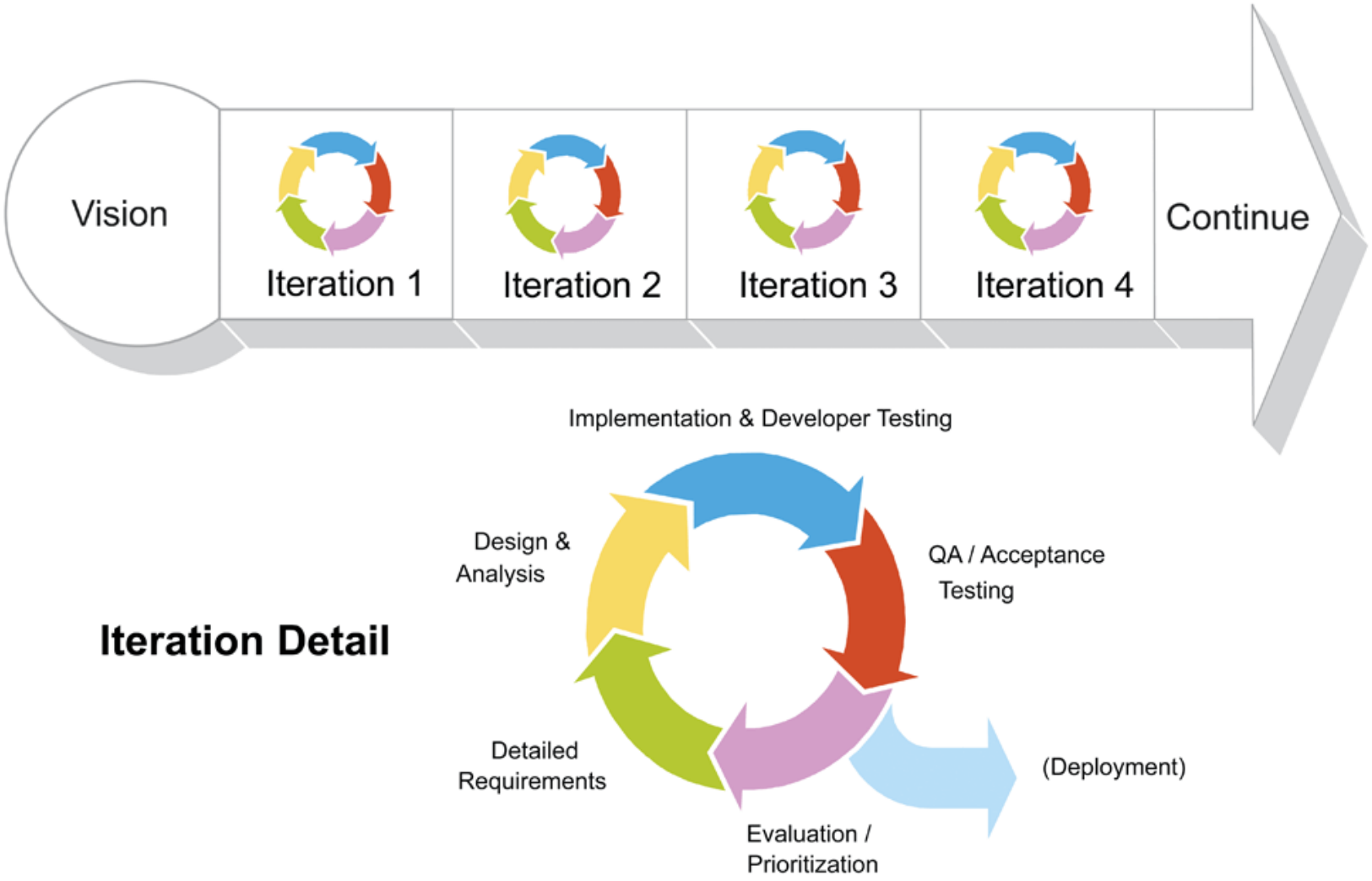
⇒ Requirements should be flexible in these cases

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 professionals that
      - Make the technical decisions
      - Evaluate the time required to perform the tasks

http://agilemanifesto.org [41]

Agile methods



Agile methods

Examples

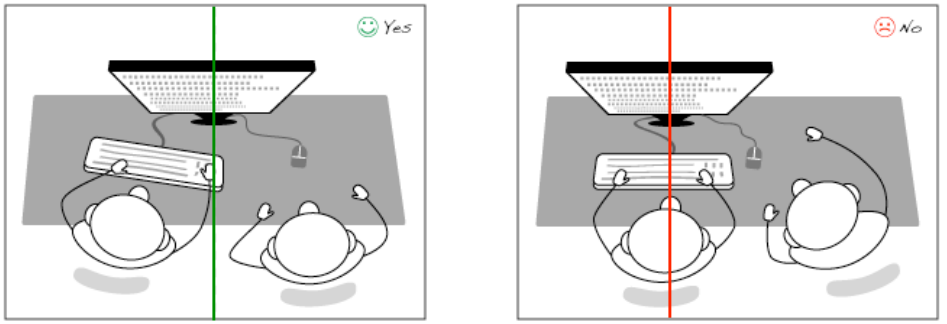
- XP (Extreme Programming)
  - Test driven development, pair programming
- Scrum
- Crystal
  - Safety, efficiency, habitability (less disciplined than XP)
- Open source process
  - Distributed contributors, parallelized debugging
- Lean software development (Lean development @ Toyota)
  - Just in time, Jidoka ("automation with a human touch")
- RUP (Rational Unified Process)
  - Use case driven, iterative, architecture centric

Pair 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 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

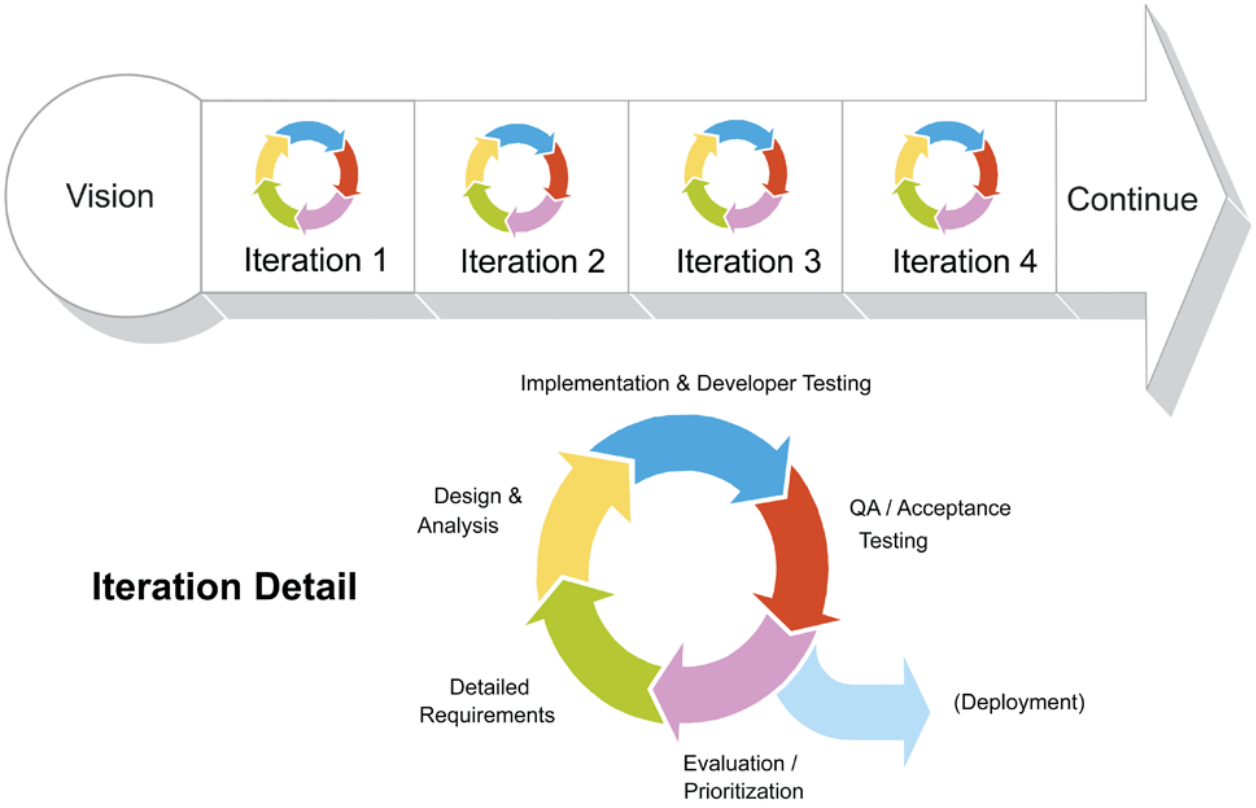
Pair programming

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 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)



Scrum

Roles

Product Owner (one person)

- Responsible for products vision
- Constantly re-prioritizes the Product Backlog
- Accepts or rejects product increment

Development team

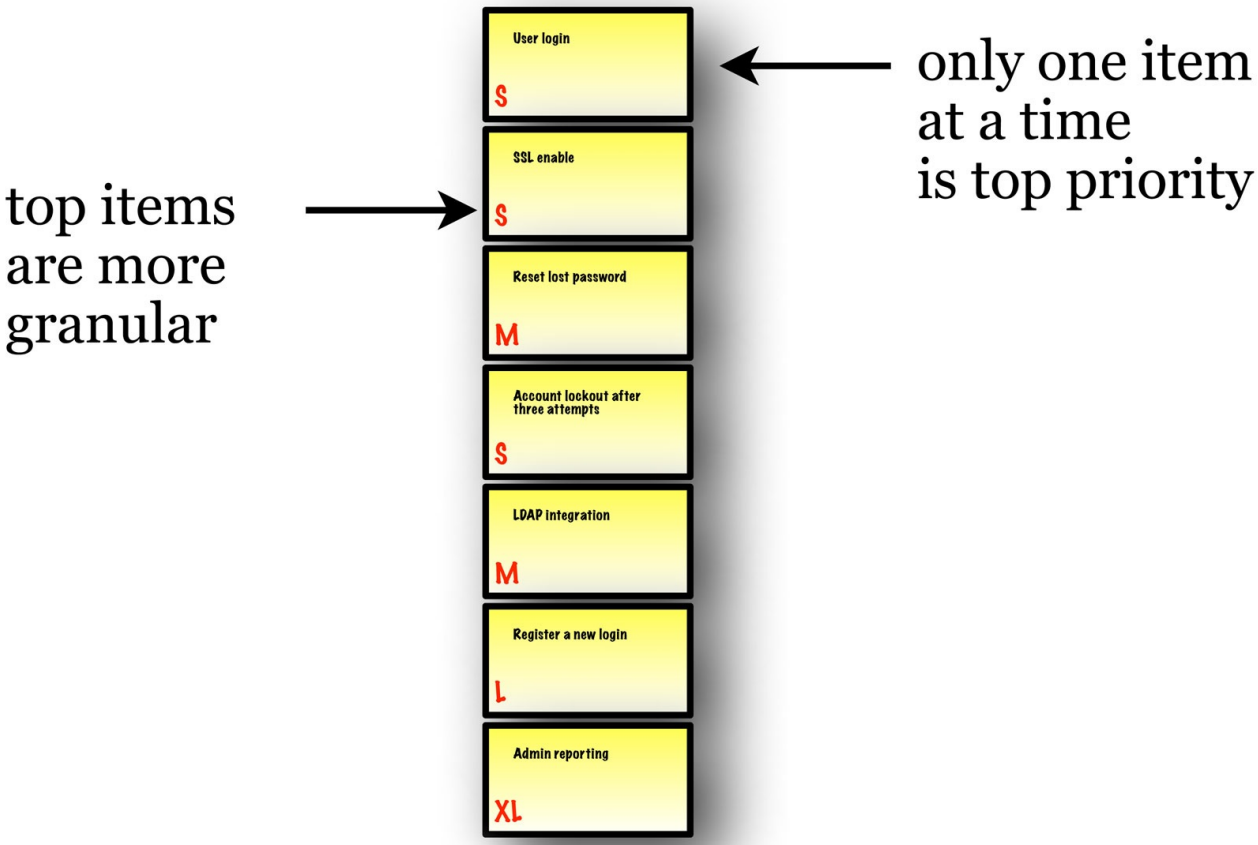
- Self-organized
- Negotiates commitments with the Product Owner
- Has autonomy regarding how to reach commitments
- Intensely collaborative

Master

- Facilitates the Scrum process
- Helps resolve issues
- Shields the team from external inferences and distraction
- Has no management authority

Scrum

Product Backlog



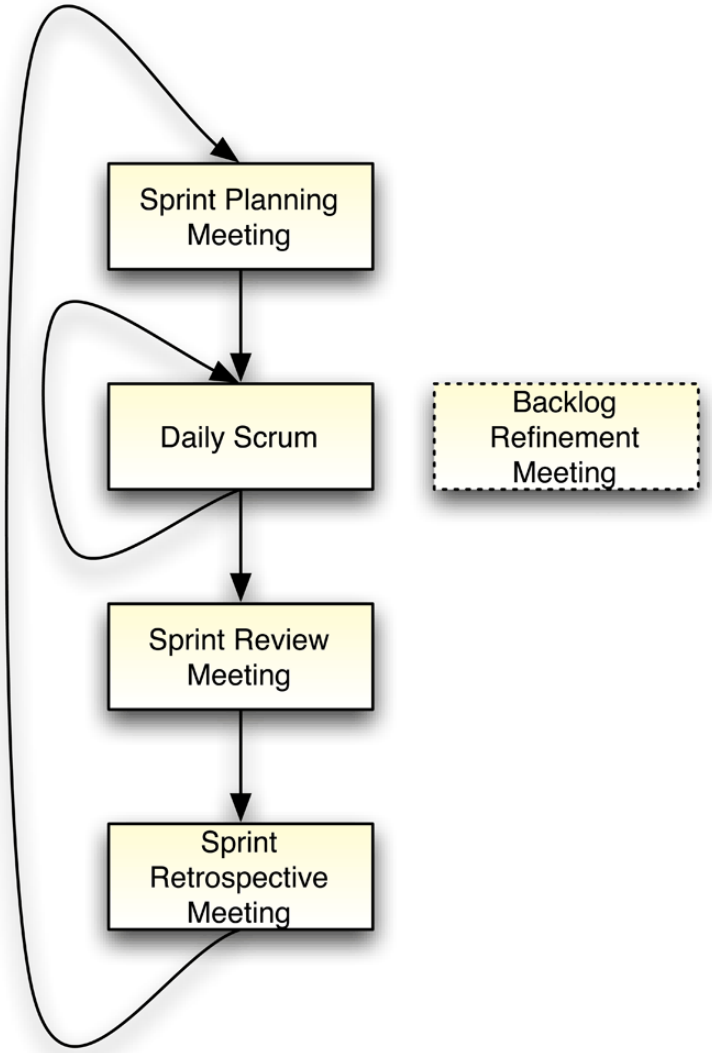
Scrum

Sprint

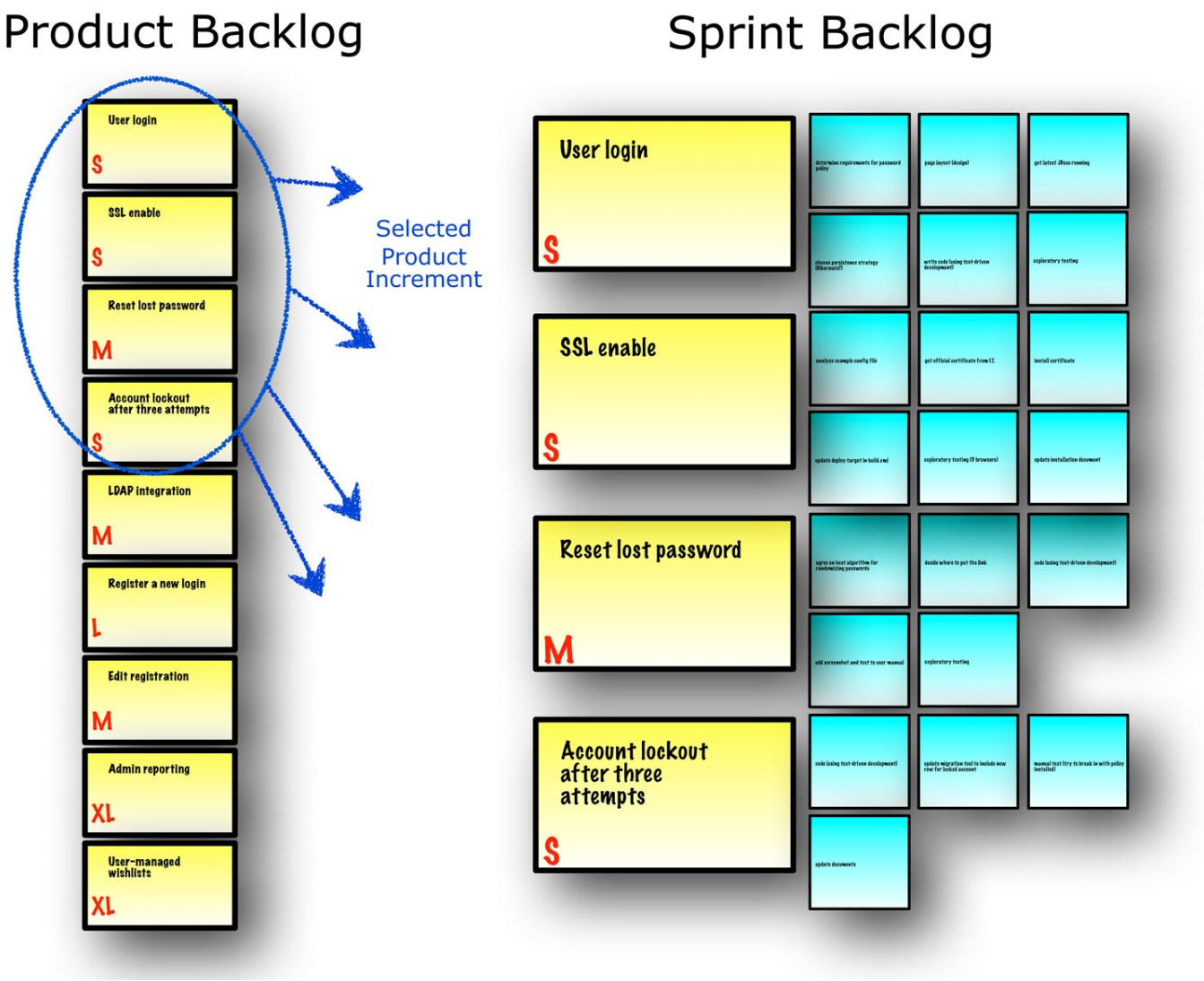
Planning Meeting

Negotiate which Product Backlog items will be processed

Break items into a list of sprint tasks



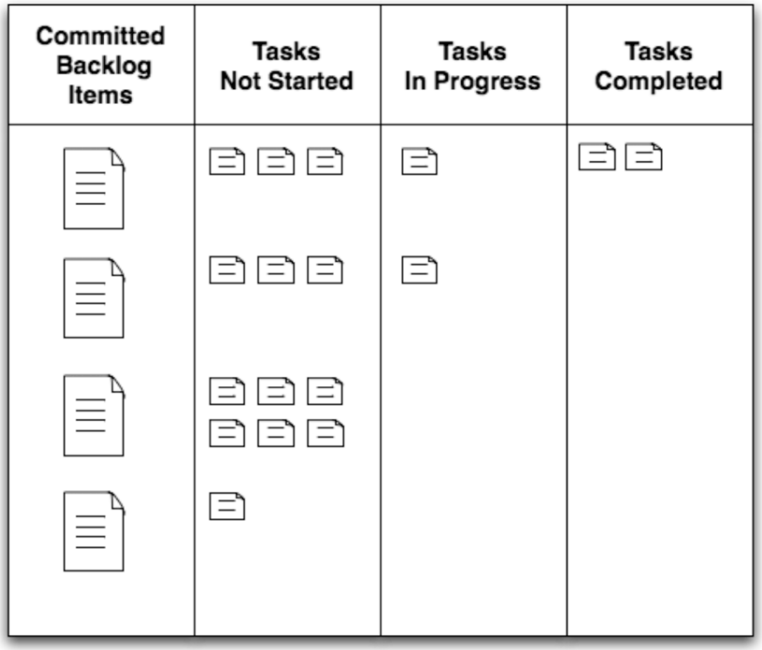
Scrum


Product Backlog
Sprint Backlog
User login
S
SSL enable
S
Reset lost password
M
Account lockout after three attempts
S
LDAP integration
M
Register a new login
L
Edit registration
M
Admin reporting
XL
User-managed wishlists
XL
Selected Product Increment
User login
S
SSL enable
S
Reset lost password
M
Account lockout after three attempts
S

Scrum

Sprint

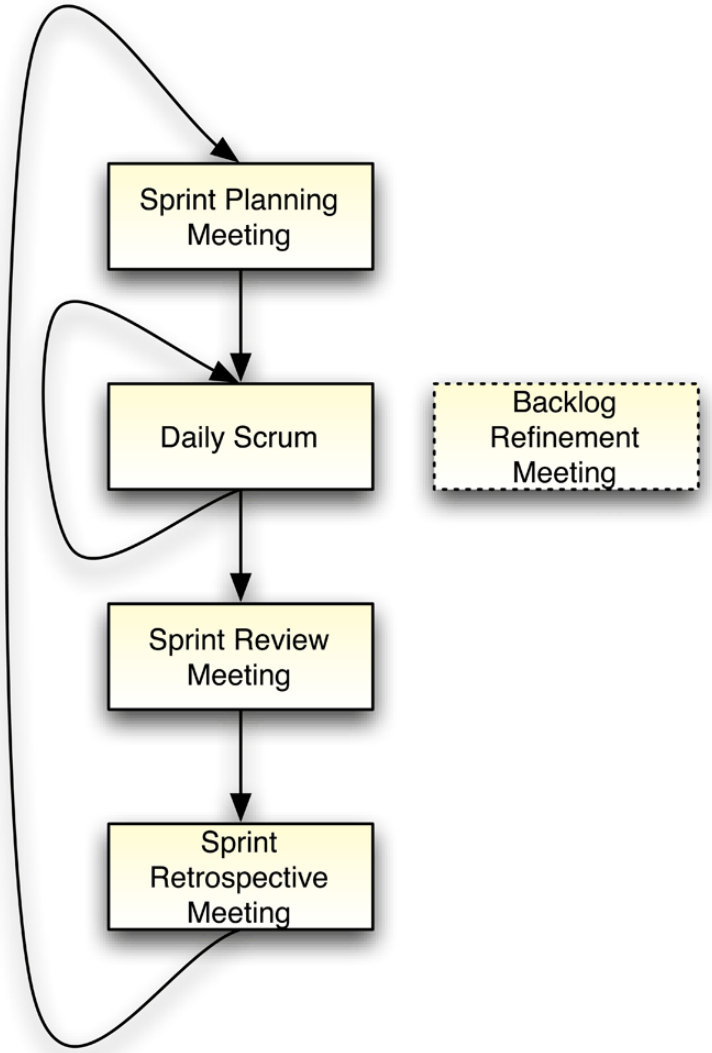
- Planning Meeting
- Daily Meeting
  - Same time and place
  - 15 minutes, standing up
  - Summarize work of previous day, work of today, issues
  - Maintain tasks list (not started, in progress, done), issues list and burn-down chart
  - Product Owner may attend



Example of Sprint Backlog

Scrum

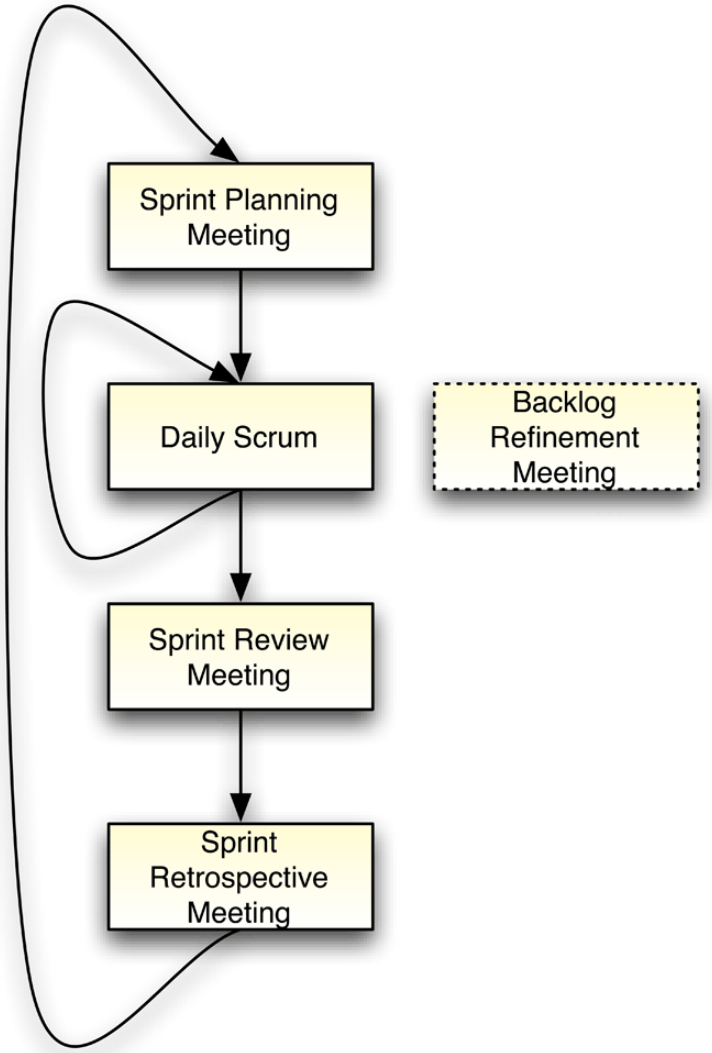
- Sprint
  - Planning Meeting
  - Daily Meeting
  - Review Meeting
    - Demonstrate the working product increment to the Product Owner
    - Product Owner declares which items are done
    - Unfinished items return to the Product Backlog
    - Master proposes new items for the Product Backlog



Scrum

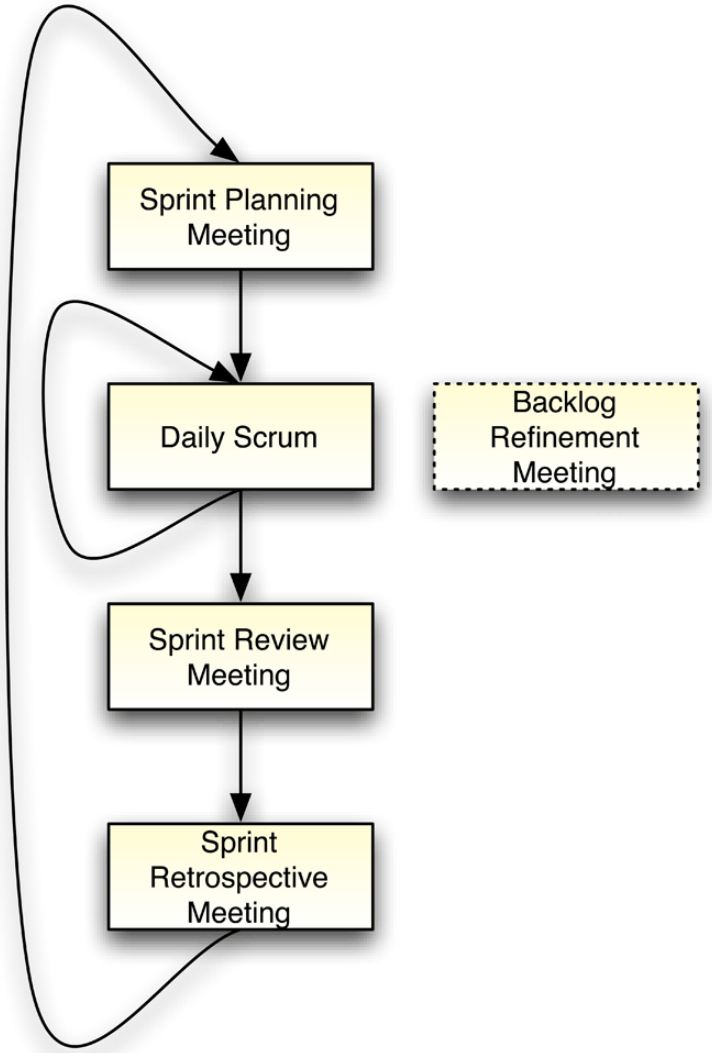
Sprint

- Planning Meeting
- Daily Meeting
- Review Meeting
- Retrospective Meeting
  - Team reviews its own process
  - Team adapts it for future Sprints
  - Master has to manage the psychological aspects of the meetings



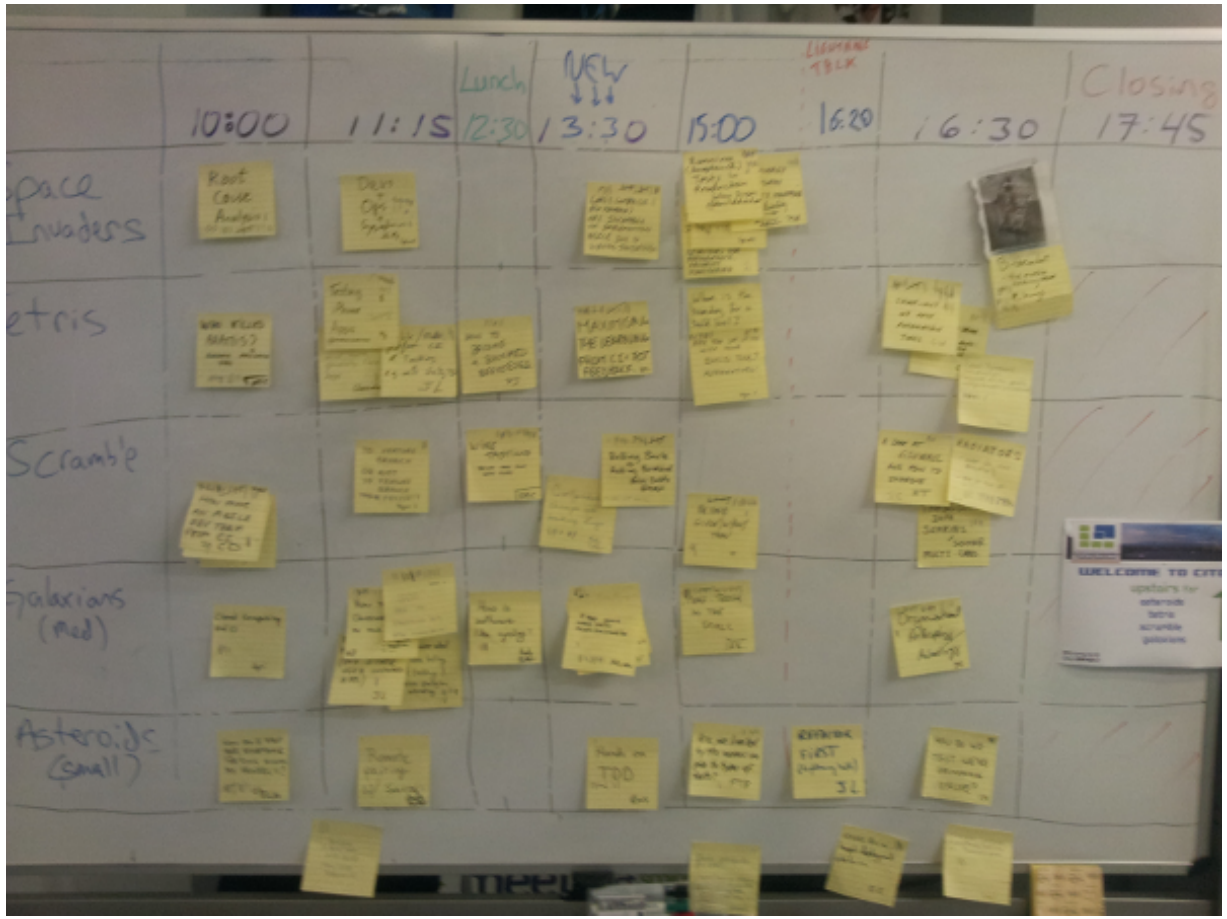
Scrum

- Sprint
  - Planning Meeting
  - Daily Meeting
  - Review Meeting
  - Retrospective Meeting
  - Backlog Refinement Meeting
    - Items are usually too large or poorly understood
    - Refine items into smaller ones
    - Master can help



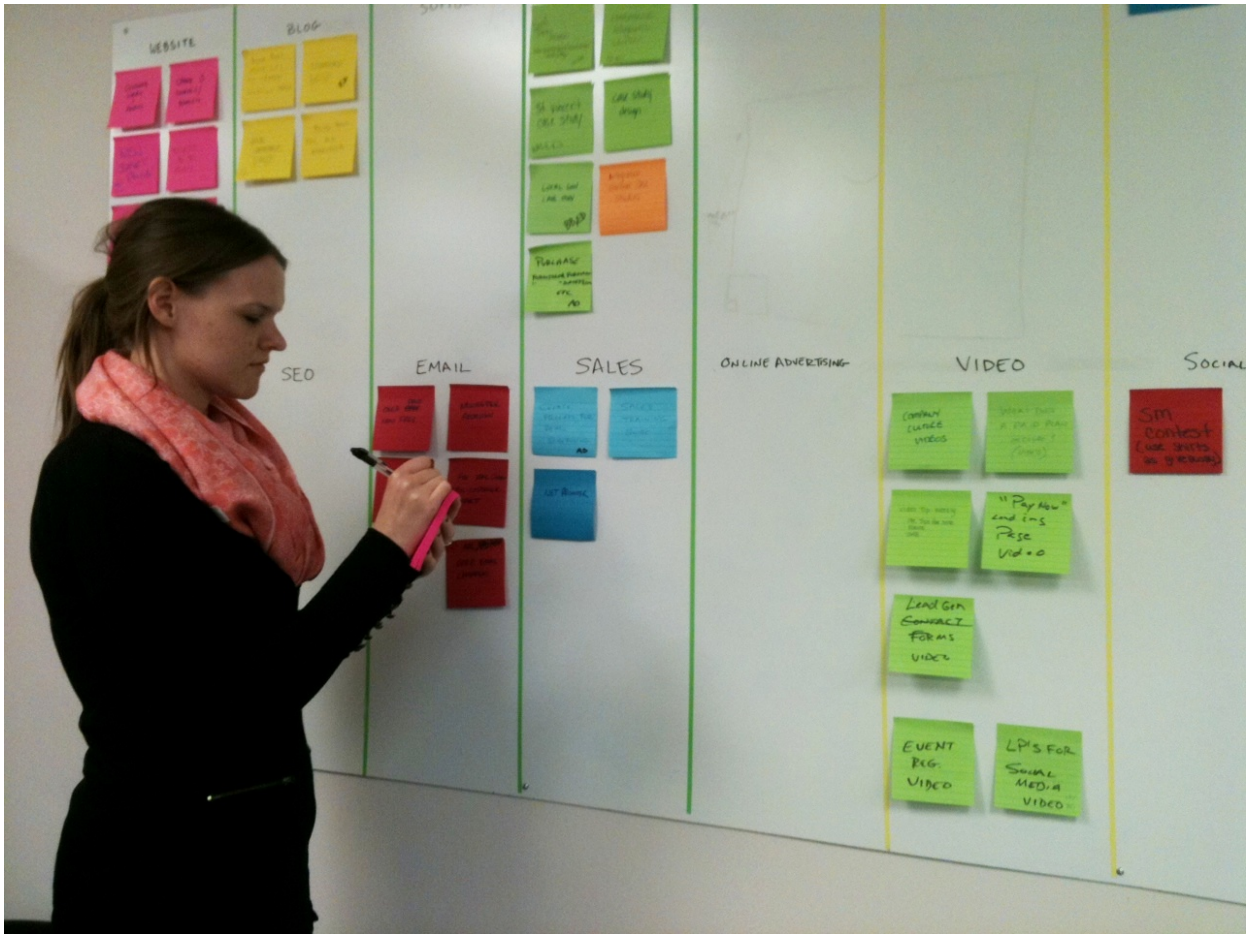
Scrum

Feedback to the team: wallboard

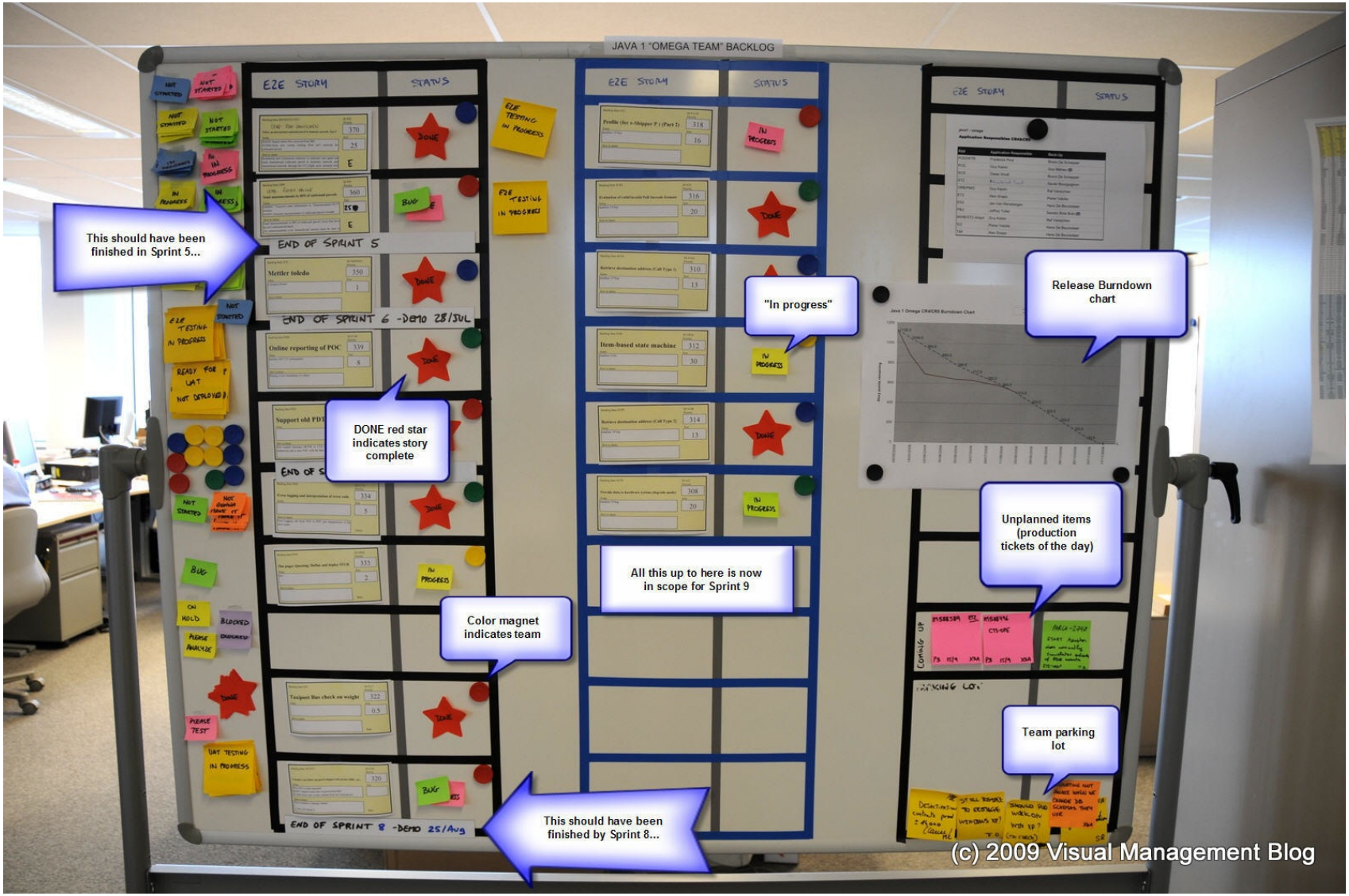


Scrum

Feedbacks to the team: wallboard



Feedback to the team: wallboard



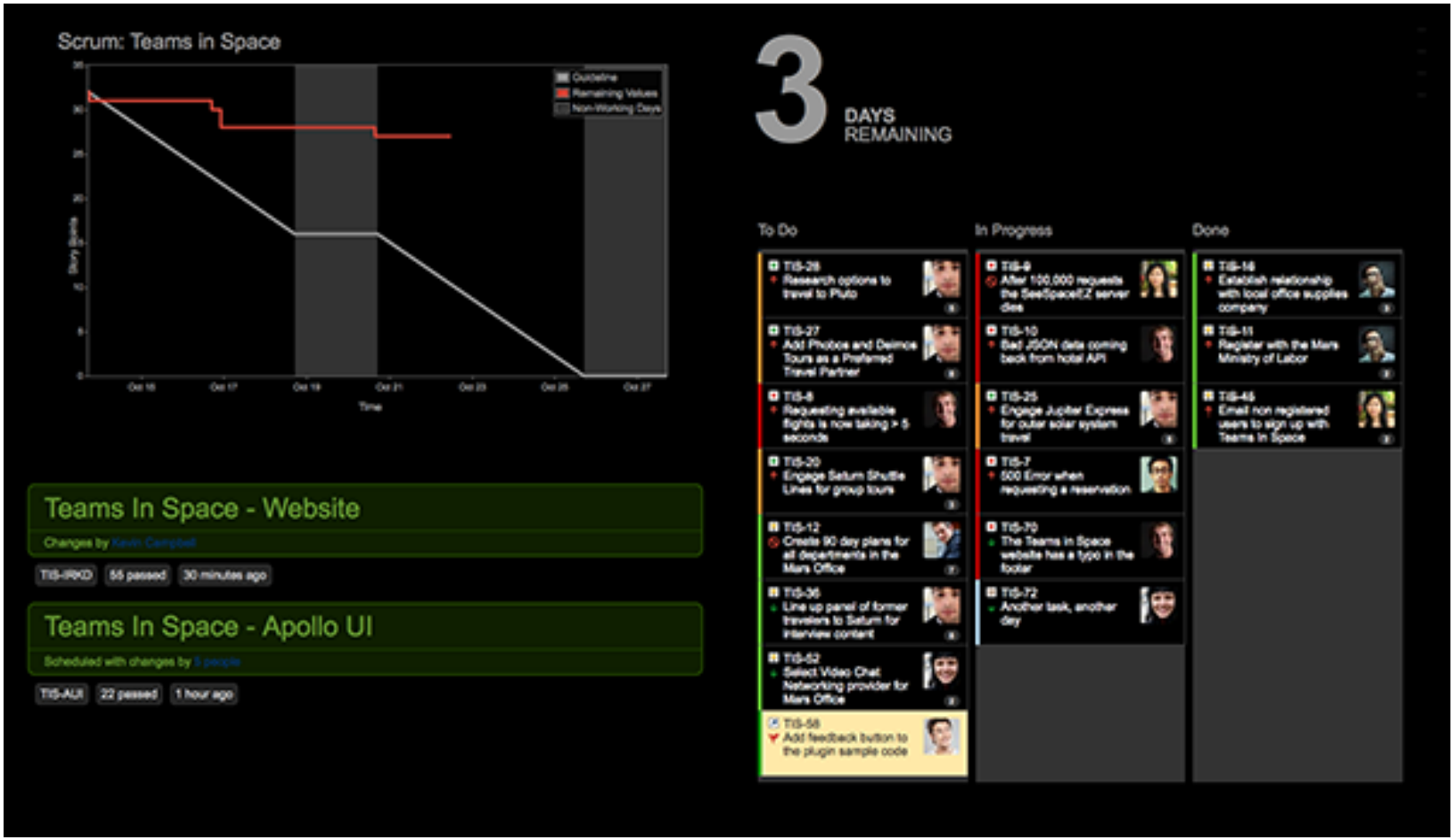
Scrum

Feedback to the team: LEGO Bit planner

https://vimeo.com/75500464

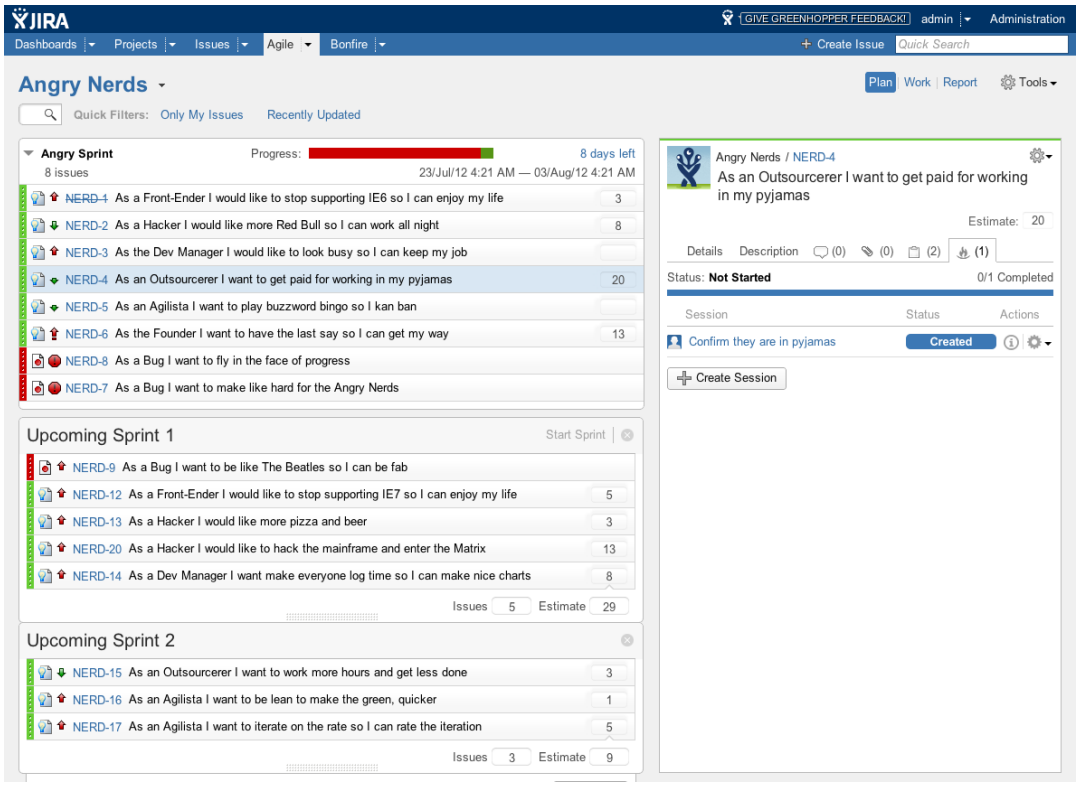
Scrum

Feedback to the team: wallboard



Scrum

Software to manage Scrum projects



JIRA - 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

Scrum

- http://scrumreferencecard.com