## Working in Teams


1

## Lecture outline

- Why is teamwork hard?
- Not getting into each other's way
- Positive teamwork

2

## Team pros and cons

- Benefits
  - Attack bigger problems in a short period of time
  - Utilize the collective experience of everyone
- Risks
  - Communication and coordination issues
  - Groupthink: diffusion of responsibility; going along
  - Working by inertia; not planning ahead
  - Conflict or mistrust between team members

3

## Communication: powerful but costly!

- Communication requirements increase with increasing numbers of people
- Everybody to everybody: quadratic cost
- Every attempt to communicate is a chance to miscommunicate
- But *not* communicating will *guarantee* miscommunication

4

## What about conflicts?

What can cause conflicts?

- Two people want to work on the same file
  - Google docs lets you do that

But…

- What about same line?
- What about relationships between different parts of the file?
- What about design decisions?

5

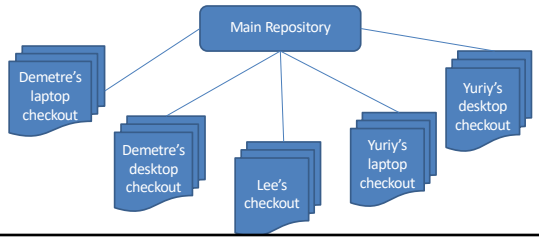
## Version control

Version control aims to allow multiple people to work in parallel.

6

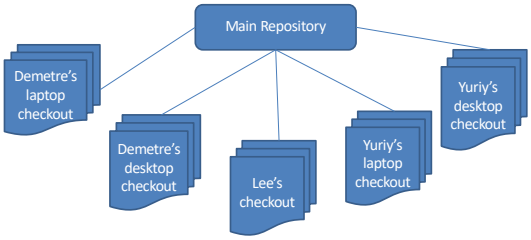
## Centralized version control

- (old model)
- Examples: Concurrent Versions System (CVS)
  Subversion (SVN)




7

## Doing work




- I update my checkout (working copy)
- I edit
- I update my checkout again
- I merge changes if necessary
- I commit my changes to the Main

8

## Problems with centralized VC

- What if I don't have a network connection?
- What if I am implementing a big change?
- What if I want to explore project history later?

9

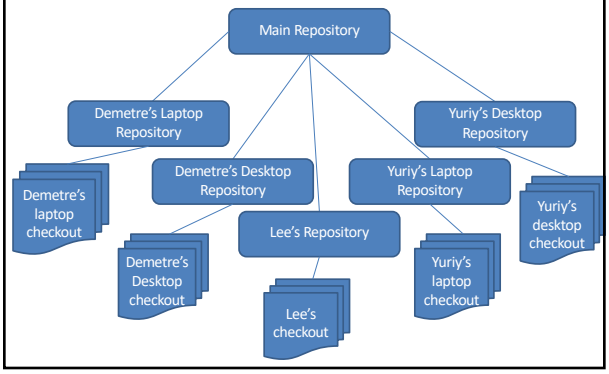
## Distributed version control

(new model)

- Examples: Mercurial (Hg), Git, Bazaar, Darcs, …
- Local operations are fast (and possible)
- History is more accurate
- Merging algorithms are far better

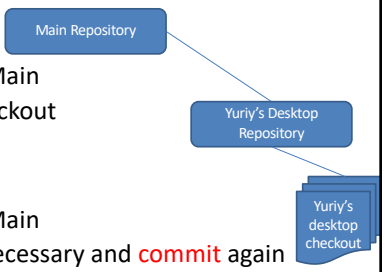
10

## Distributed version control model




11

## Doing work

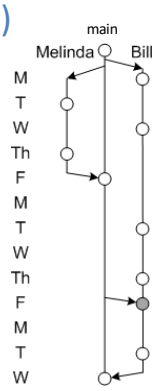



- I pull from the Main
- I update my checkout
- I edit
- I commit
- I pull from the Main
- I merge tips if necessary and commit again
- I push my changes to the Main

12

## History view (log)

- Bill and Melinda work at the same time
- At the end, all repositories have the same, rich history




13

## How frequent are conflicts?

**histories of 9 open-source projects:**

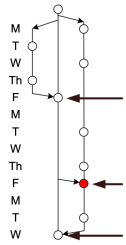
| | |
|---|---|
| size: | 26K–1.4MSLoC |
| developers: | 298 |
| versions: | 140,000 |

Perl5, Rails, Git, jQuery, Voldemort, MaNGOS, Gallery3, Samba, Insoshi

Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin, Early Detection of Collaboration Conflicts and Risks, IEEE TSE, vol. 39, no. 10, October 2013, pp. 1358–1375, https://doi.org/10.1109/TSE.2013.28

14

## How frequent are conflicts?

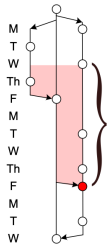



**How frequent are textual conflicts?**

16% of the merges have textual conflicts.

15

## How frequent are conflicts?




**How frequent are textual conflicts?**

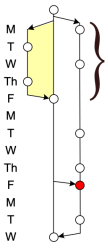
16% of the merges have textual conflicts.

**How long do textual conflicts persist?**

Conflicts live a mean of 9.8 and median of 1.6 days. The worst case was over a year.

16

## How frequent are conflicts?




**How frequent are textual conflicts?**

16% of the merges have textual conflicts.

**How long do textual conflicts persist?**

Conflicts live a mean of 9.8 and median of 1.6 days. The worst case was over a year.

**How long do textually-safe merges persist?**

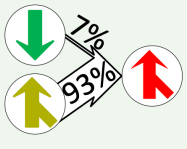
Textually-safe merges live a mean of 11.0 and median of 1.9 days.

17

## Where do conflicts come from?

**Where do textual conflicts come from?**

93% of textual conflicts developed from safe merges.




18

## Higher order conflicts

| program | conflicts | | | safe |
|---|---|---|---|---|
| | textual | build | test | merges |
| Git | 17% | <1% | 4% | 79% |
| Perl5 | 8% | 4% | 28% | 61% |
| Voldemort | 17% | 10% | 3% | 69% |

**Does merged code fail to build or fail tests?**

One in three conflicts are build or test conflicts.

19

## What VC does the cloud provide?

- ~~code.google.com has SVN and Hg~~
- bitbucket.org has ~~Hg and~~ git
- github.com has git
- sourceforge.net has SVN, ~~CVS,~~ git, Hg, ~~Bazaar~~
- You can run whatever you want locally

20

## Lecture outline

- Why is teamwork hard?
- Not getting into each other's way

➔Positive teamwork

21

## Team structures

- Tricky balance among
  - progress on the project/product
  - expertise and knowledge
  - communication needs

"A team is a set of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable."

– Katzenbach and Smith

22

## Common SW team responsibilities

- Project management
- Functional management
- Developers: programmers, testers, integrators
- Lead developer/architect ("tech lead")
- These could be all different team members, or some members could span multiple roles.
- Key: Identify and stress roles **and** responsibilities

23

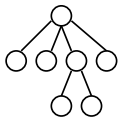
## Issues affecting team success

- Presence of a shared mission and goals
- Motivation and commitment of team members
- Experience level
  - and presence of experienced members
- Team size
  - and the need for bounded yet sufficient communication
- Team organization
  - and results-driven structure
- Reward structure within the team
  - incentives, enjoyment, empowerment (ownership, autonomy)

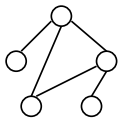
24

## Team structure models

- Dominion model
  - Pros
    - clear chain of responsibility
    - people are used to it
  - Cons:
    - single point of failure at the commander
    - less or no sense of ownership by everyone



- Communion model
  - Pros
    - a community of leaders, each in his/her own domain
    - inherent sense of ownership
  - Cons
    - people aren't used to it (and this scares them)



25

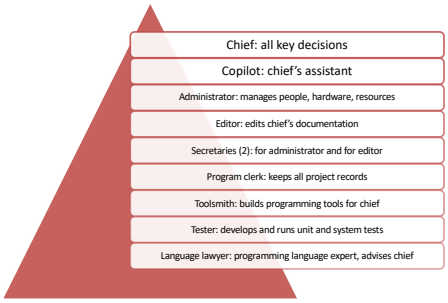
## Team leadership

- Who makes the important product-wide decisions in your team?
  - One person?
  - All, by unanimous consent?
  - Other options?...
  - Is this an unspoken or an explicit agreement among team members?

26

## Surgical/Chief Programmer Team

[Baker, Mills, Brooks]




27

## Microsoft's team structure

[microsoft.com]

- **Program Manager**. Leads the technical side of a product development team, managing and defining the functional specifications and defining how the product will work.
- **Software Design Engineer.** Codes and designs new software, often collaborating as a member of a software development team to create and build products.
- **Software Test Engineer.** Tests and critiques software to assure quality and identify potential improvement opportunities and projects.

28

## Toshiba Software Factory [Y. Matsumoto]

- Late 1970's structure for 2,300 software developers producing real-time industrial application software systems (such as traffic control, factory automation, etc.)
- Unit Workload Order Sheets (UWOS) precisely define a software component to be built
- Assigned by project management to developers based on scope/size/skills needed
- Completed UWOS fed back into management system
- Highly measured to allow for process improvement

29

## Common factors in good teams

- Clear roles and responsibilities
  - Each person knows and is accountable for their work
- Monitor individual performance
  - Who is doing what, are we getting the work done?
- Effective communication system
  - Available, credible, tracking of issues, decisions
  - Problems aren't allowed to fester ("boiled frogs")
- Fact based decisions
  - Focus on the facts, not the politics, personalities, …

30

## Motivation

What motivates you?

- Achievement
- Recognition
- Advancement
- Salary
- Possibility for growth
- Interpersonal relationships
  - Subordinate
  - Superior
  - Peer
- Status
- Technical supervision opportunities
- Company policies
- Work itself
- Work conditions
- Personal life
- Job security
- Responsibility
- Competition
- Time pressure
- Tangible goals
- Social responsibility
- Other?

31

## De-motivators

- What takes away your motivation?
  - Micro-management or no management
  - Lack of ownership
  - Lack of effective reward structure
    - Including lack of simple appreciation for job well done
  - Excessive pressure and resulting "burnout"
  - Allowing "broken windows" to persist
  - Lack of focus in the overall direction
  - Productivity barriers
    - Asking too much; not allowing sufficient learning time; using the wrong tools
  - Too little challenge
  - Work not aligned with personal interests and goals
  - Poor communication inside the team

32