[IF977] Engenharia de Software

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

AssertLab
Advanced Software and Systems Engineering Research Technologies

Centro de Informática

Universidade Federal de Pernambuco

Licença do material

Este Trabalho foi licenciado com uma Licença

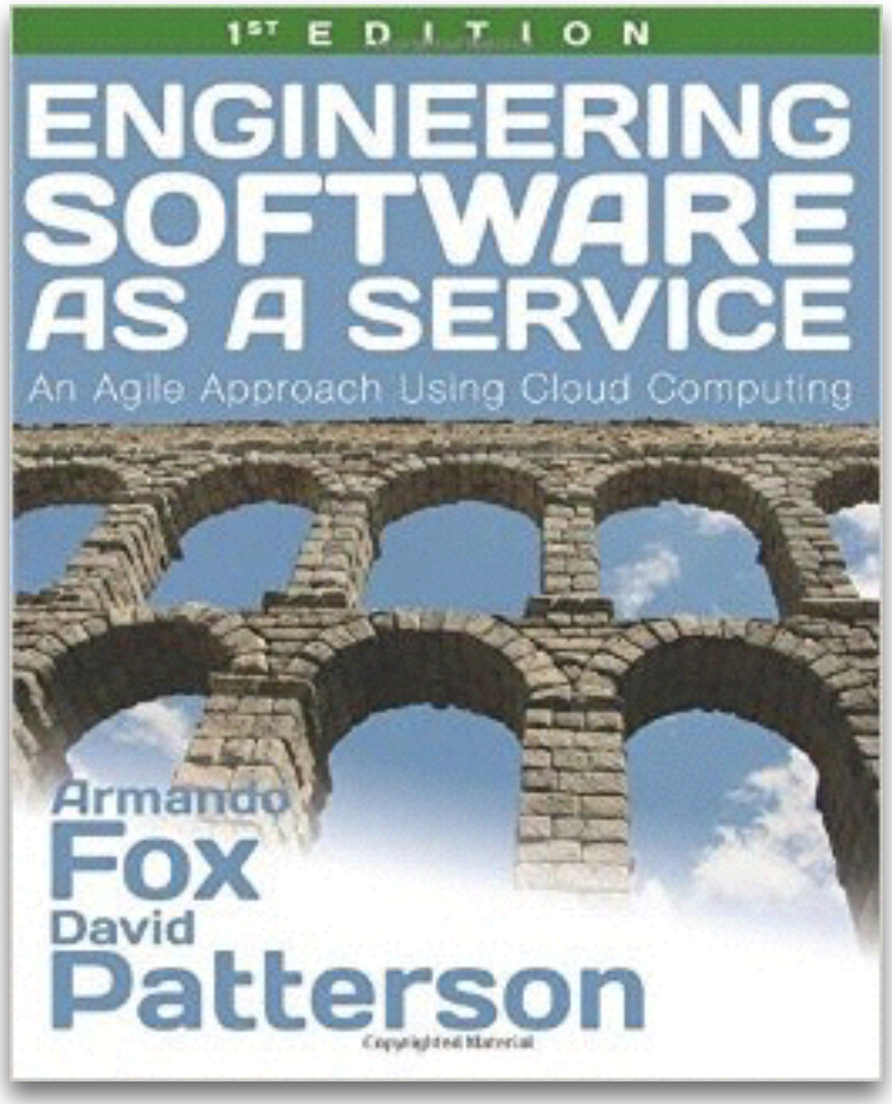
Creative Commons - Atribuição-NãoComercial-Compartilhalgual 3.0 Não Adaptada.

Mais informações visite

http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt

Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
  - http://bit.ly/TDOA5L
- SWEBOK
  - Guide to the Software Engineering Body of Knowledge (SWEBOK): http://www.computer.org/web/swebok
- Engineering Software as a Service: An Agile Approach Using Cloud Computing (Beta Edition)
  - http://www.saasbook.info/



40 Years of Version Control

SCCS & RCS (1970s)

CVS (1986)

Subversion (2001)

Git (2005)

Image © TheSun.au

Design Reviews, Code Reviews, Plan-And-Document Perspective on Project Management

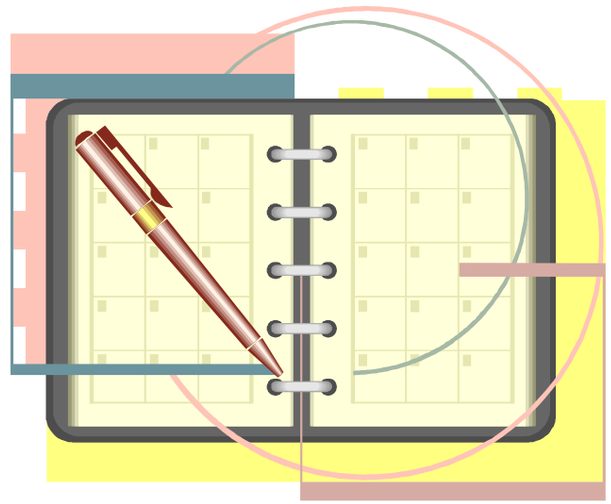
(Engineering Software as a Service §10.3, §10.7-10.9)

Design/Code Reviews

- **Design review**: meeting where authors present design
  - benefit from experience of attendees
- **Code review**: held after design implemented

Review Agenda

- Prepare with list of questions/ issues like to be discussed
- Start with High-Level Description of Customer Desires



- Give SW architecture, showing APIs and highlighting Design Patterns (Ch. 11) at each level of abstraction
- Go through code and documentation: project plan, schedule, testing plan, …: *Verification &Validation* (V&V) of project

Good Meetings: SAMOSAS

- **S**tart and stop meeting promptly
- **A**genda created in advance; no agenda, no meeting
- **M**inutes recorded so everyone can recall results
- **O**ne speaker at a time; no interrupting talker
- **S**end material in advance, since reading is faster
- **A**ction items at end of meeting, so know what each should do as a result of the meeting
- **S**et the date and time of the next meeting

Minutes and action items record results of meeting, start next meeting by reviewing action items

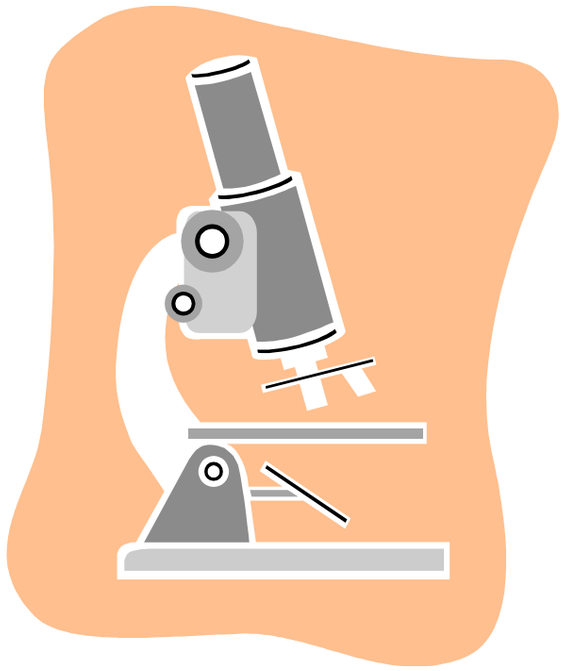
Better Reviews?

- Shalloway*: formal design and code reviews often too late in process to make big impact
- Recommends instead have earlier, smaller meetings: "approach reviews".
  - A few senior developers assist team in coming up with an approach to solve the problem
  - Group brainstorms about different approaches
- If do a formal design review, suggests 1st hold a "mini-design review" to prepare

**Alan Shalloway, Agile Design and Code Reviews, 2002, www.netobjectives.com/download/designreviews.pdf*

Quantitative Metrics vs. Reviews?

- Study many projects to record averages, set baseline for new projects, compare this one:
  - Code size (KLOC), Effort (months)
  - Milestones fulfilled, Test cases done
  - Defect discovery,repair rate / month
- Correlated so that can replace reviews??



However, we are still quite a long way from this ideal situation, and there are no signs that automated quality assessment will become a reality in the foreseeable future— Sommerville 2010

Agile & Reviews?

- Pivotal Labs – Pair programming means continuous reviewing
  => no special reviews
- GitHub – ***Pull Requests*** instead of reviews
  - A developer requests his/her code be integrated into main codebase
  - All developers see each request & determine how might affect own code
  - If concern, online discussion on pull request
  - As occur daily, “mini-reviews” continuously
    => no special reviews

Pergunta

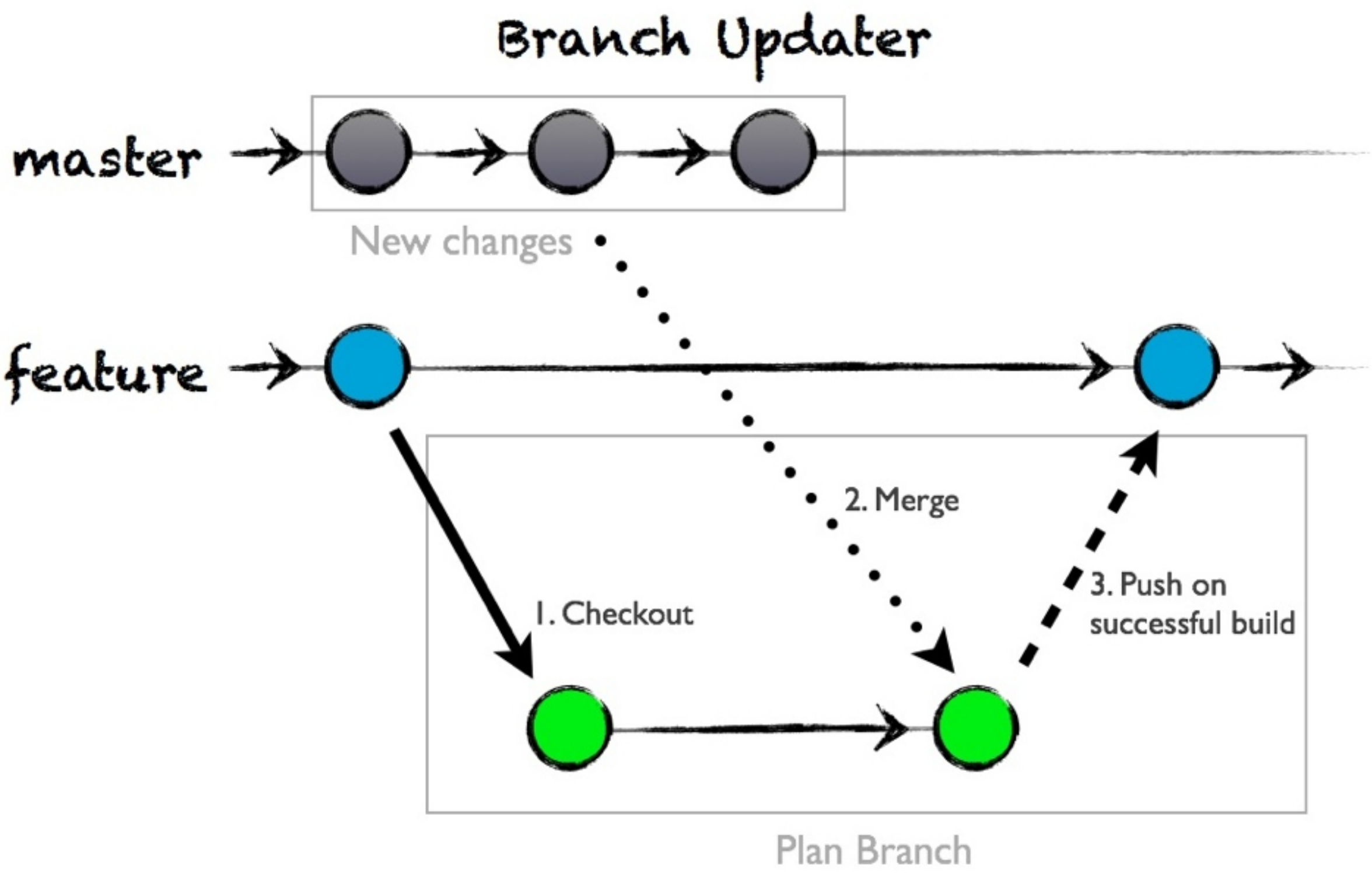
Which expression statement regarding Reviews and Meetings is **FALSE**?

A. Intended to improve the quality of the software product using the wisdom of the attendees

B. They result in technical information exchange and can be highly educational for junior people

C. Can be beneficial to both presenters and attendees

D. The A's in SAMOSA stands for Agenda and Action items, which are optional pieces of good meetings

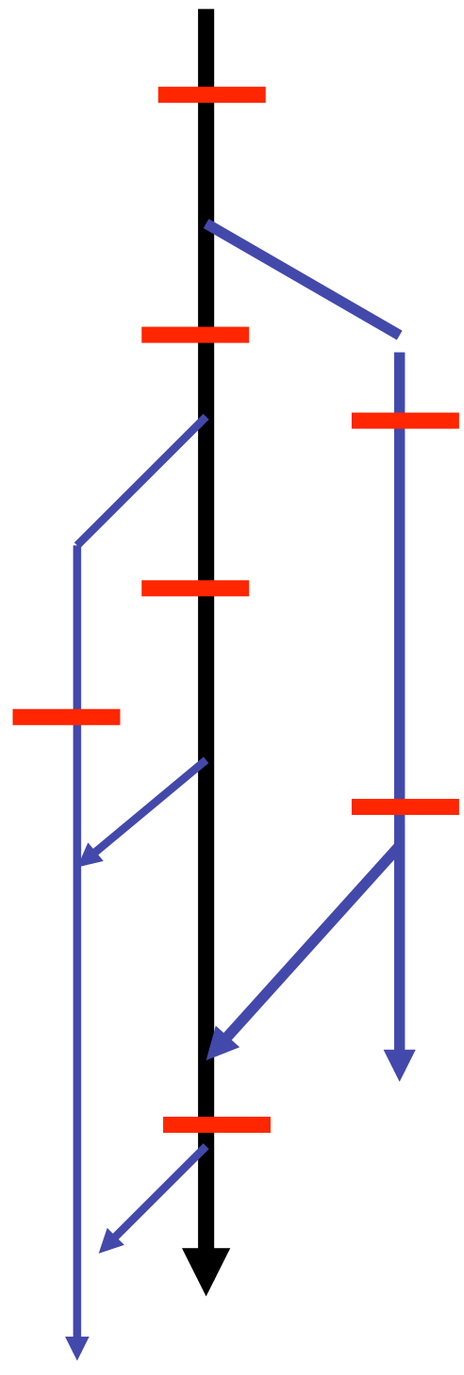


Effective Branching part 1

Branch Per Feature

Branches

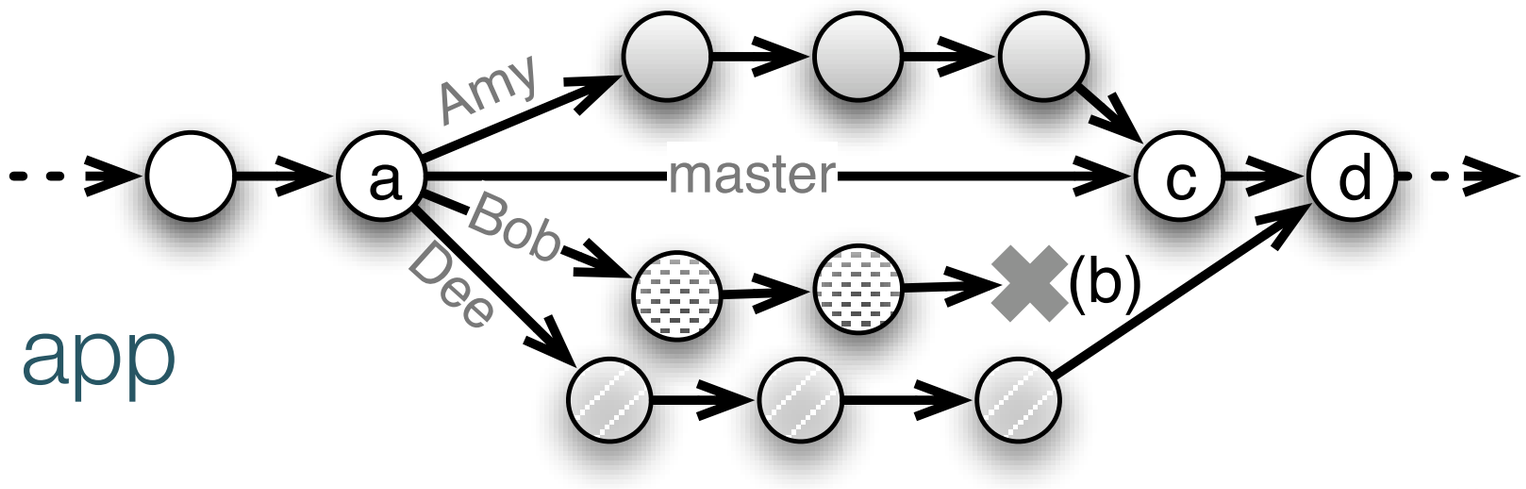
- Development **master** vs. **branches**
  - Creating branch is cheap!
  - switch among branches: checkout
- Separate commit histories per branch
- *Merge* branch back into master
  - ...or with *pushing* branch changes
  - Most branches eventually die
- Killer use case for agile SaaS:
  branch per feature



Creating new features without disrupting working code

- To work on a new feature, create new branch just for that feature
  - many features can be in progress at same time
- Use branch *only* for changes needed for this feature, then merge into master
- Back out this feature ⇔ undo this merge

In well-factored app,
1 feature shouldn't
touch many parts of app



Mechanics

- Create new branch & switch to it

```
git branch CoolNewFeature
```

```
git checkout CoolNewFeature
```
⬅ current branch

- Edit, add, make commits, etc. on branch

- Push branch to origin repo (optional):

```
git push origin CoolNewFeature
```

  - creates tracking branch on remote repo

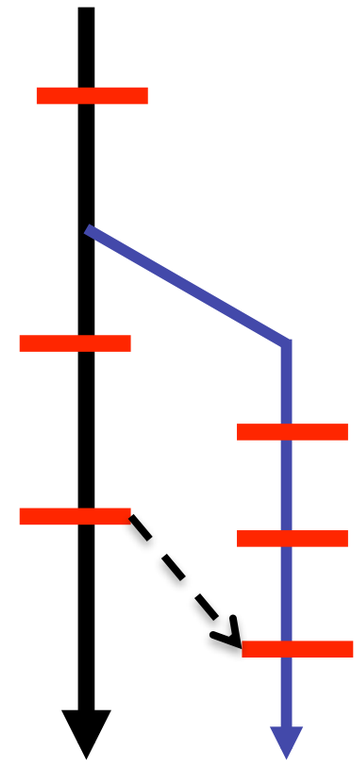
- Switch back to master, and merge:

```
git checkout master
```

```
git merge CoolNewFeature
```
⬅ warning!!

Rebasing

- Rebase branch against x ==
  try to pretend it was branched from x
- Why do it?
- Must resolve merge conflicts
  manually, as with regular merge



- Optional: can squash multiple commits into one, to simplify later merge
- Key to not having brain explode: think in terms of *changesets*, not versions

Pergunta

If you try to push to a remote and get a “non-fast-forward (error): failed to push some refs”, which statement is **FALSE**?

A. Some commits present at remote are not present on your local repo

B. You need to do a merge/pull before you can complete the push

C. You need to manually fix merge conflicts in one or more files

D. Your local repo is out-of-date with respect to the remote

Effective Branching part 2: Branches & Deploying

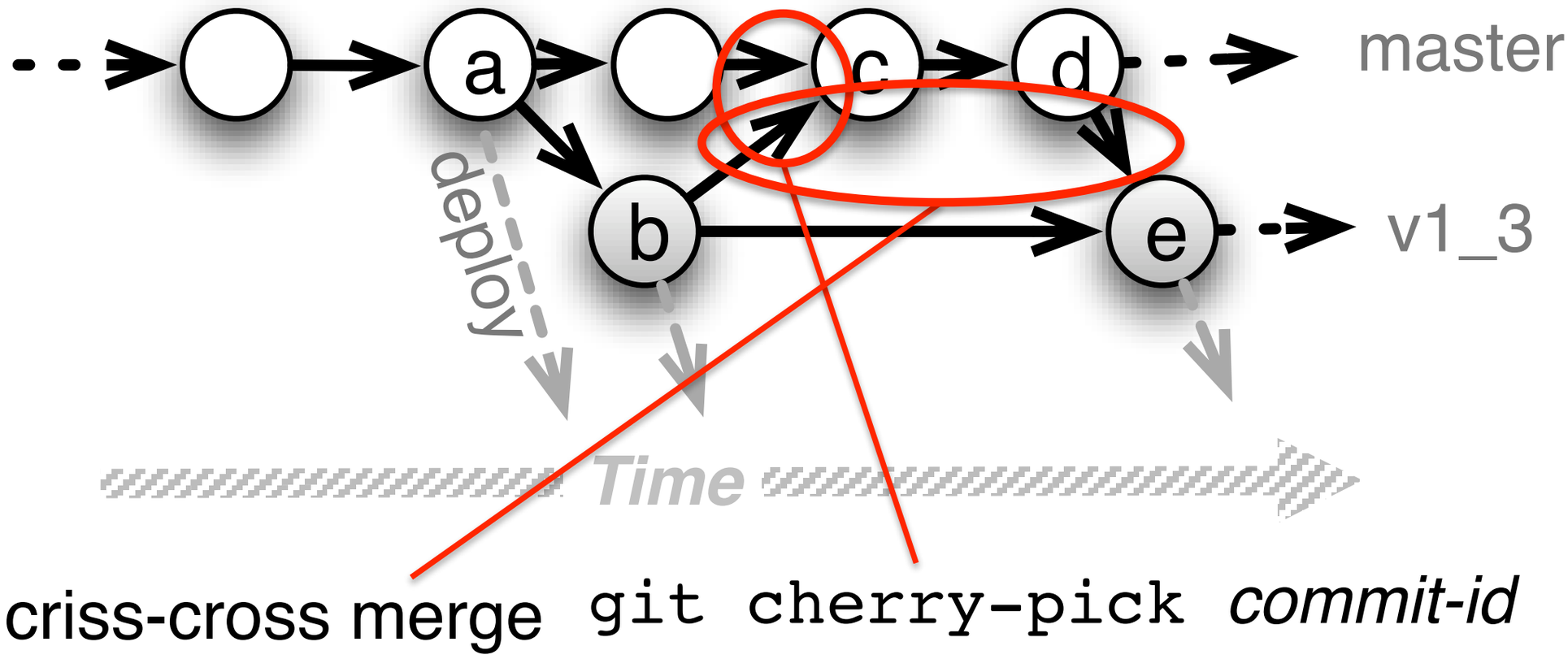
http://bit.ly/1MslHBT

Branching Out
FORD SMITH
40" x 50"
Limited Edition Giclee Canvas
& Hand Embellished
(Edition of 100 Signed and Numbered)

Branches & Deployment

- Feature branches should be short-lived
  - otherwise, drift out of sync with master, and hard to reconcile
    - `git rebase` can be used to “incrementally” merge
    - `git cherry-pick` can be used to merge only specific commits
- “Deploy from master” is most common
- “Branch per release” is alternate strategy

Release/bugfix branches and cherry-picking commits



criss-cross merge `git cherry-pick` *commit-id*

Rationale: release branch is a stable place to do incremental bug fixes

Usando branches com o Git: http://bit.ly/if977-BranchesWithGit

Branch vs. Fork

- Git supports *fork & pull* collaboration model
- If you have push/admin access on repo:
  - branch: create branch in *this repo*
  - merge: fold branch changes into master (or into another branch)
- If you don’t:
  - fork: clone *entire repo* on GitHub to one that you can branch, push, etc.
  - Finalize your work on its own branch
  - Courtesy: rebase your branch w/commit squash
  - Open pull request to pull the commit

Gitfalls

- Stomping on changes after doing a merge or switching branches
- Making “simple” changes directly on master branch

Undo!

`git reset --hard ORIG_HEAD`

- Revert your repo to last committed state just before the merge

`git reset --hard HEAD`

- Revert your repo to last committed state

`git checkout` *`commit-id`* `--` *`files…`*

- Revert the changes introduced by commit.

Undo!

- Comparing/sleuthing:

```
git diff commit-id-or-branch -- files…

git diff "master@{01-Sep-12}" -- files

git diff "master@{2 days ago}" -- files

git show mydevbranch:myfile.rb

git blame files

git log files
```

Pergunta

If separate sub-teams are assigned to work on release bug fixes and new features, you will need to use:

A. Branch per release

B. Branch per feature

C. Branch per release + Branch per feature

D. Any of these will work

http://dilbert.com/strip/2009-06-22

Fixing Bugs: The Five R's

(ESaaS §10.6)

No Bug Fix Without a Test!

- **R**eport
- **R**eproduce and/or **R**eclassify
- **R**egression test
- **R**epair
- **R**elease the fix (commit and/or deploy)

- Even in non-agile organisations
- But, existing agile processes can be adapted to bug fixes

Report

- Pivotal Tracker
  - bug = 0-points story (but not zero effort!!)
  - automation: GitHub service hooks can be configured to mark Tracker story “delivered” when properly-annotated commit is pushed
- GitHub “issues” feature
- Full-featured bug tracking, e.g. Bugzilla
- Use the simplest tool that works for your team & project scope

Reclassify?
or Reproduce + Repair?

- Reclassify as “not a bug” or “won’t be fixed”
- Reproduce with simplest possible test, and add it to regression
  - minimize preconditions (e.g. `before` blocks in RSpec, `Given` or `Background` steps in Cuke)
- Repair == test fails in presence of bug, passes in absence of bug
- Release: may mean either “pushed” or “deployed”

Pergunta

Suppose you discover that your most recent release contains a bug whose regression test will require extensive mocking or stubbing because the buggy code is convoluted. Which action, if any, is NOT appropriate?

A. Do the refactoring using TDD on the release branch, and push the bug fix as new code with tests

B. Do the refactoring using TDD on a different branch, push the bug fix as new code with tests, then cherry-pick the fix into release

C. Create a regression test with the necessary mocks and stubs, painful though it may be, and push the bugfix and tests to release branch

D. Depending on project priorities and project management, any of the above might be appropriate

10 Commandments for Being A Bad Software Team Player

```
git commit -m 'deal with it' &&
git push --force origin master
```

10 Commandments for Being a Bad Software Team Player (and suggested alternatives)

1. Those fails don't matter
2. My branches, my sanctuary
3. It's just a simple change
4. I am a special snowflake
5. Tabs save valuable bytes
6. Cleverness is impressive
7. Just change it quickly on production server
8. Time spent looking stuff up = wasted time not coding
9. "Green fever": catch it!
10. Weeks of coding can save hours of planning/thought

1. Never push red
2. Have short-lived branches
3. Mount a scratch monkey
4. 1 project, 1 coding style
5. Don't use tabs
6. Transparency is humble
7. Make every change automatable
8. Spend 5 minutes searching for less/better code
9. More tests ≠ higher quality
10. Walk through your design

Some additional commandments to make projects manageable

- any method with flog > 10 is rejected
- any branch with lifetime > ~3 days is nuked
- any merge that breaks the build is nuked and culprit **must** rebase against master
- any bugfix or code submitted without >90% test coverage is rejected

Fallacies & Pitfalls

Pitfall

- Pitfall: Dividing work based on software stack rather than features
  - E.g., front-end/back-end specialist, customer liaison, …

- Agile: better results if each team member delivers all aspects of a story
  - Cucumber scenarios, RSpec tests, views, controller actions, model logic, …
  - Everyone on team has “full stack” view of product

Pitfall

- Accidentally stomping on changes after merging or switching branches
  - In wrong branch, write over merged changes from old version in editor, …

- ***Before*** you pull or merge, commit all changes
- ***After*** you pull or merge, reload files into editor
  - Or quit editor before commit

Pitfall

- Letting your copy of the repo get too far out of sync with the origin (authoritative) copy
  - Means merges will be painful
- Do `git pull` before starting, `git push` as soon as your locally-committed changes are stable enough
- If long-lived branch do periodic `git rebase`

Fallacy

- It's fine to make simple changes on the master branch
  - Think its 1-line change, turns into 5 lines, then affects another file, then must change tests, …
- **Always** create a feature branch when starting new work
  - Branching with Git is nearly instantaneous
  - If change is small, you can delete branch after merging to avoid cluttering branch namespace

Concluding Remarks

- 2-pizza teams reduce management problem from banquet teams, but not to 0
  - Scrum is an informal organisation that is a good match to Agile Development
- Points, Velocity, Tracker => more predictable
- P&D: Project Manager as boss, plans & documents, Reviews to learn from others
- When project done, take time to think about what learned before leaping into next one
  - What went well, what didn’t, what to do differently