COMP0104 Software Development Practice: Code Review

Jens Krinke

Centre for Research on Evolution, Search & Testing
Software Systems Engineering Group
Department of Computer Science
University College London

© UCL. Unauthorised reproduction prohibited.

Overview

- Types of Code Review
- Change Review
- Gerrit

© UCL. Unauthorised reproduction prohibited.

Code Reviews

- Peer review of source code
- Goal to find and fix bugs early,
  improve overall code quality
- Humans need editors
  (see quality of self-published books…)
- Has been shown to effectively identify bugs
- Is used despite the huge costs

© UCL. Unauthorised reproduction prohibited.

Goals of Code Review

- Provide timely feedback to author
- Find bugs and flaws early
- Improve overall code quality
- Knowledge sharing
- Collective code ownership
- Mentoring new starters

© UCL. Unauthorised reproduction prohibited.

Types of Code Reviews

Synchronous Reviews:

- Reviewers meet at a defined time (in person)
- Reviewers go through a prepared list of items (code) to be reviewed
- Reviewers should prepare before the meeting

Asynchronous Reviews:

- Reviewers perform reviews on demand
- Discussion is asynchronous
- Tool support is needed (email…)

© UCL. Unauthorised reproduction prohibited.

Formal Inspections

Defined by Michael Fagan, IBM, 1976:

- Five phases and an inspection meetings
- Number of findings in meetings much higher than in individual work
- No static analysis applied

No conclusive results on the need of meetings

- Meetings may cause more false positives
- Most issues found just by reading
- No significant change when a structured technique is used

© UCL. Unauthorised reproduction prohibited.

Meeting-less Inspections

Votta (1993) suggests to minimise (or eliminate) meetings:

- Collect faults by small face-to-face meetings of two or three persons
- Collect faults using verbal or written media (phone, email, or notes)

Lightweight Inspection

- Look over shoulder
- Pass around emails
- Pair programming

© UCL. Unauthorised reproduction prohibited.

Over-the-shoulder reviews

- Most common form of code reviews
- A developer looks over the shoulder of another who walks him through a set of changes.
- The author may show results of tools
- The reviewer may ask questions
- May include small pair-programming sessions

✗ Nothing is enforced

✗ Nothing is documented

✗ Reviewer usually does not check the outcome

© UCL. Unauthorised reproduction prohibited.

Email-pass-around reviews

- Files and changes are packed up by the author who sends them to the reviewers
- Reviewers examine the files, ask questions, and discuss with the author and other developers
- Reviewers send suggestions back to authors

✗ Overhead due to the use of email (pack, unpack, track changes, etc.)

✗ Difficult to track different threads

© UCL. Unauthorised reproduction prohibited.

Pair Programming

- Reviews are constantly performed by the second reviewer during programming
- “embedded code review”

✗ No clear separation

✗ Nothing is documented

✗ Second programmer is too involved in the code (is no longer unbiased)

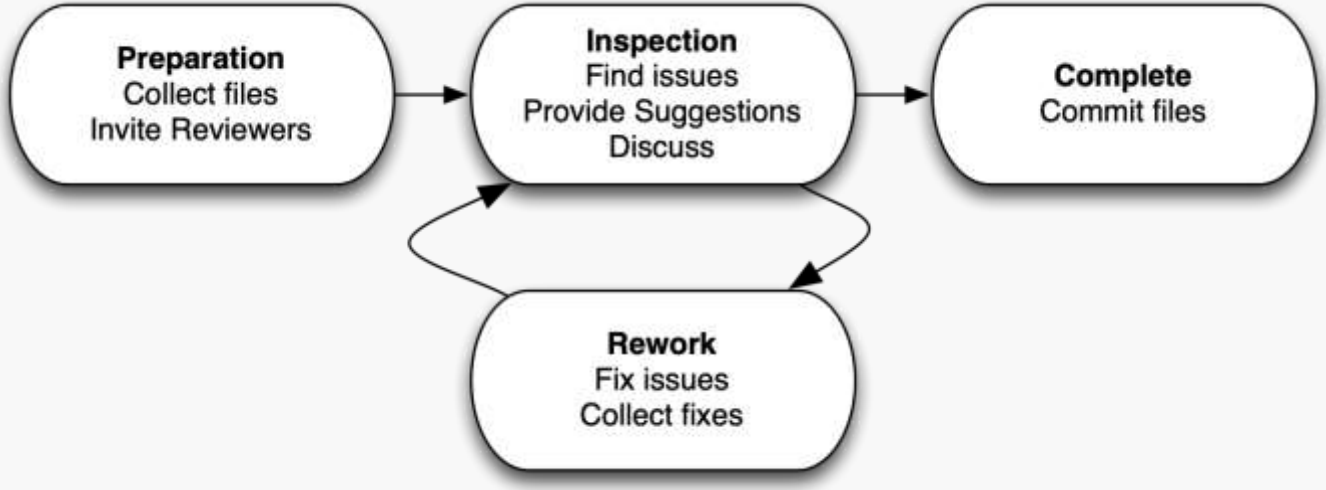
© UCL. Unauthorised reproduction prohibited.

Tool-supported lightweight code reviews

- Tool support can eliminate the drawbacks of lightweight code reviews while keeping the advantages.
- Tool-supported lightweight code reviews are as effective as formal inspection, but much more time-efficient.
- Largest study at Cisco 2005/2006:
  - 2 500 reviews
  - 3.2 million LOC
  - 50 developers

© UCL. Unauthorised reproduction prohibited.

Review Lifecycle



© UCL. Unauthorised reproduction prohibited.

Types of Code Reviews

Pessimistic Reviews:

- An artefact can only be committed if is has been reviewed (and approved)
- Commit is blocked
- All artefacts are reviewed

Optimistic Reviews:

- An artefact is reviewed after being committed
- Commit is not blocked
- Artefact may not get reviewed

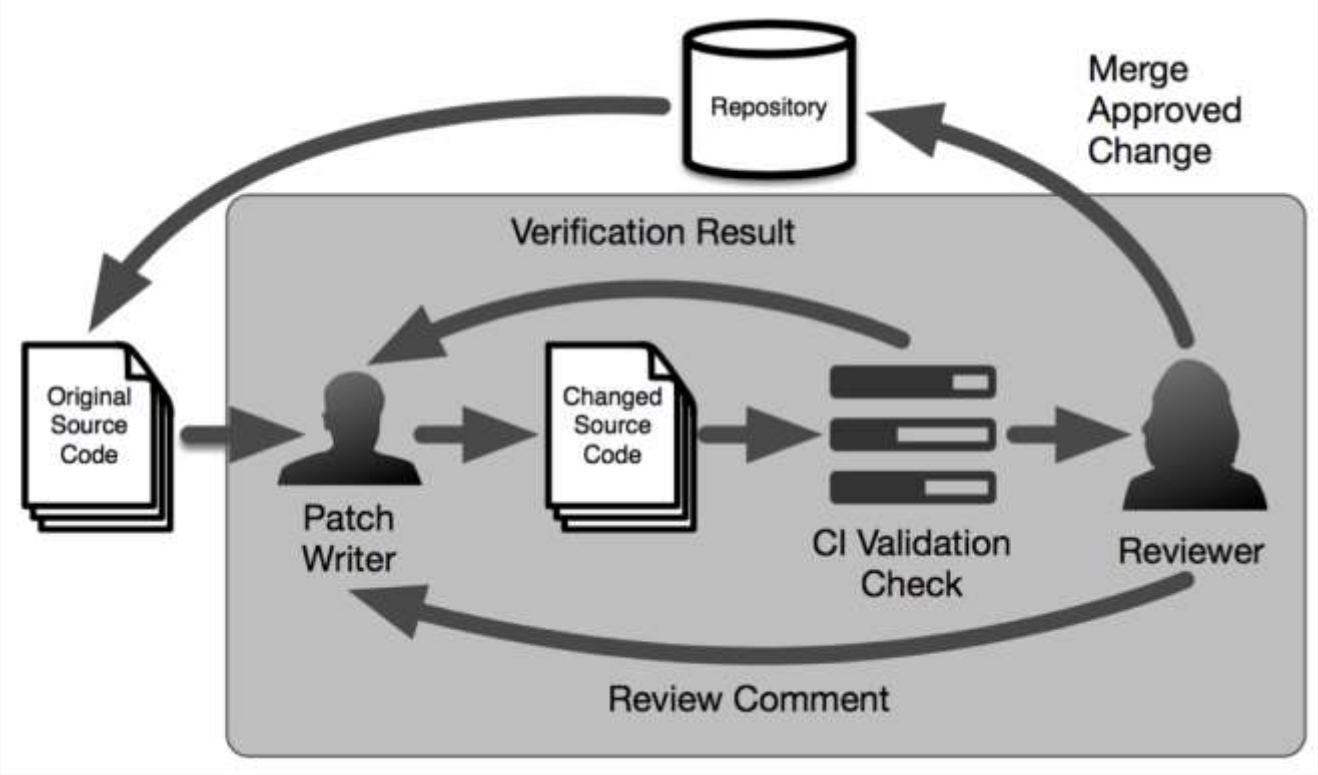
© UCL. Unauthorised reproduction prohibited.

Change Reviews

- Continuous Integration:
  Don’t break the build!
- Changes must be approved / reviewed
  before they are integrated in the main branch.
- Only approved changes are integrated.

© UCL. Unauthorised reproduction prohibited.

Modern Code Review



© UCL. Unauthorised reproduction prohibited.

Development: One Topic per Branch

- Every topic (feature) creates a new branch, is developed independently of the main branch.
- Branches are continuously merged (rebased) into the main branch.
- Changes caused by the branch are reviewed before merged into the main branch.
- Change: Difference between branch point and current state.
- Changes and branches can easily be identified.

© UCL. Unauthorised reproduction prohibited.

Pull Requests

- Branches cannot be merged directly,
  instead, a pull into the main branch is requested.
- The maintainers of the main branch can review, test, and merge the changes of the request.
- Comments stay on the platform and the history of the change stays in the (feature) branch
  (and may even be lost).

© UCL. Unauthorised reproduction prohibited.

Gerrit

- Idea: Guido Van Rossum (Python Inventor)
- Predecessors: Mondrian, Rietveld
- Review System for Android Open Source Project (AOSP)
- Tightly integrated with Git
- Integration with Jenkins

© UCL. Unauthorised reproduction prohibited.

Gerrit Roles

Contributor

- Can upload a change
- Does not need to have rights to commit code

Reviewer

- Can comment on a change and can score a change

Committer

- Has the final word on a change

Maintainer (no active role in the project)

© UCL. Unauthorised reproduction prohibited.

Change-Ids

- Commits are mapped to reviews by Change-Ids.
- Reviews consist of multiple commits,
  e.g. a second commit improves a change
  after the first round of reviews for the first commit.
- Change-Ids are usually created automatically
  and commits without Change-Id cannot be reviewed.

© UCL. Unauthorised reproduction prohibited.

Scoring

- Every team member can provide scores:
  - +1, 0, –1 for normal team members (Reviewers)
  - +2, –2 for committers
- +2 submits the change to the main branch
- –2 vetoes the change
- Scores are not cumulative (two +1 is not +2)
- Scores for different things, e.g. Review and Validation

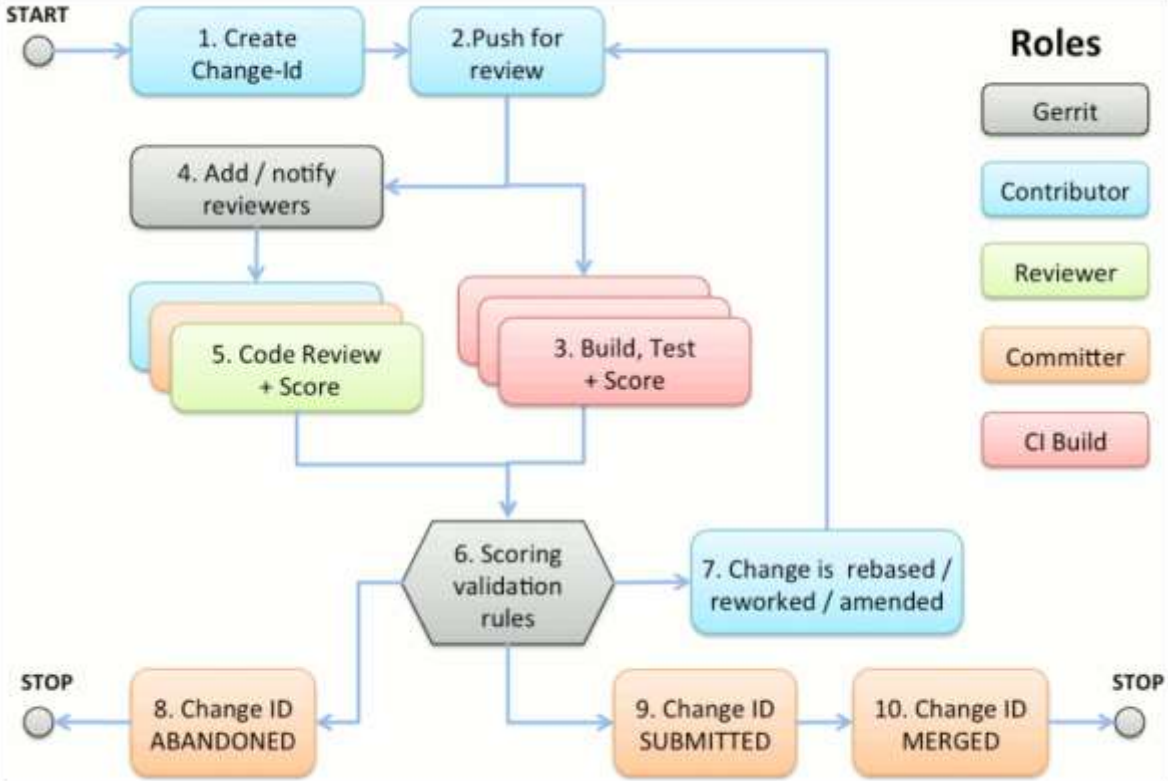
© UCL. Unauthorised reproduction prohibited.

Scores

Code Review	–2	Veto, code cannot be accepted in any case
	–1	Code looks good, but changes are needed
	+1	Code looks good, but more people need to review it
	+2	Code looks good and can be merged
Verified	–1	Code does not work
	+1	Code works fine

© UCL. Unauthorised reproduction prohibited.

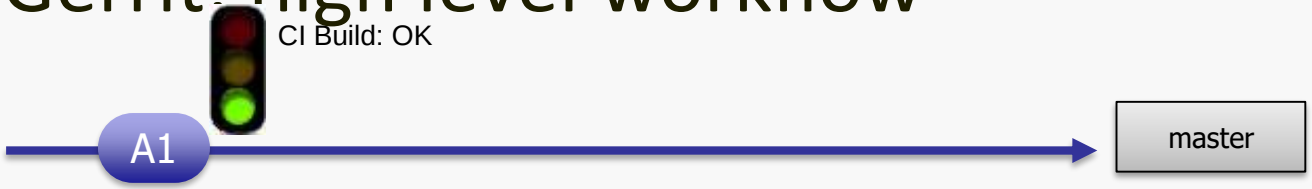
Review Workflow



Learning Gerrit Code Review, Luca Milanesio, Packt Publishing, 2013

© UCL. Unauthorised reproduction prohibited.

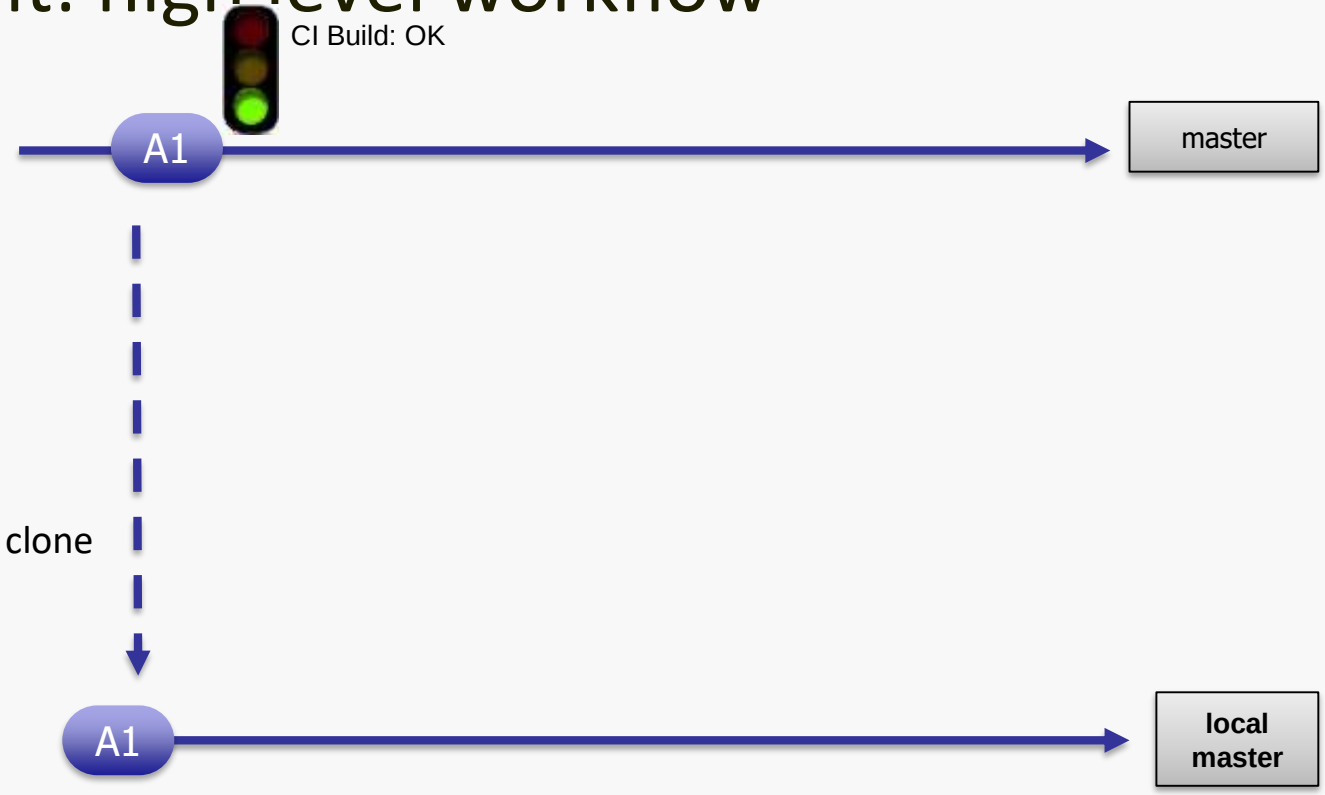
Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

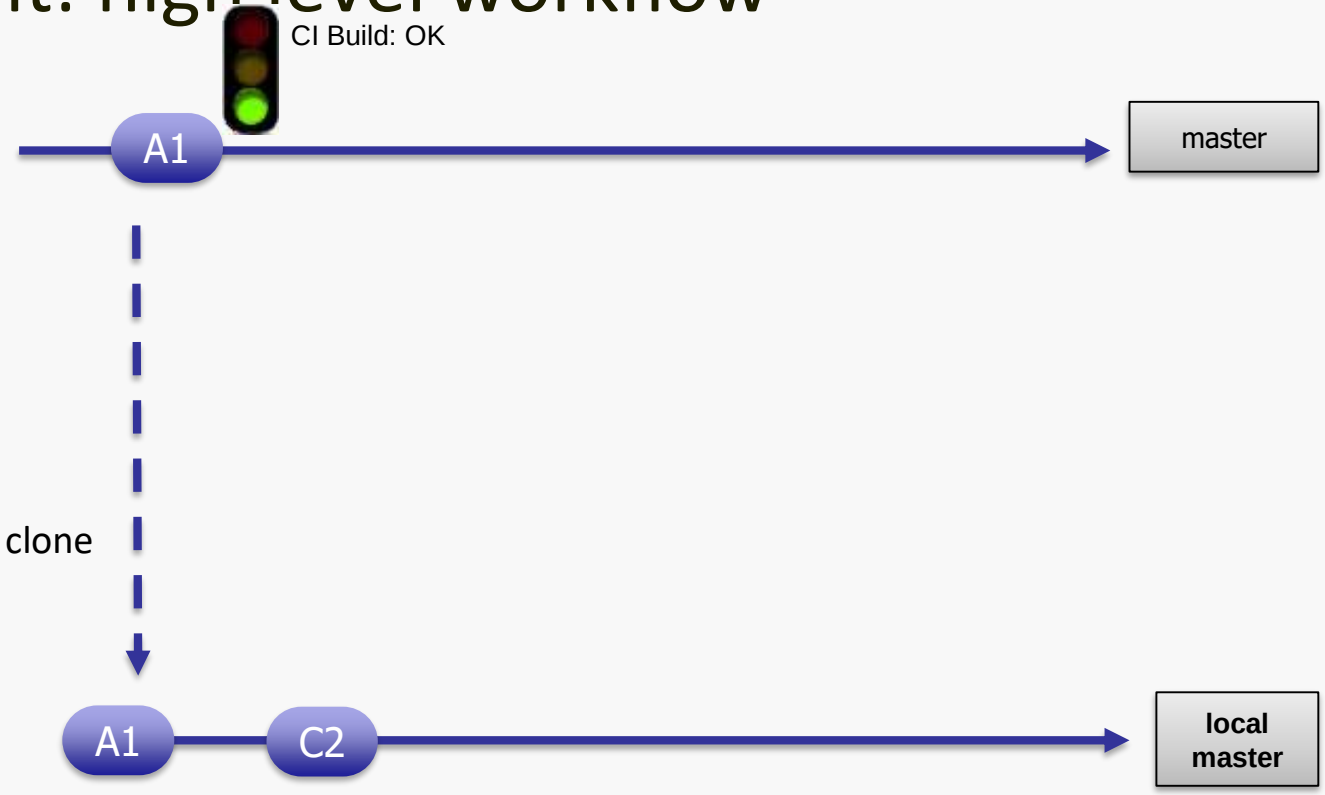
Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

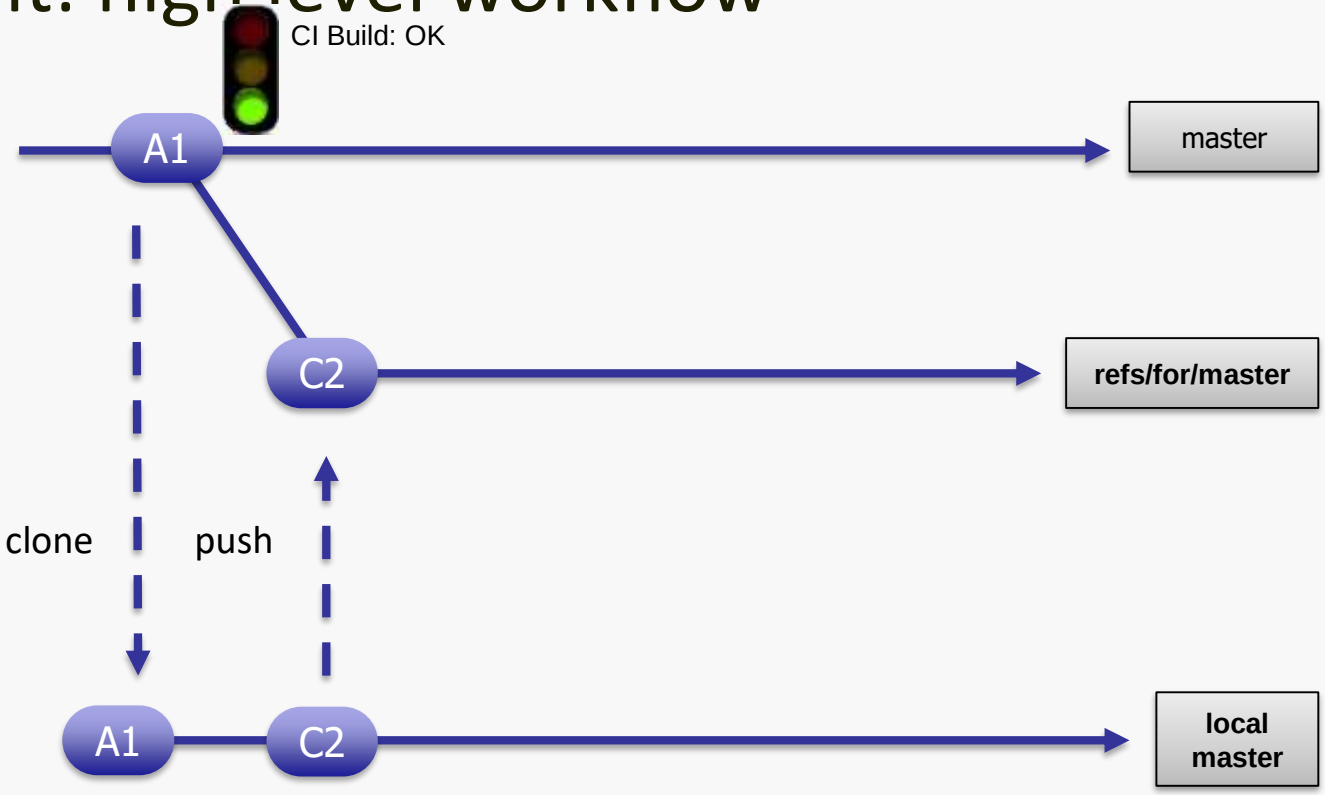
Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

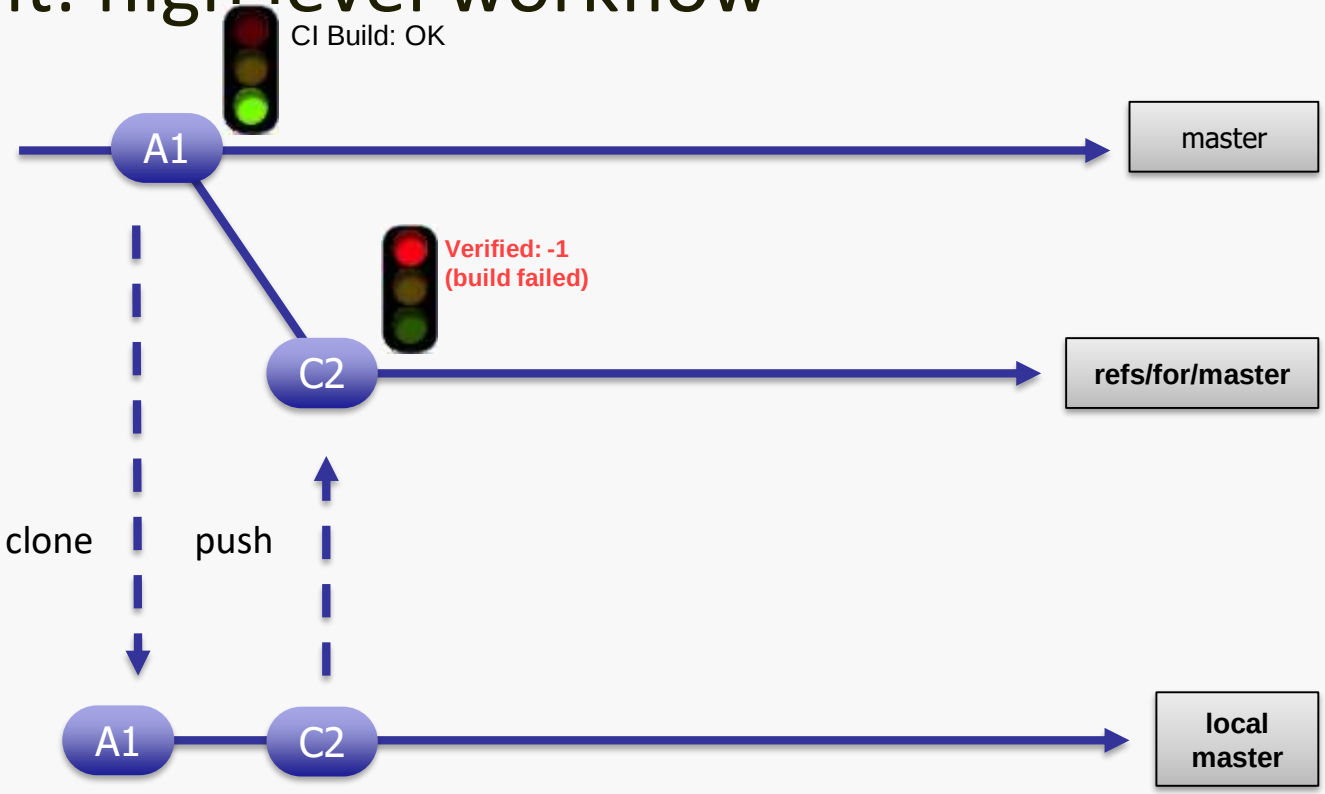
Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

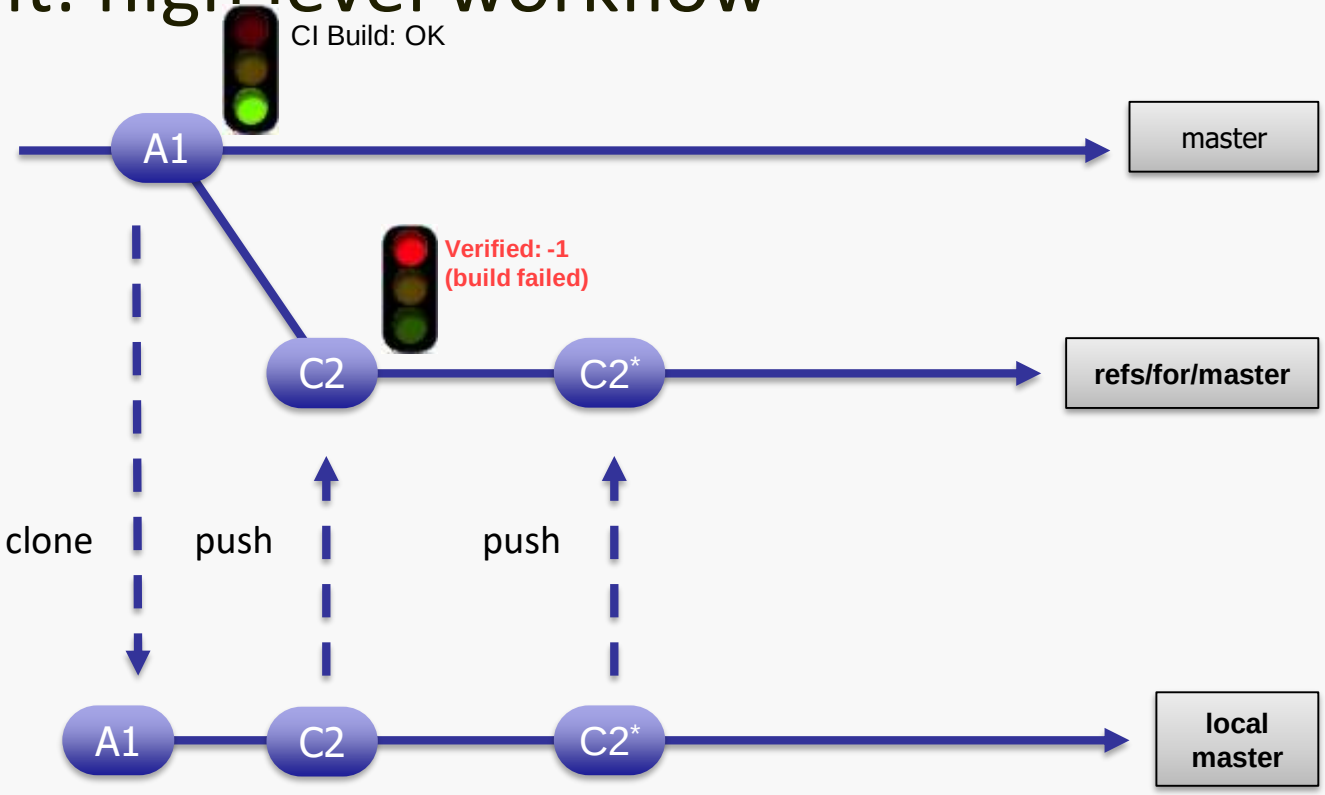
Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

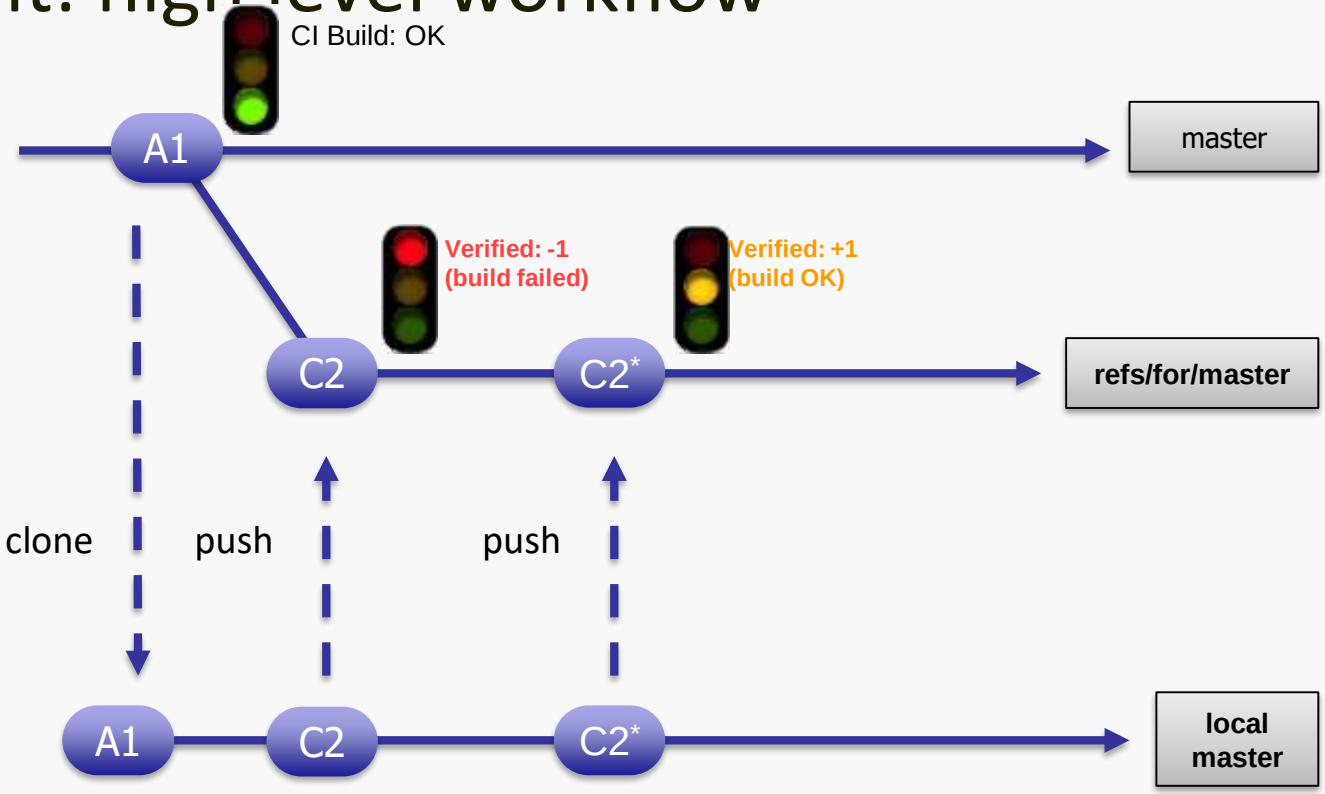
Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

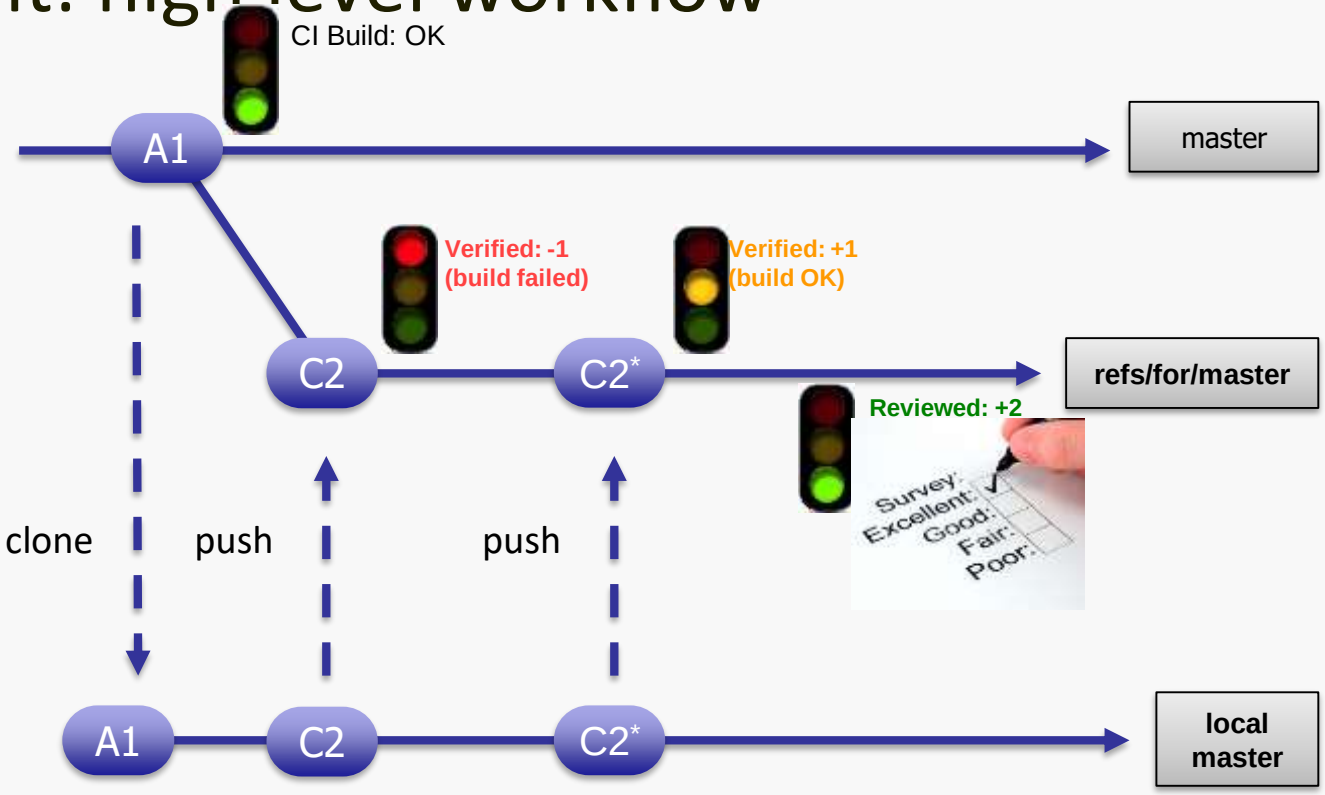
Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

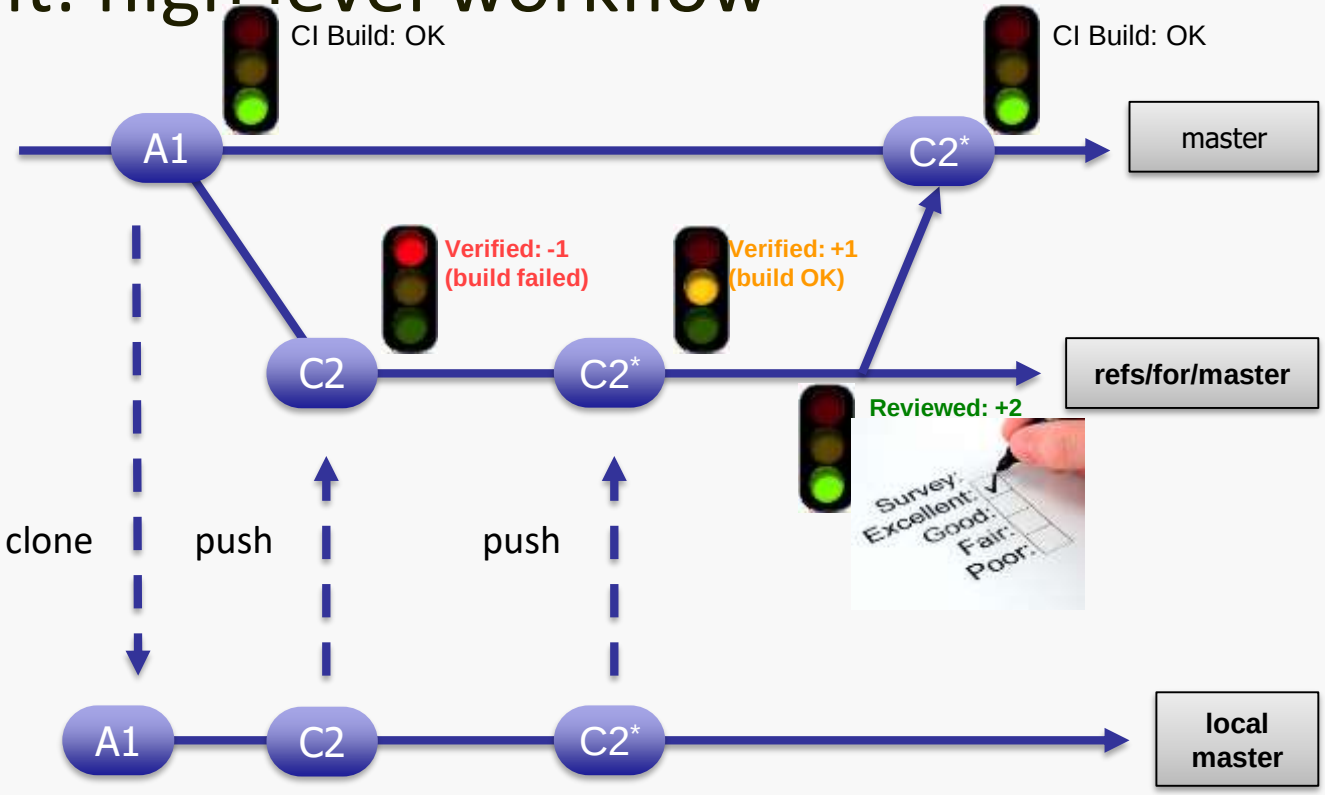
Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

Gerrit: high level workflow



© UCL. Unauthorised reproduction prohibited.

Slides by Luca Milanesio, LMIT Software, 2012

Integrating the Change

- If approved, the change can then be merged into the main branch.
- Conflicts are often avoided because only approved changes are committed, no other development commits.
- However, other changes may have been approved since the creation of the branch: Change must be “updated” (rebased).

© UCL. Unauthorised reproduction prohibited.

Merge Conflict 708530 icinga::monitor::gitlab add alerts for https and ssh for gitlab

Updated	14:59
Owner	Jelto
Assignee	
Reviewers	Daniel Zahn, jenkins-bot, JMeybohm
CC	Brennen Bearnes
Repo | Branch	operations/puppet | production
Parent	a8f1a70
Topic	gitlab_alerts
Strategy	Rebase if Necessary
Hashtags	

```
icinga::monitor::gitlab add alerts for https and ssh for gitlab

This change adds basic alerts for public facing endpoints (https
and ssh) similar to gerrit.

Bug: T275170
Change-Id: I4b3d52c6d20cadad413ca9bcf47712e48176e887
```

Test results for Patchset 3 (Only main test build and SonarQube)

SUCCESS operations-puppet-tests-buster-docker

Verified	+1	Jelto
	+2	jenkins-bot
Code-Review	+1	Daniel Zahn

© UCL. Unauthorised reproduction prohibited.

Jelto Patchset 2 Jul 28 15:06

Uploaded patch set 2: Commit message was updated.

jenkins-bot Verified +2 Main test build succeeded. - https://integration.wikimedia.org/ci/job/operations-puppet-t... Patchset 2 Jul 28 15:07

Daniel Zahn Code-Review +1 Patchset 2 Jul 28 16:57

this looks good to me! just make sure when you merge this to manually run puppet on alert* hosts and then run "icinga -v /etc/icinga/icinga.cfg" after it added the checks, to confirm icinga config is alright, should show no warnings or errors.

Jelto Uploaded patch set 3. Patchset 3 13:45

jenkins-bot Verified +2 Patchset 3 13:47

Main test build succeeded.

- https://integration.wikimedia.org/ci/job/operations-puppet-tests-buster-docker/29056/console : SUCCESS in 1m 07s

Jelto Verified +1 Patchset 3 13:51

PCC SUCCESS (DIFF 1): https://integration.wikimedia.org/ci/job/operations-puppet-catalog-compiler/30414/console

Jelto Patchset 3 14:04

> Patch Set 2: Code-Review+1
>
> this looks good to me! just make sure when you merge this to manually run puppet on alert* hosts and then run "icinga -v /etc/icinga/icinga.cfg" after it added the checks, to confirm icinga config is alright, should show no warnings or errors.

Thanks!
But I think the check_commands are wrong in the pcc run and the hostname is missing:

```
+   check_command       => check_ssh_port_ip!22!
+   check_command       => check_https_url!!/explore
```

© UC

Professional Reviewing

- Constructive feedback, no criticism
  - Nothing is personal
  - Don’t accuse
  - Ask, don’t tell
  - Review the code, not the coder
- Understand technology, conventions, and related code first
- Be concise

© UCL. Unauthorised reproduction prohibited.

Concepts (1/2)

- The goal of code reviews is to find and fix bugs early and to improve the overall code quality.
- Formal inspections require preparation in different phases, including a physical meeting to discuss source code.
- Lightweight review techniques including looking-over-the-shoulder, email-pass-around, and pair programming have been shown to be similarly effective.

© UCL. Unauthorised reproduction prohibited.

Concepts (2/2)

- Gerrit is a Git-based change review systems in which changes are submitted for review.
- Only after a change has been reviewed and approved, it will be merged to the main branch.
- In Gerrit,
  - everyone with read access to the code can contribute changes (Contributor),
  - team members review and score the change (Reviewer)
  - experienced members can approve or veto a change (Committer)

© UCL. Unauthorised reproduction prohibited.