Code Review Tools

Sven Rosvall

ACCU Conference 2017

Previous ACCU Talks

Arjan Leuwen 2013 - The art of reviewing code

Austin Bingham 2015 - Making the Case for Review

Why Review Code?

- Find bugs early
- Improve maintainability
- Share knowledge
- Harmonize code style

Better quality
Better teams

Productivity

Dangers

Viewed as waste of time

Management buy-in

Development buy-in

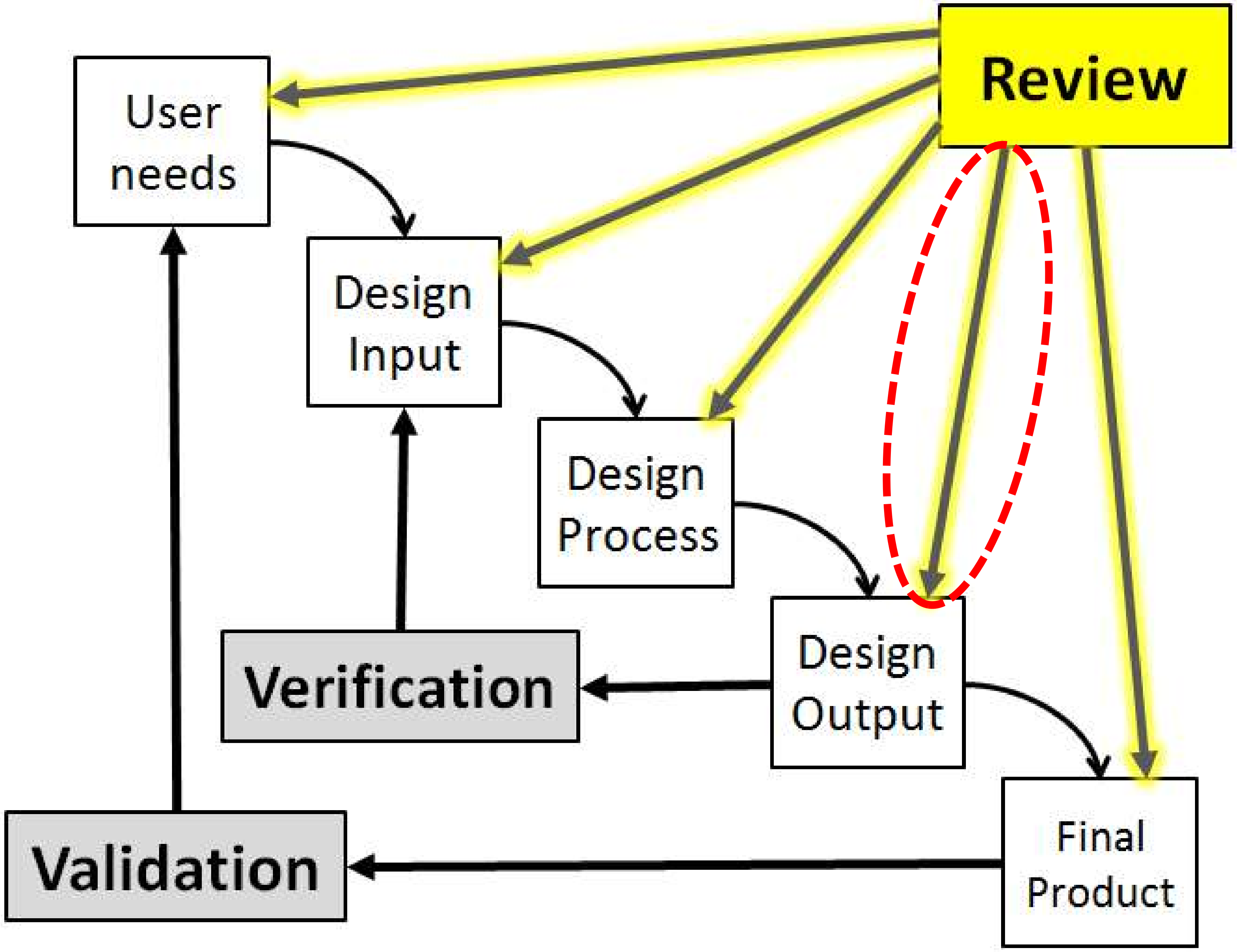
Resistance to change

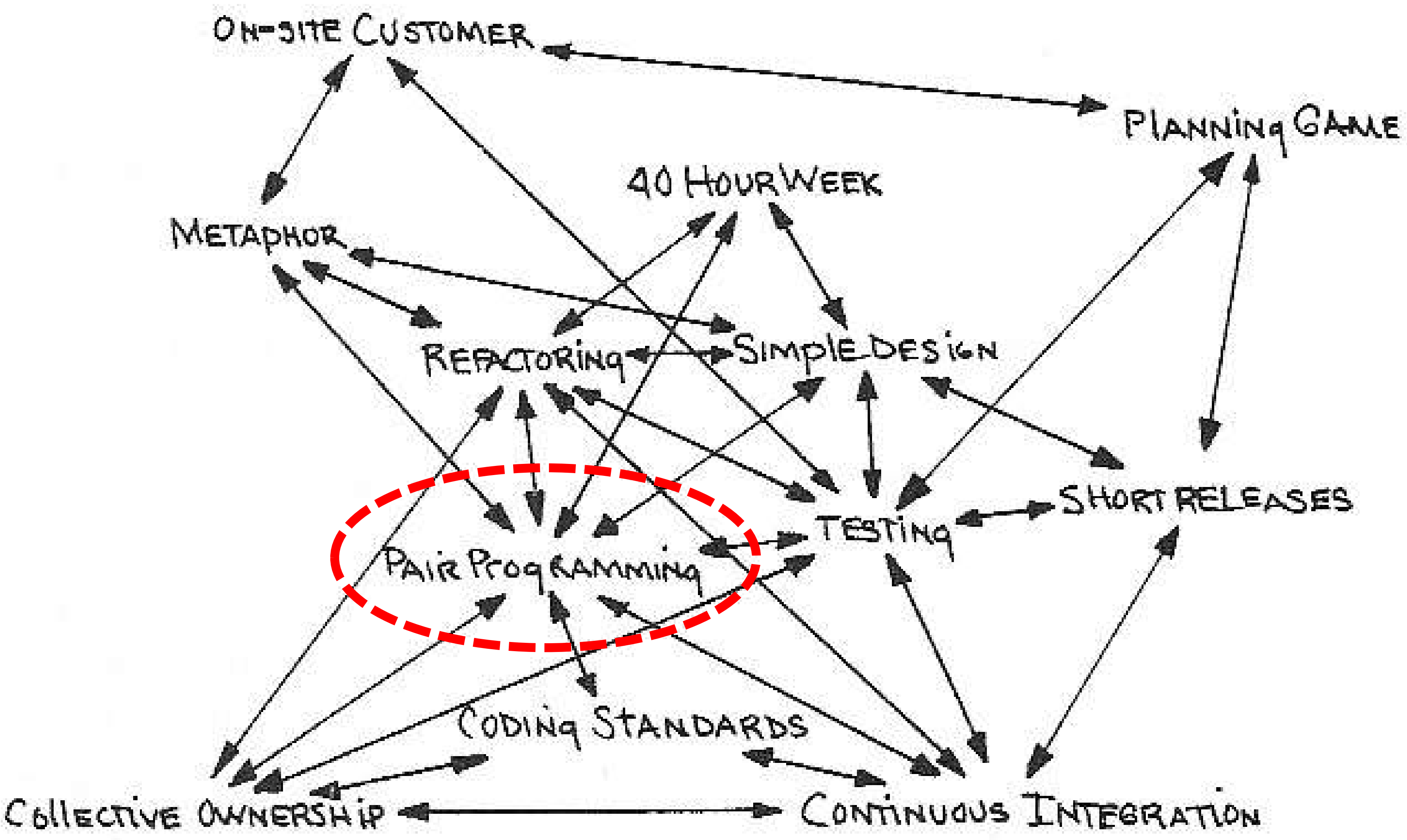
Process

Ego

Big Brother

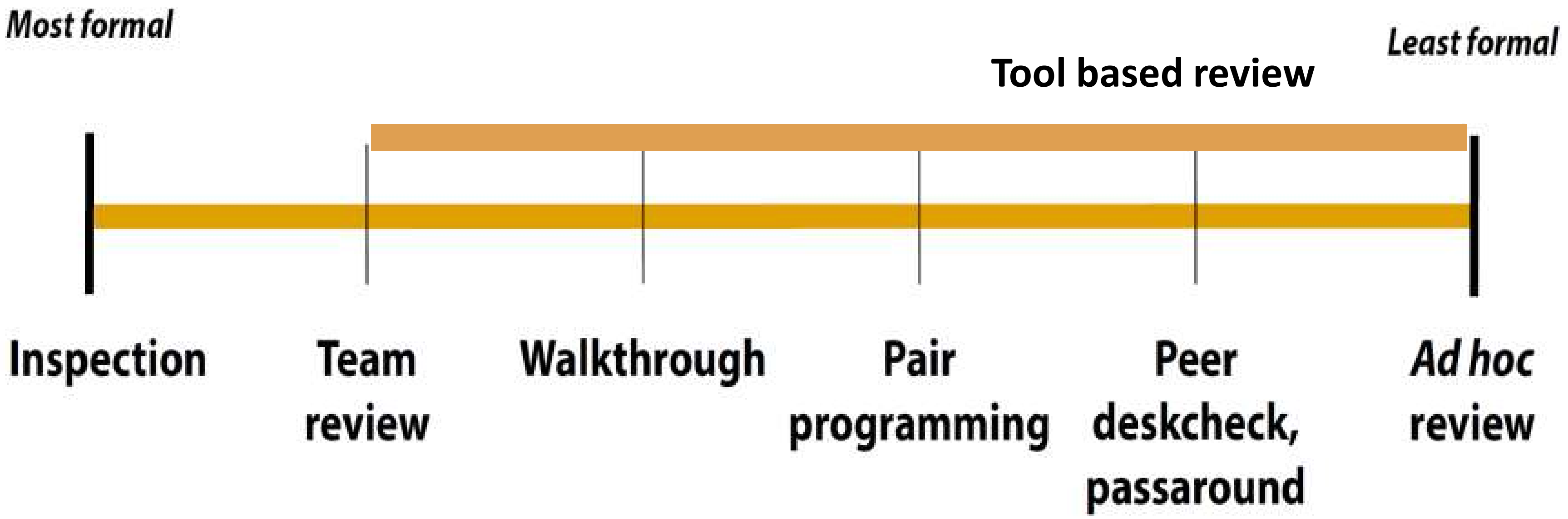
Annual Review Values


Review
User needs
Design Input
Design Process
Design Output
Final Product
Verification
Validation


On-site Customer
Planning Game
40 Hour Week
Metaphor
Refactoring
Simple Design
Testing
Short Releases
Pair Programming
Coding Standards
Collective Ownership
Continuous Integration

Review formality spectrum

Reviews can be roughly ordered from formal inspections to *ad hoc*



Based on the original diagram by Karl E. Wiegers in "Peer Reviews in Software: A Practical Guide"

Formal Review Meeting	Review Tools	Pair Programming
Many people	Many people	2 people
Schedule meeting	Individual time Individual location	While pairing
Rushed comments	Thorough comments	Creating together
Comments in big group	Offline comments	Comments while working together
Learning by observing	Learning by observing	Learning by doing

What to Review

Bugs

Style

Change matches task / bug report

Design choices

Unit tests coverage

Dependencies

Duplicate code

Library usage

Comments

Concurrency

Efficiency

Maintainability

Future proof

Improvements

Legacy stuff to avoid

Educate Author

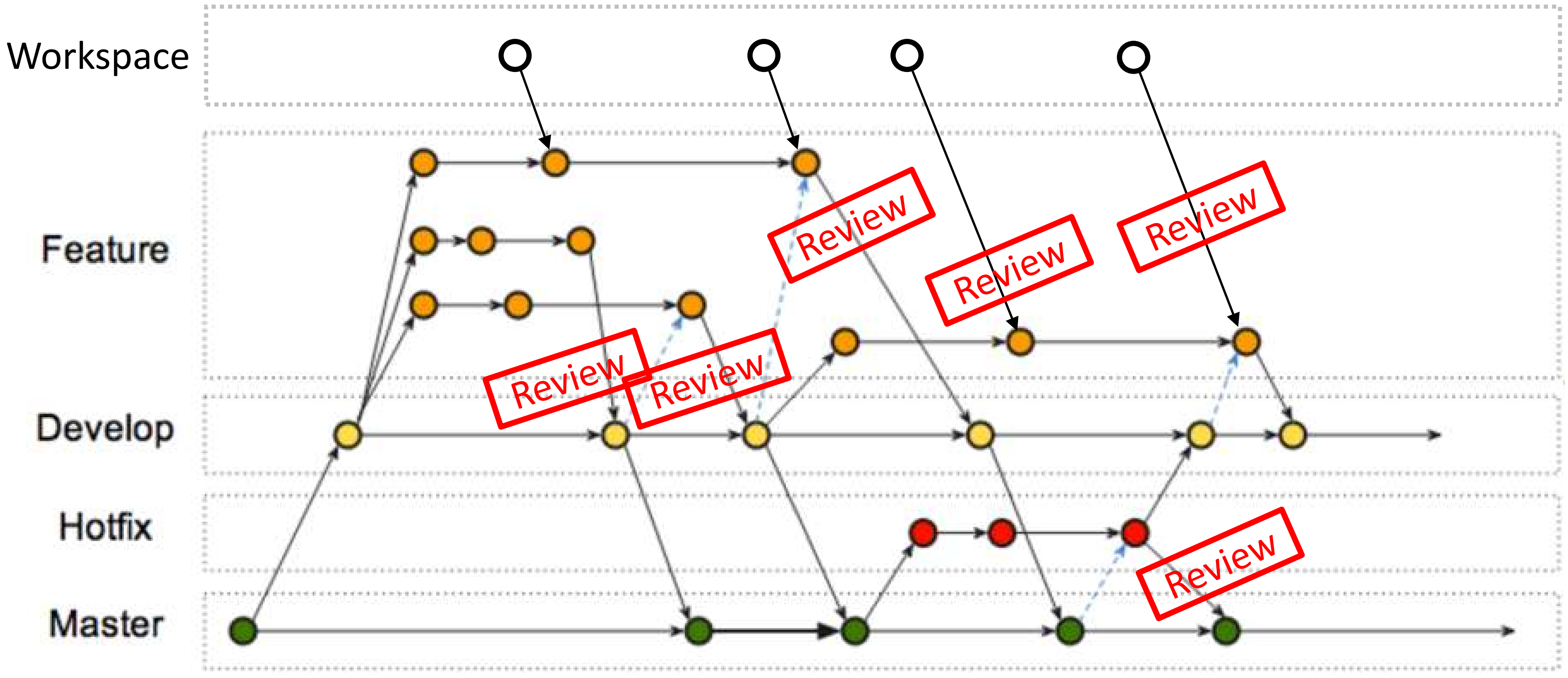
Things to learn yourself

Appraisals

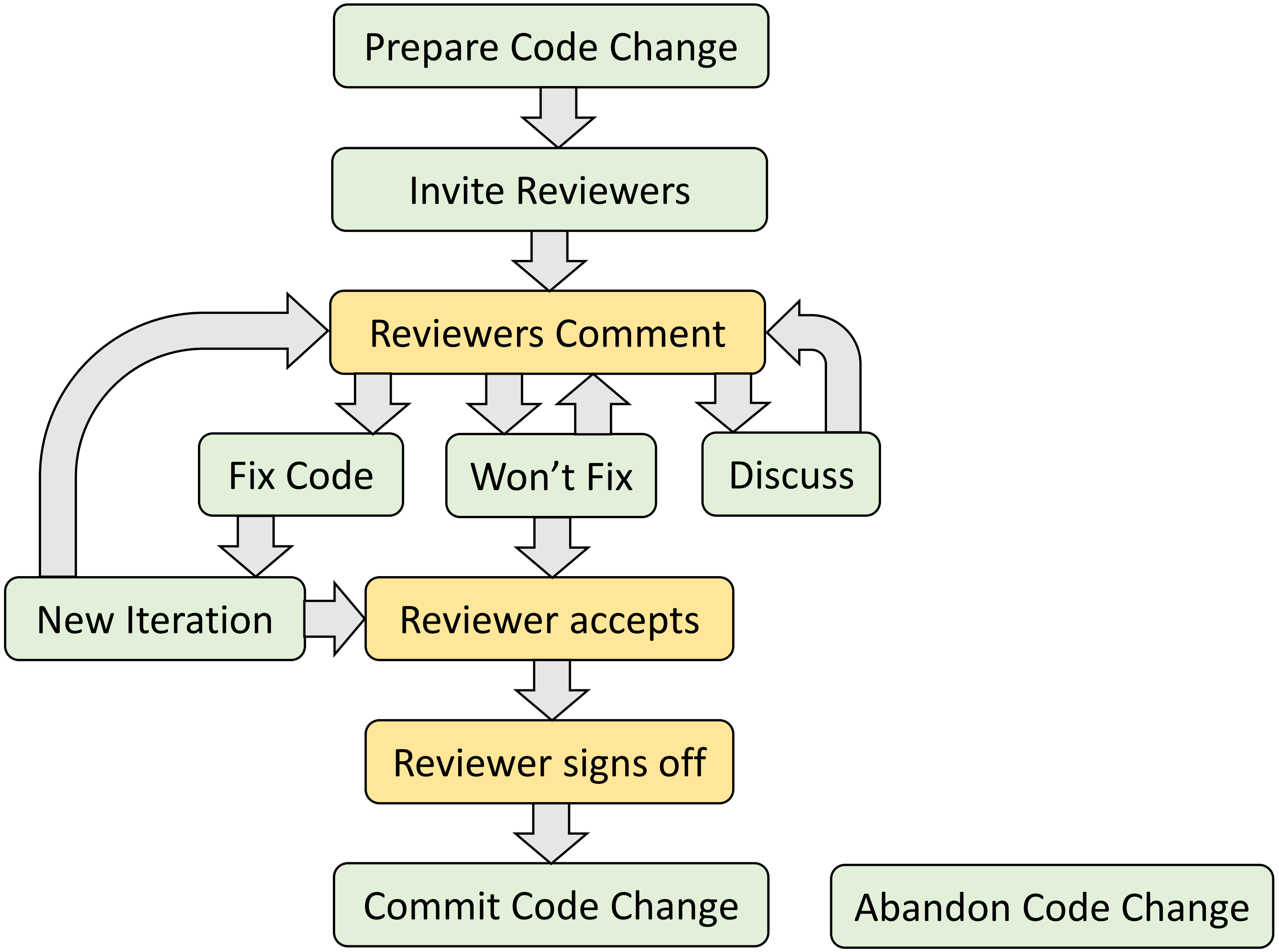
When to Review

After	**Before**
Code compiles	Committing code / Merging to release branch
Automated tests pass	Integration tests
	Handover to QA

Review for Gatekeeping



Review Workflow



Prepare Code Change

- Upload a patch to review tool
- Tell tool to find local changes
- Tell tool to find committed change
- Use a script/tool to do the above

Selecting Reviewers

Reviewers	**Observers**
Promise to do review ASAP	For Their Information
Essential skills	Contribute skills
Can block commits	Can be ignored

Reviewer Skills

Bugs

Style

Change matches task / bug report

Design choices

Unit tests coverage

Dependencies

Duplicate code

Library usage

Comments

Concurrency

Efficiency

Maintainability

Future proof

Improvements

Legacy stuff to avoid

Educate Author

Things to learn yourself

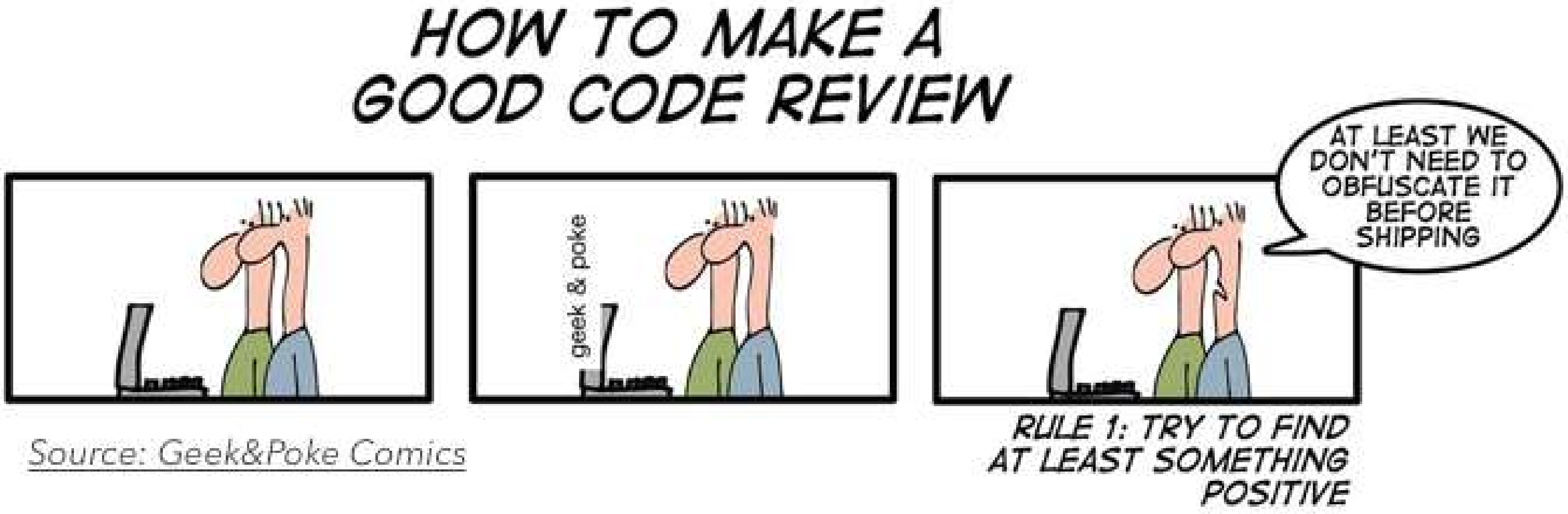
Appraisals

Pitfalls During Review

- Change creep
- Mixing feature change with refactoring
- Conflicting views
- Long discussions
- Chasing reviewers
- Negative or personal comments

Pitfalls During Review

- Change creep
- Mixing feature change with refactoring
- Conflicting views


HOW TO MAKE A GOOD CODE REVIEW
geek & poke
AT LEAST WE DON'T NEED TO OBFUSCATE IT BEFORE SHIPPING
RULE 1: TRY TO FIND AT LEAST SOMETHING POSITIVE

Source: Geek&Poke Comics

Measuring benefits of code reviews.

- Reduction in bug frequency
- Retrospective for bugs:
  - Could this bug have been caught at review time?
  - If not: Design review?
  - Any test passes?
  - Any other phase?
- Size of reviews?
- Code style coherency
- Team proficiency
- Review speed vs found issues

Review Tool Requirements

Author

- Easy to publish changes
- View outstanding comments

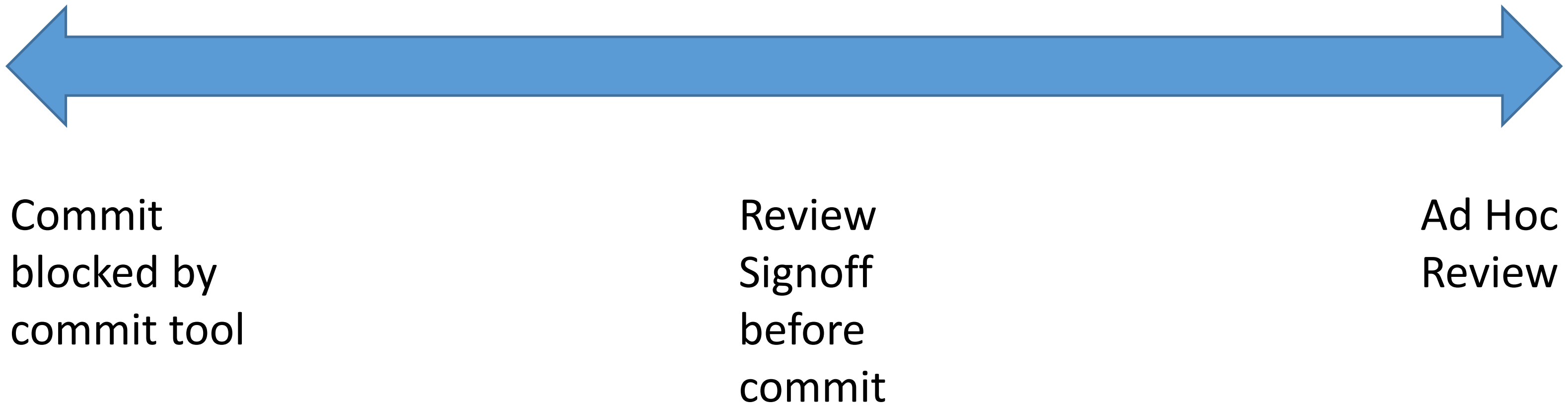
Reviewer

- View code differences
- View comments with responses
- View comments not yet dealt with by author

General

- Integration with your tool set

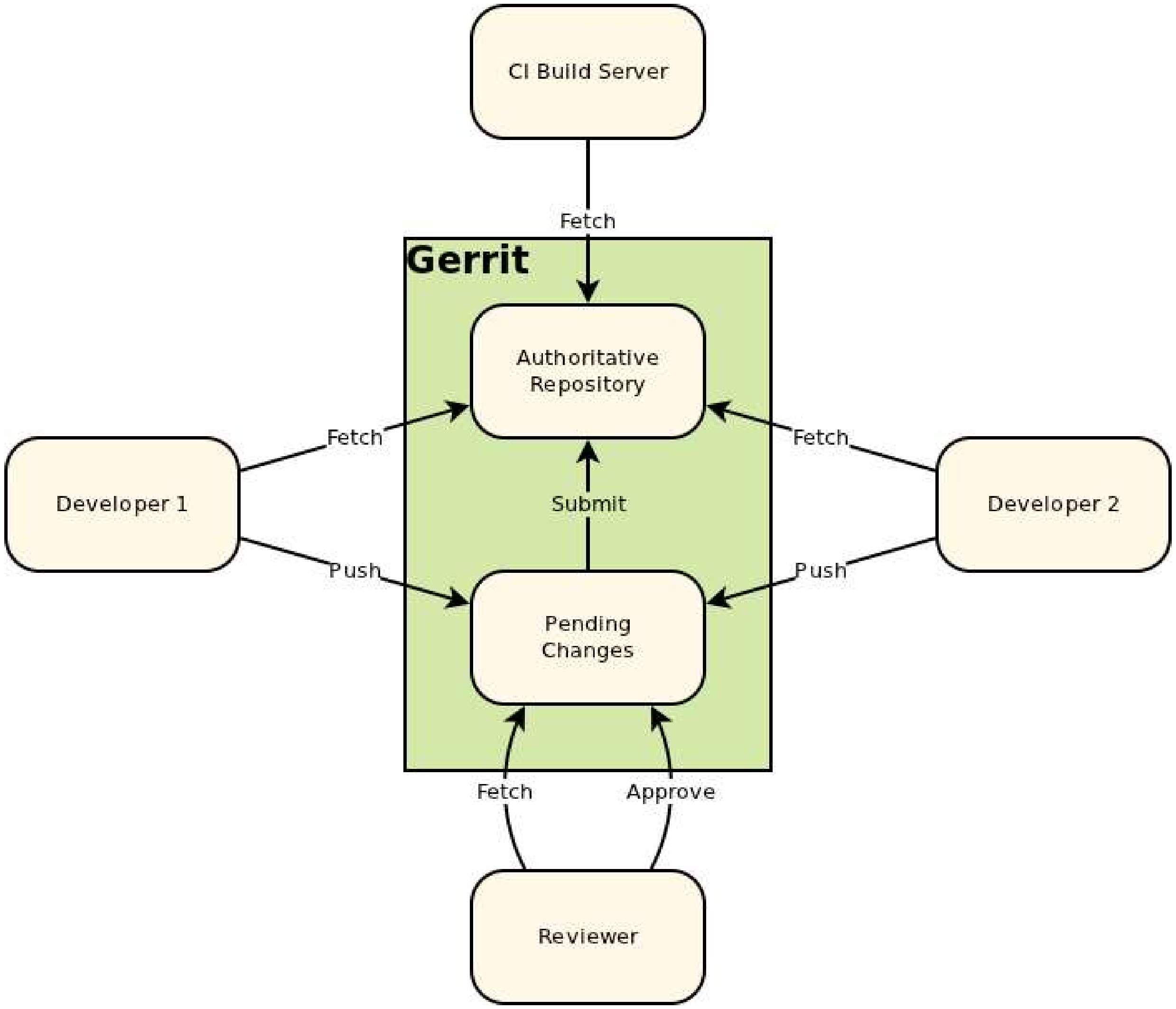
Code Review and Your Process



Resources

https://en.wikipedia.org/wiki/List_of_tools_for_code_review

Demo Gerrit


CI Build Server
Fetch
Gerrit
Authoritative Repository
Fetch
Fetch
Developer 1
Submit
Developer 2
Push
Push
Pending Changes
Fetch
Approve
Reviewer