# Code reviews

Thursday, October 11

# Announcements

Sprint 2 is released

Extra office hours on Friday, 10-noon, in KEC 3057

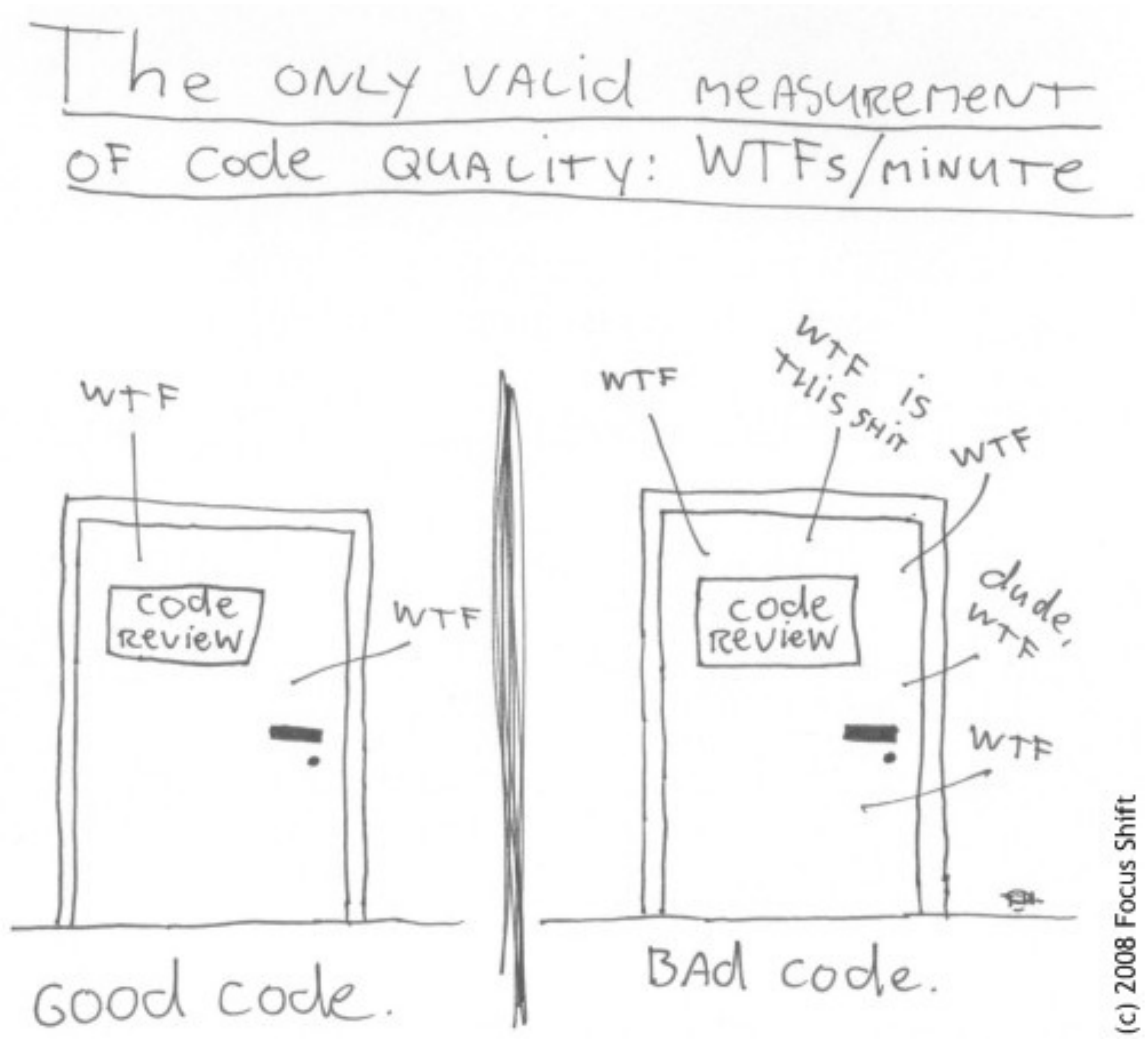
## Attribution

Much of this material inspired by a great slides from Adam Badura, available here:

https://cdn2-ecros.pl/event/codedive/files/presentations/2015/Code-review.pptx

*"Code review is having* ***other people***
*look at your* ***code***
*in order to find* ***defects."***

Oregon State University


The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
Bad code.
(c) 2008 Focus Shift

# Pros and cons

+ prevents releasing bugs

+ ensures architecture quality

+ facilitates knowledge transfer in the team

- time consuming

- don't work when the reviewer don't know the domain

- can hurt feelings

Oregon State University

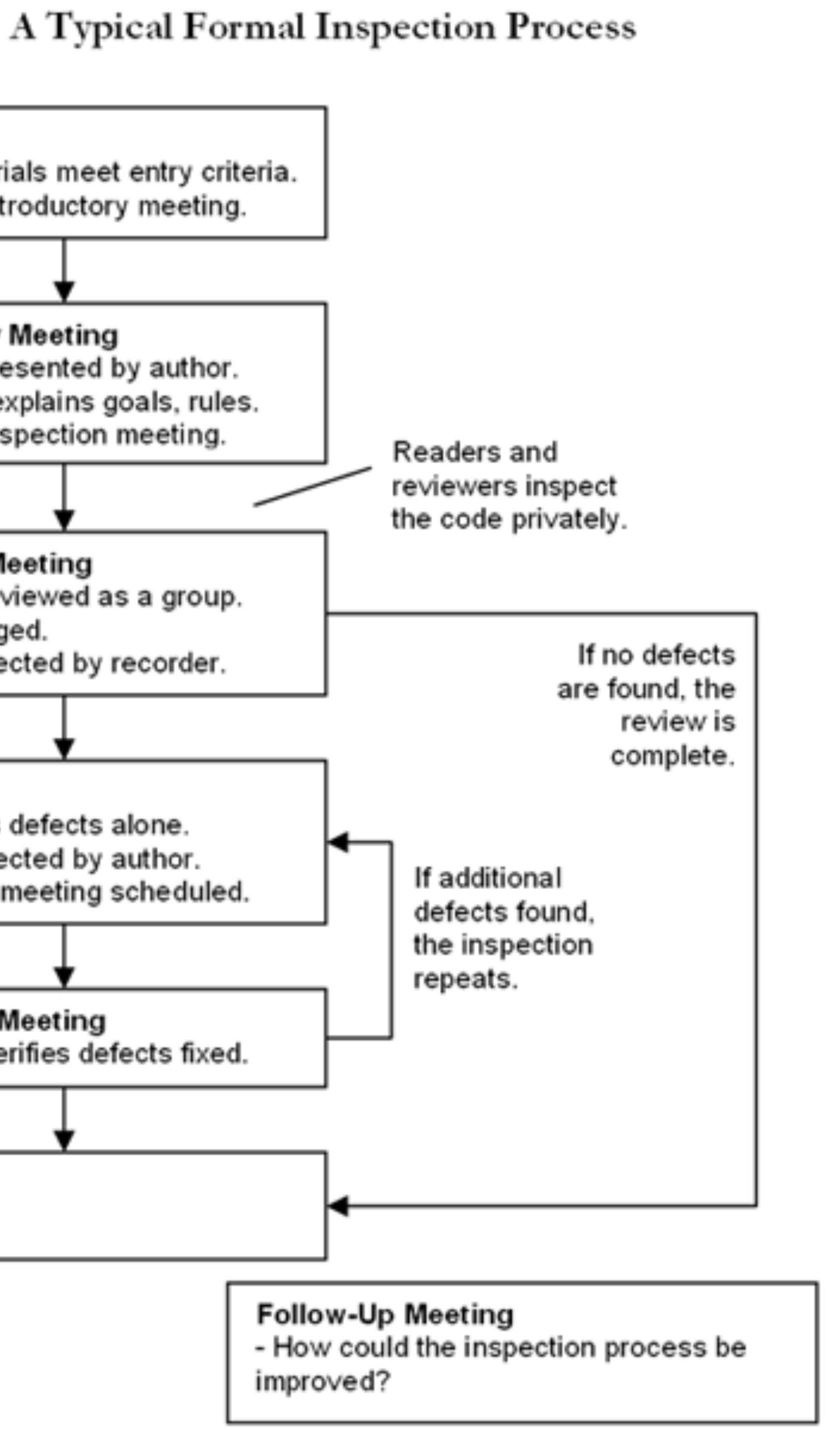
# Formal Inspections

Developed by Michael Fagan in the 1970's

A very heavyweight process

- 4 roles and 7 steps

# Formal inspection


A Typical Formal Inspection Process
Planning
- Verify materials meet entry criteria.
- Schedule introductory meeting.
Introductory Meeting
- Materials presented by author.
- Moderator explains goals, rules.
- Schedule inspection meeting.
Readers and reviewers inspect the code privately.
Inspection Meeting
- Materials reviewed as a group.
- Defects logged.
- Metrics collected by recorder.
If no defects are found, the review is complete.
Rework
- Author fixes defects alone.
- Metrics collected by author.
- Verification meeting scheduled.
If additional defects found, the inspection repeats.
Verification Meeting
- Reviewer verifies defects fixed.
Complete
- Done!
Follow-Up Meeting
- How could the inspection process be improved?

# Formal inspection

It works, but it's very expensive

~9 person-hours per 200 lines of code

Very impractical for today's realities

## Light Weight approaches

Over the shoulder

Pair programming

Pull requests

# Over the shoulder

Reviewer sits with the developer and looks "over their shoulder" at the code

The reviewer can give **informal** feedback which can be incorporated immediately (if possible).

# Over the shoulder

+ Easy to implement

+ Easy to complete

+ Easy to quickly incorporate changes

- Reviewer cannot review at their own pace

- No Verification

- Reviewer only sees what the developer shows them

Oregon State University

# Pair Programming

Code is written by a **pair** of developers.

Code Review is **"baked into" the process**

# Pair programming

+ Great for finding bugs and promoting knowledge transfer

+ Review is in-depth

- Reviewer is not objective

- Hard to do remotely

- No verification

# Pull Requests

Code is peer reviewed as part of the PR process

No PR should be merged without being reviewed by at least on other developer

# Pull Request Code Reviews

+ Can be enforced by Version Control Practices

+ PR serves as a verification of a review

+ Can be done asynchronously (great of remote teams!)

+ Reviewers can see the whole source code.

- Changes by hard to understand without explanation

- Important changes can be lost with a lot small insignificant changes

Oregon State University

# Best practices: design

Single Responsibility Principle

Code Duplication (copy/paste)

Squint Test

Left Code Better?

Potential Bugs / Missing tests

Error Handling

Efficiency

Oregon State University

# Best Practices: style

Method Names

Variable Names

Method length

Class Length

File Length

Commented Code

Number of Method Arguments

Readability

Oregon State University

# Best Practices: Testing

Test Coverage

Testing at the right level

Number of mocks

Meets requirements

## Practical Suggestions

Review < 400 LOC at a time

Don't review for > 60 minutes at a time

Use a Peer Review Checklist (language/domain specific)

Follow up with the review comments.

## Helpful Tools

https://www.codereviewhub.com/

https://www.jetbrains.com/upsource/

https://www.reviewboard.org/

https://reviewable.io/

https://www.gitcolony.com/

https://www.review.ninja/

Oregon State University

## Pair Programming

# Extreme Programming

One the first agile methods

TDD, continuous integration, refactoring were originally introduced by XP.

## XP Practices

**Pair Programming**

TDD

Continuous Integration

Refactoring

Small Releases

Coding Standards

Collective Code Ownership

Simple Design

Sustainable Pace

# Pair programming

2 programmers, 1 computer

**Driver:**

- Controls keyboard & mouse
- Deals with the details

**Navigator:**

- Thinks at a higher level
- Watches for typos, logical errors

Switch off every 10-20 minutes

Oregon State University

## Why?

Fewer defects

Higher design quality

Higher job satisfaction

Shared knowledge

Team-building and communication is enhanced

Raises your team's bus number

# Why not?

Two developers cannot be physically present

Strong personal conflicts

Task is simple and not challenging

When participants need a break