Hopefully nobody
“Looks Good to Me”

By a software engineer that probably doesn’t want to do code review, or a quick stamp of approval after a thorough code review
Code Review
What? Why? How?
A constructive review of a fellow developer’s code.

A required sign-off from another team member before a developer is permitted to check in changes or new code.
Why Code Review

Didn’t we already have tests?

- Average defect detection rates
  - Unit testing: 25%
  - Integration testing: 45%
  - Design and code inspections: 55% and 60%
- 11 programs developed by the same group of people
  - No reviews: average 4.5 errors per 100 LOC
  - With reviews: average 0.82 errors per 100 LOC
Why Code Review

Didn’t we already have tests?

- No reviews: average 4.5 errors per 100 LOC
- With reviews: average 0.82 errors per 100 LOC

IBM's Orbit project
- 500,000 LOC, 11 levels of inspections
- Delivered early with 1% of the predicted errors

After AT&T introduced reviews
- 14% increase in productivity and a 90% decrease in defects
“All code that gets submitted needs to be reviewed by at least one other person, and either the code writer or the reviewer needs to have readability in that language. Most people use Mondrian [now Critique] to do code reviews, and obviously, we spend a good chunk of our time reviewing code.”

Amanda Camp, Software Engineer, Google
Think different.
“What could go wrong?”

Famous last words
**Branch Protection**

<table>
<thead>
<tr>
<th>Branch name pattern *</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
</tr>
</tbody>
</table>

**Protect matching branches**

- **Require a pull request before merging**
  When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

- **Require approvals**
  When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.
  
  Required number of approvals before merging: 1

- **Do not allow bypassing the above settings**
  The above settings will apply to administrators and custom roles with the "bypass branch protections" permission.

**Rules applied to everyone including administrators**

- **Allow force pushes**
  Permit force pushes for all users with push access.

- **Allow deletions**
  Allow users with push access to delete matching branches.
New Workflow - Happy Path

- New Branch
- Commit, Push, (Repeat)
- Open Pull Request
- Code Review
- Merge
How the programmer wrote it
How the programmer wrote it
"At Facebook, we have an internally-developed web-based tool to aid the code review process. Once an engineer has prepared a change, she submits it to this tool, which will notify the person or people she has asked to review the change, along with others that may be interested in the change -- such as people who have worked on a function that got changed.

At this point, the reviewers can make comments, ask questions, request changes, or accept the changes. If changes are requested, the submitter must submit a new version of the change to be reviewed. All versions submitted are retained, so reviewers can compare the change to the original, or just changes from the last version they reviewed. Once a change has been submitted, the engineer can merge her change into the main source tree for deployment to the site during the next weekly push, or earlier if the change warrants quicker release."

Ryan McElroy, Software Engineer, Meta
What are we reviewing?

- Verification: are we building the system right?
- Validation: are we building the right system?
What are we reviewing?

- Verification: are we building the system right?
- Validation: are we building the right system?
What are we reviewing?

• Verification: are we building the system right?
• Validation: are we building the right system?

• Presence of good properties?
• Absence of bad properties?

• Identifying errors?
• Confidence in the absence of errors?
What are we reviewing?

• Verification: are we building the system right?
• Validation: are we building the right system?

• Presence of good properties?
• Absence of bad properties?

• Identifying errors?
• Confidence in the absence of errors?

• Robust? Safe? Secure? Available? Reliable?
• Understandable? Modifiable?
• Cost-effective?
• Usable?
public class Account {
    double principal, rate; int daysActive, accountType;
    public static final int STANDARD = 0, BUDGET = 1,
                            PREMIUM = 2, PREMIUM_PLUS = 3;

    public static double calculateFee(Account[] accounts) {
        double totalFee = 0.0;
        Account account;
        for (int i = 0; i < accounts.length; i++) {
            account = accounts[i];
            if (account.accountType == Account.PREMIUM ||
                account.accountType == Account.PREMIUM_PLUS)
                totalFee += 0.0125 * ( // 1.25% broker's fee
                    account.principal * Math.pow(account.rate,
                    (account.daysActive / 365.25))
                - account.principal); // interest-principal
        }
        return totalFee;
    }
}
“At Yelp we use review-board. An engineer works on a branch and commits the code to their own branch. The reviewer then goes through the diff, adds inline comments on review board and sends them back. The reviews are meant to be a dialogue, so typically comment threads result from the feedback. Once the reviewer's questions and concerns are all addressed they'll click "Ship It!" and the author will merge it with the main branch for deployment the same day.”

Alan Fineberg, Software Engineer, Yelp
Example Dialogue

https://github.com/apple/swift/pull/34094
Example Dialogue

Correctness
Efficiency
Alternatives
Documentation Changes
User Testing and Feedback
Making a Good Pull Request

Think like a reviewer

• Use descriptive but concise title and summary
• Describe context, rationale, and alternatives considered
• Link to relevant resources (specs, issues/bug tracker, previous PR)
• Provide screenshots/recordings for UI changes
• https://github.blog/2015-01-21-how-to-write-the-perfect-pull-request/
Logistics - CSE 403

How and where?
- Online/electronic
- In-person meeting
  - Best to prepare beforehand: artifact is distributed in advance
  - Preparation is critical and usually identifies more defects than the meeting

Who participates?
- One other developer
- A group of developers

What is reviewed?
- A specification
- A coherent module (e.g., checklist-style “inspection”)
- An entire component (“holistic review”)
- A single code commit or PR (“incremental review”)

“Can someone review this PR please? Thanks”

A message that you’ll likely see in Slack
YES
WHO?
WHEN?
Logistics

• Who should review what changes?
• How to set up (automated) notifications?
• How many reviewers per change?
• What's the expected review time frame?
• Approval for requested changes vs. general feedback
• Who submits after approval?
  • LGTM and "auto-submit" -> reviewer submits
  • Approval plus comments -> author submits
“We tried doing code reviews, but it was not useful. We never find any bugs or errors; the code is always approved.”
Logistics

• How to establish an effective and inclusive peer-review process?
• How to minimize biases in the peer-review process?
Action Items

- Figure out code review logistics with the team
- Start using pull requests and doing code reviews
- Enforce code review through branch protection
- Automate the code review process with CI checks