Code Reviews

Image: Christina Morillo on pexels.com

Topics

1) Why bother with code reviews?
2) What to look for as I do a code review?
3) How to make code reviews successful?

Purpose of a Code Review

Terminology:

Merge Request (MR) or Change List (CL):

the work to be reviewed

Based on Google’s Engineering Practices Documentation:
https://google.github.io/eng-practices/

Purpose

- Competing goals
  - Fight code rot:..
  - Allow code to..
- Principle: Seek continuous improvement of the code/product; don't allow quality to drop.
- Don't let 'perfect' be an enemy of 'good'
  - OK to accept a MR when it's good enough
  - Ex: May have *minor* inefficiencies or does only *some* of the possible improvements for an area of code
- MR should maintain or improve features and code quality

Code Review Mentality

- Everyone's goal:
  to improve the code and be better developers
- MR Author
  - ..
  - Ex: My experience with LibreOffice w/ numerous revisions
    24 patch sets: https://gerrit.libreoffice.org/c/core/+/62342
- Reviewer
  - ..
  - Politely help author to write better code
- The average code review **should** find changes;
  otherwise what's the point?
  - Reviewer points out concerns; original author fix them

Reviewer's Process

Timely

- ..
- Slow reviews are costly
  - Hard for dev to move to new task if last MR pending
  - Critical feedback is easier to accept if given quickly
- Prioritize doing a code review to unblock rest of team
  - Make doing a pending code review **next** your task
  - Don't interrupt your current task
  - Aim for multiple rounds of review in 1 day

Where to look

1. Big Picture: MR description
   - Is the description clear and concise?
   - What are the big parts of the change?
   - What issue is being solved?
2. Review the Major Part
   - Identify the file(s) that contain the core of the change;
     Review this first to understand focus of the change
   - If find a major issues,..
     don't waste time on rest of lower-level issues
     (likely to re-write)
3. Review the Rest
   - Work your way through all the files changed

Example

- Consider the following merge request:

- Preview what to look at (actual review later):
  - Title / description
  - Biggest area of change
  - Smaller changes

What to review

Use a code review checklist

Image: Pixabay on pexels.com

Design

- Look at the high-level design
- Ask:
  - Does this design make sense?

    ..
  - What abstraction does this module use?

    ..
  - What does this module's interface hide?
  - What will the code now depend on?

    ..
  - How are design heuristics used?
    - Prefer composition to inheritance

Functionality

- Ask:
  - Does the code do what the developer intended?
  - ..
    empty list, one element, null, 0, -10000000
  - For UI changes, consider trying out the feature (no expectation for the reviewer to be the tester

Code

Software's Primary Technical Imperative

McConnel, 2004

talk about essential
vs accidental complexity

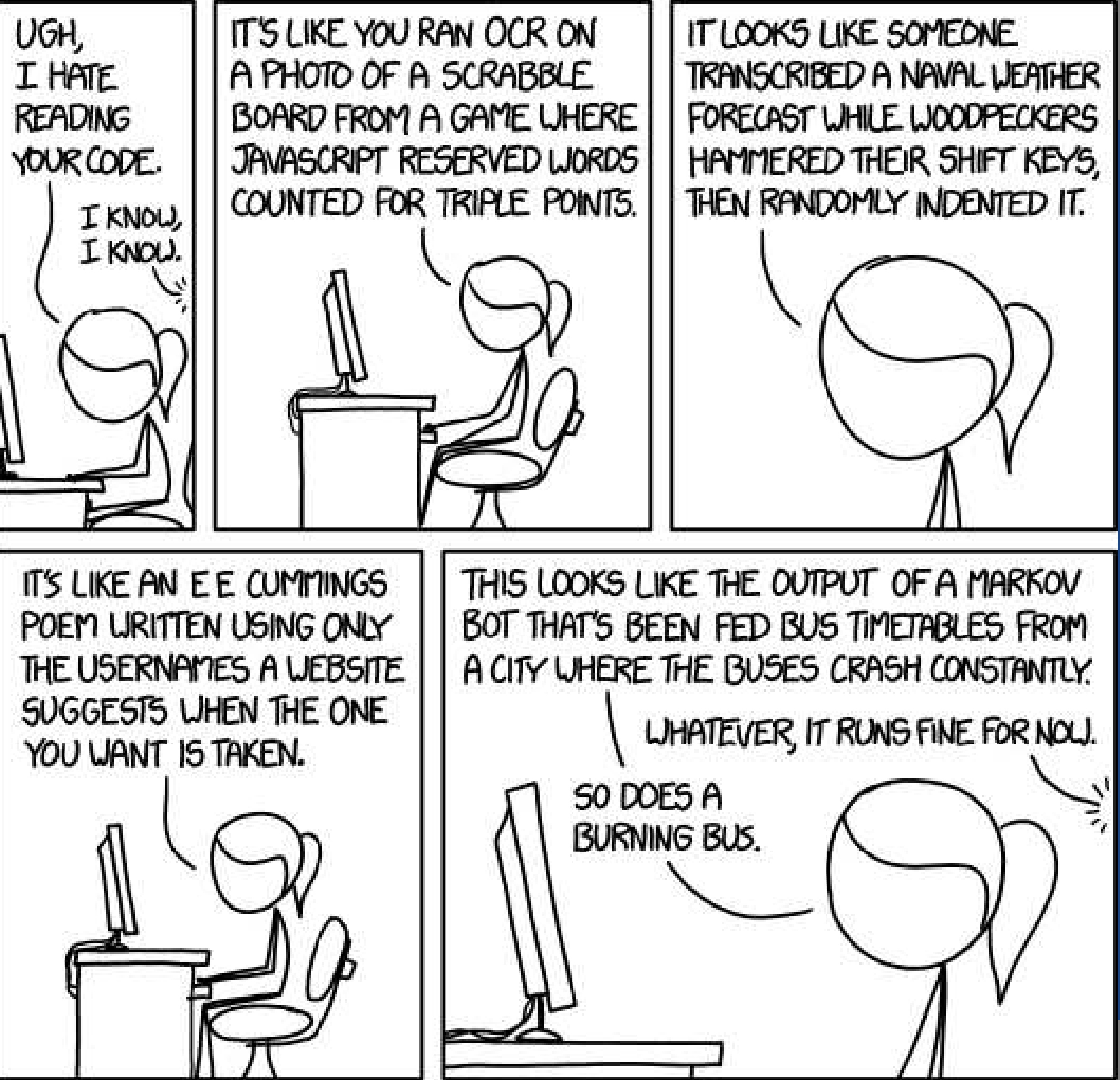
- Limit complexity
  - Is any line, function, or class too complex?
  - Can it be written simpler?
  - Does it over-engineer a solution?
  - Review every line of code
- Naming
  - Each name is descriptive and a reasonable length for its scope

Threads

- Carefully consider possible deadlocks or race conditions
  - Understand threads used in the system
- Identify data accessible from multiple functions
  - Which write? Which read?
  - Is the current locking (mutex/...) *sufficient*?
  - Is the current locking *necessary*?
- ..
  - Limit the scope of access to shared data

Comments and Documentation

- Comments
  - Clear, easy to read
  - Explains why code is needed, not what it's doing
  - ..
    include all necessary explanation in the code
- Documentation
  - If the MR changes how to build, run, test, or deploy the project, ensure documentation is updated


UGH, I HATE READING YOUR CODE.
I KNOW, I KNOW.
IT'S LIKE YOU RAN OCR ON A PHOTO OF A SCRABBLE BOARD FROM A GAME WHERE JAVASCRIPT RESERVED WORDS COUNTED FOR TRIPLE POINTS.
IT LOOKS LIKE SOMEONE TRANSCRIBED A NAVAL WEATHER FORECAST WHILE WOODPECKERS HAMMERED THEIR SHIFT KEYS, THEN RANDOMLY INDENTED IT.
IT'S LIKE AN E E CUMMINGS POEM WRITTEN USING ONLY THE USERNAMES A WEBSITE SUGGESTS WHEN THE ONE YOU WANT IS TAKEN.
THIS LOOKS LIKE THE OUTPUT OF A MARKOV BOT THAT'S BEEN FED BUS TIMETABLES FROM A CITY WHERE THE BUSES CRASH CONSTANTLY.
WHATEVER, IT RUNS FINE FOR NOW.
SO DOES A BURNING BUS.

Style

A style guide can help!

Source XKCD
https://xkcd.com/1695/

Style

- All code must match the style guide
  - If outside of style guide then:
    match existing code,
    update style guide,
    or don't fight about it.
- Prefix feedback with "Nit" (as in "nit-picking") when
  - it's a very minor issue, or
  - it's a FYI point to help teach a point, such as a comment on using the framework, etc

Tests

- Good tests with high code coverage
  ..
- MR should include tests (as needed)
  - There are no tests for tests, so
    ..
- A definition of "done":
  - Implements the feature
  - Clean code
  - Tests have ~100% (good) code coverage;
    all tests pass
- Consider a continuous integration (CI) pipeline setup with GitLab to automatically run tests

Also look for...

- Duplicate code: DRY
- 4C’s: Cohesion, Consistency, Completeness, Clarity
- Command-query separation
- Constructor makes fully-formed objects
- One-name for each idea

In-Class Demo

- Look at the following merge request in GitLab

- Comment on the overall MR
  - Title
  - Usefulness
- Comment on the code
  - Design?
  - Complexity?
  - Style, Comments & Documentation?
  - Functionality, Error handling & Tests?

Effective Code Review Tips

How to provide feedback

- Consider the following comment a reviewer might leave; what's wrong with it?

  “Why did you use threads here when there’s obviously no benefit to be gained from concurrency?”

- What's better about this?

  “The concurrency model here is adding complexity to the system without any actual performance benefit that I can see. Because there’s no performance benefit, it’s best for this code to be single-threaded instead of using multiple threads.”

Tips

- Discuss the **code** and **ideas**,
  not the author or reviewer
- Be respectful and kind
- Explain "why" when its possibly unclear
- Include complements on wonderful work
- Always include at least one feedback comment
  (for at least marking!)

Reviewing an Experienced Developer

- Hard to review a more experience developer
  - Try to learn from their code
- Look for:
  - edge cases they may have missed
  - error handling
  - code that is tricky to read
  - unclear abstraction (sometimes it's easier to see the big picture)
- Feel confident making suggestions for discussion
  - "Would it be cleaner to move this to new function?"
  - "Would it be clearer to rename this variable ...?"

After the Review

- Don't take it personally
  - All comments will be on the code;
    each is a suggestion on how to improve.
  - Don't be discouraged by others finding improvements
- Making changes
  - If code review suggests important changes,
    author makes changes and re-pushes branch;
    reviewer re-checks
- When reviewer satisfied, they thumbs-up;
  Repo manager merges

Angry Comments

Date	Sat, 13 Jul 2013 15:40:24 -0700
Subject	Re: [GIT pull] x86 updates for 3.11
From	Linus Torvalds <>

...
What the F*CK, guys?

This piece-of-sh** commit is marked for stable, but you clearly never even test-compiled it, did you?
....
Seriously, WTF? ... now I have to undo them all because this pull request was full of unbelievable sh**.
....
There aren't enough swear-words in the English language, so now I'll have to call you perkeleen vittupää just to express my disgust and frustration with this crap.

Linus

Angry Linus Van Pelt GIF by Peanuts ...
giphy.com

Linus Torvalds apologizes for years of ...
arstechnica.com

https://lkml.org/lkml/2013/7/13/132

Conflict

- Seek first to understand, then to be understood
  - What is the code or reviewer's feedback saying?
  - Give person the benefit of the doubt: what is the strongest argument they could come up with
- Discuss via video-chat (or in person)
  - Let the other person know you hear what they are saying
  - OK to let them know that you disagree
- Have another person join discussion
  - Have them find common ground
- Have the team discuss it

Summary

- Developers write code to advance the project
- Code reviews to ensure code quality
- Complete reviews within 24H
  - Review every line
- Be kind, be helpful, be critical
- Learn from every review