Code review is systematic examination of computer source code. It is intended to find mistakes overlooked in the initial development phase, improving the overall quality of software.

-Wikipedia
THE PAST

➤ ~ 2007
➤ no good tooling
➤ not yet established as good practice
➤ tooling is very good
➤ established good practice
➤ Everybody does it!
CODE REVIEWS ON GITHUB

@greysteil commented on the diff 4 days ago

```
@@ -82,7 +82,7 @@ def content_type(path)
82       end
83
84     def gzip_encoding_accepted?(request)
85     -  request.accept_encoding =~ /\bgzip\b/i
85     +  request.accept_encoding.any? { |enc, quality| enc =~ /\bgzip\b/i }
```

@greysteil added a note 4 days ago

This will now only work with Rack::Request objects. The only place that FileHandler gets called from is this file, so I think that's fine, but let me know if you think we should defend against receiving an ActionDispatch::Request here.

@matthewd added a note 4 days ago

Ruby on Rails member

I think this is fine. But we'll need to fix FileHandler#call too.

@greysteil added a note 4 days ago

Good point - done.
CODE REVIEWS ARE NOT A DISTRACTION FROM OUR JOB.

THEY ARE OUR JOB.
CODE REVIEWS ARE OFTEN DYSFUNCTIONAL

➤ not cost effective
➤ done inefficiently
➤ can even cause harm
WHAT IS WRONG?

➤ We concentrate on the wrong things.
  ➤ Technical part of the problem.

➤ We are not empathetic.
  ➤ Psychological part of the problem.
  ➤ The human factor involved.
WHAT IS WRONG?

➤ We concentrate on the wrong things.
➤ Technical part of the problem.

➤ We are not empathetic.
➤ Psychological part of the problem.
➤ The human factor involved.
WE CONCENTRATE ON THE WRONG THINGS

Which are:

➤ Syntax and coding styles
  ➤ discuss once and then move to the linter
WE CONCENTRATE ON THE WRONG THINGS

Which are:

➤ Things that are matter to personal taste
  ➤ and are not severe code smells
  ➤ and can not be handled by the linter
We concentrate on the wrong things

➤ We tend to do “bikeshedding discussions”
  ➤ You get 50 comments on a small 10 line change but none on a large pull request.
  ➤ Low complexity ⇒ high amount of discussion
  ➤ Parkinson’s Law
"The really, really short answer is that you should not. The somewhat longer answer is that just because you are capable of building a bikeshed does not mean you should stop others from building one just because you do not like the color they plan to paint it. This is a metaphor indicating that you need not argue about every little feature just because you know enough to do so. Some people have commented that the amount of noise generated by a change is inversely proportional to the complexity of the change."

(If you don't like the way we painted this bikeshed, try bikeshedding.io, or the minimalist shed.bike. Or set up your own? That's the spirit!)
Why Should I Care What Color the Bikeshed Is?

From freebsd.org/doc/en_US.ISO8859-1/books/faq/misc.html#bikeshed-painting; see also phk.freebsd.dk/sagas/bikeshed.html for historical background.

"The really, really short answer is that you should not. The somewhat longer answer is that just because you are capable of building a bikeshed does not mean you should stop others from building one just because you do not like the color they plan to paint it. This is a metaphor indicating that you need not argue about every little feature just because you know enough to do so. Some people have commented that the amount of noise generated by a change is inversely proportional to the complexity of the change."

(If you don't like the way we painted this bikeshed, try bikeshedding.io, or the minimalist shed.bike. Or set up your own? That's the spirit!)
Why Should I Care What Color the Bikeshed Is?

From freebsd.org/doc/en_US.ISO8859-1/books/faq/misc.html#bikeshed-painting; see also phk.freebsd.dk/sagas/bikeshed.html for historical background.

"The really, really short answer is that you should not. The somewhat longer answer is that just because you are capable of building a bikeshed does not mean you should stop others from building one just because you do not like the color they plan to paint it. This is a metaphor indicating that you need not argue about every little feature just because you know enough to do so. Some people have commented that the amount of noise generated by a change is inversely proportional to the complexity of the change."

(If you don't like the way we painted this bikeshed, try bikeshedding.io, or the minimalist shed.bike. Or set up your own? That's the spirit!)
Bikeshedding
Well there's your problem.
BUT WHAT IS IMPORTANT?
WHAT IS IMPORTANT?

The following are important but not discussed in detail here:

➤ understandability
➤ good naming
➤ test coverage
➤ edge cases
➤ security issues
➤ code duplication
MOST IMPORTANT:

CONCENTRATE ON EVERYTHING THAT MAKES THE CODE HARD TO BE CHANGED IN THE FUTURE.
WHAT IS IMPORTANT?

But what makes code hard to change?

➤ Too big things.
  ➤ Concentrate on the size of things.
    ➤ classes, methods, argument lists
WHAT IS IMPORTANT?

But what makes code hard to change?

➤ Doing too many things.
  ➤ Only do one thing at a time.
  ➤ Single Responsibility Principle
WHAT IS IMPORTANT?

But what makes code hard to change?

➤ Dependencies
  ➤ One object changes, other objects need to change too.
But what makes code hard to change?

- Dependencies in tests
  - The tests break with every change, though the overall functionality is not broken at all.
  - Learn to write loosely coupled and cost-effective tests.
“Most of these dependencies are usually unnecessary. They are a side effect of our coding style.”

- Sandi Metz
WHAT PRACTICAL THINGS CAN WE DO NOW?

... while we continuously improve our OOP design skills...
WHAT CAN WE DO NOW?

1. Define a **code review checklist** with your team.
   - Start the discussion
   - What is important in your special case, in your team setup, in your project?
CHECKLIST OF THE PAYMENT TEAM @BABBEL

Code reviews in Payment
Erstellt von Andre Wendt, zuletzt geändert von Pedro Vitti am Jul 01, 2016

Status: PROPOSAL

Note
There's a similar document targeted to all teams. In contrast to that, this document does not address how to communicate in reviews and focuses more on what to look for during a review.

We work on a business-critical part of Babbel. For us as a team, this means that leaving the system in an understandable and maintainable state is crucial for our long-term success as a company.

So, if you go the "long checklist" route of code review, what you wind up with, in a real sense, is a lot of busywork. Code inspection is a hugely valuable activity. Spend it doing things that require human intellect.

— Creating Your Code Review Checklist

Here's our code review checklist, heavily inspired by the post quoted above:

1. Does the PR have a clear description and provides all context and relevant/sufficient information (US, links, etc.) to the reviewer?
2. Does this code read like prose?
3. Do the methods do what the name of the method claims that they'll do?
   Same for classes?
4. Can I get an understanding of the desired behavior just by doing quick scans through unit and acceptance tests?
5. Does the understanding of the desired behavior match the requirements/stories for this work?
6. Is this code introducing any new dependencies between classes/modules and, if so, is it necessary to do that?
7. Is this code idiomatic, taking full advantage of the language, frameworks, and tools that we use?
8. Is anything here a re-implementation of existing functionality the developer may not be aware of?
9. Is the change the smallest possible unit that is deployable with no or low risk, while working towards a bigger goal?
   If not, how could it be split up?
10. Does this violate our notion of "code without tests is broken by design"?
WHAT CAN WE DO NOW?

2. Define a very **simple rule set** that declares **hard upper boundaries** for the size of things

- Classes $\leq 100$ LOC, methods $\leq 5$ LOC …

- Better design as a side effect without having to understand OO design.
WHAT CAN WE DO NOW?

Help others to get better!

3. Teach good OO design
   ➤ Share your knowledge.
   ➤ Code reviews are not always the best way to teach.
   ➤ As a company, support much more the mentoring career path.
FOCUS MORE ON TEACHING

Article “My Lawn” by Uncle Bob Martin:

➤ The amount of developers doubles every 5 years.
➤ We are dominated by novices.
➤ This is a problem for the whole software industry. Companies have to relearn their lessons endlessly.

My Lawn: http://blog.cleancoder.com/uncle-bob/2014/06/20/MyLawn.html
WHAT IS WRONG?

➤ We concentrate on the wrong things.
  ➤ Technical part to the problem.
  ✓ linter, bikeshedding vs. things that matter, CR checklists & simple rules, teaching

➤ We are not empathetic.
  ➤ Psychological part of the problem.
  ➤ The human factor involved.
WHAT IS WRONG?

➢ We concentrate on the wrong things.
  ➢ Technical part to the problem.

✓ linter, bikeshedding vs. things that matter, CR checklists & simple rules, teaching

➢ We are not empathetic.
  ➢ Psychological part of the problem.
➢ The human factor involved.
WHAT IS EMPATHY?

Empathy is defined as...

➤ the ability to understand and share the feelings of another.
➤ the ability to be nonjudgmental.
WHY IS EMPATHY IMPORTANT FOR CODE REVIEWS?

➤ We are giving feedback on something the other person cares a lot about.

➤ Giving good feedback involves a lot of empathy.
WHAT HAPPENS IF WE ARE NOT EMPATHETIC?

➤ People do not feel valued for their work.
➤ People are intimidated.
➤ We hurt each other’s feelings.

➤ Long-term damage of the open communication culture:
  ➤ People are afraid to ask questions and to ask for help.
  ➤ CODE QUALITY WILL SUFFER ENORMOUSLY.
WHAT HAPPENS IF WE ARE NOT EMPATHETIC?

In the worst case:

➤ People might quit their jobs.
➤ Recruiting and on-boarding new people is expensive.
WE ARE NOT EMPATHETIC

There are two directions of empathy involved:

- Empathy for the reviewer
- Empathy for the code author
WE ARE NOT EMPATHETIC

There are two directions of empathy involved:

Empathy for the reviewer

The code author

Empathy for the code author

The reviewer
AS AN EMPATHETIC CODE AUTHOR, YOU WANT . . .

➤ to make the reviewer’s work enjoyable.
➤ to avoid frustration for the reviewer.
➤ to put yourself in the position of the reviewer.
Code reviews change the way you write your code.

- You foresee questions when writing code.
- You make small, single-purpose commits
  - explaining your thought process
It is very important to give context.

➤ Choose a good **PR title** and **description**, add **screenshots**.
➤ Link to secondary material
➤ Ask yourself: What might not be obvious for the reviewer?
➤ Ask for specific feedback
In general:

➤ Reduce handovers by reviewing the code yourself before
  ➤ reduces 50 % of problems found later
➤ You open PRs early. Try to get feedback early.
➤ You make small PRs.
WE ARE NOT EMPATHETIC

There are two directions of empathy involved:

Empathy for the reviewer

The code author

Empathy for the code author

The reviewer
The code author...

➤ has put a lot of effort in the applied changes.
➤ might be very happy and proud of what has just been achieved.
➤ is smart and is doing a good job.
➤ cares a lot about his/her work.
➤ knows something that you don’t.
➤ You do not miss the chance to praise good work.
HOW TO MAKE A GOOD CODE REVIEW

RULE 1: TRY TO FIND AT LEAST SOMETHING POSITIVE

AT LEAST WE DON'T NEED TO OBFUSCATE IT BEFORE SHIPPING
As an empathetic reviewer, you...

Know that **written communication can be tricky:**

- It is easy to put people on the defensive.
- Your prefer to ask for clarification
  - instead of correcting people.
- You consider talking privately to the code author
  - instead of posting a huge list of comments.
know that **written communication can be tricky:**

- You don’t use sarcasm.
- You are careful with humour, animated gifs, etc.
- You avoid hyperbole. ("always", "never", "endlessly", "nothing")
TIPS & TRICKS FOR WRITTEN COMMUNICATION

Examples of good communication:

Use empathetical words:

➤ “us”, “our” and “we” instead of “you”, “your” and “mine”

➤ “We might also be able to improve our importer by …”

Be humble:

➤ “I am not sure but … We can look it up.”

➤ “If I remember correctly…”
TIPS & TRICKS FOR WRITTEN COMMUNICATION

Examples of good communication:

➤ “What do you think about…?”
➤ “It might also be possible to … Did you consider that already?”
➤ “This is interesting. What is the benefit of doing it this way?”
➤ “I didn’t understand completely … can you explain a bit more why..?”
Examples of bad communication:

Be careful with “Why” questions:
- “Why didn't you … ?”

Avoid using commands:
- “Please, stop using …”
TIPS & TRICKS FOR WRITTEN COMMUNICATION

Examples of bad communication:

Don’t talk down to someone:

➤ “... seems like a poor solution for …”
➤ “What I would suggest instead is some smart renaming …”
➤ “I can’t really see why … The whole idea was...”
FINDING COMPROMISES

➤ Be aware that you will never agree on 100% of the changes.
➤ Accept that code is always only a bunch of tradeoffs.
➤ At some point, you have to ship. The discussion has to end.
➤ You can say:

➤ “That’s interesting. But can we keep it like this for now?”

➤ “You might be right in the future. But can we revisit later when we know more?”
MORE ON COMMUNICATION

Nonviolent Communication

Second Edition

More than 1,000,000 copies sold for one simple reason: it works!

“...one of the most useful books you’ll ever read.”
—William Ury, Getting to Yes

Marshall B. Rosenberg, PhD

Endorsed by Arun Gandhi, Deepak Chopra,
Marianne Williamson, John Gray, Jack Canfield,
Tony Robbins, Dr. Thomas Gordon, Riane Eisler, and others
WHAT CAN YOU DO...

.. when you find *yourself* fighting about code?

➤ Take the discussion offline. Try to talk privately in person.
➤ Know that you are not a bad person, if it happens to you.
➤ But react like a good person. Step back. Apologise.
➤ Ask a third person for help.
WHAT CAN YOU DO...

.. when you find others fighting about code?

As a third person:

➤ Approach the people involved and ask how they feel?
➤ Offer your help as a mediator.

➤ Protect your code review culture and your healthy working environment!
THANK YOU!

My twitter: @raphaelawrede
MORE ABOUT CODE REVIEWS

➤ Book: Nonviolent Communication by Marshall B. Rosenberg
  ➤ https://www.amazon.de/Nonviolent-Communication-Language-Life-Guides/dp/189200528X

➤ Video: Implementing a Strong Code Review Culture
  ➤ https://www.youtube.com/watch?v=PJjmw9TRB7s

➤ Ruby Rogues Podcast: Code Review Culture with Derek Prior
  ➤ https://devchat.tv/ruby-rogues/216-rr-code-review-culture-with-derek-prior

➤ Blog post: Creating Your Code Review Checklist
  ➤ http://www.daedtech.com/creating-code-review-checklist/

➤ Blog post: On Empathy & Pull Requests
  ➤ https://slack.engineering/on-empathy-pull-requests-979e4257d158#.wokp23ee0

My twitter: @raphaelawrede
MORE ABOUT SOFTWARE DESIGN

➤ Book: “Practical Object-Oriented Design in Ruby” by Sandi Metz
   ➤ Examples written in Ruby
➤ Book “Growing Object-Oriented Software, Guided by Tests” by S. Freeman, N. Pryce
   ➤ Examples written in Java
➤ Video: “Rules” by Sandi Metz
   ➤ Learn how to define hard upper boundaries for the size of things
   ➤ https://www.youtube.com/watch?v=npOGOmkxuo
➤ Video: “The Magic Tricks of Testing” by Sandi Metz
   ➤ Learn how to correctly use stubs and mocks
   ➤ https://www.youtube.com/watch?v=URSWYvyc42M
➤ Refactoring talks by Katrina Owen
➤ Follow Sarah Mei on Twitter. Tweets on OO design topics.
➤ Uncle Bob Martin. Clean Coder.
➤ Level up your programming skills with Exercism.io

My twitter: @raphaelawrede