Code Review Best Practices (kevinlondon.com)
221 points by Kaedon 572 days ago | hide | past | web | 92 comments | favorite

trustfundbaby 572 days ago [-]

> If the reviewer makes a suggestion, and I don't have a clear answer as to why the suggestion should not be implemented, I'll usually make the change

This I feel is bad. Code reviews are usually between peers so you shouldn't be afraid to seek out clarification where possible. You shouldn't be making edits to code that goes in production without clearly understanding why.

The other thing that wasn't mentioned, that I think is important, is to not act as a blocker for code reviews unless its absolutely necessary. Lots of engineers take on the attitude that they're going to "gate" code they don't agree with by with holding their +1 and bogging down the review with questions and all sorts of runarounds till its what they want. this is a bad attitude to have, even when you're dealing with Junior engineers.

I'm generally going to +1 something unless I fundamentally disagree with it or think its going to break things in production. What I do, though, is leave lots of comments with questions/suggestions and mention it in the +1 with (see comments).

This builds trust on teams, and stops things getting personal, especially with people who aren't very good at dealing with criticism, even in something as banal as a CR. On a team that works well together, teammates will see those comments, think about them and make thoughtful responses, especially once they understand that you're not trying to get in their way. Giving the +1 gives them the freedom to consider your suggestions without being irritated that their PR is being blocked. They feel like they're in control not you.

In rare exceptions, someone will brush off my questions and merge ... which means that next time, I get to be tougher on the review and specifically ask for responses before the code can be merged, because they've degraded the implicit team trust. Usually repeat offenders are assholes, and assholes generally don't last on healthy teams.

> Cymen 572 days ago [-]
>
> I agree 100% with the "don't be a blocker." I personally look for about 80-90% yes and if it meets that criteria with no errors, I thumb it up. If I can't thumb it up, I try to put a "thumbs up with commented items fixed". I try to not block -- comment for bad issues, sometimes sigh but agree when it's a 10% disagreement and go forward. Life is a series of iterations. That 10% will be addressed in a future iteration.
>
> And I never close another persons PR.
>
> allsystemsgo 572 days ago [-]
>
> I agree. I have had review comments that have made no sense at all. If I didn't ask why, I wouldn't learn. Also, to be honest, there have been times where the review comment didn't understand the context of my code, so the review comment ended up being incorrect.
>
> nazbot 572 days ago [-]
>
> I'm going to differ on the 'don't be a blocker' thing.
>
> It varies by team. If your whole team agrees that the code reviewer is the 'gate' then people understand that they just need to fix the thing and move on. I personally love that style of working because you tend not to write trivial suggestions and everyone knows that what people have written isn't meant as nit-picky and just has to be changed.
>
> If you don't think it's the right change then the other person can just ask why it's important and hopefully get educated. I also tend to just trust my colleagues enough that what they suggest is a good change. I often find I may miss small things through like lack of documentation or whatever (get tired after a while) and those reminders in the CR are the equivalent of a spotter getting you to do one last rep at the gym.
>
> nimnio 567 days ago [-]
>
> From a certain perspective, you just paraphrased the quote you disagreed with:
>
> Him: "If the reviewer makes a suggestion, and I don't have a clear answer as to why the suggestion should not be implemented, I'll usually make the change."

You: "[Do] not act as a blocker for code reviews unless it's absolutely necessary."

I think these express essentially the same philosophy, with some slight differences in the particulars.

pablobaz 572 days ago [-]

While I agree with all the points listed in the article, it highlights for me a major problem of a lot of code reviews.

Most code reviews seem to focus on:

1. Examining what the change does 2. Finding ways to make the change in a nicer way. E.g. Refactoring etc.

This leaves out the key step 0 - what is actually trying to be achieved, does it need to be done and is there a better (maybe completely different) way to do it.

This leads to a focus on relatively trivial matters such as naming conventions and method lengths.

I think that the underlying reason for this is laziness. Talking about clever refactoring is an easier/faster process than understanding the 'why'.

MichaelGG 572 days ago [-]

I'm guilty of getting stuck up on trivial formatting issues. When someone pushes a commit that has random whitespace (trailing or arbitrary newlines all over, or just inconsistent spacing), it feels sloppy. Same for many other simple things. If the code is unnecessarily superficially ugly, it sets up a block in my mind that makes it harder to focus on the real issues.

Is it wrong to kick this stuff back and tell devs to make it pretty first?

dkubb 572 days ago [-]

I believe that ugly code is usually buggy code. If someone didn't take the time to deal with trivial issues like formatting then it makes me wonder if they spent time thinking hard about the actual problem they were trying to solve.

I usually explicitly do code reviews in 2 passes. I look for low level problems first, and also try to identify ways that they can be tested using tools so that next time they are caught before the code is sent for review. My second pass looks at higher level problems with the overall design and checks the functionality to make sure it matches the objectives.

yoran 572 days ago [-]

I agree with this. But I don't think code reviews are suited for this step 0 as you say. Just the way a pull request is formatted, it's very hard for the reviewer to deduce from the changeset what the high-level design of the code is.

That's why we discuss these high-level design and architectural decisions before-hand so that they are known to the reviewer at the time of the review. We're a small team so it works well. But I'm not sure how this scales up as the team gets bigger. I would like to know how bigger teams approach this problem!

jeremiep 572 days ago [-]

Same here, I've seen too many code reviews where people complain about a badly named variables and nobody saw the design was faulty leading to costly bugs to fix in production.

One thing we did on the current project is pre-commit reviews in pairs. This ensures at least two people in the team knows about the changes, let us talk about the why and how of the changes, and possibly teach a coworker new things in the process.

What it ended up doing is that every programmer now self-reviews their own changes prior to the actual review knowing they'll soon share it all face to face with a coworker. Turns out the talks are now about the design of the code, not how it looks.

dsuth 571 days ago [-]

You should not really be arguing about design and architecture in a code review - that should be done in design review.

enqk 572 days ago [-]

setting thresholds on method / class sizes seems quite arbitrary and potentially harmful.

Splitting a method into n different ones, none of which is called more than once is setting up the code for opportunistic reuse and obscures it's true function. It's especially wrong if the code that was split is mutating / non pure functional.

see http://number-none.com/blow/john_carmack_on_inlined_code.htm...

userbinator 572 days ago [-]

Agreed, I find it difficult to read code that's been split up into many small pieces because it gets difficult to keep track of the overall flow. Even with an IDE, the jumping around (and maintaining a mental call-stack) can be quite distracting.

My guideline is basically "as big as it needs to be, without having large pieces of duplicated functionality." If an algorithm requires 100 lines, none of which are repeated or very similar to the others, then so be it.

I wonder what the distribution of function lengths is like in the Linux kernel... there will be longer ones and shorter ones, but a 20-line limit seems absurdly short.

Also vaguely related: http://www.satisfice.com/blog/archives/27

zo1 572 days ago [-]

You're not supposed to "flow" through multiple levels of abstraction in your head. That is the fundamental problem, and why we so often think that it's okay to have large functions (you know, to 'keep it all in our heads', or whatever reason is most often touted).

Not only that, but as the OP mentioned that it doesn't work if the methods/functions have side-effects, then we're building up opinions on a sloppy and fundamentally broken house of cards.

First off, your methods should not be having side-effects that are not obvious from the purpose of the method. And then secondly, back to my original point about not having to flow through the code. In light of that, a function should do one thing, and that's it. And when it does so, the method name clearly states what it does. After you put those two together, you simply "assume" that method does what you expect it to, and "flow" through the logic at a higher level.

I'm not going to go down into some silly "read_input_data" method while "flowing" through the high-level code, because I know what it is supposed to do. And by extension, at some point I will review it in its singular/atomic entirety, or would have already done so. Either read_input_data returns the data, or it loads it up into a instance-variable.

mikehaggard 572 days ago [-]

>setting thresholds on method / class sizes seems quite arbitrary and potentially harmful. Splitting a method into n different ones, none of which is called more than once is setting up the code for opportunistic reuse and obscures it's true function.

Sometimes, but in general I don't agree.

If the functions are made private and giving appropriate names than it's really a way of self-documenting code, which helps.

Suppose you have:

```
some_function() {
    // Calculate interest
    // 100 lines of code

    // Apply tax
    // 120 lines of code

    // Store results
    // 30 lines of code
}
```

Then the need to have those "banner comments" there already indicate it might be a good idea to decompose the function into:

```
some_function() {
    calculate_interest(...);
    apply_tax(...);
    store_results(...);
}
```

It's not just about reuse, but about documenting what each section of code does, and

about limiting interactions between arbitrary sets of variables. Functions are create boundaries (assuming they don't intensively use global variables or class instance variables when part of a class)

enqk 572 days ago [-]

This depends on the language capabilities. In the C/Lisp/ML families of languages at least you can create isolated scopes. It's unfortunately not true of Javascript (and Ruby?).

jbbarth 572 days ago [-]

In Ruby some people achieve this by wrapping actions in their own class, which does the trick if the classes are kept independent from each other (no shared state, except global vars, which nobody uses in the ruby world).

enqk 572 days ago [-]

I should have been clearer. I meant isolated scope *within* a function.

flipperkid 572 days ago [-]

Where I work we refer to guidelines like this as code smell. These are cues that refactoring should be considered but not strict rules which would prevent a commit.

Kaedon 572 days ago [-]

Yes, exactly true. There are situations where it's probably fine to violate the guidelines if there's a good reason.

hueving 572 days ago [-]

I disagree. Breaking down a function into several one time functions that all sit at the same mental model of abstraction make code much easier to reason about.

ThrustVectoring 572 days ago [-]

A named one-time-use function has a name that describes what it does. This seems obvious, but bears repeating: instead of having to figure out what a chunk of code does, you can guess from the name and have a series of steps naturally described.

bsdpython 572 days ago [-]

You can get the same effect with a block of code by properly naming variables and adding a single comment line if need be. It's really just a personal preference.

ThrustVectoring 572 days ago [-]

You'd have to have a standard vim config with folding to get it exactly the same.

Also, there's more restriction in that the single-use code block only has access to the variables passed into it, which makes it easier to parse, since less things could be happening. It serves to prune variables from the local scope.

Fr0styMatt8 572 days ago [-]

I agree with you in principle but find that code editors let me down in that regard.

Say I split a function up into a few sub-functions because it's getting big. Now I have the problem that I'm jumping forwards and backwards through the code when I want to explore what that function does:

SomeMassiveFunction() { SubfunctionA(); SubfunctionB(); SubfunctionC(); SubfunctionD(); }

SubfunctionA() { } ....

In this case, SubfunctionC() might end up a few pages down in source code. So now it's a context switch to go there and then go back.

Now this can be somewhat avoided with good function names (so you don't HAVE to jump backwards and forwards) and keyboard shortcuts (to make the process

quicker), but it's still a trade-off that I think needs to be kept in mind.

samspot 572 days ago [-]

I think you've hit upon a good measure of function quality. If you find you have to jump around a lot when reading, that would be a sign that it's poorly organized and needs to be refactored. On the other hand, if you find you don't have to jump around and can trust the sub functions by their names, then it's been broken up well.

In the best case you should be able to follow the logic without diving into the other functions, only looking at their implementation details as that particular detail becomes relevant.

enqk 572 days ago [-]

From a static point of view this is true, however splitting the functionality into sub functions might later obscure commonalities that would have been obvious if they were still part of the upper function.

It all depends at which stage of development you are at. If the piece of code we're talking about is mature and rarely changing then yes it seems like a reasonable thing to do. If however this section is still under development then I would opt for another way of dealing with readability issue.

I think if what you want is to optimise the readability of the code then a simple comment above each section, combined with block scoping is a good set up for splitting.

As the true nature of the code emerges one can decide to turn the block into local lambdas *then* later into functions.

Again that's because of the first point I've made here in this comment. If you are still developing the functionality, splitting early means that subsequent reviews&development may miss potential interactions + potential refactorings/simplifications that would have been quite clear if the function was still "messy"

seabee 572 days ago [-]

IME refactoring optimises for a local minima of code entropy. Any time you want to add new functionality, you're going to have to jump back out into 'mess' to implement the feature, then 'fix' it again with further refactoring, with all the extra overhead this entails.

acveilleux 572 days ago [-]

If you can't understand what UsedToBeSomeMassiveFunction() does from the SubFunction*K*()'s names and arguments, they were not split out correctly.

Often I find the best way to simplify large functions is to tear out sub-blocks and give them name. Loops or large conditional blocks are usually easy to tear out and can usually get very meaningful names. Long stretch of imperative code however does not separate well and probably should remain a very large function.

arthurjj 572 days ago [-]

"Long stretch of imperative code however does not separate well and probably should remain a very large function."

As someone who uses Resharper's Extract Method a lot this is a nice counterexample to where it's probably inappropriate

swsieber 572 days ago [-]

I think it's a trade off we could get away from if we improved our tools. We should make tools that reduce our cognitive over head. I am regularly diving through code at my new job trying to figure out how everything fits together. Reading code split between various functions and figuring out program flow is a necessary skill, even if you do keep monolithic functions.

I would love an IDE that showed inline in some sane manner function calls, letting me recourse arbitrarily deep - some thing ala code collapsing and expansion. Extra points if you can show a for chart to side of function calls, class interactions and general code flow, highlighting where your cursor is. More points for a pane that shows what variables have to be in order for a the branch

of code you're working in to be reached. A changeable granularity setting would be great for all of these.

I think often times we discuss pros and cons without discussing what they could be if we put some effort into changing the current situation.

avn2109 571 days ago [-]

You have suggested so many wins here. Gallons of win.

BurningFrog 572 days ago [-]

Split into functions with clear tasks that have clear names and are relatively standalone.

If you have to jump around between functions a lot to understand something, the code probably could use some reorganizing.

ams6110 572 days ago [-]

When I have to deal with something like this in emacs I will normally split the window in two so I can look at two different parts of the buffer at the same time. I assume most other editors allow the same?

krzyk 572 days ago [-]

If you name the methods correctly (what they do) splitting actually helps readability of the code. It is quite hard to read and remember what was in the beginning of 500 line method.

AlisdairO 572 days ago [-]

IE it can help to understand the intent and flow of the code, but also make it harder to debug. If I'm in a situation where I want to follow the exact instructions a large fraction of the code performs, a call stack jumping all over the place can be pretty frustrating.

motti 571 days ago [-]

Step over

AlisdairO 571 days ago [-]

...doesn't help if I want to actually see what code is getting run

mhomde 572 days ago [-]

It's always a balance between "The Blob" and the "Poltergeist" anti-pattern, when in doubt go more towards borderline "The Blob".

I just say though it's very seldom that a huge class/method is justified. Usually there are plenty self-enclosed logic or reusable methods that can be broken out either to utility libraries or a separate class.

My rule of thumb is that you should be able to look at a method/class and understand what it does, and that diminishes quickly over a certain size.

Sure some logic might be in other classes but that's another layer of abstraction

reipahb 572 days ago [-]

I agree that arbitrary limits can be a bit restrictive at times, but that is mostly an issue when dealing with automated code checking tools.

However, in this case it is more about something to keep an eye for for the reviewer -- if the function is large and complicated it may be a good idea to take a closer look at it. Presumably the reviewer won't force the function to be broken into smaller pieces unless it actually improves readability and maintainability of the code.

yoklov 572 days ago [-]

Its extremely subjective though. For the first few years after starting, I thought that large functions were bad and should be split up unless theres a clear reason not to.

Now, I feel that it should be the other way. Most of the time you want to keep the code that does something together, splitting it up only to factor out common reusable pieces (that are actually reused).

This is also discussed in the linked article.

bliti 572 days ago [-]

I mean, some functions/methods will be longer than usual. Even if you split it into multiple ones the end product will be the same. Due to some of those derivative functions/methods being exclusive children to the caller. Sometimes it's best to leave the bigger one alone, and some times it helps to break it down. You do end up with basically the same size codebase. Dunno if it's more readable. Though this somehow shines some light into other issues. Such as language verbosity. Some languages are just syntax factories. All the logic is accompanied by a truck load of verbosity.

I agree with your point. It seems silly to be strict about such things. More so if the codebase is written in Java or any other verbose language.

Kaedon 572 days ago [-]

Yeah, thank you for the link! I use this list as a set of rough guidelines, basically, so it may or may not apply in all cases.

Sometimes it can be helpful to split up a method to be called once if it helps readability, for example if it helps split up a loop or abstracts the specific way we get a value.

USNetizen 572 days ago [-]

The one thing that is missing, which ALWAYS seems to fall by the wayside, is security. If people incorporated more iterative security testing (static AND dynamic, automated AND manual) and threat modeling into their SDLC reviews there would be a plummeting number of vulnerabilities.

But, because it doesn't fit in with the whole "Lean" approach to software (deliver features yesterday), all but the most established enterprises don't seem to care much unfortunately. Once more people experience a breach because of their desire to deliver first and remediate vulnerabilities later then perhaps more awareness will be raised. By then it's too late though.

maguirre 572 days ago [-]

I have an interesting problem. A co-worker of mine appears extremely sensitive to his code being reviewed and I honestly don't know how to deal with it. He feels attacked (and becomes defensive during code reviews) because the reviewers focus on the "bad things and mistakes" of his code instead of the accomplishments.

Has anyone here dealt with similar issues during reviews?

MaulingMonkey 572 days ago [-]

Balancing the critique with some positive feedback - parts you liked, appreciation for the features shipped, etc. - is one idea I've seen floated around, for a particularly touchy coworker.

Also, generally managing tone - "Looks good! Don't forget to add the doc comments on functions X Y and Z" vs "Why are X Y and Z missing doc comments?". Both have the same fundamental information (X Y and Z could use doc comments), but:

With the former, you're acknowledging progress, a "good" first draft, and pointing out what remains to be done in an ego friendly way ("I know you're good enough to make this even better") that doesn't demand a response.

Meanwhile, the latter demands a response, all of them negative. Either "it doesn't need them", or "I forgot them (because I suck)", or silently ignoring the question. None of these are ego friendly.

.

If the code just plain sucks, however... I'm not really sure what to do.

And if the coworker gets defensive even with positively slanted feedback... maybe there needs to be a conversation with them, to try and shift their mindset. Show them you get things pointed out in your own code reviews, that it's a team game where everyone is just looking out for each other. That bad code doesn't mean they're a bad programmer, that critique of their code isn't a critique of their skill, that everyone gets blind to the problems in their own code and benefits from a second set of eyes.

Maybe get them in your shoes - helping critique someone else's code, and help explain that they're not being an asshole when they point out edge cases and ugly code, but that they're being an asshole to future them when they have to go back and add a feature, fix a bug, or handle a corner case they'd missed.

maguirre 572 days ago [-]

Thanks for the feedback. We are a small team (3 embedded sw developers) and I am more or less the guy with actually production releases under my belt. I feel my approach has been very measured and very polite to the point where I feel like i am walking on "egg shells" when doing the code reviews.

Show them you get things pointed out in your own code reviews, that it's a team game where everyone is just looking out for each other. That bad code doesn't mean they're a bad programmer, that critique of their code isn't a critique of their skill, that everyone gets blind to the problems in their own code and benefits from a second set of eyes.

within a few days we're going to be trying exactly this in hopes to show how code reviews are not about attacking and instead about helping.

nazbot 572 days ago [-]

It's a problem for small teams and people with little experience with code reviews. The 'walking on egg shells' thing happens ALL the time.

I've never seen this get solved overnight. Your management really needs to sit down and explain their expectations around code reviews - that fixing things based on co-worker suggestions is a positive thing, that having negative comments won't affect their performance reviews, etc.

I also find that with people like that you really do need to let more things go. If it's not a critical bug or an actual problem with the code then let it slide. Some people just cannot take criticism (even constructive) and the bad blood you'll get by persisting with it will not help you.

I also tend to find these developers are not necessarily the strongest devs and often have other personality problems. This likely indicates that your company needs to focus a lot more on the hiring and culture.

mmcconnell1618 572 days ago [-]

Try taking power away from the reviewer to disarm the defense reaction. In Ed Catmul's Creativity, Inc.

(http://www.amazon.com/gp/aw/d/0812993012/ref=mp_s_a_1_1?qid=...)

he describes the Pixar "brain trust" which reviews early progress on movies with the director. They found directors would get defensive if they felt someone more powerful was telling them to make specific changes. Once they set some rules like "the director decides what, if anything gets changed" and "talk about what's not working but don't solve the problem for the director" things improved. Now instead of film veterans telling you your movie (code) stinks it becomes about opportunities to make things awesome.

halostatue 572 days ago [-]

If you know that your colleague is sensitive to what appears to be the overwhelmingly “negative” comments, make sure that there *are* positive statements. I had a code review recently for one of my guys that had a *lot* of comments that were style and logic problems.

When I finished the comments on the code, I made sure I added a comment to the top of the pull request (we use BitBucket) indicating that it was clear he understood the point of the code and had written a good first pass at it, but there were a few things that needed to be dealt with for idiomatic code.

dewiz 572 days ago [-]

Try with design sessions upfront, i.e. discuss how the problem should be solved in terms of patterns and architecture. Once that is agreed, with 2-3 people, then the code review becomes simply a matter of style and there you can only invite for consistency with the rest of the code base. Often the problem is in the tools, face to face conversations, pair programming, human interaction help with that. You can also have your colleague help with a task of yours and show that you appreciate his input.

maguirre 571 days ago [-]

+1 This is something I've been meaning to introduce. We're slowly moving from the idea of hacking away and making things work to the idea of making production ready code and this mentality is not getting through to everyone

supercanuck 572 days ago [-]

Try a compliment sandwich... Sandwich the criticism between two positive statements.

e.g. I like the way you did this, but I think this would work better but you're on the right track.

> curun1r 572 days ago [-]
>
> The feedback sandwich is something to be avoided. People are generally predisposed to either hear positive or negative feedback, but not both. When you give sandwiched feedback, those that easily hear positive feedback will miss the negative and those that easily hear negative feedback will miss the positive. Almost no one will hear both.
>
> Or so says the management training courses I've recently taken. Supposedly there's studies to back this up, too, though I've never read them myself.

Cymen 572 days ago [-]

Have you considered going out to lunch with them or grabbing a coffee and bringing it up? If you're in an organization where you wouldn't be comfortable with that, maybe talk to your manager and ask them about it? Basically, we all have to toughen up a little bit when getting reviewed and it is for the good of the project/team. So getting someone who is uncomfortable past that is a win-win for everyone.

Blackthorn 572 days ago [-]

Sounds like he told you exactly how you should deal with it. Stop focusing solely on on the bad things and mistakes: make sure to mention "hey, neat idea" or "this is pretty clever!" on parts where, you know, it is exactly that. Diplomacy is a useful skill.

tomjen3 572 days ago [-]

Sounds like he is too attached to his code - maybe make it abundantly clear that you are critiquing his code not him. If you have the ability to do so, try to have him fix bugs in other peoples code (that way it might feel less like his code).

cpitman 572 days ago [-]

I'm assuming you are performing asynchronous code reviews (like pull request comments). If you have the time, one of my teams did occasional "Code Inspections", where a team of 5 people would review a larger chunk of functionality together.

The key is that each person has an assigned responsibility, and that one of those is the "Reader". The reader presents the code for review, and the reader is NOT the author. The author is there, but I think it helps when someone else is presenting the code to reduce the feeling that the review is of the author.

shakeel_mohamed 572 days ago [-]

+1 I'm currently dealing with this. The individual doesn't offer feedback on my code when under review, so it appears like a personal attack.

> frossie 572 days ago [-]
>
> It is important that code reviews are public enough that people see other people review each other's code - a system where the only code reviews you see are the ones you do and the ones you get leads to poor expectations of what they are *for*.
>
> Having at least two very accomplished and (culturally and organisationally senior) people routinely "model" reasonable review behaviour can be stunningly effective.
>
> Also this is an area where frequent team conversations about what good code is outside a review situation helps to build a certain culture. It helps step away from nitpicking and arguing.
>
> halostatue 572 days ago [-]
>
> Go have coffee (or whatever) with your colleague and have a 1-to-1 conversation with them saying you *want* their feedback on code. You want their expertise to make you a better software developer. Even if you are better at this than they are, you *can* learn something from the sharing.
>
> The team’s lead should make sure that the individual in question knows that software development is a cooperative process and that code review helps build the team.

MaulingMonkey 572 days ago [-]

This. Although I'm less civil about it - I'll start giving people shit for rubber stamping my code reviews.

"Why are you letting me ship this terrible code I wrote? Do you want us to end up in a death spiral of technical debt and crunch? Don't be an asshole, critique my shit! We all need a second set of eyes - I'm no exception!"

...okay, I'm maybe a little more civil than that. But I'm not above leaving some mistakes unfixed to call them out on.

garthk 572 days ago [-]

Ha! I'm like that with my QA people, begging them to come after me with a blunt weapon.

hliyan 572 days ago [-]

We do the bulk of our reviews at the pull request stage, as line comments on commits (we use Github). These are *visible to all team members*. This form allows developers to take the time to phrase the feedback diplomatically.

I've also set the tone early on in my own comments (I run the development division) and I've encouraged the more experienced and talented members of the team to review my own pull requests. When they spot a problem in my code, I always make it a point to praise them (if, of course, the problem is valid). This seems to have worked.

Too 572 days ago [-]

Ask your manager to make the review a *formal and required* part of the process, for everyone. That way everybody is treated equally. If you go ask your colleague out of the blue why his code looks the way it does he might get defensive and wonder why you were suspicious to review his code behind his back.

meejah 572 days ago [-]

I have, and it was more an attitude/cultural thing: the reviewee took any criticism very personally, and was hence super defensive and unlikely to change anything anyway.

Honestly, I think it's actually really, really hard *not* to react that way (to varying degrees) especially as most code-reviews basically come at the wrong time: you've already written the code, it works, it has glorious unit-tests -- who cares if it's a for vs while loop or whatever. It's actually even worse if there are serious design or maintainability issues, as that's even *more* code to re-write/"fix".

Personally, I really like having code-reviews -- it's better to get rid of serious problems if they're there. However, I've found a lot of reviews to be rather worthless "rubber-stamp" exercises -- *especially* if the dev team has just been told Thou Shalt Do Code Reviews Because Reasons! (I've also found the opposite, for sure: getting nice solid advice or missed cases; but these are invariably with other developers who *want* reviews no matter what management said).

...but all that said, I think so-called "code reviews" really work best if they're spread out over the coding (unless it's really small chunks at a time) -- a bit of pair-programming (or "hey, can you help me with ...") and design (or whatever you call "early on in the cycle") reviews make it a *hell* of a lot easier to take a different approach.

I think they used to call this "collaboration"?

Companies that just blast out a "new rule: everything gets code reviews!" memo without helping people understand the benefits, *and* how to do more "along-the-way" validation of design, refactorings, etcetc are always the very worst at having code reviews that are in any way effective. At such places, I find "code reviews" devolve into either useless "rubber stamp" affairs or something like what you describe. Or worse.

Also, if you have "Junior Developers" blasting out code for a couple days (or more!) with no adult supervision with a hope that some last-second "code review" procedure is going to equal amazing code, you're doing mentoring wrong. It's a lot nicer for everyone involved if there is someone sitting down with (or chatting or watching-the-commits-of or whatever works) the less experienced developers. These mentors should be providing hands-on, practical advice *as they go*. Some devs might need more hand-holding, some will need "a plan", maybe you need to sketch out classes or method-signatures for them, perhaps they need a bunch of half-day tasks written down, etc...Such mentoring should, IMO, be what your Senior Developers are spending a decent chunk of their not-coding

time on.

cmpb 572 days ago [-]

Nice list. This is more or less what I look for. It's nice to see your rules of thumb.

Anyone have any suggestions for time-estimating code review? That's the biggest issue we've faced trying to implement code review into our workflow.

> Kaedon 572 days ago [-]
>
> SmartBear has their own set of best practices and they recommend about 300-400 loc per hour[1]. I think that's probably about right. It's something I think I've gotten faster at over time, partially because we're improving from previous reviews and partially because it's a skill that develops.
>
> [1]: http://smartbear.com/smartbear/media/pdfs/wp-cc-11-best-prac...

BurningFrog 572 days ago [-]

This is not a bad list, and can be useful as a checklist.

But note that it basically tries to define good programming practice in general. That's a very big topic with a lot of room for debate and disagreement.

mikehaggard 572 days ago [-]

>Variable names: foo or bar are probably not useful method names for data structures. e is similarly not useful when compared to exception. Be as verbose as you need (depending on the language). Expressive variable names make it easier to understand code when we have to revisit it later.

I so agree with this!

Properly named variables is perhaps THE first line of defense against bad code. Too many developers think they are concise and having little code if they only abbreviate their variable names enough.

Honestly, "em", "erg", "fc" and "sc" may make perfect sense to use, but it's a form of obfuscation to future developers (including your future self).

Other pet peeve; adding things to variable names that don't add anything meaningful.

E.g.

"usersList"

Does your code really care that it's a list? Should the reader be pointed at this each and every time. I much prefer just using:

"users"

Clear, to the point, and readable.

> MichaelGG 572 days ago [-]
>
> It's not a form of obfuscation if there is clear and simple context. Catch(ex) is perfectly fine, and adding 7 letters does nothing but add noise. Similarly "var us = getUsers()" is clear - it's not one u, it's multiple, and makes sense to have a loop like "for u in us".
>
> A better alternative if you find the code is still confusing is to add context by cutting a function up into inner functions. This is why I really hate working in languages that make it difficult to define little closures. In F#, I'll often end up with a couple of 1 or 2 line inner functions and it makes things much easier to read. This simply isn't practical in e.g. C#.
>
> In exported function names and certain class names or modules, sure, a bit of verbosity might help. But inside a function it just make it visually harder to understand and I've rarely found it to be beneficial. There's going to be enough context to load into my brain inside a function accurate, and unfortunately I still subvocalize when reading code and all those extra syllables add up.
>
> Additionally, removing letters means you need to split lines less based on length and focus more on when it makes sense.

bottled_poe 572 days ago [-]

Some of this is frustrating to read. Architectural and detailed design decisions should be made and approved almost entirely before the coding of those features is started. (Obviously this is

the ideal and not always possible).

This means that the code reviews should involve little more that a checklist of those approved design decisions against their implementation, perhaps a code style is verified as well.

Coding without a design is just hacking, which I believe is the primary cause of burn-out and should be avoided as much as possible.

So, the question is, do you have a design document? I know it doesn't sound very agile, but traditional engineering procedures, when managed well, have a lot of merit in controlling product quality and cost.

shakeel_mohamed 572 days ago [-]

Another thing to check for that I didn't see addressed is debug statements. There's nothing quite like seeing console.log("shit"); during a code review :)

zatkin 572 days ago [-]

Awesome. I'm joining Cisco for the summer, so I think this would help me get a head start since they do code review. Thank you!

> Kaedon 572 days ago [-]
>
> Sure! Best of luck at Cisco. I think everyone looks for different things in a code review, this is just what has worked for me.
>
> azatris 572 days ago [-]
>
> Are you the person who took my place? :) Got to the last stage, twice, in London.
>
> To me, the Code Review Best Practices seem awfully like very general knowledge, just gathered together. Not sure if it's HN-worthy per se.
>
> However, the John Carmack link is quite enlightening.
>
> > zatkin 572 days ago [-]
> >
> > I sure hope not -- I'm working at the headquarters in San Jose this summer.

Too 572 days ago [-]

This list is very very basic, most of the things like style shouldn't have to be discussed and design should preferably be done before the code is written.

Just adding "error handling and potential bugs" as a generic bullet on the list just doesn't cut it, these should basically be the only items on the list but specified in much greater detail. A serious code review checklist should contain concrete scenarios of these, preferably tailored for your specific application.

Examples of this are: What happens during system startup, before all other modules are ready to answer to requests? What happens if the user enters malformed data (sql injections etc)? Does this function shut down gracefully (transaction safe)? How will the algorithms scale? Race conditions and deadlocks? What happens if the network to the database goes down? Is localization handled properly? Backwards compatibility?

bozoUser 572 days ago [-]

While I agree with all the points in the blog, I wonder how many programmers do really follow them perfectly(even the author of the blog) because doing code review to such great detail requires plenty of time which is often not the case when you work for corporations.

> chunkstuntman 572 days ago [-]
>
> One company I worked for relied heavily on code reviews after every feature. At least two co-workers (one of whom had to be a supervisor) read, ran, and gave feedback on every piece of code. Reading others' code and providing feedback allowed me to improve my sight-linting ability, and it felt like each day my group's code as a whole was improving.
>
> Having some accountability for writing sloppy code is very sobering.
>
> > clebio 572 days ago [-]
> >
> > > supervisor ... ran ... code
> >
> > This and this again. Engineering managers, and all that. But also, don't just review,

'sight-lint', and reason about code. Rather, run the tests! It's the shared accountability and knowledge. Does the baseline capability exist (tests pass)? Can you, the reviewer, spot fallacies that the tests don't capture (if not, criticisms feed back to you later)?

Kaedon 572 days ago [-]

I'll admit that I don't follow them perfectly either. I'll usually pick one or two instances and work on those. I think most of these items are on a continuum from severe to minor basically. It's usually when it's on the severe end that I'll suggest it might be worth changing.

My code has bugs and flaws too! That's why I like code reviews, they give me an opportunity to see my blind spots.

Kiro 572 days ago [-]

> If we have to use “and” to finish describing what a method is capable of doing, it might be at the wrong level of abstraction.

When I coee, somewhere a method needs to initiate this execution flow and will therefore contain "and". Even if all it does is call two other methods where this principle is followed. How do I avoid this? I mean, somewhere in the code it makes sense to execute the methods together.

justinfreitag 572 days ago [-]

Style, complexity and coverage checks should be left to automated tooling. Code reviews should focus on whatever remains.

a3voices 572 days ago [-]

Having worked for businesses that use code reviews and those that don't, I personally favor not having code reviews. The reason is that they hinder development speed quite a lot, since you have to try to predict what other engineers will say on your reviews, which takes a lot of brainpower.

jdcantrell 572 days ago [-]

I think code reviews are critical for sharing knowledge on team projects. It helps keep your team informed about refactors and new functionality, while also giving a space for feedback on implementation (in a critical time, before the code has shipped). It allows a very organic way for people to learn from others (reading code, asking about code, thinking about others' code).

That said, you have to get useful code reviews to see any benefit. To me that means using automated tools to do most of the style checks (your braces should be on this line, no space after this foreach, etc) and having an active culture of not being human-powered code linters when doing code reviews. There is a lot of work that goes into having a team give effective code reviews.

I agree that code reviews can slow down an individual, but the speed up to the team through shared understanding *should* make up for that.

jlarocco 572 days ago [-]

Also having experienced both, I wouldn't want to work in a place that didn't have code reviews.

I ask myself, "WTF was this guy thinking when he wrote this?" far less often when I'm working on code that's been code reviewed. Most of the time, the reviewer catches code like that, and requests that it be fixed and/or commented, before I end up debugging through it 6 months later.

IMO, open office plans, reading HN and nerf guns hinder development speed far more than code reviews.

MaulingMonkey 572 days ago [-]

I experienced both within the same company - they started without code reviews, and the later introduced them. The result was overwhelmingly positive, greatly improving code quality and helping spread knowledge of new systems and utilities.

I certainly believe it's possible to do code reviews sufficiently "wrong" that they're a net hindrance. Drape them in too much ceremony, put too much emphasis on style (Bob doesn't like your whitespace preferences, and he's the code owner, so your changelist is vetoed!) and too little emphasis on subsistence (these corner cases and

possible bugs concern me).

.

We went with a relatively light touch approach. Nothing was technologically enforced - our lead dev simply told us he wanted everyone to start having everything reviewed by anyone - both for code quality and to spread knowledge - and that he'd be pissed if he found a bug in one of our changelists and found out it hadn't been reviewed, going forward.

Whitespace / naming stuff I tended to submit without review.

Quick and obvious fixes, small changes, etc., I was fairly willing to start a review request, submit, and then fix anything caught in the review in a followup changelist.

For larger changes where I had more reservations, I'd usually hold off on submitting until after a review. Maybe threaten to submit if they were dragging their feet too long :P

matwood 572 days ago [-]

> since you have to try to predict what other engineers will say on your reviews

So the code is written with higher quality up front? Sounds like reviews are working as intended.

tomjen3 572 days ago [-]

If so, don't you think that is what he would have written?

There are enough differences over what can be good quality that teams can be bogged down over discussions that just have no good answer and these take a lot of brain power.

jrbancel 572 days ago [-]

I agree that requiring deep code reviews on every code change is ridiculous. Unfortunately, I have never seen a team that doesn't have at least a developer that is much worse than the average member of the team. He can be bad at abstraction, at design, at naming things or anything else. This developer produces sub-optimal (or even worse, incorrect) code that is hard to understand, maintain and extend.

I have observed that naturally, the other teammates will review the code of this specific coworker while not looking at changes submitted by good and reliable teammates.

I see code reviews as a tool to provide feedbacks to someone and help him improve the quality of the code he writes.

OnlineRevenge 572 days ago [-]

[flagged]

Kiro 572 days ago [-]

This is the strangest spam I've ever seen on here.