10 THINGS DEVELOPERS WISHED THEIR BOSSES UNDERSTOOD ABOUT CODE REVIEW

Collaborator

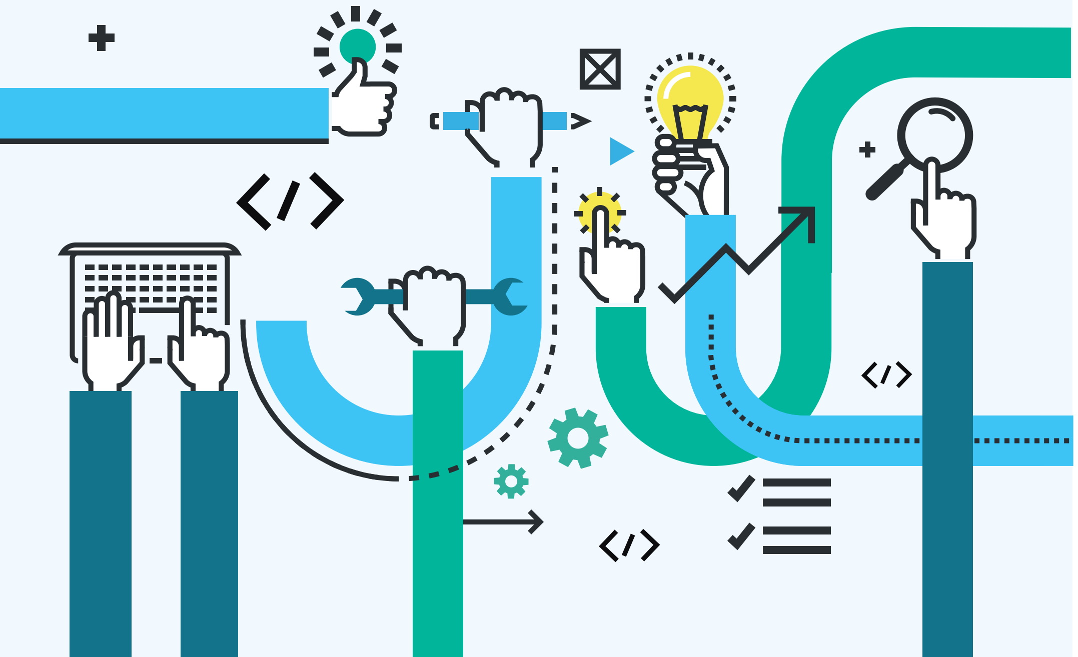
Collaborator

Collaborator is a code review tool that helps development, testing and management teams work together to produce high quality code.

LEARN MORE ABOUT COLLABORATOR

Content

Code reviews can help developers improve the quality of their applications – a goal that everyone agrees upon. To ensure that process is done right, though, sometimes developers need their managers' help and buy-in. Here's ten guidelines for application development managers (and the people they report to) to organize and run code reviews so that the process works efficiently (and with no tears).



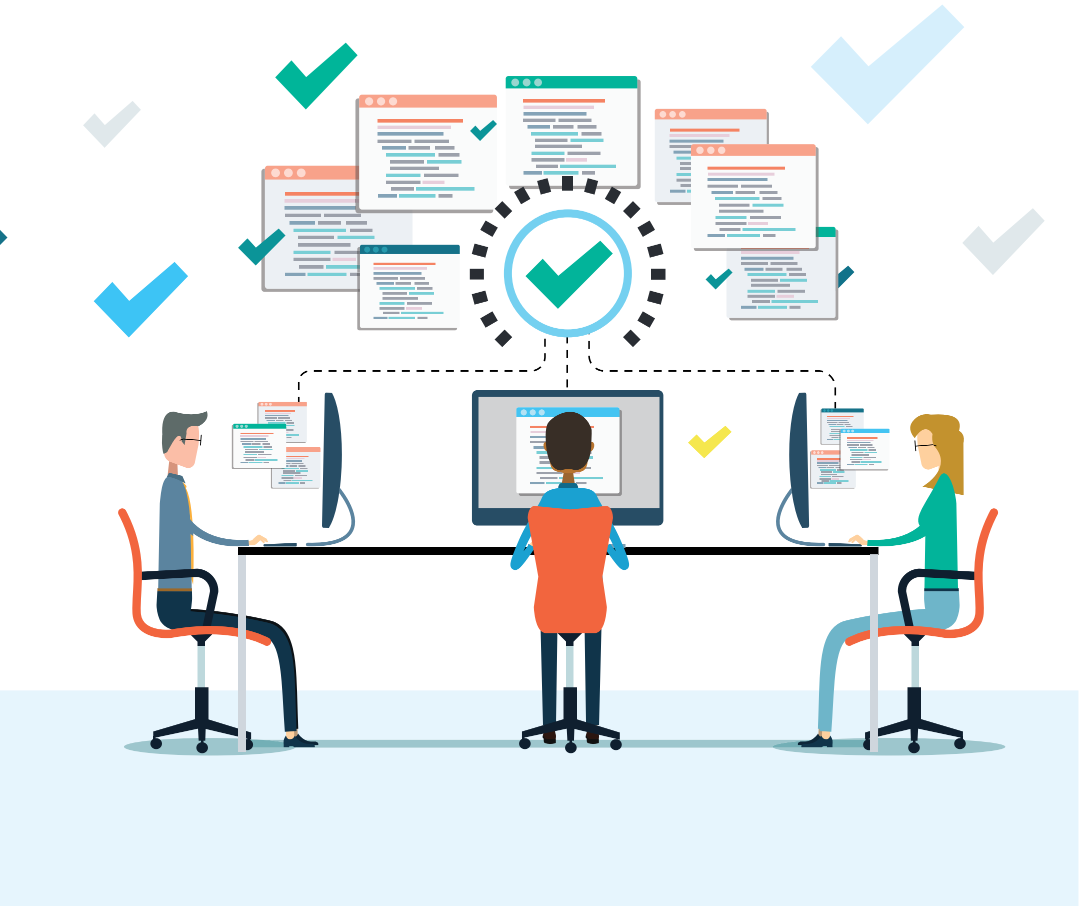
Anyone who leads a development team wants business processes that help their team members (as well as managers) to evaluate the work in-progress, improve its quality, and learn from one another so everybody does better next time. Doing a code review – whether that's as simple as an over-the-shoulder glance at a software module or a formalize process – is a great way to accomplish that.

But code review works well only when it is implemented in a way that encourages the team to embrace the process. That's where you, the manager or team lead, come in. Part of your role is to set standards, create the team culture, and establish a workflow that encourages team members to do their best.

"Code review works well only when it is implemented in a way that encourages the team to embrace the process."

Ensuring that effective code reviews happen is only one example of your responsibilities, but, developers say, it's an important one. Team members depend on their managers to set the goal – to improve code quality – and to put-and-keep attention on the work, and not on the worker.

So, how can you best make code review work? With input from dozens of software developers, we put together these guidelines for success. Here's what developers want you to know, and what they want you to do.



1. The Goal Is Improved Code Quality.

Let's start with the definition of code review: A code review is any process by which multiple people examine another developer's code. To many developers, that means a meeting-based code review, where a bunch of people pile into a room armed with annotated printouts, only to discover that someone else already ate all the good donuts.

But "code review" is anything in which, well, code is reviewed:

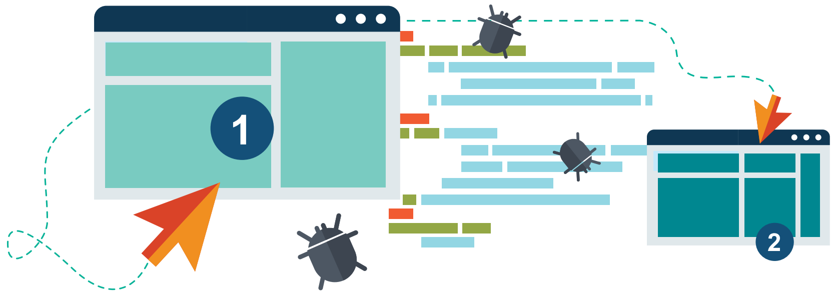
- It might be an experienced developer physically looking over the shoulder of a junior programmer and commenting, "That isn't the best way to write that SQL statement; let me show you a better technique."
- Peer programming incorporates ongoing code reviews, since the developers communicate about what they're writing (and, we like to think, why).
- Any email message in which a programmer sends a block of code asking others to "Take a look at this...?" is code review.

- Tool-assisted code reviews developers to read code, annotate it, and make suggestions from the comfort of their own desks. (No donuts required.)

We want to be clear: The advice we offer in this white paper applies to any form of code review. Because all these processes share a common goal: to improve application quality.

> *"Just because something in your code works, doesn't mean it's actually done right."*

We all want to produce higher quality code so that, when we ship the newest release, we know the code is solid. As every developer has learned painfully, just because something in your code "works, doesn't mean it's actually done right."



It is known that code review helps developers find bugs sooner, discover user-story assumptions before they're instantiated into code, and fix problems that would pass automated tests. The earlier in the software development cycle that a problem can be spotted, the cheaper it is to fix – in time, dollars, and frustration.

Among the types of bugs found during code reviews:

- Missed test cases: "You tested for an empty string, but not for null."
- Typos
- Un-optimized code
- Unnecessarily complex code
- Re-implemented code: "We already have a function that does what you're doing here, and does it better."
- Lazy documentation

At least a few of these defects would pass automated systems. Unit tests that don't exist don't fail; code coverage is 100% if you forget to write test cases; and "documentation" can be useless ("Here I added 1 to the total"). Without performing a code review, a team may not realize that a developer did loops in loops, suppressed exceptions, didn't log enough or too much, or wrote pointless tests.

2.Quality Code Saves Money.

Surely we do not need to belabor the point that finding bugs earlier has a financial benefit? Fixing defects earlier in the lifecycle translates into delivering on time, not to mention the need to go back later for expensive debugging and software updates. That has business value, even if it is difficult to measure in real dollars.

Some developers find that their management pushes back because of the time involved in doing code reviews, and "time is money." But developers wish their bosses would understand: Code review might take time now, but it's worth it in the long run.

"It slows us down in the short term. There's no denying that." explained a senior developer on a small development team at a biotech company. "It takes time for me to stop what I'm doing, to get into the code review mindset, and then switch back. It takes time for people to stop and ask us to do a code review on a pull request if they really want to merge and we haven't looked at it yet." But in the long term, the developer continued, the team catches bugs before they happen.

Having to wait for someone to review code before it can be merged can cause a delay, especially if it blocks another area of functionality. As explained by another developer, a backend and API engineer at a small start-up, the wait is never more than a couple hours. She added, the benefits far outweigh any added delay in shipping code.

"Code review might take time now, but it's worth it in the long run."

3. Code review is worth it for the knowledge transfer alone.

Beyond the opportunity to find and fix bugs early, developers see code review's value as a way to learn from other people, and to share their own knowledge. That's true both at the meta-level – making the team smarter – and in regard to a particular application. That is: Code review is one way to demonstrate that the company is committed to not only overall software quality but to respectful listening and expects people to learn from each other. Whether the knowledge imparted is detailed ("Here's how to improve that SQL statement") or at a higher level ("Let's step back a moment while I explain my philosophy on software performance…"), everyone on the team benefits.

On a team that does code reviews, everybody understands the codebase and its data flow. More importantly: everyone understands the reasoning behind the decisions that were made when changing the code, not just the mechanics of the code. For instance, if everyone recognizes that this application needs to be lightning-fast, the code written (and reviewed) will be created with that priority in mind. That makes it easier to make choices across the project, since everybody is on the same page for the "why."

Code reviews enable all the developers on a team to get to know the entire codebase, rather than just the small areas for which each individual is responsible. "This sort of bigger-picture awareness is invaluable," said the IoT developer in Colorado. "Now that I'm actually a part of all aspects of the codebase, even if just by reading code and not necessarily writing it in some areas, I can easily jump in and fix bugs wherever they come up, instead of having to take a long time to try to understand that service and then maybe contribute. Overall, we definitely have better code and better communication within the engineering team now that we do code reviews."

"I've been working on a mature and very complex code base," one developer said. "Some of the things I was asked to code review weren't just coherent nuggets of code that built something new, but tasks that involved scattered changes to scores of existing files." The code review process helped that developer become involved in other areas of the code base with which she wasn't familiar. "I would have had a lot of studying to do, just to figure out what it was all about!" she concluded.

There's all sorts of "team value" in having everybody understand the codebase, but the shared knowledge also protects the company. If only one developer knows one part of the code, the organization is at risk, especially if the developer might be in a car accident or hit by a bus. Or he might leave the company to take another job. You can lose the Wise One for any number of reasons, but they all can result in panic when the developers' code needs attention.

This isn't a matter of whether any given developer's code (the senior developer with tenure or otherwise) is well-written. Does it have enough comments to explain how the software works – where "enough" is a judgment call by the reader, not the person who wrote the documentation? If someone needs to go back to that software in six years,do the comments make sense? Do they reflect how the code currently works, not the way it operated when it was written six years ago (and much-maintained since then)?

Code review gives developers a reason and opportunity to explain what they did why they did it. This alone can have quality benefits. As with so many other things, the act of explanation sometimes helps you walk through your own assumptions and sometimes recognize their weaknesses. Often, telling someone else what you did makes you evaluate whether you approached the problem the right way.

"Code review is one way to demonstrate that the company is committed to not only overall software quality but to respectful listening and expects people to learn from each other."

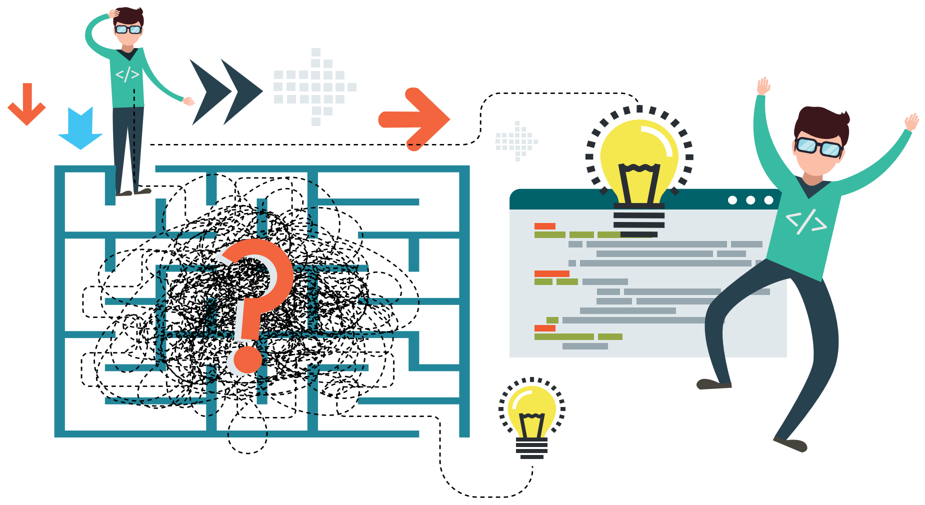
And that, of course, leads to documenting the explanations when they're unusual. During the code review, another developer's comment that the solution is elegant but confusing may encourage clarity of documentation – and that can save time later on. Annotating code lets the next programmer who touches the code better understand how things connect. Especially since that next programmer might be you, in three years from now.

4. There’s no better way to mentor someone.

In every code review session, someone should learn something new. This is not necessarily the person whose code is being reviewed.

Few managers need to be convinced of the reasons to encourage mentoring, wherein more-experienced developers teach the less experienced staff new programming techniques – the things that managers call “best practices” and the line-of-business developers describe as, “Things I wish I hadn’t learned the hard way.” The process of going over code is an ongoing process of people querying the developer’s thinking in making his algorithmic choices, making suggestions, discussing whether the resulting application addresses the requirements and user stories, and quietly sharing wisdom without necessarily recognizing it as wisdom. It’s teamwork at its best.

Times are changing; younger team members are more in-tune with global software trends, and given their more recent college education they may have even more useful recent computer science knowledge.



The “teaching” doesn’t go in only one direction, with senior developers imparting knowledge to junior staff. Older folks can learn new tips and tricks from the junior developers, too.

“Things I wish I hadn’t learned the hard way”

“I learned a language feature from someone who was many years my junior [during a code review],” said one developer. “I had a way of doing something and had no reason to research alternatives; whereas they didn’t know how to do something and researched an updated method.”

The mentoring process is not only about programming techniques and algorithmic skills. Code reviews help create internal standards. They give junior developers and new hires an opportunity to learn "how we do things around here." This happens at a natural pace: Senior developers commenting on juniors' code; the juniors asking questions of the seniors; less-experienced developers paying attention to the conversation between other developers. The end result is that a new team member is exposed to the organization's methods (programming and otherwise). By the time the new developers need to modify code, they understand globally what is going on.

> *"It's your job – as manager – to ensure those team members do understand the problems. After all, one of those reviewers might be the developer modifying the code the next time."*

We sometimes hear hesitation from application development managers – and reluctant programmers – expressing the sentiment that the reviewers do not understand the problems the programmer had to deal with, with the result that much of the reviewers' advice is not very useful. If you encounter that attitude, then it's your job – as manager – to ensure those team members do understand the problems. After all, one of those reviewers might be the developer modifying the code the next time.

5. Incorporate code review at the right stage in the process.

It's one thing for us to say, "Do code reviews," but like any other business process, it can either be performed well or poorly. And, as with other parts of the application development lifecycle, part of success is knowing when a code review should happen.

In general, we recommend that you incorporate code review early in the development process. You might not have a choice in this regard; some version control systems, such as Git, prevent code from being committed until it's been reviewed. Younger developers who got their early experience writing open-source software expect their code to be reviewed sooner rather than later.

Another reason to incorporate code reviews early into the workflow is that developers are closer to the code. If someone asks, "Why did you write it this way?" the developer remembers her reasoning. If she had made wrong assumptions about the user story, she hasn't written additional code dependent on this module. Even if you do the code review post-commit, going back to find the problem point is easier than finding the defect after you ship it to QA (or to the customer).

There's also an emotional reason to do code reviews early: the developers' sense that they were "done" with that chunk of code. If you wrote a module and sent it off to QA, you might dust off your hands and think, "Well, that's finished! I can go on to the next user story." If the code review is conducted after developers feel that they are "done" with the code, they may be upset that they have to go back to something they thought was complete. Who would blame them?

> *"Another reason to incorporate code reviews early into the workflow is that developers are closer to the code. If someone asks, "Why did you write it this way?" the developer remembers her reasoning."*

6. Make the process a sensible one.

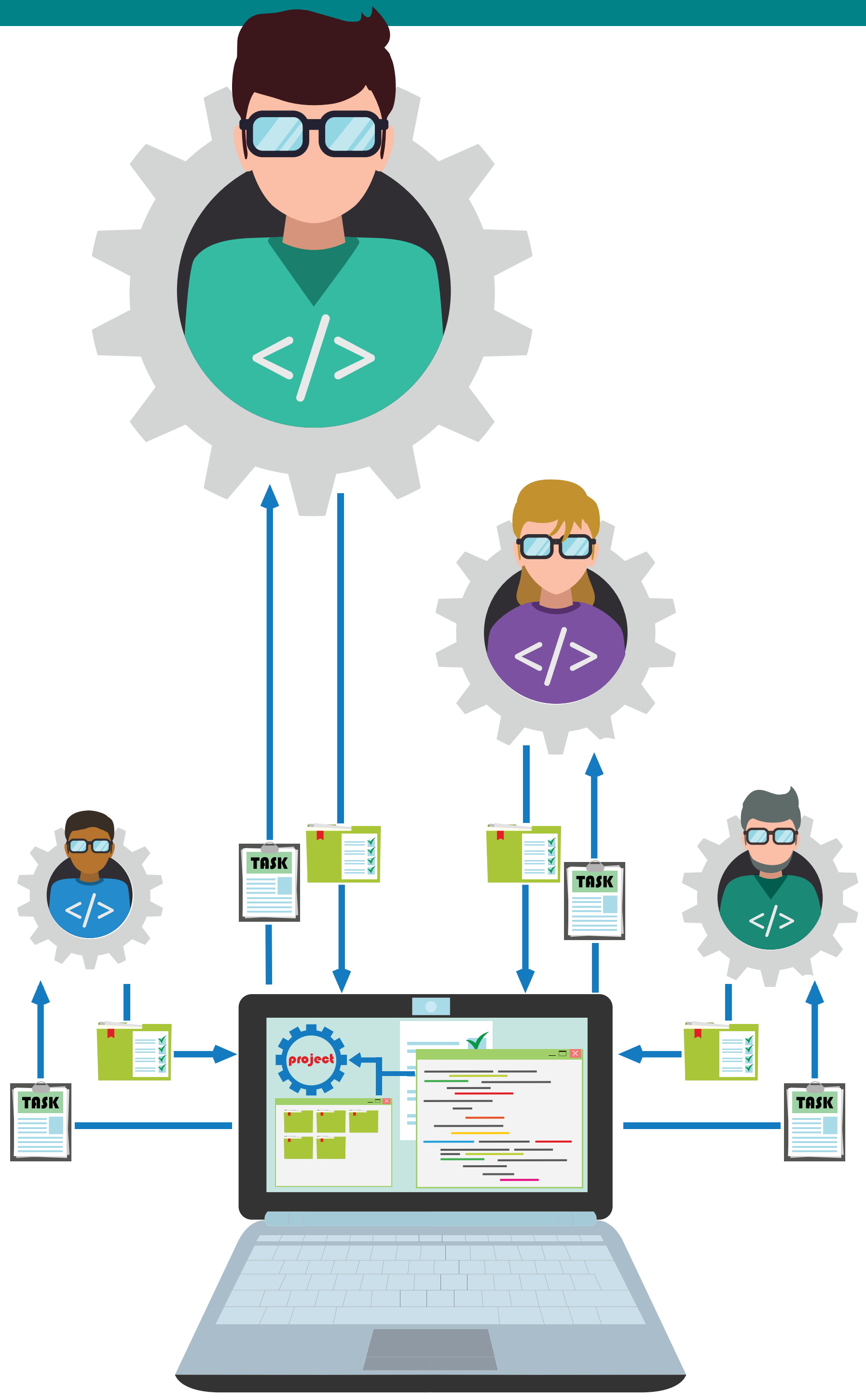
Code reviews instantiate a team's culture in one single experience. Make sure you build up a good culture around code reviews from the start. Review smaller blocks of code. Don't try to do it as one big sitting. For one thing, looking at "everything" in one session means that people may choose "only the important things" to comment about, instead of anything that might affect code quality. (Remember: The goal is always to ship higher quality software.)

Alternatively, reviewing everything-all-at-once becomes an interminable task for the entire team, all of whom wish they could spend the time working on their own code instead (because they have deadlines, too). Plus, a huge stack of suggestions has the effect of giving the reviewee hours of feedback that's hard to process mentally (and sometimes emotionally).

> *"Remember: The goal is always to ship higher quality software"*

Aim for a code review to address about 400 lines of code at a time. It should take no more than 90 minutes for any individual who is participating in the process.

Sometimes, the team does need to look at a lot of code. For instance, imagine that the company is acquiring another, and your team is asked to judge the quality of the other organization's software. In such cases, don't make every team member look at the entire project. Break it up in chunks. Developers might take each 400 lines of code to review, or each team member might devote 90 minutes each day.



7. Consider doing code reviews without in-person meetings.

Another process issue is the expectation that code review happens in a meeting. Very few people really enjoy those meetings, which are inefficient. Among the reasons:

- If ten people are locked in a room together, each has to wait for his turn to talk. Do you really imagine everybody is paying full attention?
- The "meeting" expectations encourage team members to say something, anything, no matter how nitpicky and irrelevant, because that demonstrates they did paid attention, and to justify the time investment.
- Not everyone communicates comfortably or well in meetings. You know how many introverts you have on the development staff. Developers who work remotely are at a disadvantage, too.

- And that's if the team members are prepared for that 9:00am meeting; often, they haven't had the time to look at the code, to mark it up, and to get ready to talk about it.

We encourage you to consider using a tool-based peer review. Using a tool during code reviews allows developers to log in from their desk, add a comment where they want to, and sign off. Developers can get back to work, without having to wait their turn.

8. As manager, you need to set the quality and process standards

As manager, part of your job is to foster open communication. It's your job to create the right expectations about what it means to learn from and to teach other people. You need to know how to give feedback properly, especially during code reviews (as well as other parts of your job), and to show other people how to do so, too. Because code review is all about feedback, and ideally it's also about coaching.

The manager sets the culture of a code review, and you need to reassure them – in word and deed – that a code review is not an attack. Make sure that everyone understands the purpose of the review: to help the team do the best job possible.

Done right, code review creates the team and builds trust; done poorly, it can destroy them. One developer explained, "Code reviews pretty much force collaboration and communication, especially in fast-paced teams where those might be lacking. As soon as we started reviewing each other's pull requests, I began feeling more empowered to voice my opinions about code we were writing, where before there was no platform to do so."

> *"Code reviews pretty much force collaboration and communication, especially in fast-paced teams where those might be lacking."*

Don't expect people to nitpick. Unless a developer has a valid reason to change what someone else did, the guideline should be: Let it go. Don't make people feel they have to say "something" to contribute to the process. Teach developers: Unless you have a valid reason to query the code, such as a developer ignoring predefined conventions, let it go. Don't be that guy.

As with other management tasks, sometimes you have to be the decision-maker, especially when it comes to helping your team reach a consensus, or even embarking on the path of using code reviews. It's time for you to be a leader. A software engineer at a company that makes insurance software said that her management was in favor of code reviews, but "We had to get buy in and commitment from the developers to do peer-pressure enforcement."

9. Recognize team members' emotional responses.

We all want to do good work. Whatever our professions, each of us wants to create stuff that we can be proud of, that works, that delights those who use or consume it. That's one of the nice things about the human race: We care about quality, and we recognize the value of generating the best work possible.

Yet, at the same time, each of us is very protective of whatever it is we create, whether that's a software application, marketing brochure, or modern dance recital. When the creation matters to us, it's easy to become defensive about anyone who "criticizes" the work, even if (in the perception of the speaker) it's in the direction of improvement. Often, when we have invested ourselves in the process, it's easy to see the creation as part of our identities: "If you criticize my code, you're diminishing me."

Developers, like everyone else, want to be respected and trusted. But code review is inherently about helping people to learn what they don't already know, and that can push their emotional buttons of fear ("You're going to tell me how wrong I am!") and pride ("How can you suggest this isn't great, when I'm so sure of its excellence?").

Work in the direction of creating a culture that challenges people without hurting them.

- **Always find something to praise.** It doesn't matter how small the item is. However, by its nature, a code review is looking for imperfections, and people may feel wounded ("I must be a horrible programmer!"). Go out of your way to say something complimentary, even if it's minor. (Worst case scenario: Simply say, "I can see how much effort you put into writing this code" when the result is a bunch of spaghetti code.)
- **Ask, don't judge.** "Why did you make this choice?" not "This is the wrong way." The developer's choice might turn out to be better than your own, after all.
- **Speak in "I" terms, not "you."** Back up each assertion. Instead of saying, "You should never use recursion here," say, "I would do it this way instead, for these reasons."
- **Don't criticize someone for asking a question.** It should always be safe to show what you don't know, because that opens an opportunity to learn.
- **Focus on the solution, not the problem.** If a developer's explanation is long and unwieldy, ask if it's possible to simplify the code. If it isn't, suggest that the explanation be incorporated in the code comments or documentation.
- **Remind people that it's about the work, not the person.** This might be necessary when emotions become a factor despite all the advice above.

10. It makes your company more attractive to top developers.

However the code review process works: Developers like it. In fact, many insist on it. Need evidence? Here's the remarks from several developers who consider code review a vital part of the process, and would not work on a team that did not include it.

"I personally will no longer work for companies that do not implement code review," said one developer.

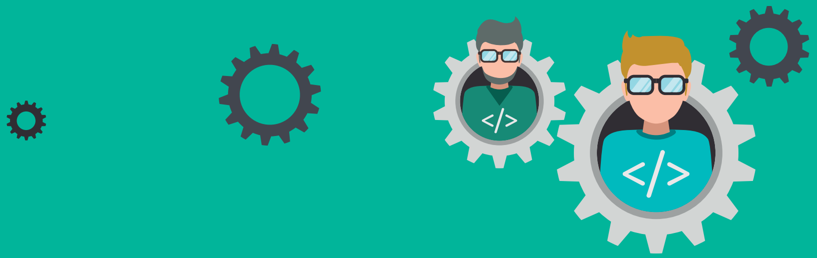
"As far as I am concerned, code review is not a subjective 'up for debate' argument, anymore. It is the correct way to do things, and not doing it is objectively the wrong way to do things. If you don't have time to do it right, you certainly don't have time to do it twice."

"Code without review is cowboy coding and we catch stuff in every review. I'd be terrified to work at a place without review."

"Code reviews don't just produce better code," explained the IoT developer in Colorado. "They increase developer engagement, provide a great platform for communication, and empower every team member to be as impactful as possible on the entire product."

Code review brings teams together, improves software quality, and ensures that the company's brain trust doesn't disappear if a key team member goes away. Why wouldn't you adopt code review in to your development workflow?

If you want the best developers, and you want them to stay, it's a good idea to incorporate code review in your application development process. Now.



With the right tools and best practices, your team can peer review all of your code.

Collaborator

TRY IT FOR **FREE**

SMARTBEAR

Over 3 million software professionals and 25,000 organizations across 194 countries use SmartBear tool

3M+	25K+	194
users	organizations	countries

See Some Succesful Customers >>

API
READINESS

Functional testing through performance monitoring

SEE API READINESS PRODUCTS

TESTING

Functional testing, performance testing and test management

SEE TESTING PRODUCTS

PERFORMANCE
MONITORING

Synthetic monitoring for API, web, mobile, SaaS, and Infrastructure

SEE MONITORING PRODUCTS

CODE
COLLABORATION

Peer code and documentation review

SEE COLLABORATION PRODUCTS

Collaborator

SMARTBEAR