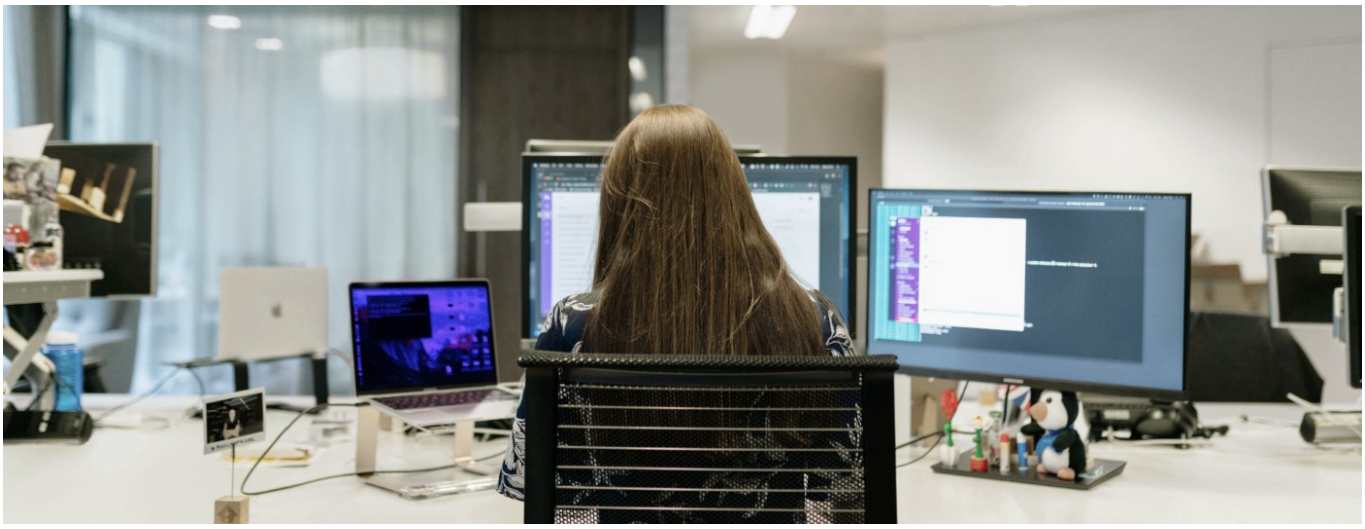
How to Review Code as a Junior Developer

Pinterest Engineering
Jun 8 · 6 min read

Emma Catlin | Software Engineer



“Just ask questions.”

That was the piece of advice that completely changed my perspective on code reviews.

I first started as a Software Engineer at Pinterest on the Shopping Partner Experience team right out of college. I focused on my personal learning by reading documentation and technical design documents to understand my team’s systems and projects, but I lacked the confidence to review my teammates’ code. Why would the senior engineers want to have a junior engineer review their code? What if I said something that was just flat out wrong?

My focus on learning was great at the beginning, but at some point in my first year, I realized something critical: I needed to help the entire team, not just myself, in order to grow to the next engineering level. To start, one of my teammates recommended I review code.

The advice was simple enough — use code reviews as a way to learn more about a piece of code and expand my knowledge of our overall system. It turned out code reviews were the perfect way for me to continue my learning journey.

I've since learned there are multiple benefits to reviewing code as a junior developer:

- Learning the code base
- Building a feedback circle
- Being an owner

Learning the code base

Beyond reading more code and reading other great engineers' code, reviewing code also gives a hint of *who* to ask if you have any questions about a given section of the code. Through code review, you can keep an eye on who knows that portion of the system well and might be a good resource for you down the road. At one point, I was investigating a bug in a part of our system that was new to me. I got stuck and didn't know how to continue investigating it, but because I had watched who on my team was working on this section of the code (and who was solving bugs in this area), I knew who had the knowledge to help guide my investigation. I reached out to that engineer and, with their help, was able to find the root cause of the bug.

Building a feedback circle

When I was a senior in high school, I did an independent study on creative writing, and was nervous about sharing my short stories with the teacher sponsor. Instead of asking to see my work, he first sent me a short story of his own for feedback. That changed everything. Once I provided feedback, I felt completely comfortable sending him my own work. He was vulnerable in his sharing, which cultivated a feedback circle where I felt trusted and supported.

In the exact same way, code review builds confidence around giving and receiving feedback with teammates. Once you give feedback to your teammates through code review, you'll be more open to receiving feedback and hearing their ideas when they push back on a piece of your code.

Being an owner

One of Pinterest's core values is to *be an owner*, and reviewing code helps take ownership over the team's codebase. Participating in code review speeds up the team's

code review process, spreads the weight of code review, and adds your new perspective to improve the codebase.

Are you thinking, “Okay, I get it. Code review sounds great, but how do I do it?”

Here are three ways I developed my ability to code review as a junior developer:

- Ask questions
- Calibrate feedback
- Emulate others

Ask questions

Asking questions in code review is some of the best advice I received as a junior developer. Once I dug into what I didn’t know and asked questions on my coworkers’ pull requests, I learned so much about our systems. Now, I feel comfortable reviewing code almost anywhere. Asking questions during code reviews is a great strategy not only for junior engineers, but also for new team members ramping onto the codebase. For example, you can ask what a specific piece of the system does, why they added that piece of logic in that file instead of further downstream, or what they meant by a particular comment. If something is not clear to you, it probably isn’t clear to everyone. So asking questions can help your teammates understand what knowledge gaps might exist for someone newer to the codebase. That way, your teammates can write clear variable names, modify their function structure, or add some code comments to help someone with little to no context jump into the code right away.

Calibrate feedback

Eventually, you will have specific things (like code style, scalability, or speed) that mean a lot to you when reviewing parts of the code, especially once someone is developing in an area that you own. But when you are new, it’s good to first identify what your team members care about to build up your relationships and get comfortable reviewing code they own. Does your team have code guidelines that you can refer back

to when reviewing other people's code? Does a particular engineer love inline functions and would greatly appreciate your idea to write a piece of their code inline?

On my team, we care about what steps you have taken to test your code, especially end-to-end testing. I often find myself asking questions about how someone tested their code in code review. Not only does it ensure that they tried their code before it lands, but it also teaches me new ways to thoroughly test my changes.

Catering feedback to your team builds positive relationships. If your team doesn't care about typos or code comments, don't tell them to fix their typo or add a code comment — unless that typo is user-facing and thus critical! It will make them less likely to be receptive to your suggestions (on their code, their technical design document, or on other aspects of your working relationship) down the line. People are receptive to feedback they care about, so work to cater recommendations to the individual. Once you have earned your team's trust and respect, by all means, give feedback on what matters to you.

Emulate others

Another way to get more comfortable reviewing code is to identify someone on your team who reviews code well and watch what they do. I look to the senior engineer on my team and observe things such as if they always look for unit tests, code comments, naming conventions, etc. If they ask for a significant change, how do they ask for it? If you see "synced offline with @so-and-so" and then a large refactor, it shows you that for more significant critiques, your teammates might prefer to message about it or talk about it one-on-one. Many feedback frameworks beyond code review suggest giving praise in front of everyone and constructive feedback in a one-on-one setting, so that might be what your team prefers.

By observing how your coworkers review code, you can learn what to pay attention to when you're writing code. If you see someone repeatedly asking for more unit tests, you can do better in your code the first time around with your test coverage before submitting your code for review. Or, if you are worried you missed some way to best test your code, draw that person into your feedback circle by preemptively asking them for recommendations. Lean on your teammates' strengths.

I hope these suggestions help you in your development journey as an engineer and give you the confidence to review code. Once I got comfortable reviewing code, I was able to read and write code all over our repositories. Ultimately, it helped me grow out of my

junior engineering role and contribute to projects that spanned multiple teams. If you are new to code review, try asking a question about someone's code tomorrow. Even if it is just in an "offline" message and not in their code review, it can start a dialogue and help you get to know your teammates and your codebase better!

Acknowledgements: Thanks to my whole team at Pinterest for their support and encouragement. Special thanks to Brian for all his advice and for setting a great example for junior coders.

To learn more about engineering at Pinterest, check out the rest of our Engineering Blog, and visit our Pinterest Labs site. To view and apply to open opportunities, visit our Careers page.