Code Reviews: Building Better Code and Stronger Teams

Sandor Dargo

Meeting C++

6th November 2025, Berlin (Germany)

TWITTER @SandorDargo WEB sandordargo.com

Who Am I?

Sándor DARGÓ

- Senior Engineer at **Spotify**
- Enthusiastic blogger: sandordargo.com
- Curious oenophile
- Fortunate father of two

Have you ever received a code review comment that hurt you?
Frustrated you?
Left you totally confused?

Why This Talk?

Engineering is as much about people as it is about code

But we mostly talk about about code or processes

Code reviews often cause stress and conflict

Done right, they amplify learning, trust, and quality

What Are We Not Covering?

The formatting / style

The technical parts

The “business” process

What Are We Covering?

The Reasons of Code Reviews

The Emotional Parts

Some Language-Agnostic Elements

The Consequences of Good (or Bad) Code Reviews

Agenda

What code reviews are

Different ways to review code

Arguments against dedicated code reviews

Why you should have code reviews

Common pitfalls of code reviews

The AIR Formula for writing better code review comments

What are code reviews?

A Tool for Quality Assurance

Catch inconsistencies and maintain standards

Enforce style and architectural patterns

Support readability and maintainability

An opportunity to catch flaws

A Way of Knowledge Sharing

Educate about the used language

Share unfamiliar APIs or internal tools

Explain decisions and context

Avoid “*only Sandor knows this code*” syndrome

Sometimes Even a Way of Mentoring

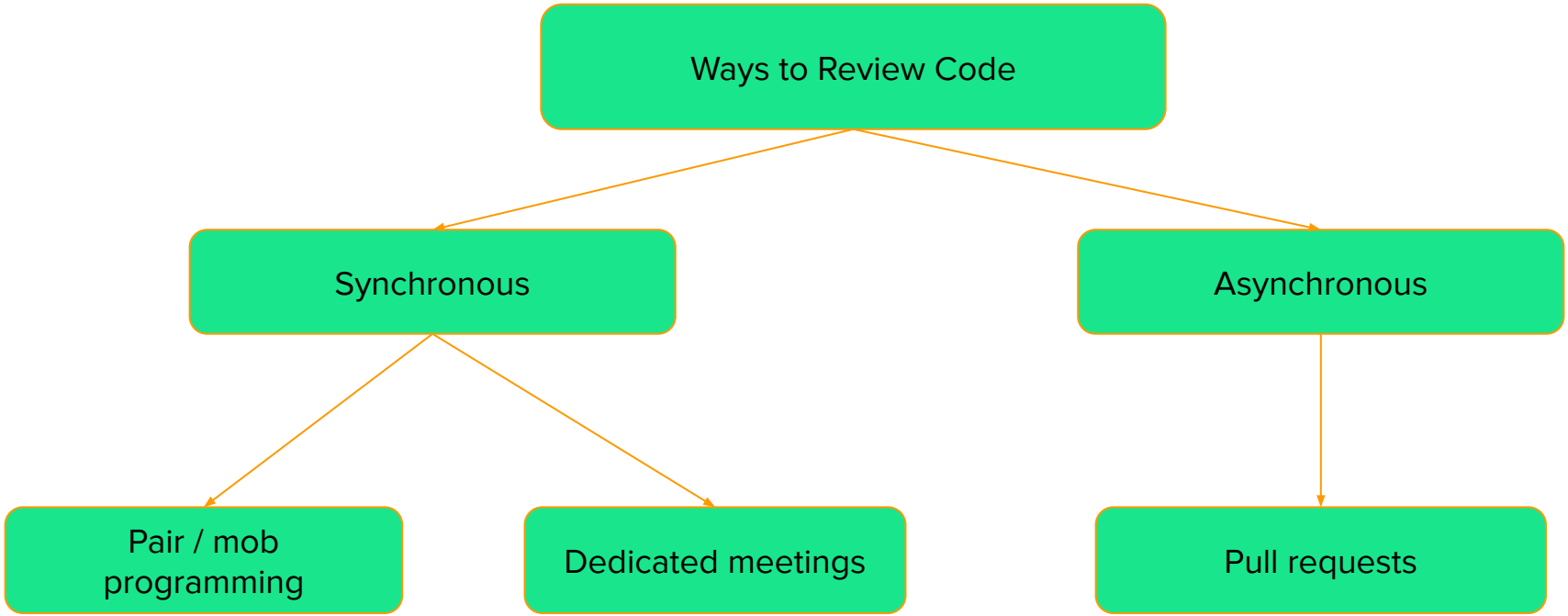
You might even comment your own code

Elevate ~~junior~~ less experienced devs by explaining *“why”* and *“how”*

Use reviews to grow confidence and skills

A great place to give feedback with empathy

Different ways to review code


Ways to Review Code
Synchronous
Asynchronous
Pair / mob programming
Dedicated meetings
Pull requests

Pair programming

Real time collaboration

Constant and immediate feedback loop

Great for onboarding and complex problems

The review is a byproduct of the coding process

Mob Programming

One person types, others guide

Builds shared understanding

Best for

- Exploratory work
- Architecture decisions
- Bringing the team to the same level

Dedicated code review meeting

Typically pre-scheduled

Can include multiple stakeholders

Best for critical code or architecture decisions

The async solution: pull requests

Most common style nowadays

It’s purely written which can lead to misunderstandings

Not real-time at all

Enables flexibility, but can slow things down

Quick Recap: Forms of Code Review

	Pros	**Cons**
Pair/mob	Instant feedback, shared knowledge	Can be time-intensive, not scalable
Meetings	Good for alignment, stakeholder input	High coordination cost
PRs	Scalable, flexible	Delayed feedback, tone misunderstandings

Arguments against dedicated code reviews

“Replace Code Reviews by Pair Programming!”

Pairing catches issues early and improves shared understanding

But lacks reflection time and broader input

Reviews offer a different kind of approach

- More deliberate
- And less biased by shared context

“Code Reviews Slow Us Down”

Increases the *“raise-to-merge”* latency

The latency can lead to more often merge conflicts

But good reviews prevent costly rework and fallbacks

Async reviews scale better than other review practices

Especially if there is a culture of review-first policy

“They Don’t Catch Bugs”

True: reviews are not a substitute for testing

They’re more effective at catching design flaws, naming, complexity, unclear logic

Review feedback is more about understandability than correctness

But still can catch bugs too

Tradeoffs Are Real

Yes, reviews take time

Yes, they’re imperfect

But when done well, the payoff is

- Better code
- Stronger team
- And shared ownership

Why you should have code reviews

It’s The Last Line of Defence

Essential for the long-term health of your codebase

The final human check after all the automation

AI helps – and will help more – but it’s not enough

Human insight catches what machines cannot (yet)

Offer a Fresh Perspective

Author bias is real – we miss our own mistakes

“What’s obvious to you isn’t obvious to others”

Reviewers can question assumptions or point out edge cases

Leverage Diverse Insights

Different specialties notice different things

Different levels of experience focus on different angles

Code quality improves when multiple viewpoints are considered

Educate Team Members

Pull requests let senior devs coach through code reviews

Juniors learn idioms, architecture, internal APIs

Great place to share reasoning and alternatives

Stronger Personal and Team Relationships

Respectful feedback builds psychological safety

Teams where everyone reviews communicate better

Helps dissolve silos and cliques

Code reviews don't just improve code

They improve coders

Common Pitfalls in Code Reviews

What are code reviews?

Different ways to review code

Arguments against dedicated code reviews

Why you should have code reviews

Common Pitfalls in Code Reviews

The AIR Formula for Better Code Review Comments

Feedback Arrives Too Late

Reviews should happen while the author still remembers the details

Reviews should not block merging for too long

Late reviews lead to frustration and resistance

Can result in ignored or rushed changes


Many devs in a
code review

Focusing on the nits

People often focus on the details – it’s easier

The big picture is often missed

Making the details right is important

But the bigger picture is even more important

Watch Your Tone

Written feedback lacks tone of voice and body language

Can sound (or be) aggressive or passive-aggressive

Might erode trust and discourage open discussion

Reviewing the reviewer

Avoid language that feels personal or judgmental

It’s not about “who” but about “what”

Don’t blame, help

We are all learning

Kindness scales better than harsh criticism

Bossy or Commanding Language

Avoid phrasing feedback as commands

Invite collaboration, not compliance

Even senior devs should stay humble

Comments Lack Priority

Make the intent of your comment understandable

- Is it a blocker?
- A maybe?
- A nit?
- Just a question?
- Or even a kudos!

Make it clear what you expect as a reviewer

Comments Lack Explanation or Educational Value

Vague or unexplained comments frustrate authors

They miss the chance to teach or share reasoning

Without context, authors may blindly comply – or push back

When Everything Gets Commented

Receiving feedback on every single line can feel overwhelming and discouraging

It creates the impression of rigid control (*there's only one "right" way*)

Developers need some autonomy

Not every decision has to be perfectly optimal

Poorly Prepared Pull Requests

Not green PR shared

Self-review not done

Too big PRs

Mix of unrelated changes

No description and entry points shared

Quick Recap: Common pitfalls

❌ **Pitfall**	💥 **Impact**
Feedback arrives too late	Context is lost, causes frustration or extra work
Focusing on the nits	The bigger picture is not examined
Harsh, judgmental or personal tone	Erodes trust, (feels) personal rather than helpful
Comments sound bossy or commanding	Feels like micromanagement, discourages discussion
Comments have unclear priority	Leaves author unsure what needs action
Comments lack explanation	No learning, feels arbitrary
Everything is commented	Overwhelming, disempowering – kills autonomy
Poorly Prepared Pull Requests	Too much work for reviewers, they lose focus

The AIR Formula for Better Code Review Comments

A = Action: What Should Be Done?

Phrase feedback as a suggestion, not a command

Use softening language: "consider...", "perhaps...", "could we..."

Encourage discussion, not compliance

I = Information: Why It Matters

Explain your reasoning clearly

Helps the author understand intent

Builds shared knowledge

R = Reference: Where to Learn More

Link to style guides, docs, or relevant discussions

Helps justify your feedback without debating in the PR

Encourages self-directed learning

Putting It All Together

❌ Typical bad comment:

“Rename this.”

What’s the problem:

- No context
- No reasoning
- No learning opportunity
- Sounds a bit too direct

✅ Improved Comment (With AIR):

*“Consider renaming this variable to make it clearer that it represents a configuration object **(Action)**. It confused me at first because I assumed it held the results **(Information)**. Our naming convention suggests clarity over brevity—see ”Choosing names” section of the style guide **(Reference)**.”*

Another example

❌ Typical bad comment:

“Don’t use magic numbers.”

✅ Improved Comment (With AIR):

*“Consider replacing 42 with a named constant **(Action)**. It’s not clear what this value represents, so maintaining or changing it later might be risky **(Information)**. We recommend symbolic constants for readability—see C++ Core Guidelines ES.45 **(Reference)**.”*

Yet another one

❌ Typical bad comment:

“Why did you do it like this?”

✅ Improved Comment (With AIR):

*“Could you explain why this approach was chosen? **(Action)** I’m wondering if it’s for readability or if there’s a constraint I’m missing **(Information)**. We try to document non-obvious design choices in code comments—see Implementation Comments **(Reference)**.”*

Quick Recap: The AIR Formula

Component	Purpose	Example
Action	What to do	*“Please consider renaming”*
Information	Why it matters	*“To clarify its meaning”*
Reference	Learn more or justify	*“See our team style guide…”*

Should you always use this formula?

Maybe – in a perfect world

But the world is not perfect

- We are tired
- Overwhelmed
- Often stressed

Plus a typo is just a typo

Use it when there is a teaching opportunity

Time to conclude!

Code Reviews: More Than Just a Process

Code reviews are a tool and investment for quality, learning, and mentorship

Main challenges are tone, timing and clarity

Action, **I**nformation, **R**eference makes comments educative and actionable

With better reviews, have stronger teams, deeper learning, lasting impact

Call to Action

✅ Encourage self reviews to catch the obvious

✅ Pick one thing to improve about how you give feedback

✅ Try using the AIR formula in your next review

✅ Talk with your team about your review culture

✅ Start seeing reviews as mentorship opportunities

Code Reviews: Building Better Code and Stronger Teams

Sandor Dargo

Meeting C++

6th November 2025, Berlin (Germany)

TWITTER @SandorDargo WEB sandordargo.com