Everything here is my opinion. I do not speak for your employer.

← *November 2025*

2026-03-16 »

Every layer of review makes you 10x slower

We've all heard of those network effect laws: the value of a network goes up with the square of the number of members. Or the cost of communication goes up with the square of the number of members, or maybe it was n log n, or something like that, depending how you arrange the members. Anyway doubling a team doesn't double its speed; there's coordination overhead. Exactly how much overhead depends on how badly you botch the org design.

But there's one rule of thumb that someone showed me decades ago, that has stuck with me ever since, because of how annoyingly true it is. The rule is annoying because it doesn't *seem* like it should be true. There's no theoretical basis for this claim that I've ever heard. And yet, every time I look for it, there it is.

Here we go:

Every layer of approval makes a process 10x slower

I know what you're thinking. Come on, 10x? That's a lot. It's unfathomable. Surely we're exaggerating.

Nope.

Just to be clear, we're counting "wall clock time" here rather than effort. Almost all the extra time is spent sitting and waiting.

Look:

- Code a simple bug fix
  30 minutes

- Get it code reviewed by the peer next to you
  300 minutes → 5 hours → half a day

- Get a design doc approved by your architects team first
  50 hours → about a week

- Get it on some other team's calendar to do all that
  (for example, if a customer requests a feature)
  500 hours → 12 weeks → one fiscal quarter

I wish I could tell you that the next step up — 10 quarters or about 2.5 years — was too crazy to contemplate, but no. That's the life of an executive sitting above a medium-sized team; I bump into it all the time even at a relatively small company like Tailscale if I want to change product direction. (And execs sitting above large teams can't actually do work of their own at all. That's another story.)

AI can't fix this

First of all, this isn't a post about AI, because AI's direct impact on this problem is minimal. Okay, so Claude can code it in 3 minutes instead of 30? That's super, Claude, great work.

Now you either get to spend 27 minutes reviewing the code yourself in a back-and-forth loop with the AI (this is actually kinda fun); or you save 27 minutes and submit unverified code to the code reviewer, who will still take 5 hours like before, but who will now be mad that you're making them read the slop that you were too lazy to read yourself. Little of value was gained.

Now now, you say, that's not the value of agentic coding. You don't use an agent on a 30-minute fix. You use it on a monstrosity week-long project that you and Claude can now do in a couple of hours! Now we're talking. Except no, because the monstrosity is so big that your reviewer will be extra mad that you didn't read it yourself, and it's too big to review in one chunk so you have to slice it into new bite-sized chunks, each with a 5-hour review cycle. And there's no design doc so there's no intentional architecture, so eventually someone's going to push back on that and here we go with the design doc review meeting, and now your monstrosity week-long project that you did in two hours is... oh. A week, again.

I guess I could have called this post Systems Design 4 (or 5, or whatever I'm up to now, who knows, I'm writing this on a plane with no wifi) because yeah, you guessed it. It's Systems Design time again.

The only way to sustainably go faster is fewer reviews

It's funny, everyone has been predicting the Singularity for decades now. The premise is we build systems that are so smart that they themselves can build the next system that is even smarter, that builds the next smarter one, and so on, and once we get that started, if they keep getting smarter faster enough, then the incremental time (t) to achieve a unit (u) of improvement goes to zero, so (u/t) goes to infinity and foom.

Anyway, I have never believed in this theory for the simple reason we outlined above: the majority of time needed to get anything done is not actually the time doing it. It's wall clock time. Waiting. Latency.

And you can't overcome latency with brute force.

I know you want to. I know many of you now work at companies where the business model kinda depends on doing exactly that.

Sorry.

But you can't just *not review* things!

Ah, well, no, actually yeah. You really can't.

There are now many people who have seen the symptom: the start of the pipeline (AI generated code) is so much faster, but all the subsequent stages (reviews) are too slow! And so they intuit the obvious solution: stop reviewing then!

The result might be slop, but if the slop is 100x cheaper, then it only needs to deliver 1% of the value per unit and it's still a fair trade. And if your value per unit is even a mere 2% of what it used to be, you've doubled your returns! Amazing.

There are some pretty dumb assumptions underlying that theory; you can imagine them for yourself. Suffice it to say that this produces what I will call the AI Developer's Descent Into Madness:

1. Whoa, I produced this prototype so fast! I have super powers!

2. This prototype is getting buggy. I'll tell the AI to fix the bugs.
3. Hmm, every change now causes as many new bugs as it fixes.
4. Aha! But if I have an AI agent also review the code, it can find its own bugs!
5. Wait, why am I personally passing data back and forth between agents
6. I need an agent framework
7. I can have my agent write an agent framework!
8. Return to step 1

It's actually alarming how many friends and respected peers I've lost to this cycle already. Claude Code only got good maybe a few months ago, so this only recenlty started happening, so I assume they will emerge from the spiral eventually. I mean, I hope they will. We have no way of knowing.

Why we review

Anyway we know our symptom: the pipeline gets jammed up because of too much new code spewed into it at step 1. But what's the root cause of the clog? Why doesn't the pipeline go faster?

I said above that this isn't an article about AI. Clearly I'm failing at that so far, but let's bring it back to humans. It goes back to the annoyingly true observation I started with: every layer of review is 10x slower. As a society, we know this. Maybe you haven't seen it before now. But trust me: people who do org design for a living know that layers are expensive... and they still do it.

As companies grow, they all end up with more and more layers of collaboration, review, and management. Why? Because otherwise mistakes get made, and mistakes are increasingly expensive at scale. The average value added by a new feature eventually becomes lower than the average value lost through the new bugs it causes. So, lacking a way to make features produce more value (wouldn't that be nice!), we try to at least reduce the damage.

The more checks and controls we put in place, the slower we go, but the more monotonically the quality increases. And isn't that the basis of *continuous improvement?*

Well, sort of. Monotonically increasing quality is on the right track. But “more checks and controls” went off the rails. That’s *only one way* to improve quality, and it's a fraught one.

“Quality Assurance” reduces quality

I wrote a few years ago about W. E. Deming and the "new" philosophy around quality that he popularized in Japanese auto manufacturing. (Eventually U.S. auto manufacturers more or less got the idea. So far the software industry hasn’t.)

One of the effects he highlighted was the problem of a “QA” pass in a factory: build widgets, have an inspection/QA phase, reject widgets that fail QA. Of course, your inspectors probably miss some of the failures, so when in doubt, add a second QA phase after the first to catch the remaining ones, and so on.

In a simplistic mathematical model this seems to make sense. (For example, if every QA pass catches 90% of defects, then after two QA passes you’ve reduced the number of defects by 100x. How awesome is that?)

But in the reality of agentic humans, it’s not so simple. First of all, the incentives get weird. The second QA team basically serves to evaluate how well the first QA team is doing; if the first QA team keeps missing defects, fire them. Now, that second QA team has little incentive to produce that outcome for their friends. So maybe they don’t look too hard; after all, the first QA team missed the defect, it’s not unreasonable that we might miss it too.

Furthermore, the first QA team knows there is a second QA team to catch any defects; if I don’t work too hard today, surely the second team will pick up the slack. That's why they're there!

Also, the team making the widgets in the first place doesn’t check their work too carefully; that’s what the QA team is for! Why would I slow down the production of every widget by being careful, at a cost of say 20% more time, when there are only 10 defects in 100 and I can just eliminate them at the next step for only a 10% waste overhead? It only makes sense. Plus they'll fire me if I go 20% slower.

To say nothing of a whole engineering redesign to improve quality, that would be super expensive and we could be designing all new widgets instead.

Sound like any engineering departments you know?

Well, this isn't the right time to rehash Deming, but suffice it to say, he was on to something. And his techniques worked. You get things like the famous Toyota Production System where they eliminated the QA phase entirely, but gave everybody an "oh crap, stop the line, I found a defect!" button.

Famously, US auto manufacturers tried to adopt the same system by installing the same "stop the line" buttons. Of course, nobody pushed those buttons. They were afraid of getting fired.

Trust

The basis of the Japanese system that worked, and the missing part of the American system that didn't, is trust. Trust among individuals that your boss Really Truly Actually wants to know about every defect, and wants you to stop the line when you find one. Trust among managers that executives were serious about quality. Trust among executives that individuals, given a system that can work and has the right incentives, will produce quality work and spot their own defects, and push the stop button when they need to push it.

But, one more thing: trust that the system *actually does work*. So first you need a system that will work.

Fallibility

AI coders are fallible; they write bad code, often. In this way, they are just like human programmers.

Deming's approach to manufacturing didn't have any magic bullets. Alas, you can't just follow his ten-step process and immediately get higher quality engineering. The secret is, you have to get your engineers to engineer higher quality into the whole system, from top to bottom, repeatedly. Continuously.

Every time something goes wrong, you have to ask, "How did this happen?" and then do a whole post-mortem and the Five Whys (or however many Whys are in fashion nowadays) and fix the underlying Root Causes so that it doesn't happen again. "The coder did it wrong" is never a root cause, only a symptom. Why was it possible for the coder to get it wrong?

The job of a code reviewer isn't to review code. It's to figure out how to obsolete their code review comment, that whole class of comment, in all future cases, until you don't need their reviews at all anymore.

(Think of the people who first created "go fmt" and how many stupid code review comments about whitespace are gone forever. Now that's engineering.)

By the time your review catches a mistake, the mistake has already been made. The root cause happened already. You're too late.

Modularity

I wish I could tell you I had all the answers. Actually I don’t have much. If I did, I’d be first in line for the Singularity because it sounds kind of awesome.

I think we’re going to be stuck with these systems pipeline problems for a long time. Review pipelines — layers of QA — don’t work. Instead, they make you slower while hiding root causes. Hiding causes makes them harder to fix.

But, the call of AI coding is strong. That first, fast step in the pipeline is *so fast!* It really does feel like having super powers. I want more super powers. What are we going to do about it?

Maybe we finally have a compelling enough excuse to fix the 20 years of problems hidden by code review culture, and replace it with a real culture of quality.

I think the optimists have half of the right idea. Reducing review stages, even to an uncomfortable degree, is going to be needed. But you can’t just reduce review stages without something to replace them. That way lies the Ford Pinto or any recent Boeing aircraft.

The complete package, the table flip, was what Deming brought to manufacturing. You can’t half-adopt a “total quality” system. You need to eliminate the reviews *and* obsolete them, in one step.

How? You can fully adopt the new system, in small bites. What if some components of your system can be built the new way? Imagine an old-school U.S. auto manufacturer buying parts from Japanese suppliers; wow, these parts are so well made! Now I can start removing QA steps elsewhere

because I can just assume the parts are going to work, and my job of "assemble a bigger widget from the parts" has a ton of its complexity removed.

I like this view. I've always liked small beautiful things, that's my own bias. But, you can assemble big beautiful things from small beautiful things.

It's a lot easier to build those individual beautiful things in small teams that trust each other, that know what quality looks like *to them*. They deliver their things to customer teams who can clearly explain what quality looks like *to them*. And on we go. Quality starts bottom-up, and spreads.

I think small startups are going to do really well in this new world, probably better than ever. Startups already have fewer layers of review just because they have fewer people. Some startups will figure out how to produce high quality components quickly; others won't and will fail. Quality by natural selection?

Bigger companies are gonna have a harder time, because their slow review systems are baked in, and deleting them would cause complete chaos.

But, it's not just about company size. I think engineering teams at any company can get smaller, and have better defined interfaces between them.

Maybe you could have multiple teams inside a company competing to deliver the same component. Each one is just a few people and a few coding bots. Try it 100 ways and see who comes up with the best one. Again, quality by evolution. Code is cheap but good ideas are not. But now you can try out new ideas faster than ever.

Maybe we'll see a new optimal point on the monoliths-microservices continuum. Microservices got a bad name because they were too micro; in the original terminology, a "micro" service was exactly the right size for a "two pizza team" to build and operate on their own. With AI, maybe it's one pizza and some tokens.

What's fun is you can also use this new, faster coding to experiment with different module *boundaries* faster. *Features* are still hard for lots of reasons, but refactoring and automated integration testing are things the AIs excel at. Try splitting out a module you were afraid to split out before. Maybe it'll add some lines of code. But suddenly lines of code are cheap,

compared to the coordination overhead of a bigger team maintaining both parts.

Every team has some monoliths that are a little too big, and too many layers of reviews. Maybe we won't get all the way to Singularity. But, we can engineer a much better world. Our problems are solvable.

It just takes trust.

Related

Highlights on "quality," and Deming's work as it applies to software development (2016)

Systems design 3: LLMs and the semantic revolution (2025)

Unrelated

SimSWE part 2: The perils of multitasking (2017)

I'm CEO at **Tailscale**, where we make network problems disappear.

Why would you follow me on twitter? Use RSS.

apenwarr on gmail.com