Linus's Law

Idea In Short

Assign five reviewers to a deliverable and expect five different objections; assign fifty and the marginal catch rate collapses within the first dozen. Linus's Law, coined by Eric Raymond after Linux creator Linus Torvalds, holds that with enough eyeballs, every defect becomes shallow. Leaders often stop there and mistake headcount for thoroughness, stacking sign-off chains that slow decisions without catching more errors. The evidence points elsewhere: what matters is reviewer diversity and independence, not reviewer volume. Teams that get real value from peer review pick a handful of people with different vantage points, give them real authority to flag problems and stop adding reviewers once returns diminish. Executives who want fewer client-facing errors and fewer costly recalls should redesign their review boards around that principle, not around adding more names to a distribution list.

Software engineers have a long-running argument about who should look at code before it ships and the answer they settled on four decades ago still shapes how consulting firms, audit teams and product organizations catch mistakes today. The label for that answer is Linus's Law, a principle born in the world of open source software but built on a claim any business leader can test against their own review processes. It says that defects which look impossible to spot from one angle become obvious once enough of the right people examine them from different angles. The idea sounds like a case for adding more reviewers to every process and that is exactly where most organizations misapply it. The actual mechanism behind the law has less to do with headcount and more to do with the variety of perspective a review group brings, a distinction that determines whether a peer review process finds real problems or just adds delay.

Where the law came from

Eric Raymond, a software developer and essayist, formulated the principle in his 1999 work "The Cathedral and the Bazaar", an essay comparing two models of software development.

The cathedral model relied on a small, closed team working in isolation before releasing a finished product. The bazaar model, which Raymond observed in Linus Torvalds' development of the Linux kernel, opened the code to a large, loosely coordinated group of volunteer contributors who tested, patched and argued over changes in public. Raymond summarized what he saw working in the bazaar with a line that outlived the essay itself: given enough eyeballs, all bugs are shallow. He named the observation after Torvalds because the Linux project demonstrated it at a scale nobody had tried before, turning thousands of scattered contributors into an informal but effective quality control system.

1 The Linux kernel's growth over the following decades gave the claim staying power, but it also invited decades of testing by researchers who wanted to know whether the mechanism generalized beyond one famous project. That testing produced a more precise and more useful, version of the law than the original slogan suggests.

Why more reviewers change what gets seen

The statistical logic behind Linus's Law is straightforward once stated plainly. Every reviewer carries a different mental model of what the work should look like, built from their own training, experience and blind spots. A defect invisible to one reviewer, because it sits outside their frame of reference, often sits squarely inside another reviewer's area of attention. Add reviewers with genuinely different backgrounds and the union of what they collectively notice grows faster than any individual reviewer's coverage would suggest. This is why a single, exhaustive review by one expert reviewer still misses categories of error that a shorter review by three differently trained people would catch.

The catch is that this logic depends entirely on the reviewers being different from each other in relevant ways, not just numerous. Ten reviewers with the same training, working from the same checklist, behave statistically like one reviewer repeated ten times. Their overlapping blind spots stay blind no matter how many of them look.

Beyond defect-hunting

A large study of code review practice inside Microsoft found something that complicates the simple defect-counting version of the law. Researchers who interviewed developers and analyzed review comments discovered that formal reviews caught fewer outright bugs than participants expected going in.

> Reviews are less about defects than expected

The same study found that reviewers who did not catch a defect still delivered value through knowledge transfer, raised awareness of how a change fit into the wider codebase and alternative approaches that improved the work even when no bug was present.2 That finding matters for any organization designing a review process, because it means the return on adding a reviewer is not just measured in defects caught. It also shows up in the shared understanding a review builds, which reduces the odds of a related defect appearing later.

The business case for more eyeballs

Consulting engagements, audits and product launches all produce deliverables where an undetected error can cost far more to fix after release than before it. IBM's research on software quality points to the CrowdStrike update that grounded Delta Air Lines flights in 2024, an incident tied to over 500 million dollars in losses, as an example of what an inadequately reviewed change can trigger once it reaches production.3 The same arithmetic applies to a mispriced client proposal, a contract clause that survives one round of legal review but not two, or a financial model that balances on an assumption nobody challenged. Catching these problems earlier, while they are still cheap to fix, is the entire economic case for building review into the workflow rather than treating it as a final gate.

Google's engineering organization built its review culture around this logic long before most consulting firms formalized theirs. A study of code review at Google, based on interviews, a survey of 44 engineers and an analysis of nine million reviewed changes, found that the company had refined lightweight, tool-supported review into a standard practice woven through daily work rather than an occasional audit.4 The scale of that dataset offers something rare:

> evidence that a review discipline can operate continuously, across an enormous volume of work, without collapsing into either rubber-stamping or gridlock, provided the organization designs the process deliberately rather than layering approvals on top of each other

Applying it to consulting deliverables

A consulting firm that wants the same effect needs to identify which risk category each reviewer actually owns rather than routing every deliverable through the same fixed chain of sign-offs. A pricing model benefits from a reviewer trained to stress-test assumptions and check the arithmetic. A client-facing narrative benefits from a reviewer who can judge tone, clarity and whether the argument will land with a specific audience. A contractual clause benefits from someone with legal training who can spot exposure that a strategist would miss entirely. Structuring review this way turns a generic approval chain into a set of targeted checks, each one matched to a specific failure mode. Firms that skip this step and simply add more partners to the sign-off list often find that the same category of error keeps slipping through, because nobody with the relevant expertise was ever actually asked to look for it.

Where the law breaks down

Jakob Nielsen's research on usability testing offers a useful counterweight to the idea that more reviewers is always better. His analysis found that a single test participant reveals roughly 31% of usability problems in a product and that value from additional participants drops sharply after that, with a fifth participant mostly repeating what earlier ones already found.

> You are wasting your time by observing the same findings repeatedly

Nielsen's recommendation, testing in small batches of five and fixing problems between rounds rather than running one large study, reflects a pattern that holds well beyond usability testing.5 Whatever the domain, a review group built from a handful of well-chosen participants tends to outperform a much larger group assembled without much thought to what each person adds.

The coordination tax

Adding reviewers past that point does not just fail to help; it actively costs time and attention that the organization could spend elsewhere. Research on collaboration inside large companies found that the total time employees spend in meetings and responding to collaborative requests has grown by 50% or more over two decades and that only 3 to 5% of employees generate the bulk of the value in that collaborative work.6 A review chain that keeps growing follows the same pattern: each additional sign-off adds scheduling friction and spreads accountability thinner, until no single reviewer feels fully responsible for catching a problem. That diffusion of ownership is the opposite of what Linus's Law depends on, since the law works only when each reviewer genuinely engages with the material rather than assuming someone else already checked it.

Building a review system that works

Getting real value from Linus's Law starts with treating review design as a deliberate exercise rather than a habit inherited from whoever ran the process last year. Leaders should list the specific risks a deliverable carries, then assign a reviewer to each risk based on expertise, not availability. They should set a hard limit on review group size and resist the instinct to add another name whenever something goes wrong, since a bigger group rarely fixes a problem caused by the wrong reviewers being in the room. They should also build in a short, structured window for review rather than an open-ended one, because deadlines force reviewers to focus on what matters instead of skimming everything superficially. Finally, they should track which reviewer actually caught which class of problem over time, because that record reveals whether the review group still matches the risks the organization faces or needs to be rebuilt around new expertise.

None of this requires the scale of a project like Linux or the review infrastructure of a company like Google. It requires the same underlying discipline both examples share:

> matching genuinely different perspectives to a piece of work, giving each reviewer real ownership of a specific risk and stopping once the marginal reviewer stops adding anything new

- 1Linus's Law
- 2Expectations, Outcomes And Challenges Of Modern Code Review
- 3Software Testing

- 4Modern Code Review: A Case Study At Google
- 5Why You Only Need To Test With 5 Users
- 6Collaborative Overload

Summary

Linus's Law survives forty years of scrutiny because it describes something real: independent, differently informed reviewers catch different mistakes and a defect invisible to one reviewer is often obvious to another. What the law does not say and what business leaders keep adding to it, is that more reviewers automatically means more coverage. The research on code review, usability testing and collaboration all points the same direction: value peaks with a handful of well-chosen reviewers and declines once the group grows large enough to create coordination overhead and diffused accountability. Firms that treat review as a checkbox exercise, stacking approvals without changing who looks or how, get the cost of scrutiny without the benefit. Firms that treat it as a design problem, matching reviewer expertise to the specific risk in a deliverable, catch more with less effort and move faster because nobody is waiting on a redundant sign-off.