

Code Review, Done Formally: The Fagan Inspection

Mon, Aug 17, 4:00 PM EDT · <https://scottslab.io/posts/code-review-fagan-inspection>



TL;DR

Fagan inspection is the most formal code review there is: six defined steps, five named roles, entry and exit criteria, and a hard ceiling on how fast you're allowed to read. Fagan's own 1976 IBM data found 38 defects per thousand lines by inspection against 8 by unit test, and 82% total detection efficiency. Almost nobody runs the full ceremony. Every shortcut your team takes maps to a specific step, and the rate limit is the one that actually decides whether review works.

Code review usually means someone skims your pull request, leaves two comments, and clicks approve. That's fine for a lot of changes. It's worth knowing what the rigorous end of the spectrum looks like, though, because everyday review is a shortcut for it. If you can name which part you cut, you can predict what leaks through.

That rigorous end is the Fagan inspection, published by Michael Fagan in the *IBM Systems Journal* in 1976. It is not a vibe. It is a defined process with roles, criteria, metrics and a stopwatch.

The six steps, and what each one is actually for

Planning. The moderator confirms the work product meets *entry criteria*: it compiles, it passes its own tests, the author isn't sending half-finished work to be finished by committee. Then materials go out and the meeting is scheduled. Entry criteria are the step most teams don't realise exists, and they are the whole reason an inspection isn't a design session in disguise.

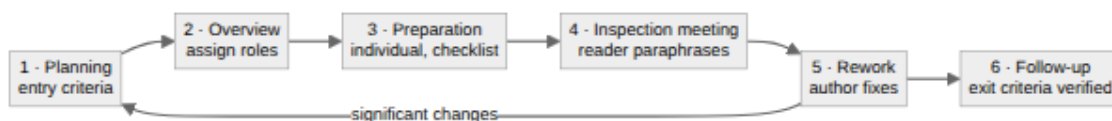
Overview. The author briefs the team on what the code is for and roles are assigned. This is the only step Fagan considered optional. If everyone already knows the subsystem, skip it.

Preparation. Each inspector reads the material *alone*, against a checklist, and logs candidate defects before anyone meets. This is where inspections actually find bugs. Skip it and the meeting becomes a group read-through, which finds a fraction as much.

Inspection meeting. The reader paraphrases the code aloud. Not the author. That distinction matters more than it looks: if the author narrates, they explain what they *meant*, and the team reviews the intention rather than the text. A second person forced to say out loud what the code does is how you find the gap between the two.

Rework. The author fixes what was logged. Significant changes loop back to planning and the thing gets inspected again.

Follow-up. The moderator verifies every logged defect was actually resolved and checks the *exit criteria* before closing. Not "the author says it's fixed."



Five roles, and why the author isn't in charge

Fagan separated the work into distinct roles, and the separation is the point. Every one of them is a check on a different failure mode.

The **moderator** runs the process: schedules, enforces entry and exit criteria, keeps the meeting on defects, and owns follow-up. The **author** wrote the thing, answers questions, and does the rework. They deliberately do not lead. The **reader** paraphrases the work product to the group. **Inspectors** find defects. The **recorder** logs every defect raised, with enough detail that follow-up can verify it.

The role most teams collapse is recorder. Nobody writes anything down, so "we discussed it" becomes the record, and three weeks later nobody can prove whether the fix landed. If you adopt

exactly one thing from this article, make it a written defect log.

The rate limit is the whole ballgame

Here is the number that decides whether your review works: roughly **150 lines of code per hour**. That's the ceiling, not a target. Read faster and detection falls off a cliff.

This is the finding people skip past, and it is the most useful thing in the entire method. It reframes the common complaint. When someone says "we do code review and bugs still get through," the question isn't whether they review. It's how many lines they approved per hour. A 2,000-line pull request skimmed in twenty minutes is running at roughly 6,000 lines an hour, forty times the rate at which humans reliably find defects. That review didn't fail. It was never a review.

The practical consequence is that **pull request size is a security control**. If you want review to catch anything, cap the diff. Everything else in this process is negotiable; the rate is physics.

The numbers Fagan actually reported

Fagan's 1976 study found **38 defects per thousand lines of code caught by inspection, against 8 per thousand caught by unit testing** on the same system, with total detection efficiency of **82%** across design and code inspection on a COBOL application.

Treat those as an existence proof rather than a promise. They came from 1970s IBM, with full-time inspection discipline and waterfall-era work products, and the modern replication literature reports a wide range depending on preparation quality. The durable claim isn't "inspections find 82%." It's that reading code carefully, before it runs, finds a different and larger class of defect than executing it does. And that this was measured, not asserted, half a century ago.

Where it lives now

Inspection isn't folklore; it's standardised. **IEEE 1028** defines five distinct review types: management reviews, technical reviews, inspections, walk-throughs, and audits. They are genuinely different things. An inspection is defect-focused, led by a trained moderator, with metrics. A walk-through is author-led and educational. If you're in a regulated environment and someone asks for evidence of "formal review," those words have specific meanings and you should use them precisely.

Almost nobody runs the full ceremony today, and that's a reasonable trade. But a modern pull request is a Fagan inspection with most of the steps deleted: no entry criteria, no individual preparation against a checklist, no reader, no recorder, no verified exit. Sometimes that's fine. On the code that authenticates users, handles money, or parses untrusted input, it isn't. Those are exactly the diffs worth putting through something closer to the real process.

The value here isn't adopting 1976 wholesale. It's that when a bug sails through review, you can name the step you cut. Nobody prepared. The author narrated. Nothing was written down. Nobody checked the fix landed. Or, most often, the diff was too big to read at human speed, and everyone approved it anyway.

Written by Scott S. — Contact: scott@scottschlangen.com — scottslab.io — <https://scottslab.io/posts/code-review-fagan-inspection>