



Article By: Jason Cohen,
www.SmartBear.com

Lightweight Code Review Episode 2: Why Code Inspections Fail

This article explains why Fagan Inspections and similar formal code review processes don't work in practice, and what we can do about it.

In [Episode 1](#) we gave the case for peer code review. In this Episode we explore why almost no one does "proper" inspections and set up the case for lightweight peer code review.

The story begins

Two years ago I was *not invited* to a meeting with the CTO of a billion-dollar software development shop, but I didn't know that until I walked in the room. I had been asked by the head of Software Process and Metrics to come and talk about a new type of lightweight code review that we had some successes with.

But the CTO made it clear that my presence was Not Appreciated.

"You see," he explained, "we already do code inspections. Michael Fagan invented inspections and his company is teaching us how to do it." His face completed the silent conclusion: "And you sir, are no Michael Fagan."

"Currently 1% of our code is inspected," offered the process/metrics guy. "We believe by the end of the year we can get it up to 7%." Here Mr. Metrics stopped and shot a glance over to Mr. CTO. The latter's face fell. Whatever was coming, they obviously had had this discussion before.

"The problem is we can't inspect more than that. Given the number of hours it takes to complete a Fagan inspection, we don't have the time to inspect more than 7% of the new code we write."

My next question was obvious: "What are you going to do about the other 93%?" Their stares were equally obvious — my role here was to convince the CTO that we had the answer.

This story has a happy ending, but before we get there I have to explain what it means to "inspect" code because this is what most developers, managers, and process engineers think of when they hear "code review." It's the reason this company couldn't review 93% of their code and why developers *hate* the idea. And changing this notion of what it means to "review code" is what we've been doing at companies like this one for the last four years.

Michael Fagan — father of a legacy

If you've ever read anything on peer code review you know that Michael Fagan is credited with the first published, formalized system of code review. His technique, developed at IBM in the mid-1970's, demonstrably removed defects from any kind of document from design specs to OS/370 assembly code. To this day, any technique resembling his carries his moniker of "code inspection."

Take a deep breath...

I'm going to describe a "code inspection" in brief, but brace yourself. This is heavyweight process at its finest, so bear with me. It will all be over soon, I promise. A code inspection consists of seven phases. In the Planning Phase the author gathers Materials, ensures that they

meet the pre-defined Entry Criteria, and determines who will participate in the inspection. There are four participants with four distinct roles: The Author, the Moderator, the Reviewer, and the Reader. Sometimes there is an Observer. All participants need to be invited to the first of several meetings, and this meeting must be scheduled with the various participants. This first meeting kicks off the Introduction Phase where the Author explains the background, motivation, and goals for the review. All participants get printed copies of the Materials. (This important — it's not a Fagan Inspection unless it's printed out.) The participants schedule the next meeting and leave. This starts the Reading Phase where each person reads the Materials, but each role reads for a different purpose and — this is very important — no one identifies defects. When the next meeting convenes this starts the Inspection Phase. The Moderator sets the pace of this meeting and makes sure everyone is performing their role and not ruining anything with personal attacks. The Reader presents the Materials because it was his job to "read for comprehension" since often someone else's misunderstanding indicates a fault in the Materials. During the meeting a Defect Log is kept so the Author will know what needs to be fixed. Before the meeting ends they complete a rubric that will help with later process improvement. If defects were found the inspection enters the Rework Phase where the Author fixes the problems, and later there will be a Verification Phase to make sure the fixes were appropriate and didn't open new defects. Finally the inspection can enter the Completed Phase.

...you can let it out now

The good news is, this works. It uncovers defects, it helps when training new hires, and the whole process can be measured for process insight and improvement. If you have extra money laying around in your dev budget, Mr. Fagan himself will even come show you how to do it.

The bad news should be obvious in this day of Agile Methodologies. Studies show that the average inspection takes 9 man-hours per 200 lines of code, so of *course* Mr. CTO couldn't do this for every code change in the company.

Over the years there have been experiments, case studies, and books on this subject, almost always using some form of "code inspection" as the basis. In our survey of published case studies and experiments in the past 20 years, we found that 95% of them tried inspections only in small pilot groups, and that in *no case* were they able to apply the technique to all their software development projects.

If "Agile" can do it, why can't we?

But surely there is another way. Fagan inspections were designed in the days when business logic was written in assembly language and "computer science" wasn't a major and dinosaurs roamed the earth.

Have we learned nothing since then? Don't we need different techniques when reading object-oriented code in a 3-tier application? Don't the challenges of off-shore development require

new processes? Hasn't the rise of Agile Methodologies shown us that we can have process and metrics and measurement and improvement and happy developers all at the same time?

E-mail isn't enough

Of course other ways exist. Most of them are (rightfully) rejected by the process community because they are typically unmeasurable and uncontrollable. For example, many groups have their version control server email code-diffs every time something is checked in. Those developers responsible for that code — or just interested — have a chance to look it over, start conversations about it, and possibly to open issues to have someone fix something that isn't right.

But whether your reviews are by email, by developers standing over each other's shoulders, or something similarly informal, how do you know whether all your code has been reviewed? If defects are discovered and authors are told, how do you know whether authors really do fix the defects? How much time do developers and reviewers spend locating, opening, and jumping to "line 732 of file //depot/foo/bar/baz/bat.java" and then running back to Outlook to make a comment? How does a reviewer manage the 50 reviews that pile up because each takes seven days because the author is in Taipei and every question takes 12 hours to answer? And then the higher-level questions: How much time are developers spending during review? How many defects are they finding per 1000 lines of code or per hour? How does that compare to QA? Is this really worth it?

The Epiphany

Over the last four years we've arrived at the answer: We can review code in an Agile, lightweight way using a development tool that integrates source code viewing with chat room collaboration. The tool must gather metrics *automatically*, because although metrics are critical to process improvement they are eschewed by developers. The tool must do just enough to support developers, but not be so inflexible or all-encompassing that the tool starts dictating the process.

This tool — [CodeCollaborator](#) — is the culmination of everything we've learned about lightweight peer code review at SmartBear. We even have the data that proves it works.

But as a critical reader you might read this and think "Please, no more tools, I'm drowning in tools, I don't believe you anyway."

In the six remaining articles in this series we will present our stories, our data, and our theories on how code review can be lightweight and fun, yet effective, measurable, and give a satisfactory answer to all the questions above.

So finish the story already!

By now you can guess how the story ends. Using arguments not unlike those above, Mr. Metrics and I convinced Mr. CTO to at least try our lightweight code review technique in a pilot program with a one development group that was already hopelessly opposed to Fagan inspections. The metrics that came out of that group demonstrated the effectiveness of the lightweight system, and within 18 months CodeCollaborator was deployed across the entire organization.

What's Next: In the [next article](#) we describe the pros and cons of four kinds of code review. Later we'll provide hard data to back up our theory of lightweight code review and talk about the social ramifications of personal critiques.

Jason Cohen is the author of [Best Kept Secrets of Peer Code Review](#) and the founder of [SmartBear Software](#). SmartBear sells [CodeCollaborator](#), the only lightweight peer code review tool and is well known for conducting the largest case study of peer review ever published.



SmartBear Software

+1 978.236.7900

www.smartbear.com

© 2010 SmartBear Software. All rights reserved. All other product/brand names are trademarks of their respective holders.