# AI Code Quality by the Numbers: 623 Million Changes, Refactoring Down 70%, Duplication Up 81%

Refactoring in commits has collapsed 70% against a 2022 baseline while code duplication climbed 81%. That comes from 623 million code changes, not a survey — and it points the opposite way from every adoption stat you've been quoted.

TL;DR

GitClear's June 2026 report "The Maintainability Gap" analyzed 623 million code changes from 2023–2026 and found that as roughly a quarter of commits now show measurable AI assistance, eight maintainability signals moved the wrong way at once: refactoring −70%, duplication +81%, copy/paste +41%, error-masking catch blocks +47%, cross-file reuse −35%, legacy maintenance −74%. The findings are correlational, not causal. But the pattern matches what developers report themselves — 84% adoption, 3% high trust — and the cost shows up on a delay, in code you'll be reading in 2028.

## The headline number

623 million code changes. That's the sample in [GitClear's "The Maintainability Gap: AI Code Quality in 2026"] (version 2026.6.1, published June 2026, page last updated 2026-08-26). It matters because almost everything else in the AI productivity debate is self-report: developers saying they feel faster, executives saying deployments feel smoother. GitClear didn't ask anyone anything. It read the diffs, across 2023 through 2026, and tracked eight signals tied to code reuse and risk accumulation rather than raw throughput.

**623M**
code changes analyzed, 2023–2026
GitClear, The Maintainability Gap, June 2026

**−70%**
refactoring line moves vs 2022 baseline
GitClear via tech-insider.org, 2026-07-12

**+81%**
code block duplication, 2023 → 2026
GitClear via Pagerly, 2026-08-30

**~25%**
of all 2026 commits show measurable AI assistance
LeadDev, 2026-07-07

## The breakdown

**Refactoring: −70%.** One secondary summary puts moved-or-refactored code at 21% of changed lines in 2022 versus 3.8% in 2026. Moving code is how a codebase stays navigable — it's the signal that someone read what was already there before adding to it. A drop of that magnitude means the reading step is being skipped.

**Duplication: +81%.** Concretely, 40.3 duplications per million changed lines in 2023 to 73.0 per million in 2026, per [Pagerly's summary of the GitClear data]. Duplication is the cheapest thing in the world to create and the most expensive thing to fix later, because every copy has to be found before any of them can be corrected.

**Copy/paste within commits: +41%.** Copy-pasted code went from 9.4% of new code in 2022 to 15.7% in the first half of 2026. This is the mechanism behind the duplication number — not an independent finding, but confirmation that the duplication isn't a measurement artifact.

**Error-masking constructs: +47%.** Catch blocks that swallow exceptions. This is the one that should make you uncomfortable, because it doesn't degrade maintainability slowly — it degrades observability immediately. A swallowed exception is a production incident you find out about from a customer.

**Cross-file function calls: −35%.** GitClear uses this as a proxy for reuse. Fewer calls across file boundaries means less code is being invoked from elsewhere and more is being written fresh in place. GitClear CEO Bill Harding put it directly: "Every time you want something, AI creates a new package for it. That general approach to building has all sorts of consequences."

**Long-term legacy maintenance: −74%.** The steepest decline in the set. Harding again: "It's not just duplication, it's about not tending to legacy code." That's the finding that reframes the rest. Duplication is a symptom. Not touching old code is the behavior.

> **Every one of the eight signals moved in the same direction at the same time. That is not noise; that is a change in how software gets written.**

## The comparison

| SIGNAL | BASELINE | 2026 |
|---|---|---|
| **Moved / refactored code** | **21% (2022)** | 3.8% |
| **Copy-pasted code, share of new code** | **9.4% (2022)** | 15.7% (H1 2026) |
| **Duplications per million changed lines** | **40.3 (2023)** | 73.0 |
| **Two-week code churn** | **~16% (2023–24)** | just under 19% |

Two-week churn — code rewritten or deleted within a fortnight of being committed — rose from about 16% in 2023–2024 to about 19% in 2025 and held just under 19% in 2026, roughly a 15% increase, per Big Agile's analysis. That's the fastest-returning cost in the dataset. Duplication bills you in 2028; churn bills you this sprint.

The DORA trend Big Agile summarizes fits the same shape. In 2024, AI adoption was associated with *lower* throughput and worse delivery stability. By 2025 throughput turned positive — but the instability, change failure and rework, did not resolve. Speed recovered. Stability didn't. Meanwhile a Google Cloud post dated 2026-06-09 summarizing DORA data reports 90% of respondents now use AI at work.

Developers themselves are not confused about this. The Stack Overflow Developer Survey 2026 — 49,000 respondents across 177 countries — found 84% AI coding tool adoption against 3% who highly trust AI-generated output, with 46% actively distrusting it. The top frustration, cited by 66%, is "AI solutions that are almost right, but not quite." And 45% say debugging AI-generated code takes longer than writing it themselves.

| THE CLAIM | THE REALITY |
| --- | --- |
| AI coding assistants make teams faster, and the sentiment data proves it — 84% adoption, 90% of DORA respondents using AI at work. | Adoption is not endorsement. The same Stack Overflow survey that reports 84% adoption reports 3% high trust and 45% saying debugging AI output takes longer than writing it manually. Throughput gains in DORA data arrived without stability recovering. |

## What the data does not say

! **Correlation, stated plainly**
Available coverage presents GitClear's findings as correlational: maintainability signals worsen as the share of AI-assisted commits rises. The published material does not establish that AI assistance *caused* the decline. What's also true: no named researcher has published a point-by-point methodological rebuttal, and the direction is consistent across all eight signals.

I'd add a caveat GitClear can't address from diffs alone. Refactoring line moves measure editing behavior. If an assistant rewrites a function in place rather than moving it, the diff may not register as a "move" even though the maintenance work happened. I don't think that explains a 70% drop, or a 74% drop in legacy maintenance — but it's the instrumentation question I'd want answered before treating the exact magnitudes as

precise.

## So what

The throughput gains are real. I've written before about [the gap between developers feeling faster and delivery actually accelerating], and this dataset explains part of the mechanism: the work isn't disappearing, it's being deferred into the codebase.

MY TAKE

*The +47% in error-masking constructs is the number I'd act on first, ahead of duplication. Duplication is a maintainability tax you can pay down on a schedule. A catch block that swallows an exception is a silent failure mode shipping to production today, and it is trivially detectable with a lint rule. If you do one thing after reading this, ban empty catch blocks in CI. Cost: an afternoon. That is the highest-leverage rule change available in this dataset.*

MY TAKE

*The second thing I'd change is what code review looks for. Reviewers were trained to catch bugs, and AI-generated code is often not buggy — it's the "almost right" that 66% of Stack Overflow respondents complain about. Review should be asking a different question now: does this duplicate something that already exists in the repo? A duplication check in CI catches what a human reviewer, reading one PR in isolation, structurally cannot. I'd weight that ahead of hiring another reviewer.*

Third, and this is the uncomfortable one for engineering leadership: if legacy maintenance activity fell 74%, no amount of tooling fixes that. That's an incentive problem. Nobody is measured on tending old code, and AI made writing new code so cheap that the relative cost of maintenance went up. My inference, not GitClear's finding: teams that explicitly budget maintenance time will separate from those that don't within about two years, and the separation will show up as incident rate rather than velocity.

One related pattern worth checking against your own dependency graph: Sonatype found [AI coding assistants hallucinating 27.75% of package upgrades]. Harding's "AI creates a new package for it" observation and the −35% in cross-file reuse describe the same instinct from the other side — reaching outward instead of inward.

✓ **What I'd instrument**
Track two-week churn and duplications-per-million-changed-lines on your own repo before arguing about whether the industry numbers apply to you. Both are measurable from your own git history. If your churn is flat at 16% and your duplication is stable, this report describes someone else's problem. If it's tracking toward 19% and 73, you have the same bill coming.

BOTTOM LINE

**Across 623 million code changes, every maintainability signal GitClear tracked moved the wrong way while AI-assisted commits reached ~25% of the total — correlational, but consistent, and matched by developers' own 3% trust rating. The throughput is real and the debt is real; the question is whether you're measuring the second one. If you want an outside read on what your git history says about your codebase's trajectory, Book a call →**

WRITTEN BY

Artur Markus

Software engineer in Basel building AI infrastructure and agentic systems, with a background in pharmaceutical automation, GxP compliance, and validation. I help teams turn AI from a chatbot into a reliable operational layer.

Book a meeting →