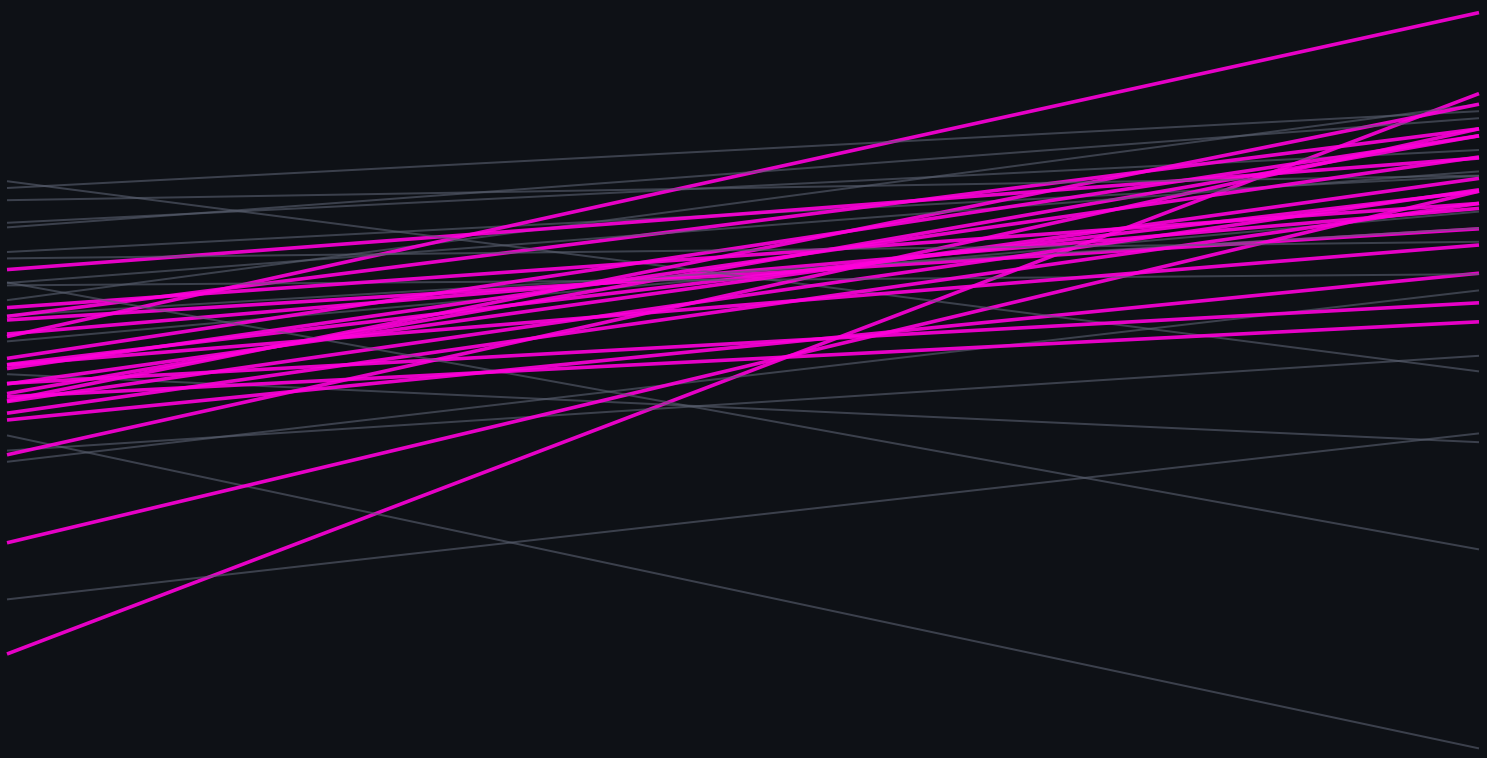
Forewit



FOREWIT 2026: THE IMPACT OF AI ON CODE QUALITY

The Complexity Bill

Code quality in the age of AI: what 201,576 commits across 46 open-source codebases show about complexity, change size and rework.

FOREWIT · OCTOBER 2026 EDITION

AI is helping developers ship more code, but it is also accelerating complexity that compounds fast. Left unchecked, that complexity will build up silently, in code that is becoming slower and riskier to change.

Forewit research · 201,576 commits · 46 open-source codebases · October 2026

Key takeaways

— **New code is denser.** Since AI adoption, every 100 new lines carry 8.1 decision points (`if` statements, loops and similar), up from 5.2. That is 50 percent more paths to understand and test.

— **Each engineer ships 71 percent more code**. In the median codebase, each engineer now writes 424 production lines a week, up from 293. The 71 percent is the median of each codebase's own change; the change between those two medians is 45 percent. Team size hasn't changed in any consistent way.

— **Less code is thrown away, but AI commits are revisited more.** The share of new code rewritten within 90 days fell from 18 to 12 percent. AI-marked commits are revisited 43 percent more often than other commits and are 17 percent more likely to add a hard-to-test function.

— **More testing hasn't reduced bug fixes.** Test commits rose from 4.7 to 8.3 percent of all commits, but bug fixes still make up a quarter. One possible reason, which the data can't confirm, is that testing has kept up with the volume of code but not with its complexity.

— **Complexity was flat until 2025, then rose.** It held near 5 decision points per 100 lines from 2022 through 2024 and reached about 10 by October 2026. Codebases where AI use isn't marked also rose, but codebases that adopted AI rose faster.

— **Claude-marked commits are bigger.** Claude signs 69 percent of AI-marked commits, largely because Claude Code marks commits by default and most other tools don't. In the same codebases, Claude's commits are 54 percent larger than unmarked ones and are revisited 55 percent more often.

— **Complexity keeps rising.** On the current trend, new code will carry 42 percent more decision points by late 2028, rising from 9.0 to 12.8 per 100 lines. If growth keeps compounding at 3.5 percent a month, the increase would be 128 percent, to 20 per 100 lines. Every extra branch is another path to reason through, test and maintain.

Percent changes are the median of each codebase's own change. They can differ from the change between the two medians shown.

KEY POINT

What this means for engineering leaders

Each engineer now owns more code, and that code is denser. The cost is building up in code that is slower and riskier to change. We expect delivery metrics such as DORA to be the last to show it.

INTRODUCTION

Since AI arrived, new code carries more complexity, and it's compounding faster

By Sean Madigan and Antoine Frau

This is the first edition of The Impact of AI on Code Quality, Forewit's study of how AI is changing the code teams ship. In under two years, AI coding assistants and agents have gone from experiment to the default way much software is written. To see what this means for the codebases themselves, we analyzed 201,576 commits across 46 open-source projects between October 2024 and October 2026, and traced complexity back to October 2022.

The first finding is about density. Every 100 new lines now carry **8.1 decision points, up from 5.2**. That is about 50 percent more paths to understand and test, in pull requests that are already larger than before.

The second is about speed. From 2022 to 2024, complexity was flat and even eased slightly, at −0.4 percent a month. **Since January 2025, the complexity of new code has grown 3.1 percent a month, or about 44 percent a year.** Across whole codebases the build-up is slower but still real: excess complexity rose 14 percent after AI adoption.

Most teams won't see this, because **the industry measures delivery, not structure.** DORA metrics track how fast teams ship, not how each change reshapes the codebase. That gap was manageable when engineers understood all the code they shipped. With agents writing more of it, we expect the problem to stay hidden until it shows up as incidents and stalled features.

In Forewit's view, which this data doesn't test, none of this is inevitable. Teams that invest in agent-native Local Code QA can keep complexity in check by moving quality checks into the agent loop. That means running static analysis (SAST), runtime analysis and verbosity checks locally while code is generated, rather than after a pull request is opened, when the feedback loop is already strained.

> **The fundamentals of code quality haven't changed.** What has to change is where and when teams apply them.

Since AI adoption, new code carries 50 percent more decision points per 100 lines, and complexity growth has gone from −0.4 to +3.1 percent a month

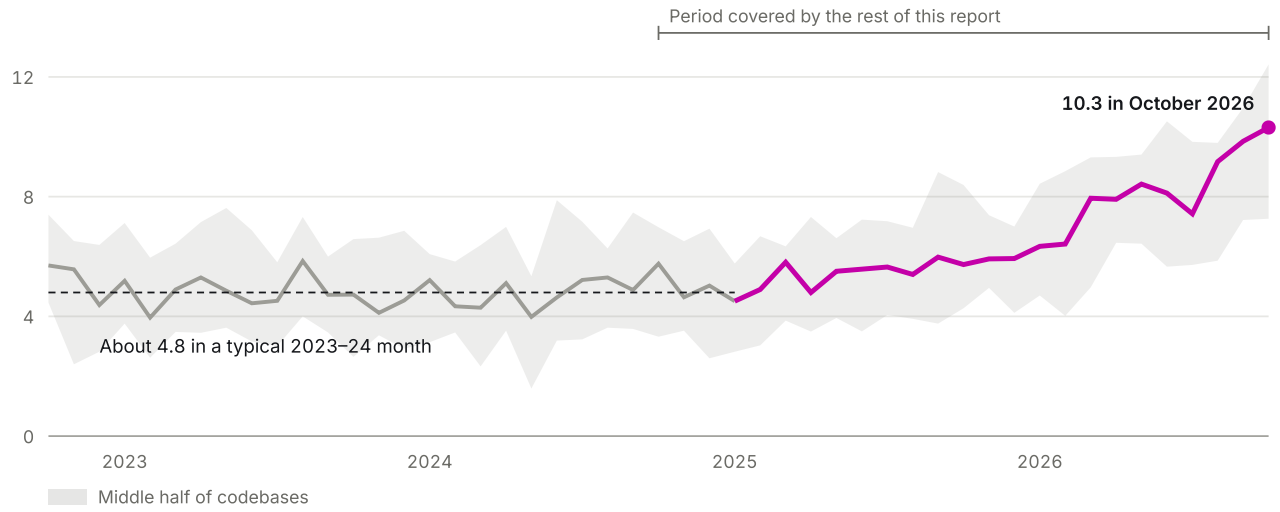
A decision point is any place where code can take one of two paths: an `if`, a loop or a `case` in a `switch` (McCabe, 1976). Each one is another path an engineer has to understand and a test has to cover.

The rise is recent. From October 2022 to December 2024, complexity held steady at about 5 decision points per 100 new lines. From January 2025 it began to climb, slowly at first and then sharply in 2026, reaching 10.3 in October 2026 (Exhibit 1). The median across all codebases grew about 3.5 percent a month. The typical codebase's own growth was 3.1 percent, the figure in this section's heading (Appendix C).

Exhibit 1

Complexity held near 5 for two years, then climbed to about 10.

Decision points per 100 new lines, median codebase each month, October 2022 to October 2026



Four-year re-run of all 46 codebases, with 29–41 codebases in each month. A fixed set of 24 codebases shows the same rise (Appendix C). Months with fewer than 1,000 production lines are left out. Shaded band: middle half of codebases.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2022 to October 2026

Forewit

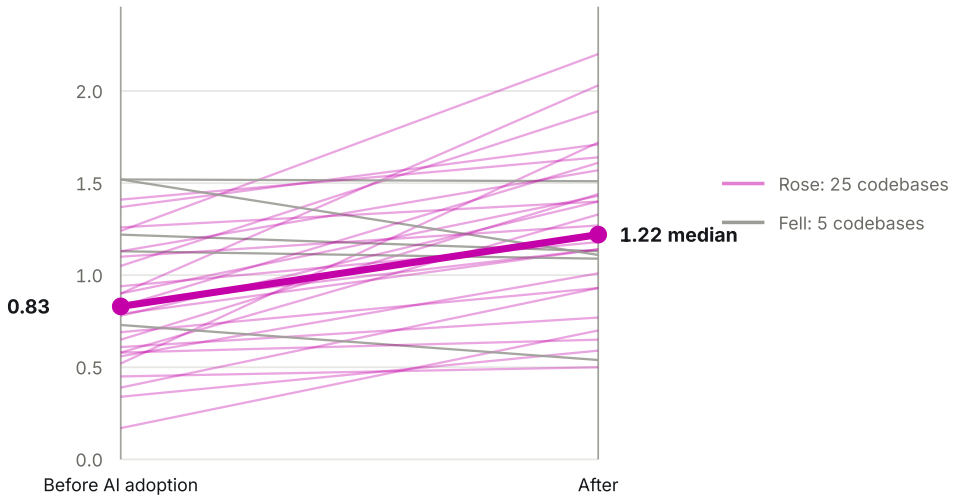
The rise is just as clear when we measure it around each codebase's own adoption date. Before adoption, the typical codebase added 5.2 decision points per 100 new lines. After adoption, it added 8.1. The rise appeared in 27 of 30 codebases.

It also shows up in the functions that are hardest to work with, those with more than 10 decision points, where testing every path becomes impractical (Exhibit 2).

Exhibit 2

25 of 30 codebases now add more hard-to-test functions per 1,000 new lines.

Functions with more than 10 decision points added per 1,000 new production lines, before vs after AI adoption



30 codebases with an AI adoption date; bots excluded. Each line is one codebase.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

KEY POINT

Why we count per 100 lines

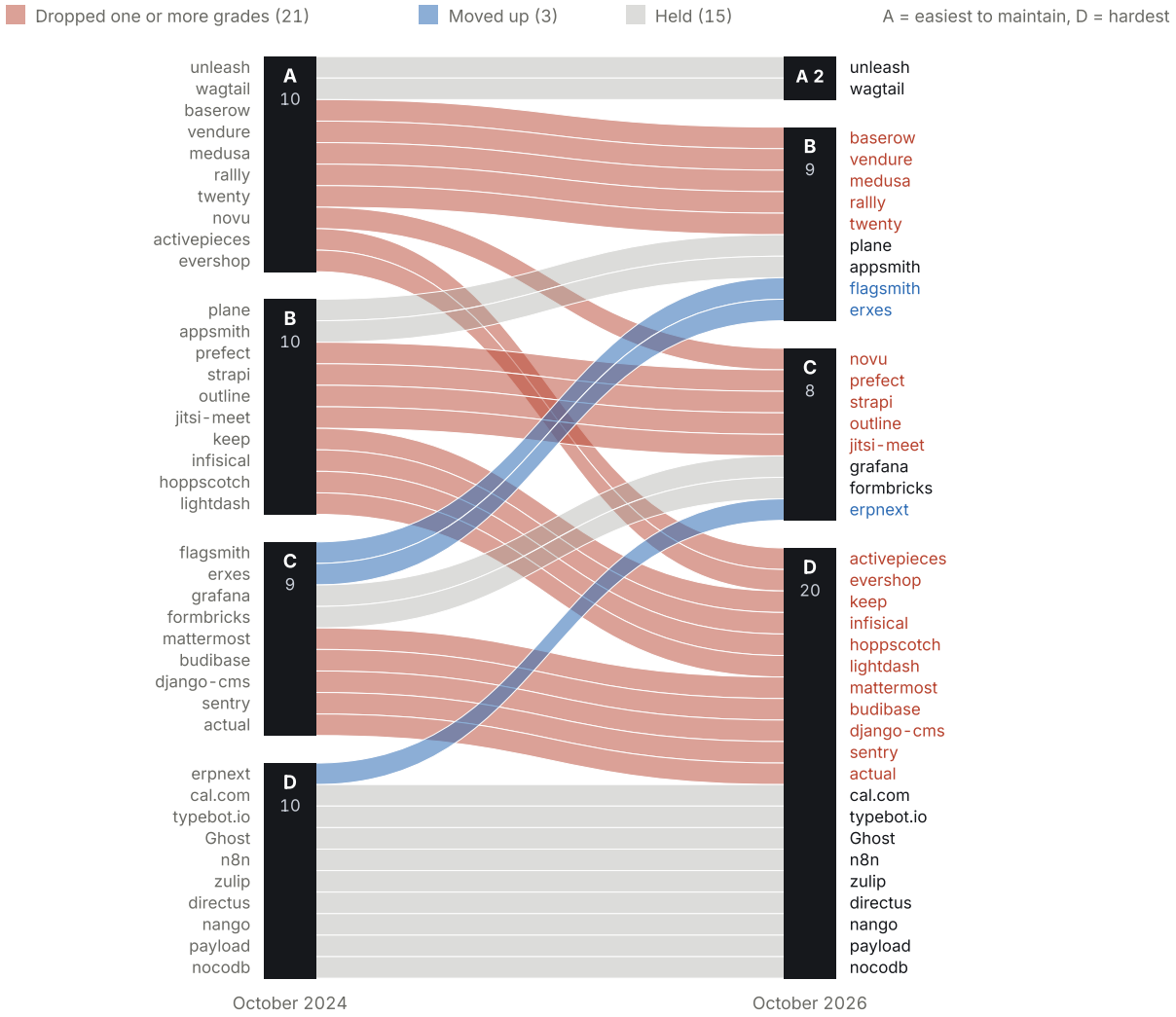
Counting per line stops a large change from looking complex simply because it is large. The unit doesn't affect the result: a 50 percent rise is 50 percent whether it is measured per 100 lines or per 1,000. We use 100 because a typical commit adds about 40 lines, so 100 lines is roughly one small pull request, and the numbers stay readable (5 to 8 rather than 52 to 81).

This hard-to-change code is also building up across each codebase as a whole. We graded every codebase from A to D by its excess complexity: the decision points its overly complex functions carry beyond 10, per 1,000 lines. We split the 39 codebases into quarters by this score at the start of the period, with A the lowest and D the highest, then graded them again at the end using the same cut-offs (Exhibit 3).

Exhibit 3

The number of codebases with the lowest maintainability grade doubled, from 10 to 20 of 39.

Each codebase's grade, from excess complexity per 1,000 lines, in October 2024 (left) and October 2026 (right)



39 codebases with a complexity trend. Grades are quarters of this benchmark at the start (cut-offs 4.1, 7.2, 9.6), not an industry scale. Each band is one codebase. 21 of 39 codebases dropped at least one grade, 3 moved up and 15 stayed the same.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

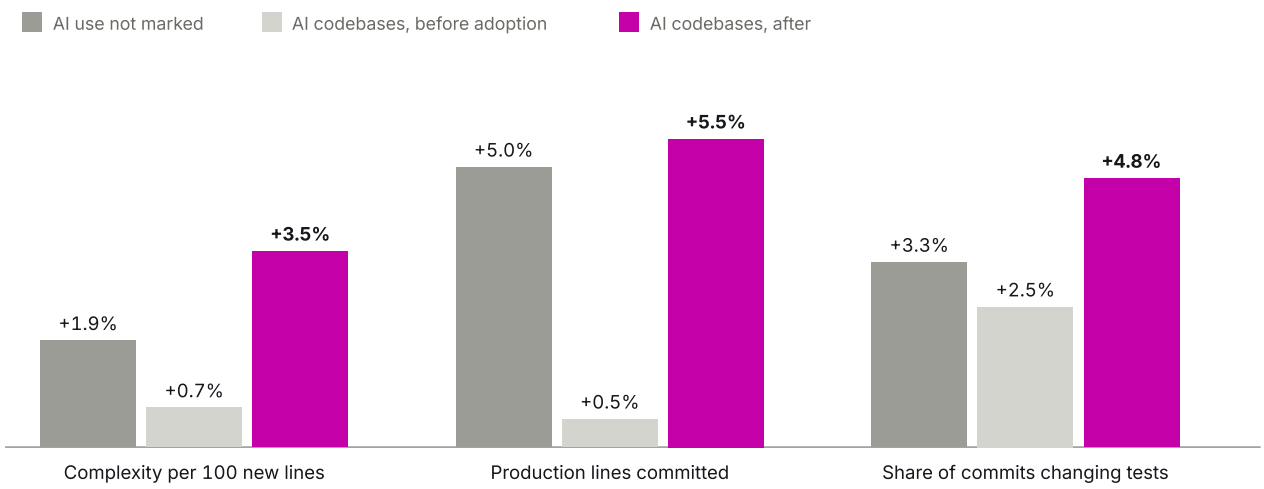
Complexity began rising in 2025 across the board, including in unmarked codebases, but it rose faster where AI use is marked

We looked for a trend that appears only after AI adoption and didn't find one. Rising complexity, more code and more testing all appear in codebases where AI use isn't marked (unmarked codebases, from here on). These codebases have few AI-marked commits, but 9 of the 10 in Exhibit 4 have some (12 to 54 each), so AI is present in them. How much AI use goes unmarked can't be measured. This means they are not an AI-free comparison group: their rise doesn't count against an AI effect, but it can't confirm one either. In codebases that did adopt AI, all three trends ran faster after adoption (Exhibit 4).

Exhibit 4

The same trends appear in unmarked codebases, and run faster where AI use is marked.

Median growth per month



AI use not marked: 8–10 of the 16 codebases without an adoption date, with almost no AI-marked commits. The other 6 have too few active months to fit a trend. AI codebases: 25–26 with enough months on each side of adoption. Growth rates come from a trend line fitted to each codebase's monthly values.

Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

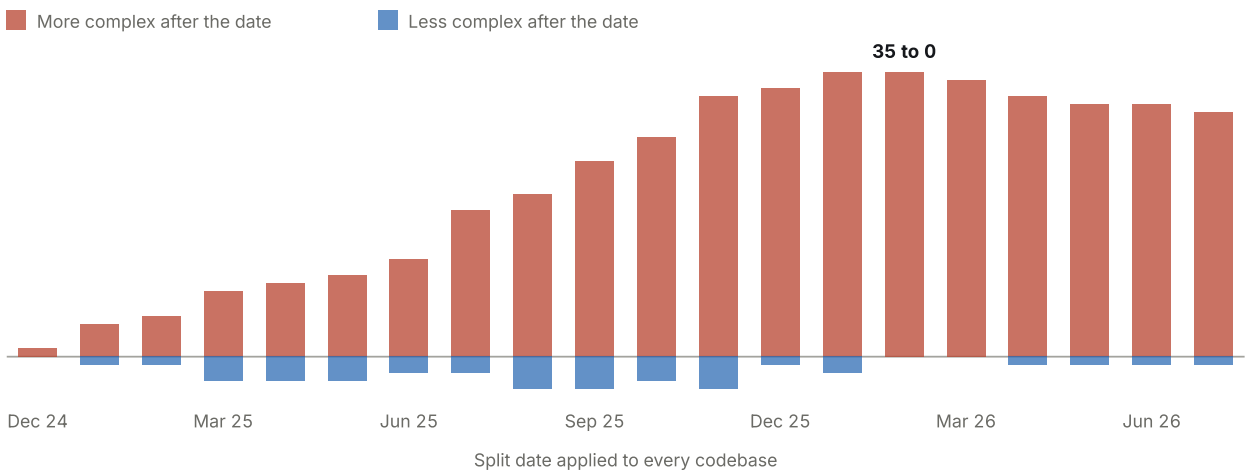
Forewit

The rise follows the calendar rather than each codebase's adoption date. If adoption caused it, the gap would peak at each codebase's own adoption date and fade at other dates. Instead, it appears at all 20 dates we tested (Exhibit 5). That fits an industry-wide shift starting in 2025, including AI use that isn't marked in commits. Within the same codebases after adoption, AI-marked code wasn't clearly more complex per line than other code. AI-marked commits were, however, more likely to add a hard-to-test function, because they were bigger (Exhibit 8).

Exhibit 5

Wherever the split date falls, more codebases become more complex than less complex.

Codebases whose complexity per 100 new lines rose or fell when the same split date is applied to every codebase



All codebases with enough history; split dates from December 2024 to July 2026.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

KEY POINT

What this rules out, and what it doesn't

Through 2024, complexity held near 5 everywhere (Exhibit 1). From 2025, it rose almost everywhere. Codebases with few AI-marked commits rose too, by 1.9 percent a month, but 9 of the 10 show some AI use, so they can't tell us what codebases without AI would have done. In codebases that adopted AI, complexity climbed faster after adoption (Exhibit 4), but the data can't say how much of that is due to AI. In short, complexity rose as AI went mainstream, but this data can't prove that AI caused the rise.

After AI adoption, each engineer ships 71 percent more production code a week, with no clear change in team size

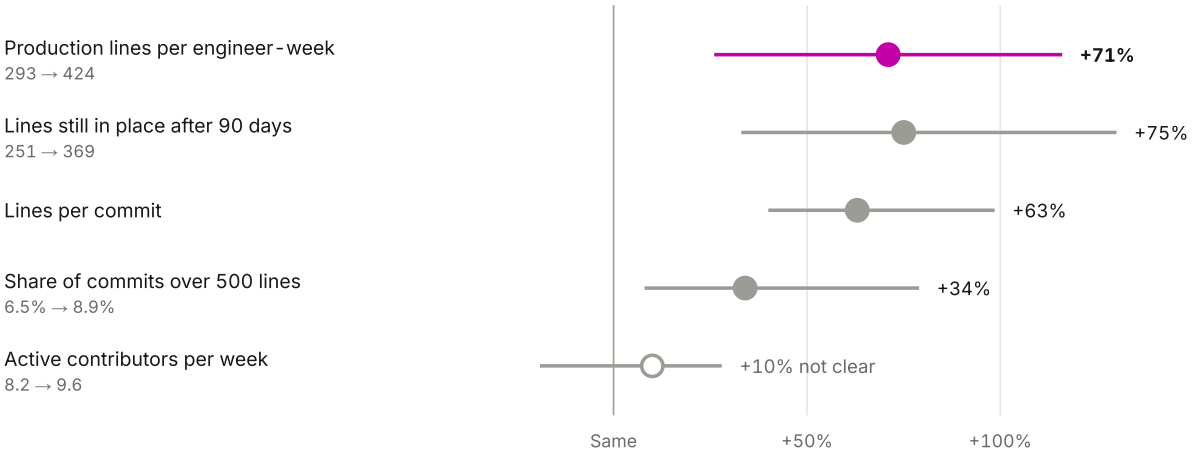
The extra code isn't just churn. Counting only lines still in place 90 days later, output per engineer rose 75 percent. Commits also got bigger, and very large commits became more common. Over the two years, the typical codebase grew 40 percent, from 230,000 to 350,000 lines of production code (Exhibit 6).

More code per person means more code for each reviewer to read and for each engineer to own.

Exhibit 6

Each engineer ships 71 percent more code a week, while team size hasn't clearly changed.

Median change per codebase, after vs before AI adoption; line = 95% confidence range



30 codebases with an AI adoption date; bots excluded. Percentages are the median of each codebase's own change. Measured between the two medians shown, lines per engineer-week rose 45% (293 to 424) and lines still in place after 90 days rose 47% (251 to 369). The 90-day figure includes recent commits that had less time to be rewritten (Appendix C). Hollow marker = not statistically clear: the 95% range includes zero or a sign test gives p > 0.05.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

Less new code is rewritten after AI adoption, but AI-marked commits are revisited 43 percent more often than other commits

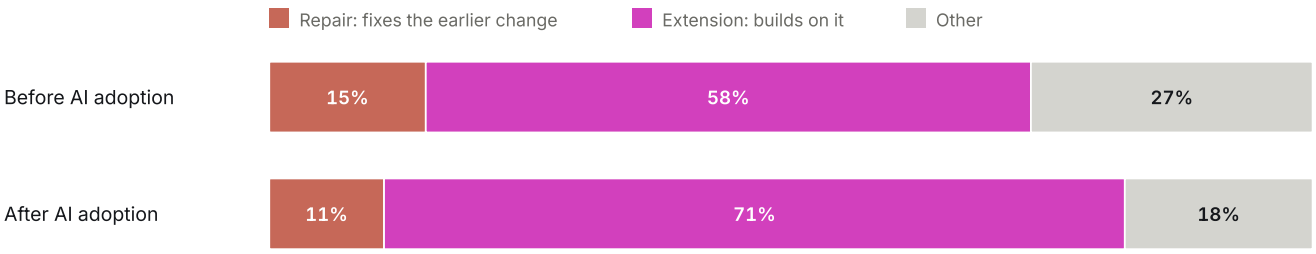
This section uses two measures, and neither is a bug count. **Rework** is new code that is rewritten or deleted within 90 days. A **follow-up** is any later commit that touches the same code. A follow-up can be a **repair** (it fixes the earlier change), an **extension** (it builds on it) or unrelated. So a follow-up is not the same as rework, and only some follow-ups are repairs.

After AI adoption, rework fell in 24 of 30 codebases. The decline holds in the codebases with the fewest recent commits, whose code had the most time to be rewritten (Appendix C). The number of follow-ups per change didn't clearly change, but fewer of them were repairs (Exhibit 7).

Exhibit 7

When code is revisited, it is now more often to build on it than to fix it.

What follow-up commits did to earlier changes, median codebase, % of follow-ups



30 codebases with an AI adoption date. Follow-up types were judged by an AI model, and verdicts below 0.5 confidence were dropped. Repair and extension are medians; Other is the remainder.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

KEY POINT

Less rework is not all good news

It can mean code was right the first time. It can also mean code is being left alone when it should be cleaned up. The rise in complexity suggests at least some of the latter.

In the same codebases, AI-marked commits are revisited 43 percent more often than other commits and are 17 percent more likely to add a function with more than 10 decision points. This held in 21 of 26 codebases. When commits of the same size are compared, the gap disappears, and AI-marked code isn't clearly more complex per line than other code (Exhibit 8).

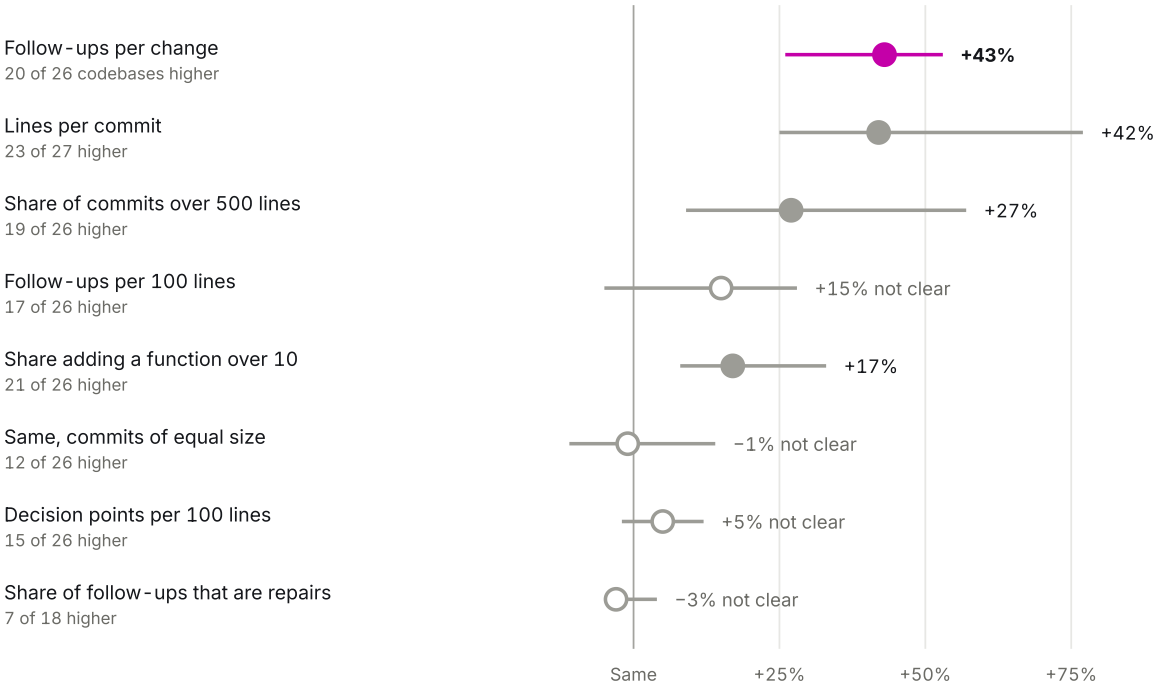
AI-marked commits also land where code changes most. They made up 9 percent of all commits but 20 percent of the edits to each codebase's most-edited functions, a pattern that held in 24 of 30 codebases. Part of the reason is size: bigger commits touch more functions.

Denser code hasn't yet led to more fixes. Commits that added more decision points per line were no more likely to be fixed within 90 days than commits of the same size that added fewer (+3 percent, range −8 to +23 percent, 32 codebases; not statistically clear). Earlier research links complexity to failures over longer periods (Nagappan et al., 2006), which a two-year window can't yet test.

Exhibit 8

AI-marked commits are bigger, so each one adds more complexity. Per line, the difference isn't clear.

AI-marked vs other commits in the same codebase after adoption, median difference with 95% range



26 codebases with an AI adoption date and at least 20 AI-marked commits. Hollow marker = not statistically clear: the 95% range includes zero or a sign test gives p > 0.05. Other commits include unmarked AI use, so true gaps may be larger.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

Test changes rose from 4.7 to 8.3 percent of commits, while bug fixes held at a quarter of commits, as they did before AI adoption

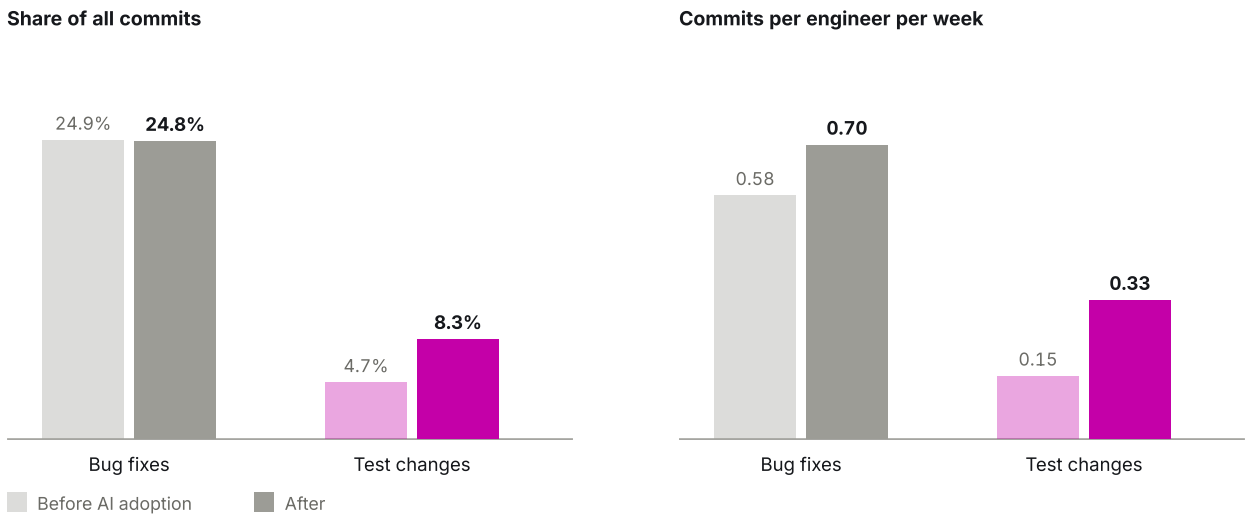
Test commits per engineer rose in 29 of 30 codebases. Each engineer made more than twice as many test commits a week as before: 0.33, up from 0.15. Measured as test commits per 1,000 new lines, however, testing grew only about as fast as the code. The data has no test line counts or coverage.

Bug fixes stayed at about one commit in four. Because each engineer commits more often, bug-fix commits per engineer rose from 0.58 to 0.70 a week, although this rise isn't statistically clear. Reverts, where a change is undone, stayed rare at under 0.5 percent of commits (Exhibit 9).

Exhibit 9

Test changes nearly doubled, while bug fixes held at a quarter of commits.

Median codebase, before and after AI adoption



30 codebases with an AI adoption date. Commit types are read from commit messages and were 70–88% accurate on 100 commits checked by hand. The per-engineer rise in bug fixes (median +14% per codebase) is not statistically clear.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

KEY POINT

More testing has not reduced bug fixes

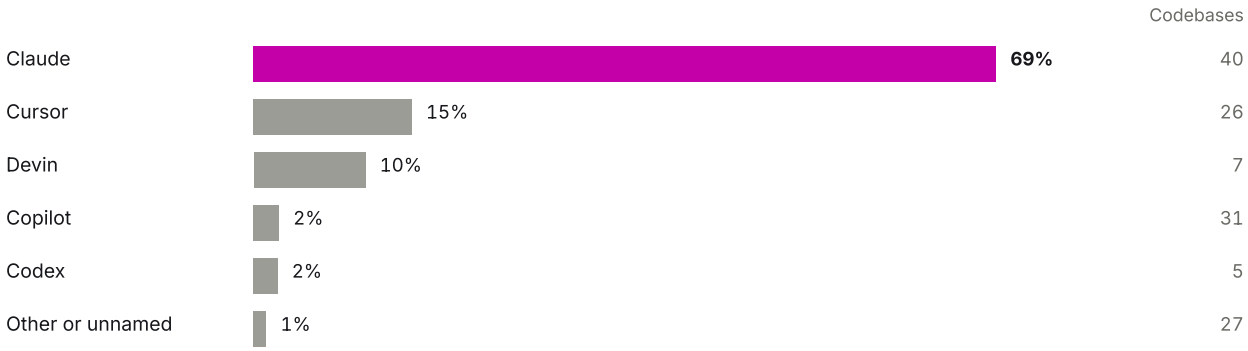
One possible reason is that testing has kept pace with the volume of code but not with its complexity. Every 100 new lines now carry 50 percent more decision points, so the same amount of testing covers a smaller share of the paths. The new tests may also share the assumptions of the code they check. The data can't tell these explanations apart, because it has no test coverage.

Claude signs 69 percent of AI-marked commits, which are 54 percent bigger than unmarked commits, though newer Claude models make smaller ones

Exhibit 10

Claude signs 69 percent of AI-marked commits.

Share of 21,760 AI tool marks found in commit authors, co-author lines and messages



This shows which tools sign commits, not which tools teams use. Claude Code adds a co-author line by default. Two engineers named Devin, whom the detection rule counted as the Devin agent, are excluded.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

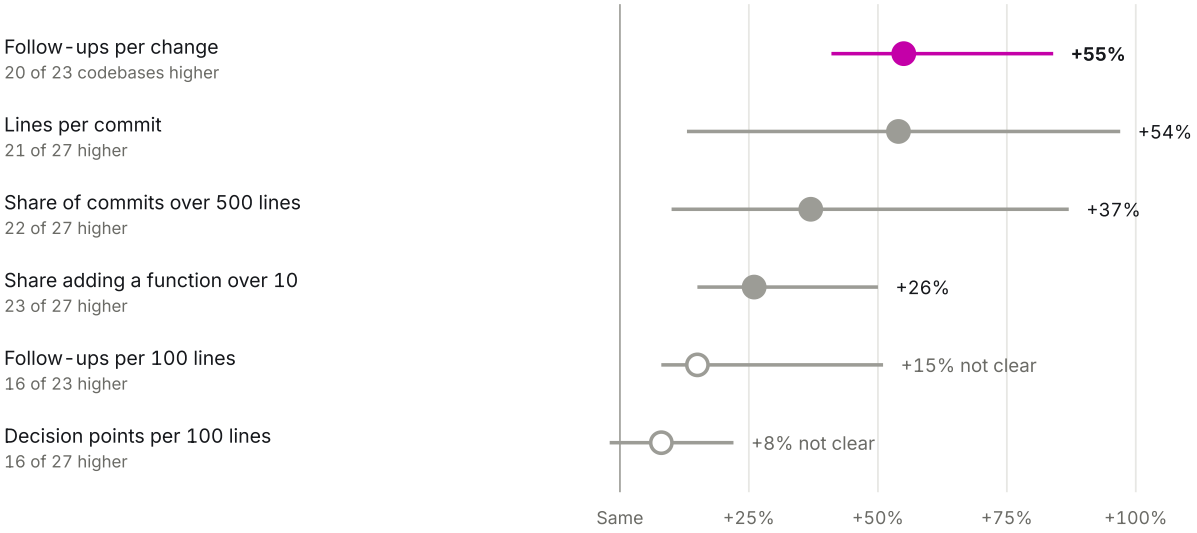
Forewit

We compared each tool's commits with unmarked commits in the same codebase and the same months. Only Claude appears in enough codebases to give a clear answer. Its commits are bigger, more often very large and revisited more often, but they are not clearly denser per line (Exhibit 11).

Exhibit 11

Claude-marked commits are 54 percent bigger than unmarked commits and revisited 55 percent more.

Claude-marked vs unmarked commits in the same codebase and months, median difference with 95% range



27 codebases with at least 20 Claude-marked and 20 unmarked commits in the same months. Bots excluded; follow-ups counted within 30 days.
Hollow marker = not statistically clear: the 95% range includes zero or a sign test gives p > 0.05.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

Claude Code also records which model wrote each commit. In the 13 codebases that used both, Opus 5 commits were 26 percent less likely than Opus 4 commits to add a function with more than 10 decision points (lower in 10 of 12 codebases) and 28 percent less likely to exceed 500 lines (lower in 11 of 13). Lines per commit and follow-ups also fell, but not clearly, and branching per line didn't clearly change. Because Opus 5 is newer, teams may also have learned to scope changes better in the meantime, and the data can't separate the two effects.

KEY POINT

Why we don't compare tools

Cursor, Devin, Copilot and Codex each appear in too few codebases for a fair comparison. Almost all of Devin's commits are in two codebases and almost all of Codex's are in one, so their numbers would describe those codebases rather than the tools. Claude's 69 percent share also reflects marking: Claude Code adds a co-author line by default, so its commits are easier to see than those of other tools.

On the current trend, complexity per 100 new lines will rise another 42 percent by late 2028

Complexity rarely breaks anything on the day it is written. It slows down the next change, and the one after that. It is felt when something forces a broad change, such as a security patch across many services, a framework upgrade or an incident in code that few people fully understand.

KEY POINT

A quiet codebase is not necessarily a safe one

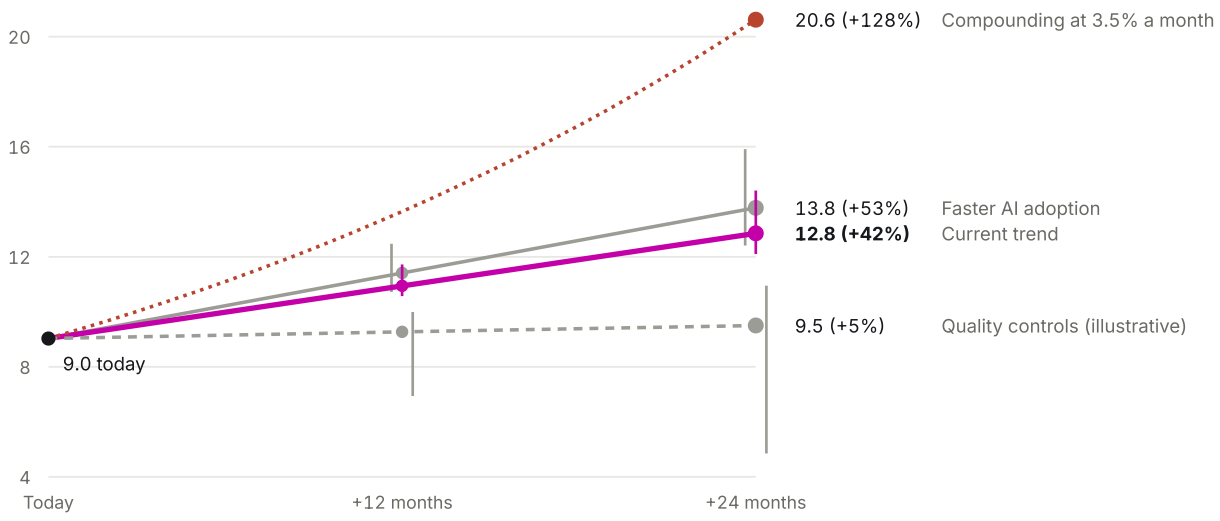
Bug fixes and reverts catch problems that appear within weeks. The cost of complexity comes later, in how long each subsequent change takes. We also can't yet see whether the new tests cover the new branches, because the data has no test coverage or test line counts. That is the main question for the next edition.

These projections are conservative. They assume complexity grows by a fixed amount each month. In practice, the median across all codebases has compounded at 3.5 percent a month since January 2025 (Exhibit 1), and each codebase's own growth after adoption was similar. If that pace held, complexity per 100 new lines would more than double in two years, well above even the faster-adoption line (Exhibit 12).

Exhibit 12

On the current trend, complexity rises 42 percent in two years. If growth keeps compounding, it more than doubles.

Projected decision points per 100 new lines, median codebase, with 95% range at 12 and 24 months



"Today" is the trend level (9.0), not the latest single month (10.3, Exhibit 1). Dotted red line: compounding at 3.5% a month, no range shown. Other lines: straight-line projections of each scenario's median monthly slope across 39 codebases. Faster adoption = the 20 codebases with the highest AI share. Quality controls (illustrative) = the 10 codebases with the flattest trends; no quality intervention was observed.
Source: Forewit Codebase Analysis of 46 open-source codebases, October 2024 to October 2026

Forewit

What engineering teams can do

Action	Why	Effort
Track decision points per 100 lines alongside DORA metrics	Complexity rises before delivery metrics move	Low: most static-analysis tools already report it, or Forewit's free scanner can
Check AI-written code against the running system as it is written	AI-marked commits are revisited 43% more often	Medium: tooling and pipeline changes
Set a size limit for AI-assisted changes	Commits got 63% bigger, and AI-marked commits are a further 42% bigger	Low: a team policy and a pipeline check
Measure your own trend before judging AI	Unmarked codebases got more complex too	Low, given 8 or more months of history
Mark AI-assisted commits	Only 9% of commits are marked in the typical codebase, so AI use is undercounted	Low: a tool setting

About the research

We ran Forewit's Codebase Analysis tool on the full commit history of 46 well-known open-source codebases from October 2024 to October 2026, with a second pass on complexity going back to October 2022. Most are commercial open-core products, and 35 are written mainly in TypeScript, so the results describe that kind of codebase. We chose the 46 by hand based on their size and popularity, so they are not a random sample of open-source projects. The data and analysis scripts behind every number in this report are public at github.com/forewitio/forewit-impact-ai.

Commits	201,576
Authors	6,847
Codebases	46 (30 with an AI adoption date, 16 without)
Window	October 2024 to October 2026

How we identified AI use. A commit counts as AI-written only when it carries an agent's mark: an agent in the author field, a co-author line or a “generated with” note. A codebase's adoption date is the month in which AI-marked commits pass 25 percent of the total, or step up and stay up. Most tools don't mark commits by default, so AI use is undercounted.

How we compared. Each codebase is compared with itself, before and after its own adoption date, and counts once. We report the median codebase. We call a result a finding only when it holds across most codebases and is statistically clear, and we say so when it isn't. We tested 16 before-and-after metrics (Appendix B) without correcting for multiple comparisons, so at this threshold about one of them could pass by chance. Percent changes are the median of each codebase's own change, so they can differ from the change between the two medians shown.

What the data can't show. This is observational data. “After adoption” means after the adoption date, not “because of AI”. The data has no information on incidents, defects, test coverage or pull-request lead times.

kerno

ABOUT FOREWIT

The future will be built on code no human wrote. Forewit makes sure you can trust it.

Forewit is an agentic QA platform. It runs in every agent session to stop agents from duplicating code that already exists, catch runtime issues before they reach a PR, and give developers full proof of what the agent did.

Less rework. Fewer tokens. Higher-quality code.

Know what your code is becoming

Benchmark your codebase against this report. Run Forewit's free CLI codebase audit on your own repository. You'll see your complexity trend, commit size and rework rates.

useforewit.com

© Forewit 2026 · Data: Forewit Codebase Analysis, 46 open-source codebases

Appendix

A. Definitions

Term	Plain meaning	How it's measured
Decision point	A place where code can go two ways: `if`, `else if`, loop, `case`, `&&`, `||`	Cyclomatic complexity (McCabe, 1976): decision points + 1 per function. In TypeScript and JavaScript, ternaries (`? :`), `??` and `catch` also count; optional chaining (`?.`) does not
Decision points per 100 lines	How much branching new code brings, independent of commit size	Decision points a commit adds ÷ production lines it adds × 100, summed over each period
Excess complexity	Branching in functions too complex to test easily	For each function above 10 decision points, the amount above 10, per 1,000 lines of the whole codebase
Maintainability grade	A to D rating of excess complexity	Quarters of the 39 codebases with a trend at the start (cut-offs 4.1, 7.2, 9.6); same cut-offs at the end
Rework	New code rewritten or deleted soon after it's written	Share of lines from sampled commits changed within 30 or 90 days
Follow-up	Any later commit that comes back to the same code	Count of later commits touching a change's lines, per change
Repair	A follow-up that fixes the earlier change	Judged by an AI model; verdicts below 0.5 confidence dropped
Extension	A follow-up that builds on the earlier change	Same as repair
Reimplementation	A new function that duplicates one the codebase already had	Matched by name, then logic; judged by an AI model
Engineer-week	One person with at least one commit in a given week	Bots excluded
Commit type	Bug fix, test change, revert, feature	Read from the commit's subject line; 70–88% accurate on 100 checked by hand

B. Every metric, before vs after AI adoption

Metric	Median change	95% range	Rose / fell (of 30)	Clear?
Decision points per 100 new lines	+51.5%	+32% to +72%	27 / 2	Yes
Excess complexity per 1,000 lines	+14%	+2% to +27%	24 / 6	Yes
Functions over 10 decision points added per 1,000 lines	+38%	+21% to +78%	25 / 5	Yes
Share of commits adding a function over 10 decision points	+62%	+45% to +90%	25 / 5	Yes
Lines per commit	+63%	+40% to +98%	26 / 2	Yes
Share of commits over 500 production lines	+34%	+8% to +79%	23 / 7	Yes
Production lines per engineer-week	+71%	+26% to +116%	23 / 7	Yes
Active contributors per week	+10%	−19% to +28%	16 / 14	No
Lines rewritten within 30 days	−16%	−32% to −6%	8 / 22	Yes
Lines rewritten within 90 days	−26.5%	−34% to −20%	5 / 24	Yes
Follow-ups per change	+7.5%	−5% to +41%	18 / 12	No
Share of follow-ups that are repairs	−23%	−40% to −7%	7 / 21	Yes
Reimplementations per 1,000 lines	−26.5%	−40% to −3%	7 / 23	Yes
Share of commits that are bug fixes	−4%	−16% to +6%	13 / 17	No
Share of commits that change tests	+81%	+41% to +127%	25 / 5	Yes
Share of commits that are reverts	−50%	−61% to −21%	6 / 17	Yes

“Clear” means the 95% range excludes zero and a sign test gives $p \le 0.05$. Percent changes are calculated as described under How we compared.

C. Method notes

Monthly growth rates come from a straight line fitted to the logarithm of each codebase's monthly values. Months with fewer than 1,000 production lines are left out.

The step-up in complexity growth, from +0.7 to +3.5 percent a month, is not statistically clear on the two-year data alone (range −2.1 to +5.8 points). The four-year re-run makes it clear. In each codebase, growth went from −0.4 percent a month (October 2022 to September 2024) to +3.1 percent (January 2025 to October 2026), a step of +3.3 points (95% range +1.7 to +5.3), and was faster in 31 of 39 codebases ($p < 0.001$). The 3.5 percent a month used in the forecast is the growth of the median across all codebases since January 2025, while 3.1 percent is the median of each codebase's own growth.

Four-year re-run: the same 46 codebases from October 2022 to October 2026, covering complexity and attribution only, with adoption months taken from the main run. Monthly medians leave out months with under 1,000 production lines. The main before-and-after result reproduces: 5.2 to 8.2 decision points per 100 lines, up in 28 of 30 codebases. Leaving out any single codebase moves the October 2026 median by at most 0.03. Leaving out the three biggest risers (Vendure, Strapi and Directus) lowers it from 10.3 to 9.7.

Fixed set of codebases: on the two-year data, the 24 codebases present in at least 20 of 24 months grew 3.1 percent a month from January 2025, compared with 3.2 percent for all codebases. The 3.5 percent quoted elsewhere comes from the four-year re-run, a separate run with slightly different monthly values.

Recent commits and 90-day measures: in the median codebase, 42 percent of after-adoption commits fall in the window's final 90 days, so they had less time to be rewritten. In the 14 codebases where under 40 percent were that recent, 90-day rework still fell 29 percent (12 of 14, $p = 0.01$). Lines still in place after 90 days have the same exposure and were not re-run.

Moving the date: we applied the same adoption date to every codebase, for each month from December 2024 to July 2026. At all 20 dates, more codebases became more complex than less complex.

Forecast: today's median level (9.0) plus each scenario's median monthly slope, projected 12 and 24 months ahead. The range is a 95% bootstrap interval for the median slope. Straight lines fitted to 8 to 30 monthly points are a weak method, so read the ranges rather than the lines. The straight line is deliberately conservative. Compounding at the after-adoption rate of 3.5 percent a month ($1.035^{24} \approx 2.3\times$) would give a far steeper rise, but that rate rests on only about a year of data per codebase.

By tool and model: each tool's or model's commits are compared with unmarked commits in the same codebase and the same months, and codebases need at least 20 commits in each group. Follow-ups are counted within 30 days, and commits from the last 30 days are left out. Model names come from the Claude Code co-author line, because other tools don't record one. The Opus 4 vs Opus 5 comparison measures each model against its own human baseline in the 13 codebases that used both. The detection rule counted two engineers named Devin, at Prefect and Mattermost, as the Devin agent, so their roughly 350 commits are excluded.

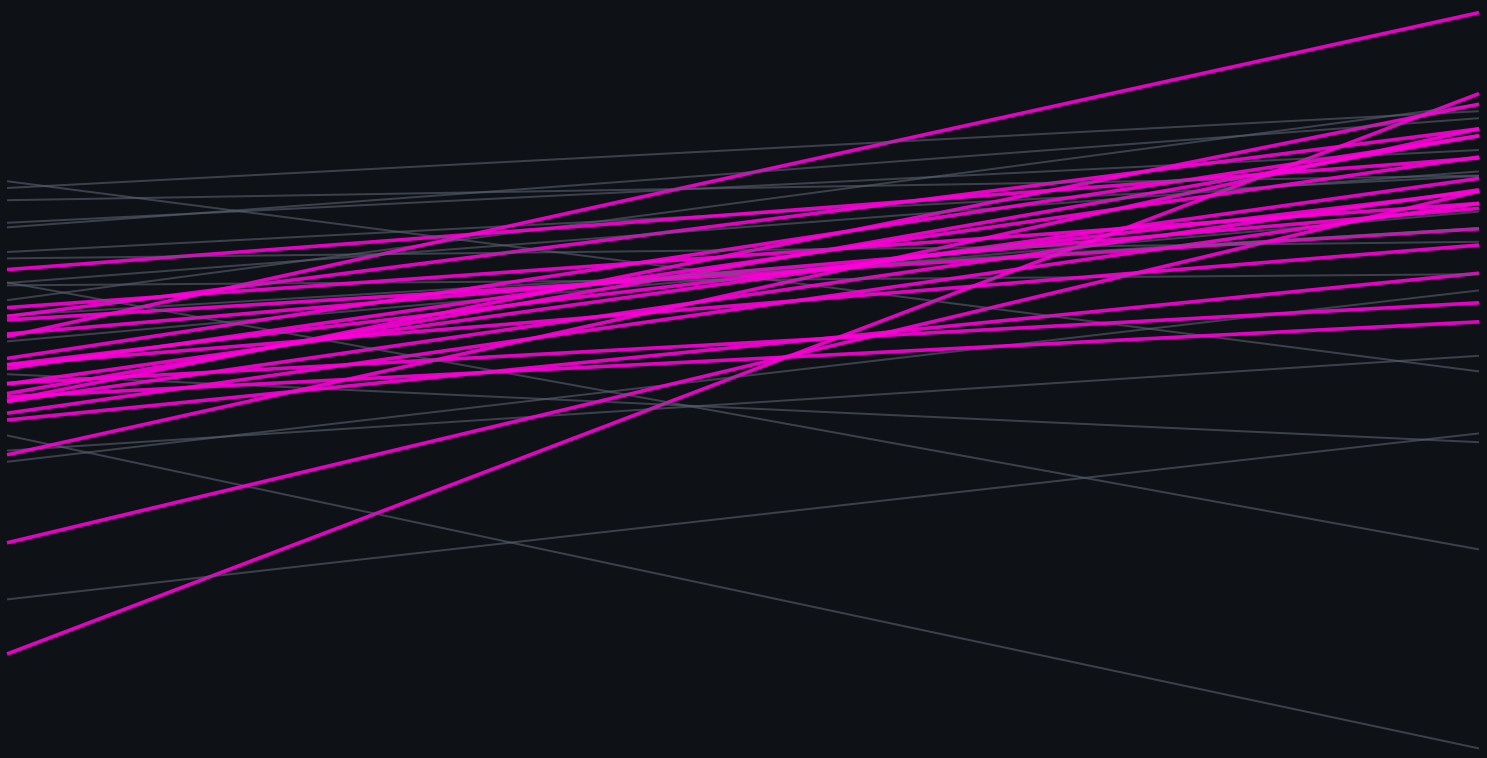
Data quality: bots (7 percent of commits) are excluded from complexity and output measures. Lockfiles and vendored code are excluded, and generated code makes up 0.1 percent of file changes. Vue, Handlebars and ERB files aren't parsed, and 8 codebases have under 90 percent parser coverage (the lowest is 58 percent).

D. Raw data and references

Every codebase's numbers, the raw benchmark output and the analysis scripts are available at github.com/forewitio/forewit-impact-ai.

1. Nagappan, N., Ball, T., Zeller, A. (2006). Mining metrics to predict component failures. ICSE 2006
2. McCabe, T. J. (1976). A Complexity Measure. IEEE Transactions on Software Engineering, SE-2(4)

Forewit



THE COMPLEXITY BILL

The Impact of AI on Code Quality, 2026

useforewit.com

© FOREWIT 2026 · DATA: FOREWIT CODEBASE ANALYSIS, 46 OPEN-SOURCE CODEBASES