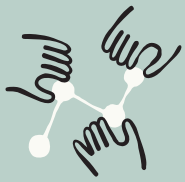

ANTHROP\C

Agentic coding and persistent returns to expertise

Published
June 16, 2026

Authors
Zoe Hitzig, Maxim Massenkoff, Eva Lyubich, Ryan Heller, and Peter McCrory

Acknowledgements

With acknowledgements to: Jake Eaton, Sarah Pollack, Hanah Ho, Szymon Sacher, Anton Korinek, Santi Ruiz, Kerry Persen, Ankur Rathi, Alex Tamkin, Heather Whitney, Cat Wu, Kacie Jenkins, Jennifer Martinez, Amie Rotherham, Boris Cherny, Eleanor Dorfman, Miles McCain, and Jack Clark.

Key findings

- Building on prior work, we introduce a framework for studying interactive agentic coding based on a privacy-preserving analysis of ~400,000 Claude Code sessions from between October 2025 and April 2026. We evaluate the composition of tasks, human-AI collaboration, and success rates.
- In a typical session, people make most of the planning decisions (what to do) and Claude makes most of the execution decisions (how to do it). The greater domain expertise a person brings to a session, the more work Claude does per instruction. On coding tasks, every major occupation succeeds—accomplishes what the person set out to do, with verifiable evidence like passing tests or committed work—at nearly the same rate as software engineers, on average.
- The more domain expertise a person has, the more often the session ends in success—though the gap between intermediate and expert users is modest. Over the seven months we observe, the share of sessions spent debugging fell by nearly half, and usage shifted toward more end-to-end agentic use: deploying and running code, analyzing data, and writing non-code documents.
- Over those seven months, the value of the typical task, which we estimate through a comparison to freelance job postings, rose in almost every kind of work, and about 25% on average.

1. Introduction

Agentic coding has taken off. The share of Github projects with coding agent activity has more than doubled since late 2025,[1] and Claude Code users now spend an average of 20 hours per week using the tool.[2] Can people without formal coding experience successfully direct an agent through complex technical work? And what will rapid adoption and improvement of these tools mean for knowledge work broadly? While we don't have full answers to these questions yet, we look to Claude Code usage data for early signals.

This report provides evidence on how Claude Code is used in practice, based on a privacy-preserving analysis of ~400,000 interactive sessions from ~235,000 people between October 2025 and April 2026. It builds on prior work focused on measures of autonomy in Claude Code sessions, and how Claude Code is changing work at Anthropic.[3] Here, we introduce a framework for describing

interactive AI coding-assistant usage: what kind of work is being done, who is doing it, and whether it succeeds. We focus on Claude Code usage through a command-line interface (CLI), claude.ai, or the Claude Code desktop app.[4] By tracking how agentic coding usage changes as models get more capable, we can better understand how these tools affect the labor market for coding professionals and knowledge workers.

What happens on Claude Code may be a preview of where knowledge work is headed, as agents become embedded in non-coding work. We find that Claude is handling more complex and more valuable tasks. At the same time, there remains a clear division of labor in agentic coding: People decide what to build, and the agent decides how to build it.

We also see evidence that domain expertise, and not coding proficiency, amplifies effective use of the tool. In particular, domain experts, succeed more often, and more easily recover from errors and misunderstandings. However, the gap between experts and intermediates is modest—suggesting that proficiency in a domain is enough to use the tool almost as effectively as those with deep mastery.

These findings give us an early read on possible transitions in the labor market. In our data, success is determined by how well a person understands the problem they are trying to solve, not whether they're trained in coding. If these patterns hold across the economy, it suggests that while agentic coding tools may be absorbing some implementation-heavy work, they are also rewarding those with firm understanding of the problems they solve on the job. Coding agents are not substituting for domain expertise—the more understanding a worker brings to an agent, the more quality work the agent is able to do.

2. The division of labor

2.1 What people use Claude Code for

To understand what people are using Claude Code for, we classify each session into one of nine **work modes**—the single activity that best describes what the

session is trying to accomplish.[5] Four modes involve writing or maintaining code directly: building something new, fixing something broken, testing code, and orchestrating other agents or automated pipelines. Another category is operating software—deploying, configuring, running pipelines, monitoring systems. Two categories are more about working out what to do: understanding how an existing system works, and planning a change before making it. And two take actions unrelated to code, or where code is incidental to the final product: analyzing data, and communicating via presentations and other prose-based documents.

About 56% of sessions consist of writing (25%) fixing (26%), or testing and orchestrating code 5%. Operating software comprises 17%, while 14% of sessions are planning or exploring, and 13% produce analysis or prose (Figure 1).

Most sessions are anchored to an existing codebase: 48% primarily modify existing code and another 17% explore code, while 14% create new code from scratch. Roughly a fifth of sessions touch no codebase at all.

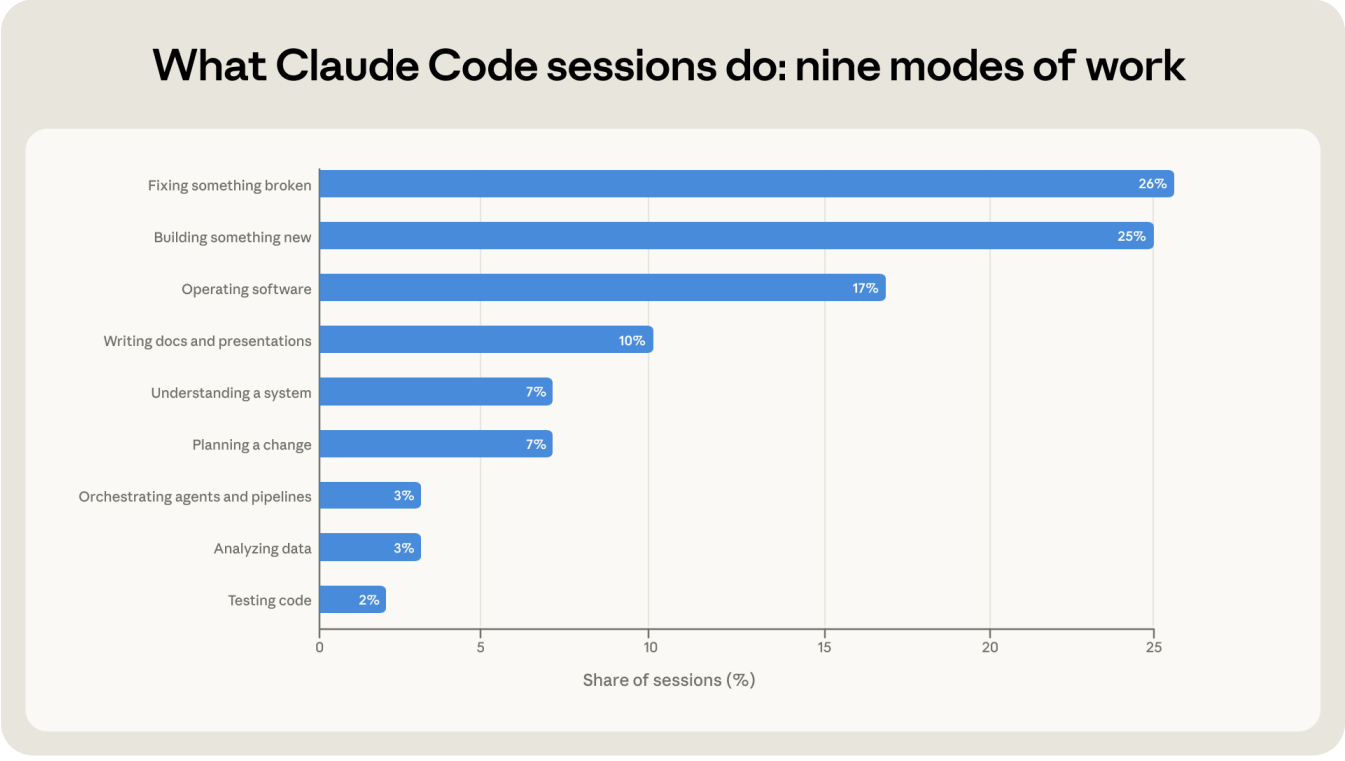

Figure 1: The nine modes of work
Each interactive session is classified into the single mode that best describes what it is trying to accomplish.

We classify each session by having a model read its transcript, then using our privacy-preserving analysis tool, we check them against telemetry that's recorded automatically for every session, including whether any lines of code were added or deleted. The two sources have high agreement for instance, more than 90% of sessions our classifier labeled as creating or modifying code showed code changes in the telemetry. See the Appendix for details.

2.2 Who decides what

How autonomous is Claude Code? Capability evaluations suggest the ceiling is high and rising: on benchmarks such as METR's time-horizon evaluations, frontier models can now complete software tasks that would take a person hours, autonomously working through obstacles along the way. But what does usage actually look like in practice? Here, we look at how much steering is done by the user and by Claude in real sessions.

We investigate this question from two angles. First, we focus on the extent to which people are entrusting decisions to Claude, and second we look at how many actions they give to Claude. To understand the division of decision-

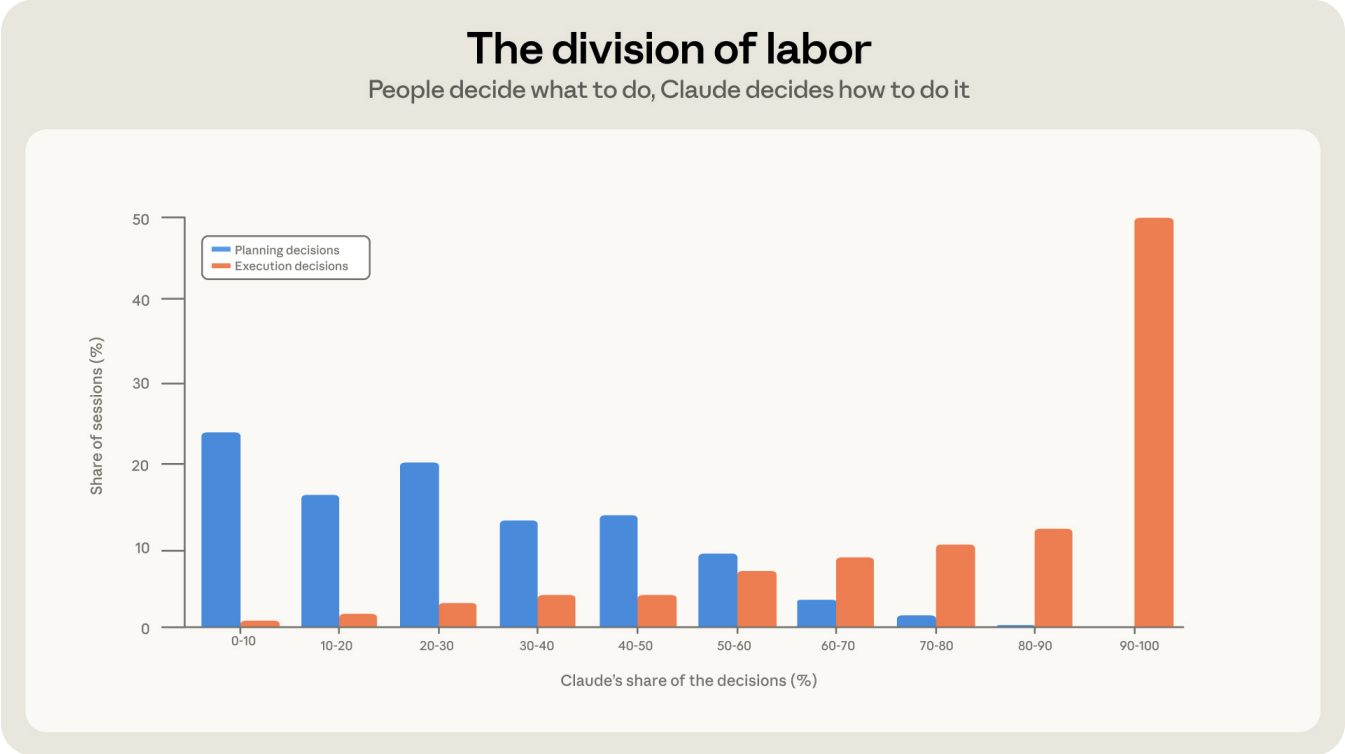

Figure 2: Claude's share of planning and execution decisions
Distribution across sessions of the share of planning decisions (what to do) and execution decisions (how to do it) attributed to Claude rather than the user. In the typical session, the user makes about 70% of planning decisions while Claude makes about 80% of execution decisions.

making in a session, we build a privacy-preserving decision attribution classifier based on the content of a session. We ask a classifier to list all the meaningful decisions in a session. We separate these decisions into planning (what to do, which approach to take, what counts as done) and execution (which files to change, what code to write, what language to write in, which commands to run). The classifier then attributes each decision to Claude or to the user, giving every session two numbers: the user's share of planning decisions and the user's share of execution decisions.

On average, people make about 70% of the planning decisions but only 20% of the execution decisions (Figure 2). In practice, there is a clear division of labor in agentic coding—people decide what to build, and the agent decides how to build it.

To understand the delegation of actions in a session, we look at the session's structure instead of its content. A Claude Code session involves Claude and the user going back and forth trading prompts (from the user) and actions (taken by Claude)—the user writes a prompt and Claude goes off and does some work, and then the user writes another prompt, and so forth. In a typical session, there are about 4 such turns. In our historical data from October to April, each prompt the user sends sets off a chain of around 10 actions taken by Claude on average—and sometimes over a hundred.[6] In each turn, Claude reads files, edits code, runs commands, and writes on average 2,400 words of output.

How much Claude does between check-ins largely tracks who is making the decisions. When the user keeps control of execution (i.e., makes over 80% of execution decisions), Claude takes fewer actions per turn (about 8 actions). And when Claude takes control of planning (i.e., makes over 80% of planning decisions), it takes on the highest number of actions (about 16).

2.3 Level of expertise

From each transcript, Claude rates the user's apparent expertise at the task on a five-point scale from novice to expert. The expertise classifier looks for three signals: how precisely the user frames their directions, what they ask Claude to verify, and whether the user tends to correct Claude or Claude tends to correct the user. Note that expertise is capturing something quite different from job title or general ability, and, crucially, it is *task-specific.* A senior engineer asking their first Rust question is a beginner at Rust. An accountant who has never

Expertise	What the classifier looks for	Representative requests across a conversation
1 (Novice)	User requests have no domain-specific nomenclature. Verification requests, if any, are generic (e.g. "double check this"). User doesn't recognize Claude's errors.	**1st prompt:** Can you analyze this data and make a chart? **3rd prompt:** Can you also make it show the trend over time? **6th prompt:** That's not what I expected please double-check what you did.
2 (Beginner)	User requests have some domain terminology. Verification requests are untargeted. User pushes back only on obvious errors.	**1st prompt:** What is big query **2nd prompt:** Can you help me do a small run? walk me through it **5th prompt:** How can I tell if I have approval **12th prompt:** Wait did you use the exact specification my teammate did?
3 (Intermediate)	User frames requests with some domain specificity, but does not engage deeply on methodology or tradeoffs. User asks for some non-generic checks, and may notice Claude's errors.	**1st prompt:** can you check whether this branch is ok to merge? **7th prompt:** should we do seperate fetchers for each part of the page and then wouldn''t that optimize caching for each section? like we could cache basic details more than say performance data? **19th prompt:** ok all good so far – where are we on caching? do you think the changes you made will bring down egress on `[database provider]` ? resolve whatever's left from my changes please
4 (Advanced)	User exhibits domain knowledge and anticipates some tradeoffs unprompted. Verification requests are targeted. User catches at least one of Claude's domain mistakes.	**2nd prompt:** what's the right way to test this stage before going to stage 3? ? **5th prompt:** wait how's an agent console different from normal chat? I'm pretty sure the only way to talk to an agent at all is through this session console view **88th prompt:** it looks like the parsing fix didn't work – line count of the file is still 742 : `wc -l [file name]` **106th prompt:** instead of regex is there a better / more bullet proof way to pull out user turns i.e. like key off the record field when prsing the jsonl
5 (Expert)	User employs sophisticated domain-specific jargon and anticipates intricate tradeoffs and design decisions. Verification is precise, targeting weak points. User corrects Claude, Claude almost never corrects the user.	**1st prompt:** I need to dig into the the issue `[user]` reported here: `[url]` note that the fix we did in the last release's PR wasn't enough. any other ideas? **3rd prompt:** once the code is cleared it shouldn't be returning `****` **64th prompt:** yeah okay. should also note that we might need to break hard refresh down further by managed vs unmanaged slots e.g. managed ones could refresh every 30 min but rhe rest once a day 108th prompt: should we do retries instead of best effort? sync needs to reliably know what's on the lock. Remember the original bug where the valuedb was stale and it created a loop trying to set the pin over and over. retries aren't necessarily the best solution but neither is best effort

Table 1: Expertise classifier The examples paraphrase, anonymize and condense real sessions labeled by our classifiers. Many of the sessions used in the table come from a public dataset of agentic coding sessions,

used Python, but tells Claude exactly which reconciliation rules a Python script must enforce and catches the edge case it mishandles at month-end close, is an expert at that task.

The table below shows how we defined each expertise level in the classifier

along with an example request from a public dataset of coding agent sessions, SWE-chat. The conversation categorized as Novice gives generic instructions with no implied domain-specific knowledge. The Expert conversation conveys deep knowledge of the codebase and technical environment.

We quantify how expertise relates to Claude's output and activity per prompt. In typical novice sessions, each prompt sets off about 5 Claude actions and roughly 600 words of output, while expert sessions set off action chains more than twice as long (12 actions) carrying five times the output (3200 words) (Figure 3). This gap between novice and expert sessions appears within every kind of work and every band of task value.

These measures complement the autonomy measures in our prior report on Claude Code. Those measures tracked how long the agent runs and how often people approve its actions automatically. Our decision attribution measure, by contrast, captures who makes the substantive decisions in a session as a whole while our measures of output and actions per prompt measure how much autonomous activity from Claude human directions yield.

3. Who uses Claude Code, and for what

3.1 The users

To understand who is doing this work, we infer each user's occupation from the session transcript, mapping it to one of 23 major groups in the Bureau of Labor Statistics' Standard Occupational Classification (SOC) taxonomy. The classifier is instructed to rely only on signals such as the project context the agent loads at the start of a session, the names and structure of their files, any artifacts they reference (i.e., legal filings, clinical data, financial reports, a curriculum, etc.) and vocabulary they use.[7] It is explicitly instructed not to treat the act of coding as evidence of a coding profession. A session is classified into the coding SOC code (Computer and Mathematical Occupations) only when there is clear signal that software or data work is the user's job. A session in which a lawyer builds a script to automatically flag missing clauses across a folder of contracts is mapped into Legal Occupations, even if the session's work is primarily software. The session is left unclassified when there is no signal about the user's occupation.

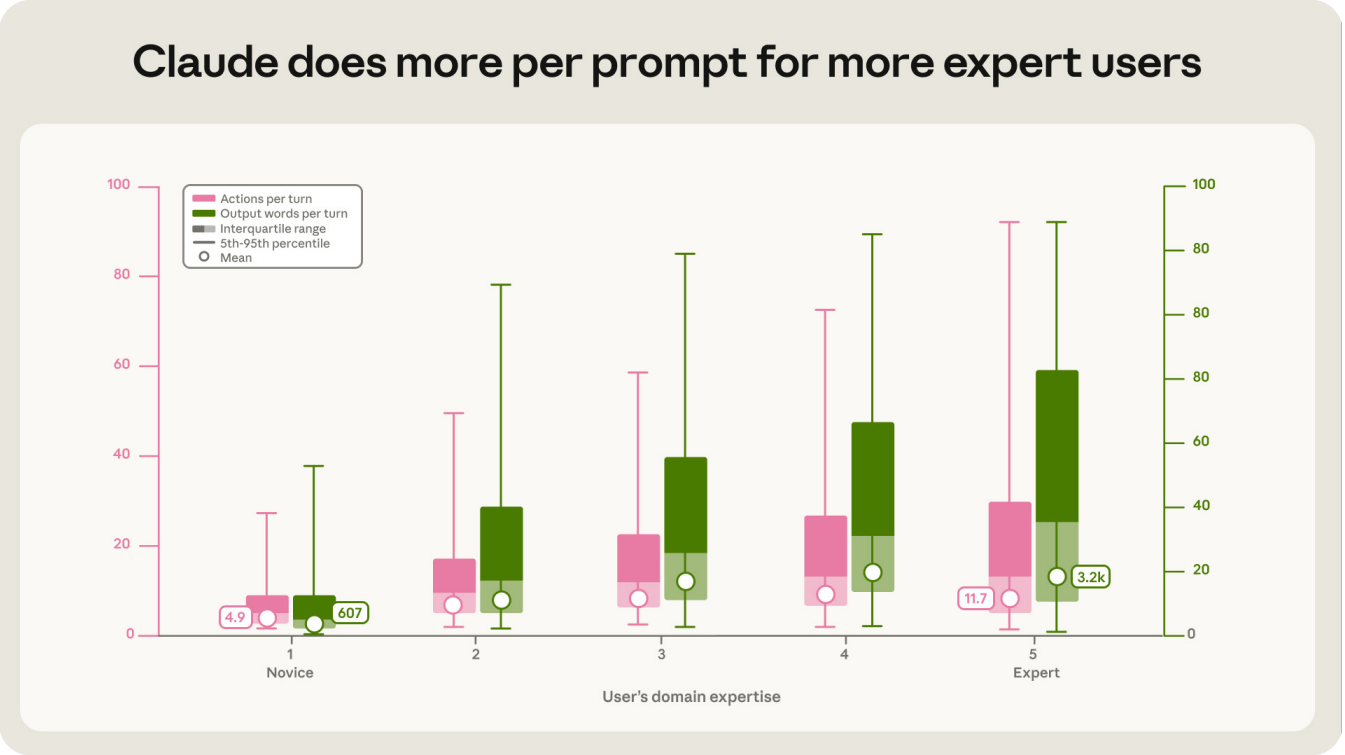

Figure 3: Claude does more per prompt for more expert users
Claude produces more actions (left bar) and text output per prompt (right bar) for more expert users. Boxes span the interquartile range (split at the median). Whiskers represent the 5th to 95th percentile. White dots are geometric means. Both upward trends are statistically significant ($p < 0.001$), as is each adjacent-level step, and they remain significant (at +9% actions and +13% output per expertise level) in a regression controlling for work mode, task value, month, occupation, and model family, with standard errors clustered by user.

We were able to infer occupation in about 70% of sessions. Within this set, Computer and Mathematical Occupations, a category which encompasses most software-related jobs, is unsurprisingly the largest group. The next largest are Business and Financial Operations, Arts, Design and Media, Management, and Life, Physical, and Social Sciences. The fastest-growing non-software occupation groups in our sample are management, sales, and legal occupations.

3.2 The work

The composition of the work done with Claude Code changed substantially between October 2025 and April 2026. The clearest change is that the share of sessions spent fixing broken code fell from 33% to 19% (Figure 4). In its place, we saw a greater share of the work that surrounds code. Operating software grew from 14% to 21% of sessions. Writing and data analysis roughly doubled, from about 10% to 20% of sessions.

The tasks themselves also grew more valuable. We approximate each

session's economic value by asking what the work would cost on a freelance marketplace, calibrated against a public dataset of real postings. By this measure, the estimated value of the average session rose by 27% between October and April. The rise holds across many kinds of work. Building, operating, and fixing tasks all grew more valuable by roughly a third or more (about 43%, 34%, and 32% respectively). These price estimates are coarse, so we use them primarily to compare tasks to one another over time, not as dollar values to be read literally.[8] For details about the construction of the task estimator, see the Appendix.

4. Success depends on what the people brings

The estimated value of a task is one way to get a sense of how Claude Code is helping people do their work. Another angle is to look at how many sessions are successful, and what characteristics of a session are linked to success. Across all our measures of success, we see a clear pattern: the more expertise a person exhibits in a session, the higher the likelihood of success. Most of the gain is concentrated at the lower end of the expertise scale—the gap between novice sessions and intermediate sessions is bigger than the gap between intermediate and expert.

Before turning to the characteristics of successful sessions, we should be precise about how we measure success. We do not observe users' real-world outcomes, and we cannot ask them directly whether they got what they wanted out of Claude. Instead, we rely on two complementary transcript-based measures. The first, *judged success*, comes from a classifier that reads the full transcript and decides whether the person succeeded in doing what they set out to do (with options: succeeded, partially succeeded, failed, no clear goal). Two companion classifiers then rate the strength of the evidence for that judgment to determine verified success. A success signal classifier looks for verifiable evidence of success. In particular, it looks for git activity like commits and pull requests matching the work, as well as test suites passing, and explicit affirmation from the user. It scores the session from "no signal" to "weak signal" (1) to "multiple hard signals" (5). A parallel failure signal scores the evidence that things went wrong—errors, failed tests, retries, the user pushing back on the output. Verified success requires both that the session is judged successful and there is at least one hard verifiable signal of success. For the following

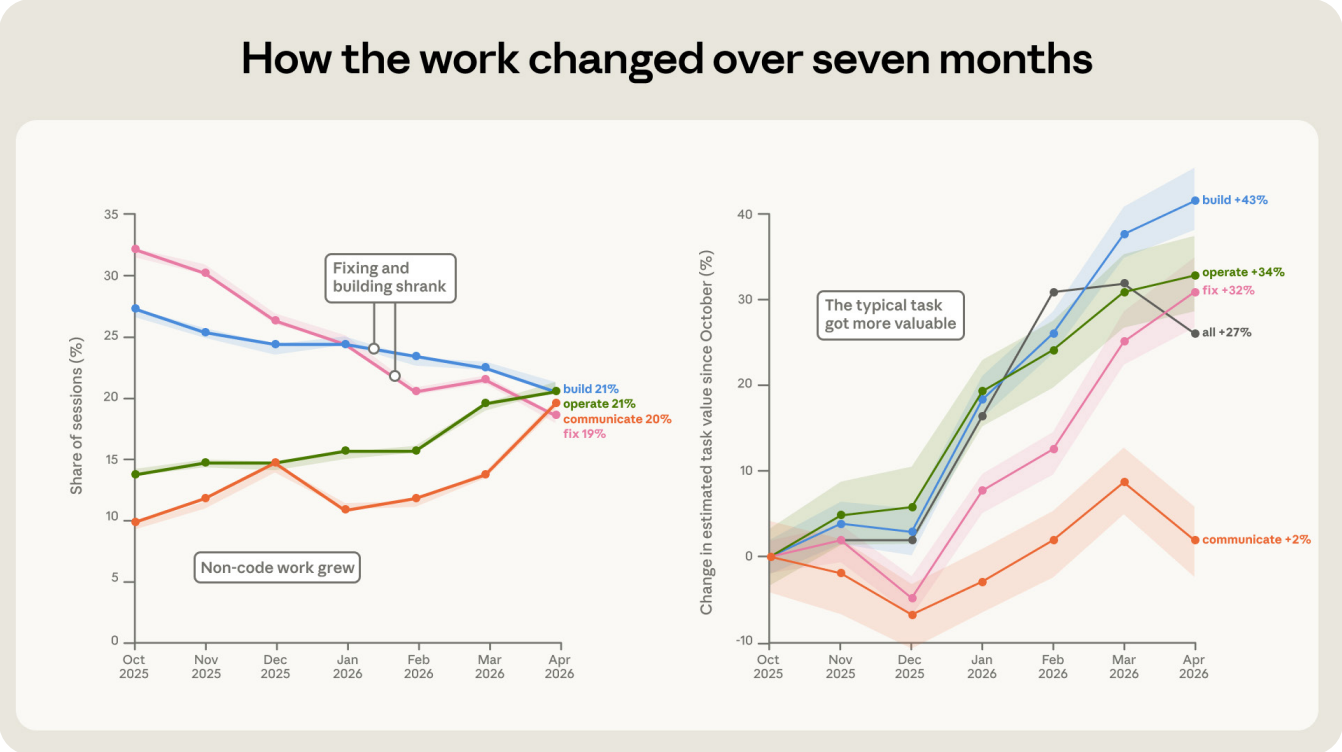

Figure 4: The composition and value of Claude Code work, October 2025 to April 2026
Share of sessions in each work mode over the seven-month window. The share of sessions fixing broken code fell from 33% to 19%, while operating software, analyzing data, and writing documents grew.

analysis, which is focused on the degree of success or failure in a session, we exclude sessions classified as having "no clear goal," which comprise about 7.7% of our full sample.

4.1 The returns to expertise

So what kinds of sessions are most successful? It turns out that the expertise rating of a session, described above, matters a great deal for the success of a session.

One might worry that expertise isn't the real driver—perhaps experts simply pick different tasks, or differ in other ways. Throughout this section, we partially address this worry by comparing sessions doing the same kind of work, at the same estimated value, in the same month, on the same subject, from people in the same broad occupation group, and ask how outcomes differ by the person's rated expertise.

Across all of our success measures, the more expertise a person exhibits in a session, the more likely it is that the session succeeds. A novice-rated session

Measure	Definition	Example session and its outcome Paraphrased and summarized examples from a public dataset of agentic coding sessions
At least partial success	The session partially completes the user's main goal `outcome = {success, partial success}`	**Paraphrased request:** "Set up automatic deploys of the docs site to GitHub Pages, with a subfolder for each open PR." **Outcome:** The workflow ships and runs. The session ends with link fixes applied but unconfirmed. Progress made but not finished.
Judged success	The session completes the user's main goal `outcome = {success}`	**Paraphrased request:** "Write an implementation plan for separating the featured-image field from the OpenGraph image." **Outcome:** Claude drafts the plan, the user says "yes," the plan file is written, and the session ends. Nothing is committed or tested.
Verified success	Judged success plus hard evidence of success `outcome = success AND success signal = {4,5}`	**Paraphrased request:** "Fix the README's deployment section — we use Docker and Render now, not GitHub Actions." **Outcome:** Claude edits the file, the user requests that Claude commit and push the edits. The push lands on main.
Judged failure	The session does not complete the user's main goal `outcome = failure = NOT {success, partial success}`	**Paraphrased request:** "Add tests for the single-sample edge case—let's do it TDD-style." — Claude asks one clarifying question; the user leaves without answering and nothing is written. No error and no complaint.
Verified failure	Judged failure plus hard evidence of failure `outcome = failure AND failure signal = {4,5}`	**Paraphrased request:** "Wire the new search endpoint into the router." **Outcome:** The build fails on an undefined import. Three retries don't fix it. The user types "nevermind" and leaves.
Abandoned	Judged failure and zero lines of code added `outcome = failure AND lines of code added = 0`	**Paraphrased request:** "Only the first checkpoint in each date group loads —the rest just spin." **Outcome:** The user reports two more errors over three messages. Claude never produces a response. The session ends with zero lines written.

Table 2: Definitions of success and failure derived from classifiers
The examples paraphrase and summarize real sessions from a public dataset of agentic coding interactions, SWE-chat, labeled by our classifiers.

reaches our strictest measure, verified success, 15% of the time and at least partial success 77% of the time. A session rated intermediate or up reaches verified success 28-33% of the time and partial success 91-92% of the time (Figure 5).

In each measure, most of the gain comes from moving between novice to intermediate; between intermediate and expert, the slope decreases. In the Appendix, we give details about the regressions behind Figure 5. A similar gradient appears in sessions that run into challenges along the way. We say a session hits trouble when the failure signal records verified evidence of failure.

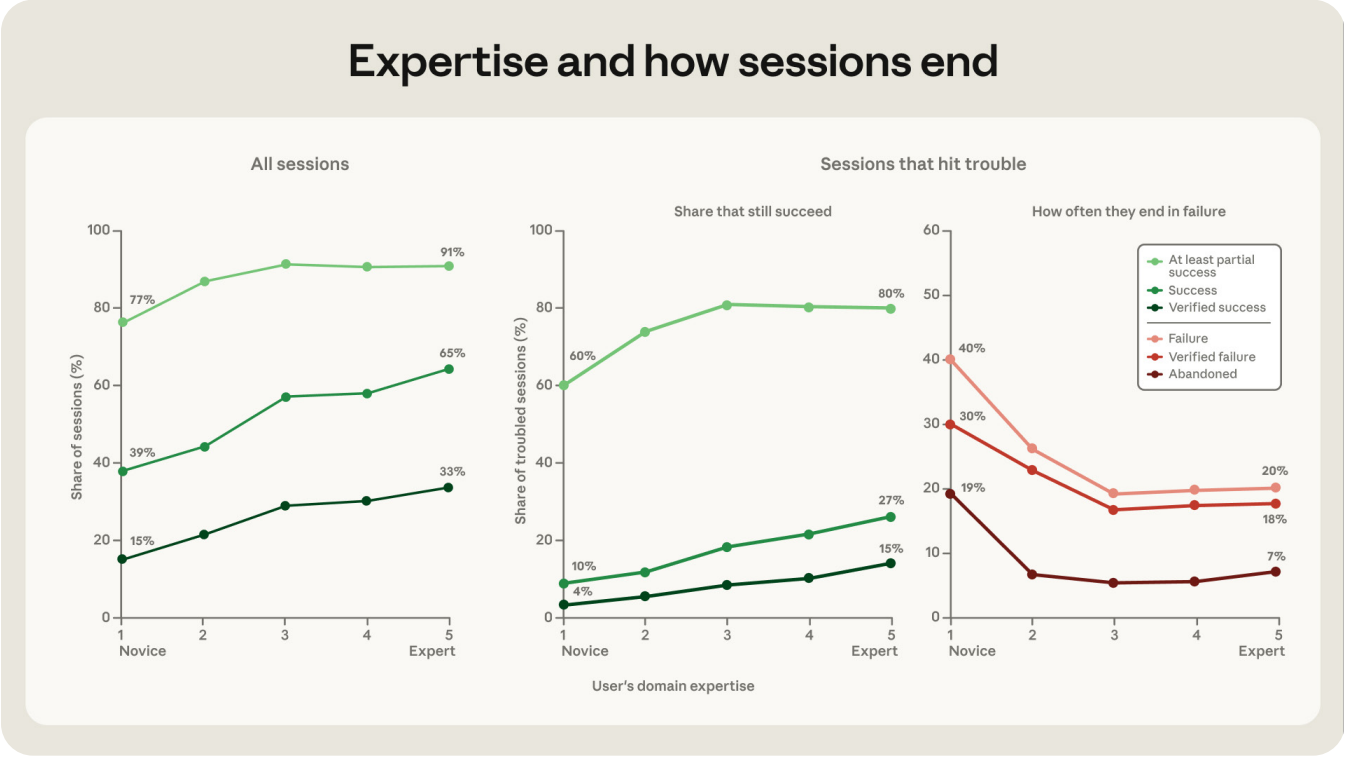

Figure 5: Expertise and how sessions end
Session outcomes by the user's rated expertise at the task, on a five-point scale from novice to expert. The left panel shows the share of all sessions judged succeeded or partially succeeded, those judged succeeded and those reaching verified success. The middle and right panels restrict to sessions that hit trouble (failure signals ⩾ 3) and show the share that still end in various definitions of success and the share that meet various definitions of failure, respectively. Each point is an adjusted rate—we estimate the differences between expertise levels by comparing only sessions that share the same work mode, the same task-value band, the same month, the same task subject, and the same kind of user (software-related occupation or not). Details about the regressions behind these points are in the Appendix. Whiskers are confidence intervals on sample means (most are too small to be visible in this plot). These plots exclude sessions judged by the success outcome classifier to have no clear goal.

This could be an error, a failed test, multiple attempts to do the same thing, or the user expressing frustration or dissatisfaction. Among sessions that hit trouble, the share that are verified successes rises from 4% for novice-rated sessions to 15% for expert-rated ones, accounting for all the controls described above (Figure 5). Looking at the looser measures, we find that the share of at least partial success is 60% for novice and 80-81% for intermediate through expert sessions.

We also track the inverse relationship—expertise versus various measures of failure. Note that in this analysis, the sessions judged as failures are those that do not even partially succeed. We say a troubled session is *abandoned* if it is judged as failed *and* zero lines of code are written: 19% of sessions where the user appears to be a novice end abandoned, against 5-7% for everyone else. In other words, the least experienced users are more likely to give up when they

are struggling to get the outcome they are after. Part of the value of expertise appears to be the ability to steer the agent in the right direction.[9]

4.2 Occupation may matter less than expertise

Software engineers and users in other "computer and mathematical occupations" reach verified success in about 30% of their sessions overall, where users from other professions reach verified success about 26% of the time. Among sessions that produce code (i.e., sessions that add or modify at least one line of code), those numbers are 34% and 29% respectively. That five-point gap is small, and it has neither widened nor narrowed over seven months, even as the success rates in both groups increased. In code-producing sessions, every one of the ten largest occupations in our dataset lands within seven points of software engineers in terms of their success. Management occupations are highest on verified success, slightly above the software engineering occupations. Their higher verified success rates may reflect management skills that transfer to directing an agent. But they may also partly reflect our measurement: verification rests partially on explicit confirmation in the transcript, and managers may be more likely to communicate when they get what they ask for.[10]

Looking ahead

The results in this report offer an emerging picture of how agentic coding amplifies some forms of knowledge and skills, while substituting for others. In sessions that produce code, every major occupation succeeds at rates within a few points of those in software-related occupations. It appears that coding agents are making a coding background less relevant to successful programming.

At the same time, successful sessions are more likely to exhibit domain expertise. Sessions rated expert reach verified success more than twice as often as those rated novice, and when a session hits trouble, novices abandon the session at several times the rate of everyone else. The shape of the collaboration gives this picture more color—domain experts are able to direct Claude to do more work with each instruction they give. So, the ability to steer Claude toward success comes more from command of a domain than from the ability to write code. A person with such command, in any field, may now be

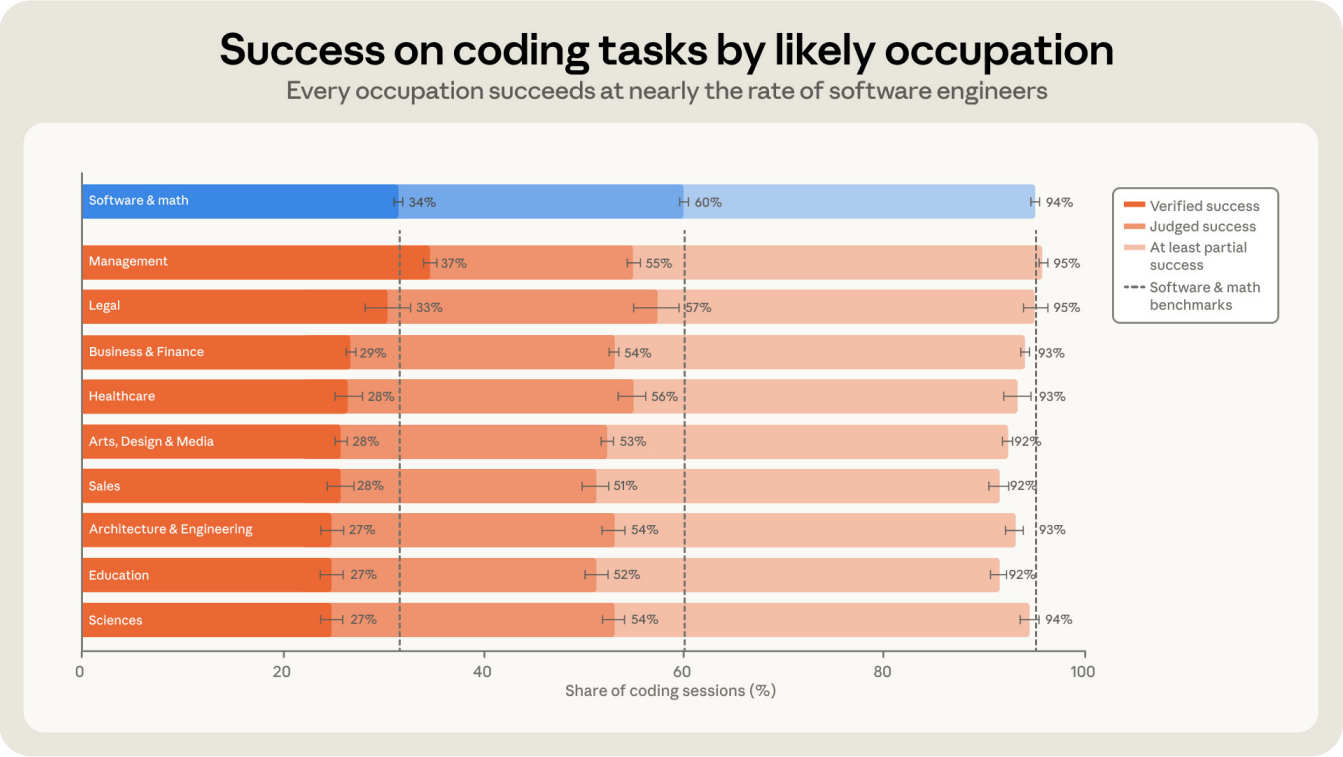

Figure 6: Success rates in coding sessions by inferred occupation
Share of sessions meeting strict definitions of success—judged success and verified success—among sessions that add or change at least one line of code, by the user's inferred occupational group, for the ten largest groups. Every group is within seven percentage points of software/math users (SOC Code Computer and Mathematical Occupations). Error bars are 95% confidence intervals computed on distinct accounts.

able to do technical work they previously could not. A person without any such expertise will get far less from the same tool. And the gains come mostly from competence, not mastery—a working grasp of the domain captures most of the benefit, while deep specialization adds only a bit more beyond that.

These findings are preliminary. As in most of our research, we cannot measure real-world outcomes, like whether code written in a session is actually used or discarded thereafter, or whether it produces an economically valuable artifact. In addition, the non-interactive usage this report excludes is a substantial share of activity. Developing a framework to measure it is a priority for future work. And all of our classifications of sessions depend on a model's reading of the transcript. In the Appendix, we show that our classifiers track independent telemetry in expected directions, and agree with a strong reference model on the majority of sessions. But classifiers remain challenging to validate at scale, and Claude Code sessions add further difficulty, as they may be too long and complex for human labels to serve as ground truth.

The picture in this report will be updated as the models, the users, and the division of labor between them change. We hope that these measures will allow us to track consequential shifts as they happen. For instance, if the returns to expertise begin to decrease over time, that would suggest that models are starting to supply the essential judgment that users currently bring, and that the gains from these tools are broadening beyond domain experts. If the share of coding sessions completed successfully by users outside software occupations continues to grow, it could indicate that software production is becoming a part of ordinary work in every field, rather than the product of a single occupation. These shifts would change who benefits from agentic coding, and by how much, and would have implications for what is most valued in the labor market.

And coding is a leading case—what happens in software is likely a preview of what may come as agentic tools take on other forms of knowledge work.

[1] A first study, covering 128,000 public repositories , detected coding-agent activity in an estimated 16-23% of projects as of the end of October 2025. A follow-up study using the same methodology found adoption rates more than twice as high among projects created after that period. Detection of agentic coding activity relies on agent co-authorship tags and configuration files, which likely undercount actual usage.

[2] Note that this measures hours in which Claude Code was actively running, not the user's hands-on time typing to Claude.

[3] In addition, Sarkar (2026) and Baumann et al. (2026) have offered lenses through which to understand agentic coding, by studying Cursor IDE sessions and publicly available sessions, respectively.

[4] Note that we exclude Claude Code usage that runs through third party integrated developer environments, and software development kits. We also therefore exclude sessions in "headless" mode where a user runs a single prompt in the CLI via `claude -p "<prompt>"`. We exclude this usage since it differs in two key ways—much of it is programmatic, with Claude Code embedded in automated tools and pipelines rather than conversing with a user, and even when a user is present, we do not see a user's session end-to-end the way we do on the surfaces we include.

[5] All classifiers in this report use Claude Sonnet 4.6 unless otherwise noted. Details about the classifiers, including their exact full text and validation results, can be found in the Appendix.

[6] The tail of actions per prompt is long. About 2% of sessions average more than 100 actions per prompt, about 1 in 270 average more than 200, and about 1 in 2,300 average more than 500.

[7] Like all measures in this report, these inferences are produced using our privacy-preserving analysis tool. No researcher reads individual transcripts, occupation labels are never linked to identifiable users, and we only observe aggregates over a minimum number of distinct users.

[8] The estimation approach we take here is intended to get at relative differences in the value of sessions, not absolute value. The dollar amount is based on

comparisons to the freelancer market—not salaried work—and comes from an ultimately fuzzy match between the Claude Code session and the job posting. Since the relative estimates will remove any consistent bias from these issues, we place more emphasis there.

[9] Conditioning on trouble selects different sessions for different users. Experts hit trouble less often overall, so the troubled sessions they do have are likely to be on harder problems—using the price estimate of the session as a proxy for the complexity of the session, we see that the average estimated value of a troubled session roughly doubles from the bottom of the expertise scale to the top. Part of the gap in recovery rates may therefore reflect that novices get stuck on routine problems while experts get stuck on challenging hard problems.

[10] Even if the model misclassifies managers, the signals relied upon to determine that the user is a likely manager—perhaps in how tasks are delegated and specified—tend to be associated with greater success. In other words, perhaps acting like a manager confers greater success.

BibTex Citation

```
@online{hitzig2026agentic,
  author = {Zoe Hitzig and Maxim Massenkoff and Eva Lyubich and Ryan
Heller and Peter McCrory},
  title = {Agentic coding and persistent returns to expertise},
  date = {2026-06-16},
  year = {2026},
  url = {https://www.anthropic.com/research/research/claude-code-
expertise},
}
```

Appendix

Available here