SWE-CoWork: Benchmarking Coding Agents as Digital Employees in Organizations

TokenWave AI[1]

A coding agent working within a software organization must establish what to build, collaborate with colleagues, and verify what it delivers. We introduce SWE-CoWork, a benchmark comprising 101 development assignments across 101 real repositories, 11 programming languages, and 12 application domains. Each assignment places an agent in a simulated engineering organization with 26 colleagues who have distinct responsibilities, bounded knowledge, and controlled collaboration behavior. Starting from a brief handoff, agents must recover requirements distributed across colleagues and workplace records, reconcile conflicting evidence, and implement changes evaluated by private executable tests.

We evaluate eleven models and find a substantial gap between partial implementation and complete task delivery. The two leading models satisfy approximately 63% of fail-to-pass (F2P) tests but complete fewer than 8% of assignments. Providing complete requirements upfront improves F2P by approximately 16 percentage points for Seed Evolving and GPT-6 Astra, yet leaves complete task success unchanged. Requirement-level trajectory analysis reveals failures throughout the delivery process: Claude-Opus-5.5 directs a mean of 93% of first questions about queried cards to their accountable owners across runs but obtains only 64% of colleague-held requirements. Deferred questions go unfollowed, implementations remain incomplete even after requirements are acquired, and 31% of requirements reported as completed fail their tests.

These findings highlight sustained requirement tracking, from clarification through implementation, verification, and handoff, as a key challenge for reliable autonomous software engineering.

Date: November 15, 2026
Website: https://tokenwave-ai.github.io/swe-cowork-page
Code: https://github.com/TokenWave-AI/swe-cowork-bench
Dataset: https://huggingface.co/datasets/Tokenwave-AI/swe-cowork
Authors: Full author list in Appendix A

1 Introduction

Real-world software development is inherently a collaborative human endeavor rather than an isolated coding exercise. On one hand, requirements originate from human stakeholders and evolve continuously: as intermediate milestones are reached or runtime traces surface, engineers and clients routinely refine their assumptions and introduce follow-up demands. On the other hand, professional software engineering cannot be accomplished from a single static specification alone; it requires close coordination across diverse roles, where asking the right clarifying questions is indispensable (Ko et al., 2007; LaToza et al., 2006; Sillito et al., 2008; Xia et al., 2018). Throughout this process, developers must navigate a sprawling web of project boards, versioned design documents, chat logs, merge requests, and CI histories—artifacts that frequently disagree with one another. A design document may describe an approach that has since been superseded; a passing CI run may correspond to an outdated revision; a colleague may know a local compatibility constraint without knowing which downstream component depends on it; and a trace uncovered during implementation may invalidate an initial diagnosis. For decades, software engineers have borne the heavy cognitive burden of

[1]Located in Palo Alto, United States.

checking evidence provenance, reconciling conflicting accounts, and aligning shifting requirements across the organization.

Recent advances in large language model (LLM) coding agents have achieved remarkable success on software engineering tasks, driving standard benchmarks from single-digit scores to near saturation within two years (Jimenez et al., 2024; The Anthropic Institute, 2026). In frontier industrial workflows, AI now accounts for the vast majority of code production: Anthropic reports that as of May 2026, over 80% of the code merged into its internal codebase was authored by Claude; in the second quarter of 2026, the typical engineer merged 8× as many lines of code per day as in 2024 (The Anthropic Institute, 2026). Yet scaling raw code generation does not resolve the broader organizational challenges of software delivery. Existing coding benchmarks (Jimenez et al., 2024; Huang et al., 2026; Raghavendra et al., 2026; Kondra et al., 2026; Cursor, 2026; Das et al., 2026) typically evaluate isolated, pre-scoped task descriptions. Studies of underspecification vary the information in a fixed initial request (Kim et al., 2026) or let agents query a designated simulated user who holds the missing requirements (Pu et al., 2026; Edwards and Schuster, 2026). Consequently, today's coding agents largely function as passive *"spec executors"*: they accelerate low-level implementation, but still require humans to curate organizational context, resolve conflicting historical records, and translate evolving stakeholder needs into self-contained specifications.

To further reduce human involvement in low-level execution and elevate human engineers to high-level direction setting, prioritization, and final acceptance, coding agents must evolve from passive *"spec executors"* into true *digital software engineers* possessing four core organizational capabilities:

- **Continuous:** Interacting with colleagues across familiar workplace interfaces (e.g., IM chat, project boards, and code reviews) while maintaining coherent context across multiple sources and evolving revisions—such as adapting execution when an early draft proposal is superseded in a later document version or when new runtime evidence arrives mid-implementation.
- **Proactive:** Operating without step-by-step human prompting from a brief handoff (e.g., "finish the 2026.09 upgrade; the previous task owner is on leave"), actively identifying and consulting accountable owners with concrete code context, while exercising restraint against scope distractors such as canceled cards or neighboring projects.
- **Efficient:** Communicating with colleagues autonomously within a bounded working budget and advancing multiple complex requirements in parallel—continuing productive work on other modules rather than blocking when a colleague requests a reproduction or defers an answer.
- **Persistent:** Remaining embedded in a realistic workplace environment over long execution horizons and context compactions, continuously tracking the state of every requirement from initial clarification through implementation, verification, and an accurate final handoff report.

Because current evaluation paradigms lack tasks that measure these organizational capabilities end to end, we introduce SWE-CoWork to benchmark coding agents as digital employees inside software organizations. Each SWE-CoWork task combines a real open-source repository with an authored workplace populated by 26 *digital employees*—rule-governed, LLM-backed models of colleagues with distinct responsibilities, bounded local knowledge, and controlled collaboration behavior (e.g., disclosing a held requirement only when given sufficient code context, requesting a reproduction, or referring the agent to another owner), yielding 2,626 employee profiles across the benchmark. Figure 1 contrasts a single Click requirement presented directly as an issue against the same requirement embedded within an ongoing SWE-CoWork assignment, while a typical full assignment distributes requirements across a median of six accountable owners (such as module maintainers, compatibility liaisons, and release coordinators). In total, SWE-CoWork comprises 101 cumulative development assignments across 101 upstream repositories, 11 languages, and 12 application domains, where a median task adds 2,955 production lines across 60 files and is graded by 32 executable verifier *test nodes*. Starting from a short onboarding note, a frozen repository, and a workplace containing project boards, versioned documents, merge-request histories, CI records, and 40,836 chat messages across the benchmark (Figure 4), the agent must inspect records, consult colleagues, reconcile conflicting evidence, implement and verify changes, and deliver a final handoff document.

Each scored requirement is linked to an accountable owner, a discovery route through workplace records or

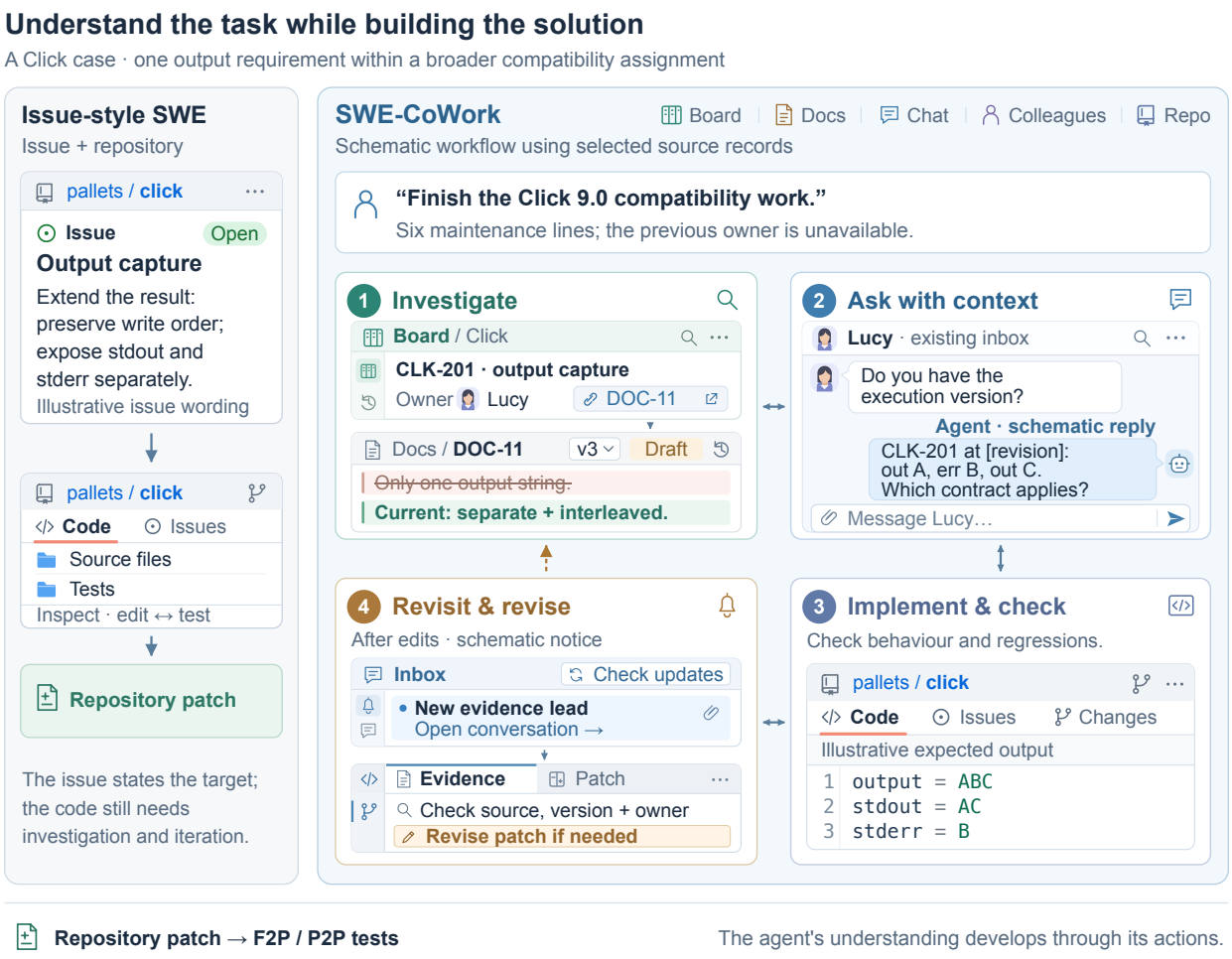


Figure 1 The same Click requirement presented directly as an issue-based task (left) and within an ongoing SWE-CoWork assignment (right). In SWE-CoWork, knowledge is distributed among digital employees and workplace records. The agent recovers the requirement, consults the responsible colleague, implements and verifies the change, and updates its interpretation when new evidence appears. Interfaces are schematic; Appendix D provides the source records.

colleague interactions, and private executable F2P/P2P tests. This linkage allows us to trace requirements from initial discovery to verified delivery. We further introduce a paired Oracle-Spec condition that provides complete requirements upfront while holding the repository, tests, scaffold, and execution budget constant. Together with agent trajectories, this comparison helps distinguish failures in requirement acquisition from those that persist during implementation, verification, and handoff.

We evaluate eleven models on 101 tasks and report F2P, P2P, and complete task pass rates (Table 2). Claude-Opus-5.5 and GPT-6 Astra pass 63.84% and 62.91% of F2P test nodes but complete only 6.93% and 7.92% of tasks; the remaining nine models reach at most 39.09% F2P and 0.99% complete task success. Supplying complete requirements upfront in Oracle-Spec raises pooled F2P from 39.09% to 55.76% for Seed Evolving and from 62.91% to 78.84% for GPT-6 Astra (Table 3), yet complete task pass rates remain unchanged (0.99% and 7.92%). Requirement-level trajectory analysis reveals why: weaker models remain trapped in passive spec-executor behavior (e.g., DeepSeek V4 Pro sends a median of 0 colleague messages per run and obtains only 1.8% of colleague-held requirements), whereas frontier models find the right people but fail at persistent requirement tracking. Specifically, Claude-Opus-5.5 directs 93% of first questions to the accountable owner and obtains 64% of colleague-held requirements, with only 3% belonging to uncontacted owners. However, failures persist after initial contact: relevant cards go unraised with already-contacted owners, deferred questions go unfollowed (153 of 450 deferred threads are abandoned with a median of 7.1 hours of budget remaining), disclosed requirements still account for 56% of failed linked tests, and 31% of

requirements reported as "done" in final handoffs fail their tests.

In summary, we make the following contributions:

- **The first benchmark for organizational software engineering intelligence.** SWE-CoWork provides 101 development assignments across 101 repositories, 11 languages, and 12 domains, shifting evaluation from isolated spec execution to autonomous collaboration inside software organizations. In each assignment, requirements are distributed across rule-governed colleagues with bounded responsibilities and evolving project records, with every requirement tied to a fixed F2P/P2P test contract.
- **A paired evaluation of organizational overhead and implementation bottlenecks.** The CoWork and Oracle-Spec conditions share the repository, tests, scaffold, and execution budget, allowing us to jointly examine the organizational overhead of establishing requirements through colleagues and records alongside downstream implementation bottlenecks. We report this comparison for Seed Evolving and GPT-6 Astra together with CoWork results across eleven models.
- **A requirement-level account of where sustained tracking fails.** By tracing each requirement across colleague conversations, code changes, verification, and handoff notes to individual tests, we show that weaker models struggle to proactively obtain requirements from the organization. Frontier models such as Claude-Opus-5.5 reliably reach the right owners, yet fail at sustained requirement tracking at later stages—through unraised cards, abandoned deferrals, unsatisfied disclosed requirements, premature stops, and completion claims contradicted by tests.

2 Related Work

Engineering tasks in real repositories. Recent software engineering benchmarks evaluate a growing range of development activities in realistic repositories. DeepSWE (Huang et al., 2026; Datacurve, 2026) evaluates original tasks in real repositories; SWE Atlas (Raghavendra et al., 2026) covers codebase questions, test writing, and refactoring; FrontierSWE v2 (Kondra et al., 2026) examines implementation, optimization, and research tasks under long interaction budgets; CursorBench 4.0 (Cursor, 2026; Jain, 2026) derives tasks from real developer activity; and Real-SWE (Das et al., 2026) extends evaluation to private production codebases and business tooling. These benchmarks expand the scope, complexity, and execution horizon of coding tasks, but generally provide agents with an identified development objective. In SWE-CoWork, agents instead join an ongoing project with a brief handoff and must establish its current requirements from organizational records, colleagues, and repository history before implementing and verifying the changes.

Clarification and requirement recovery. Several benchmarks investigate how coding agents handle incomplete or ambiguous requirements. RealSWE (Kim et al., 2026) varies the information provided for a fixed task, while LHAW (Pu et al., 2026) systematically removes goals, constraints, inputs, and contextual information. *Ask or Assume* (Edwards and Schuster, 2026) and Dialogue SWE-Bench (King and Flanigan, 2026) introduce interactive users who answer clarification questions, and ICAE-Bench (Peng et al., 2026) grounds user responses in withheld product constraints. τ^{τ}-Bench (Shi et al., 2026) requires developer agents to recover requirements through business artifacts and client interactions. These works demonstrate the importance of information acquisition, typically within a task whose scope is predefined or whose missing requirements can be clarified through a designated user or client.

SWE-CoWork examines requirement recovery within a broader engineering organization. Requirements are distributed among multiple colleagues with distinct responsibilities and bounded knowledge, as well as project boards, versioned documents, conversations, reviews, and execution records. Agents must identify the appropriate owners, ask questions with sufficient context, follow up on deferred answers, and reconcile information from different sources. Historical records may contain superseded decisions or observations tied to earlier revisions, while new evidence encountered during implementation may require revisiting an existing solution. These interactions require agents to establish both the content and the authority of requirements as the work progresses.

Benchmark	Engineering task	Initial information	Information sources	Information process	Evaluation
DeepSWE 1.1 (Huang et al., 2026; Datacurve, 2026)	Repo change	Request Underspecified	Repo	n/r	Patch tests
SWE Atlas (Raghavendra et al., 2026)	Code Q&A Test writing Refactor	Request Underspecified	Repo	No live Q&A	Execution Rubric
FrontierSWE v2 (Kondra et al., 2026)	Build Optimize Research	Brief	Workspace Self-check	n/r	Task score
CursorBench 4.0 (Cursor, 2026; Jain, 2026)	Edit Refactor Investigate	Request Underspecified	Code Tools	n/r	Agentic grader
Real-SWE (Specific Labs) (Das et al., 2026)	Enterprise change	Brief Underspecified	Repo Docs Board Chat	n/r	n/r
RealSWE (paper v2) (Kim et al., 2026)	Bug fix Feature	Issue variants Underspecified	Repo	Static request No live Q&A	Fixed target
τ^{τ}-Bench (Shi et al., 2026)	Agent build	Underspecified	Docs APIs Code	Client Q&A Conflicting history Superseded records	User simulation
SWE-CoWork (ours)	Multi-PR Repo change	Onboarding brief Underspecified	Repo Board Docs Chat MR/CI	Multi-owner Q&A Versioned records Conflicting history Execution-time leads	F2P/P2P Fixed contract

task initial information information source interaction / dynamics evaluation

n/r = not reported in the cited public description.

Table 1 How recent software engineering benchmarks present engineering work to agents. *Initial information* describes what the agent receives at the outset; *information sources* identifies where task-relevant evidence resides; and *information process* describes how agents clarify requirements, reconcile evidence, and respond to information encountered during execution. "n/r" indicates that the property is not described in the cited public material.

Agents in workplace environments. Workplace-agent benchmarks evaluate professional tasks involving organizational records, applications, and collaboration with colleagues. TheAgentCompany (Xu et al., 2025) places agents in a simulated company with coworkers and workplace services to evaluate their ability to complete consequential professional tasks. Recent system efforts also explore more autonomous software development workflows. DeepSeek's Super-agent prototype (DeepSeek AI, 2026a), for example, connects coding agents to GitHub and team messaging, using event-driven task dispatch and dedicated execution sessions to support ongoing engineering work.

SWE-CoWork provides an evaluation environment for coding agents operating within such organizational contexts. Each task combines a real software repository with an authored engineering workplace populated by digital employees whose knowledge and responsibilities are grounded in the project. Agents must collaborate with these employees, recover relevant decisions from organizational records, and implement the required changes. Each requirement is linked to a fixed executable test contract, allowing evaluation of both requirement acquisition and verified delivery.

The reliability of simulated colleagues is also important for interactive evaluation. Evaluation outcomes can depend on the user simulator, and simulated interactions may differ systematically from interactions with human users (Seshadri et al., 2026). In SWE-CoWork, fixed knowledge boundaries and disclosure rules constrain colleague interactions across runs (Section 3.1).

Analyzing agent behavior and failures. Prior work also investigates the processes through which coding agents complete engineering tasks. SWE-RPG (Zhou et al., 2026) provides representations for studying planning and clarification in software agent trajectories, while τ-bench (Yao et al., 2024) examines reliability across repeated interactive executions. Coding-agent frameworks such as SWE-agent, OpenHands, and Agentless (Yang et al., 2024; Wang et al., 2025; Xia et al., 2024) further demonstrate the influence of agent scaffolding on execution behavior and task outcomes.

SWE-CoWork enables requirement-level analysis of the full engineering workflow. By linking individual requirements to colleague interactions, workplace evidence, executable tests, and final delivery reports, we can identify where requirements are lost between discovery, clarification, implementation, verification, and handoff. This distinguishes failures to acquire necessary information from failures to implement requirements that the agent has already obtained, and reveals discrepancies between reported completion and verified outcomes.

Table 1 compares recent benchmarks by their initial task information, available evidence sources, and information processes. Appendix B discusses additional repository, interactive, and workplace benchmarks.

3 Engineering Organizations of Digital Employees

Each SWE-CoWork task places a coding agent in an engineering organization consisting of a real software repository, a shared workplace, and a population of digital employees. Starting from a brief handoff, the agent establishes requirements through workplace records and interactions with colleagues, then implements, verifies, and delivers the requested changes. To evaluate this process, each task defines a fixed executable contract through fail-to-pass (F2P) and pass-to-pass (P2P) tests. For every scored requirement, the design connects the colleague or record that holds it, the responsibility boundary that determines who can answer for it, and the test nodes that check it. These links let us trace a requirement from the first question about it through the delivered code to the final handoff note.

3.1 Digital Employees

We construct an organization of 26 modeled colleagues around each repository, yielding 2,626 employee profiles across the 101 tasks. Each digital employee represents a named employee with a defined role, responsibilities, local knowledge, and rules governing interactions with the agent, with responsibilities and knowledge grounded in the repository's files, declarations, and modules. The organization spans one target project and four neighboring projects that provide additional context; within this population, a median of six employees hold requirements that determine the task's executable outcome (Table 6).

What a digital employee represents. Each digital employee has three properties: responsibility, local knowledge, and collaboration behavior. *Responsibility* identifies the employee accountable for a module or decision. *Local knowledge* defines which requirements the employee can explain and which questions require referral to another colleague. *Collaboration behavior* governs how the employee responds. An owner may request a reproduction or caller path, address one topic at a time, or defer an answer until the agent follows up. The environment controls which information can be disclosed at each step, while a language model generates the employee's response.

From individual digital employees to an organization. The digital employees share project records that connect people to engineering work. Board cards identify owners, documents record decisions, and chat and review threads preserve earlier discussions. Information obtained from one employee may need to be checked against another employee's constraints or a more recent repository revision, requiring the agent to identify appropriate owners and resolve outstanding questions as implementation proceeds.

The agent interacts with the organization through a dedicated tool interface. It can search the staff directory, inspect the project board, open versioned documents, search and read chat, message colleagues, inspect merge requests and CI records, and check for new activity. Digital employees are accessible through the same workplace that contains their decisions and work history.

The handoff. The agent joins an engineering assignment already in progress. It begins with a short onboarding note that states the development objective and identifies workplace resources relevant to the current project state. The note specifies restrictions on modifying existing tests, excludes neighboring projects from the assignment, and identifies historical terminology that may differ from the current repository. The remaining acceptance criteria are distributed across workplace records and colleagues.

The agent decides which records to inspect, which references to follow, and which colleagues to contact. It interprets the available evidence against the current repository state and resolves questions that affect the implementation. At completion, it produces a delivery note describing the requirements established, the changes made, the validation performed, and any unresolved questions.

3.2 Composition

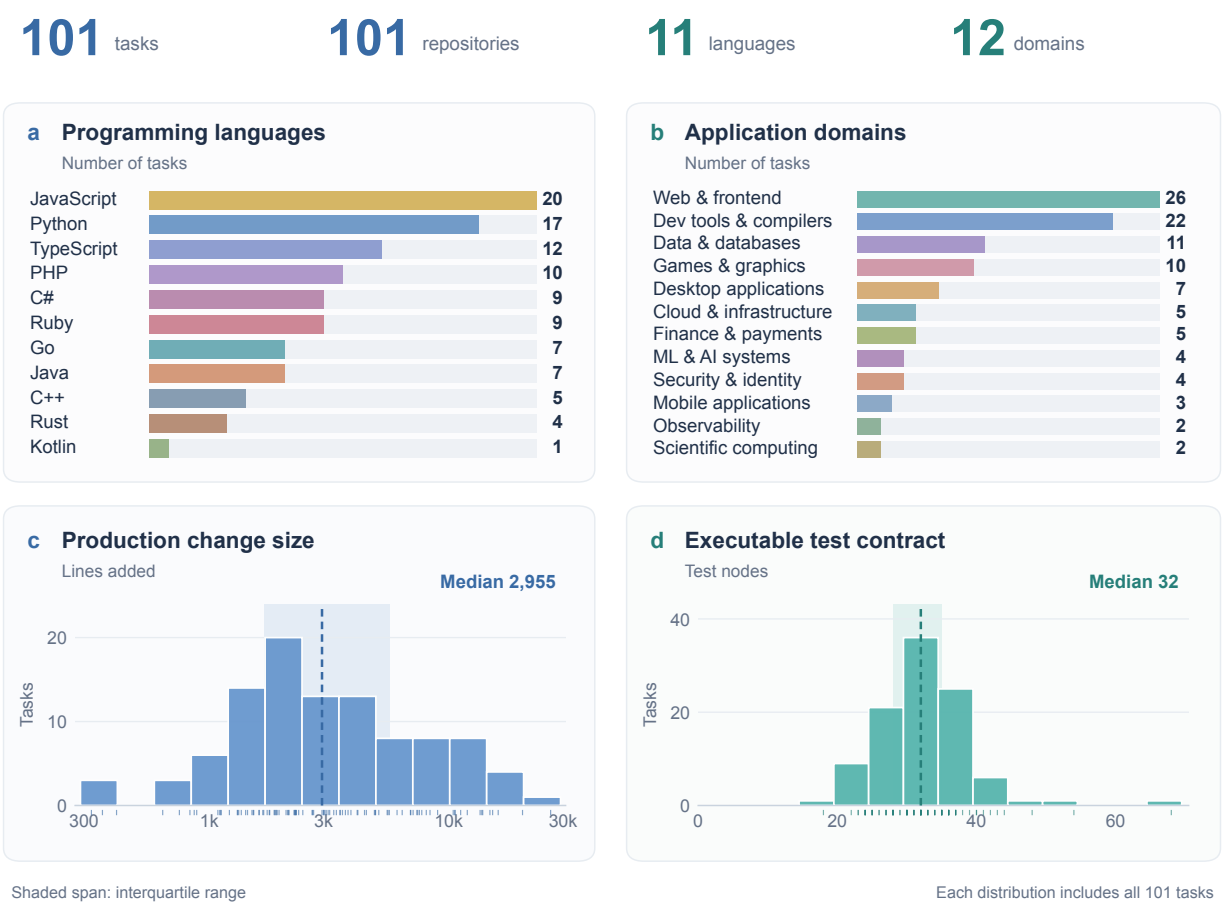


Figure 2 Composition of the 101 SWE-CoWork tasks. (a) Primary language of the production changes. (b) Application domain of the upstream repository. (c) Production lines added by the reference changes, shown on a logarithmic axis. (d) Verifier test nodes per task. In (c) and (d), dashed lines mark medians, shaded bands span the interquartile range, and ticks below each axis mark individual tasks.

SWE-CoWork contains 101 tasks drawn from 101 upstream repositories across 11 programming languages and 12 application domains, including web frameworks, compilers, games, payments, and scientific software (Figure 2; Appendix C). 11 tasks involve substantive changes in more than one programming language.

The reference solutions involve substantial engineering work. The median solution adds 2,955 lines of production code across 60 files, excluding tests, documentation, configuration, fixtures, and generated artifacts. For each task, we maintain a complete Oracle-Spec document that enumerates the requirements encoded by the same executable contract used to evaluate the workplace assignment. Across the 101 tasks, these contracts contain 2,699 F2P test nodes and 562 P2P test nodes, giving 3,261 executable checks in total.

3.3 Shared Records and Organizational History

Each assignment is situated within the records of a wider engineering organization. Across the benchmark release, these organizations contain 10,329 board cards, 5,097 versioned documents, 40,836 chat messages, 1,194 merge requests, and 2,626 colleague profiles. Whereas the initial onboarding note contains a median of 206 words, the authored document corpus contains a median of 5,453 words, with requirements and supporting evidence distributed across these records and their histories.

The agent determines which artifacts to inspect, which historical decisions still apply, which execution records correspond to the current revision, and which colleagues can provide missing information. We keep the overall organization size roughly comparable across tasks to limit variation in the volume of available workplace material.

Organizational history. The workplace records how projects develop over time. Versioned documents, board histories, review threads, CI records, and conversations contain intermediate diagnoses, superseded plans, rejected approaches, and observations tied to particular revisions. Historical records may use interface, module, or protocol names that differ from those in the current codebase. The agent establishes which claims remain applicable and reconciles them with subsequent decisions and the current repository.

Access follows the organization's information boundaries. Private groups and direct messages that do not include the agent are inaccessible. Records from neighboring projects provide organizational context, while those projects remain outside the scope of the assignment.

3.4 Scoring

We report three outcome metrics: **F2P**, the proportion of fail-to-pass verifier test nodes satisfied by the submitted implementation; **P2P**, the proportion of pass-to-pass test nodes preserved; and **Pass rate**, the proportion of tasks for which every F2P and P2P test node passes in the same run. F2P and P2P are computed by pooling passed test nodes and total test nodes across tasks.

Because each task contains multiple independently testable behaviors, F2P captures partial delivery, while Pass rate requires completion of the full executable contract. Section 5 provides the aggregation formulas and describes the trajectory diagnostics.

4 Working a SWE-CoWork Task

In a SWE-CoWork task, the agent works with the digital employees introduced in Section 3.1. Starting from a brief handoff, it establishes the requirements through repository evidence, workplace records, and conversations with colleagues. The agent then implements and verifies the requested changes, revisiting earlier decisions as new information becomes available. Requirement discovery and implementation are interleaved: an owner's clarification can change the behavior that a patch must satisfy.

4.1 Working with the Digital Employees

Agents continuously move between investigating requirements, consulting colleagues, and implementing changes. Starting from a brief handoff, an agent explores the repository and workplace records to establish the current scope, identify unresolved requirements, and locate responsible colleagues. It must reconcile conflicting evidence, trace references across historical revisions, and ask questions grounded in the available code and project context. A typical run involves the following activities:

1. **Orient.** Read the onboarding note, inspect the repository, identify the target project, and establish the initial scope and relevant contacts.
2. **Investigate.** Search project boards, documents, chat, and engineering history for requirements and implementation constraints. Check record provenance, reconcile version differences, and trace references to renamed interfaces.

3. **Ask.** Identify the colleague responsible for an unresolved decision and ask a question supported by the available evidence. Follow referrals and respond to requests for additional context.
4. **Build and verify.** Implement the required changes, run relevant tests, and check for regressions in existing functionality and neighboring projects.
5. **Revise.** Examine new messages, traces, and corrections. Confirm their implications with the responsible colleague and update the implementation when necessary.
6. **Deliver.** Document the confirmed requirements, implementation changes, and verification results in the delivery note.

These activities are interleaved throughout execution. A colleague may request additional evidence, refer the agent to another owner, or introduce a constraint that requires revising an existing implementation. While some questions remain unresolved, the agent can continue working on other requirements, validate its changes, and revisit earlier decisions as new information becomes available. At handoff, it reports the implemented changes, verification results, and outstanding issues.

4.2 Inside a Run: An Owner Clarification Changes the Patch

Task 094 involves a cumulative upgrade of Kimai, a PHP time-tracking application. One requirement, KIM-210, concerns exporting timesheet durations in decimal hours. The agent initially implemented the numerical conversion and verified that a 5,400-second entry produced 1.5 hours. However, the design document did not specify how the feature should be activated (Figure 3).

The agent then contacted the export owner to clarify the intended behavior:

> "My XLSX cell and total calculate to 1.5. Please confirm whether that is the intended decimal behavior..." — *agent*

> "... when the current user's `timesheet.export_decimal` preference is true, the value is 1.5 hours... The preference is evaluated on every export, even when one renderer is reused for different users." — *export owner*

The clarification revealed two requirements absent from the initial implementation: decimal formatting must follow the current user's preference, and that preference must be evaluated on every export. The agent revised the renderer accordingly and extended its tests to cover consecutive exports by different users sharing the same renderer.

This interaction illustrates how a seemingly correct implementation can remain incomplete until a colleague clarifies the intended behavior. Across the full assignment, the agent satisfied 17 of 21 new behaviors while preserving all 15 existing behaviors.

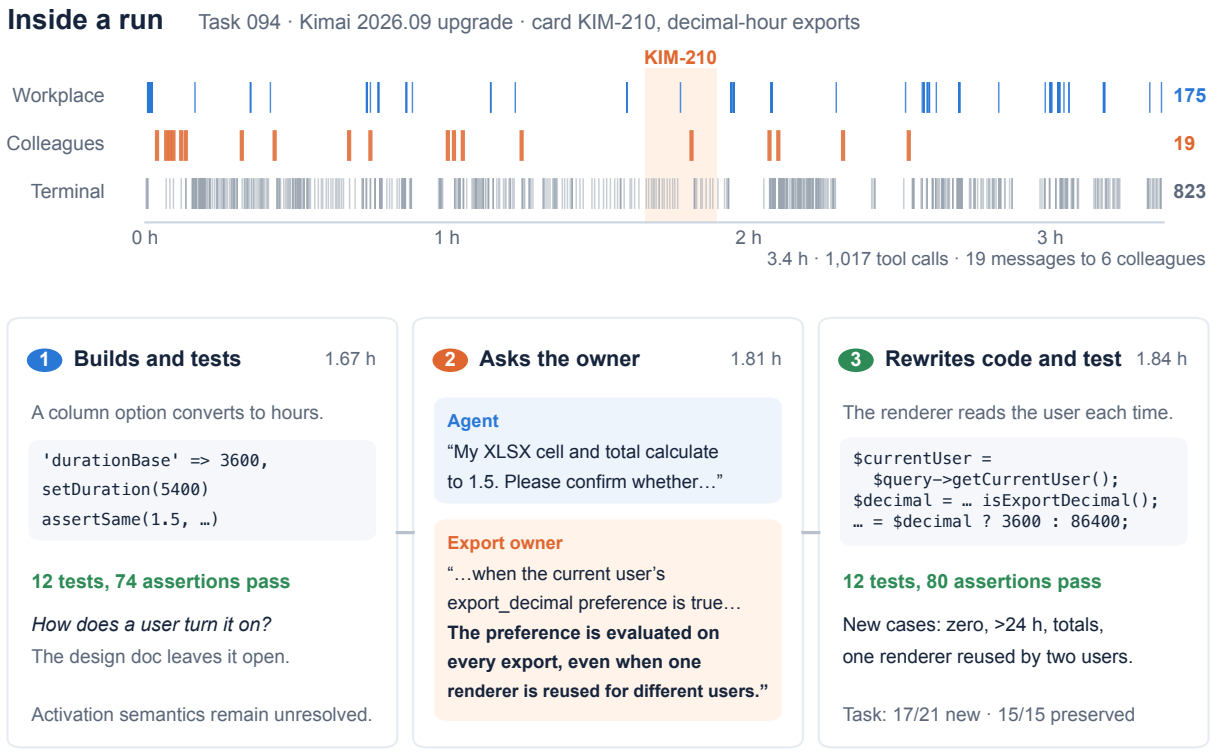


Figure 3 An owner clarification changes a timesheet export implementation in a recorded SWE-CoWork run (task 094, Kimai). The agent initially implements and tests decimal conversion, then contacts the export owner to resolve how users activate the feature. The owner specifies that the current user's preference must be evaluated on every render. The agent updates the implementation and expands the tests accordingly. Times indicate elapsed hours from the start of the run, and quotations come from the recorded exchange.

4.3 Workplace Information Challenges

Because requirements are distributed across records and colleagues, establishing what a record means requires checking when it became available, where it came from, and who has authority over the decision. Figure 4 groups these information challenges into five situations and shows the engineering judgment required in each. These situations affect how the agent establishes the requirements, with each misleading record accompanied by an accessible correction or a responsible colleague who can establish which evidence applies.

Late evidence. A recovered trace, corrected version label, or message from another team may arrive after implementation begins. In the schematic example, a successful browser record is followed by an SDK observation and a request to confirm the client version. The agent must follow this lead to the responsible owner, establish which behavior it concerns, and reassess the affected code and tests. Owners remain available for consultation before a scheduled notice appears.

Superseded conclusions. An earlier proposal can remain searchable after a published correction. For Stencil's `AUR-202`, the draft of `DOC-13` proposes an empty object when no source map is generated. The published third version instead requires the returned `map` field to remain `undefined`, while retaining the chunk's other output information. The agent must compare revisions and publication states to identify the applicable rule.

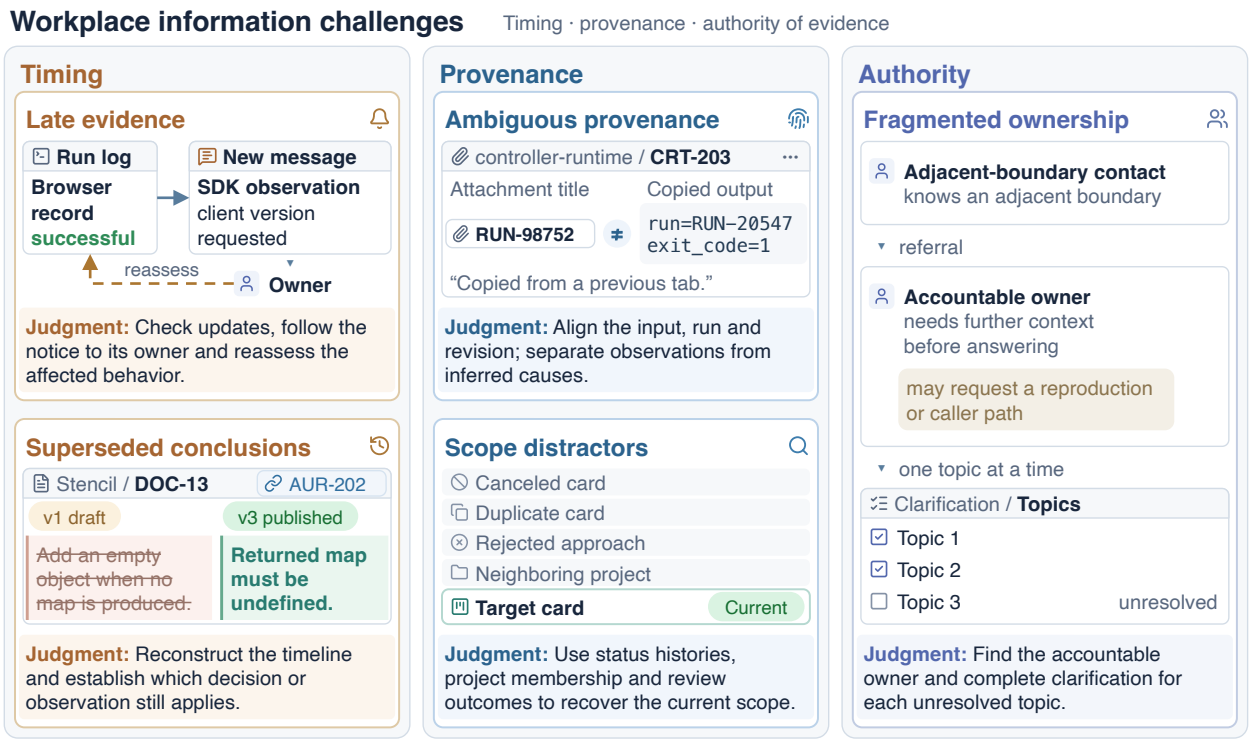


Figure 4 Five workplace information challenges and the judgments they require, organized by when evidence becomes available or is superseded, which run or project it belongs to, and who can resolve the outstanding decision. The panels use schematic situations and condensed workplace-record examples. Source details for Stencil and controller-runtime appear in Appendix D.

Ambiguous provenance. A log's title, contents, and surrounding discussion may refer to different executions. In the controller-runtime example, the attachment title names `RUN-98752`, but the copied output identifies `RUN-20547` and states that it came from a previous tab. The agent must align the output with its input, run, and code revision before using it to diagnose a failure.

Scope distractors. Canceled cards, duplicates, rejected approaches, and neighboring projects remain visible alongside the current assignment. Similar wording can therefore lead to a record outside the required change. Status histories, project membership, and review outcomes establish which card is current and which approaches have already been rejected. The agent uses these distinctions to recover the target scope.

Fragmented ownership. A colleague may know an adjacent interface boundary yet lack authority over the unresolved decision. A referral leads to the accountable owner, who may request a reproduction, caller path, or version before answering. Clarification also proceeds topic by topic: resolving one question can leave others open. The agent supplies the requested context and tracks which decisions each exchange establishes before carrying them into the implementation.

These situations make investigation and coding interdependent. A newly located record can prompt another question, whose answer changes the patch or its tests. Appendix D gives the underlying records for the Stencil and controller-runtime examples and a further Click example involving output semantics and stream ownership.

5 Evaluation

We evaluate eleven coding models in the CoWork setting using executable outcomes and collaboration records. A paired comparison with Oracle-Spec measures the performance of Seed Evolving and GPT-6 Astra when

complete requirements are supplied at the start. To examine how requirements are obtained, incorporated into code, verified, and reported at handoff, we analyze Claude-Opus-5.5 trajectories.

5.1 Primary metrics

We report **F2P**, **P2P**, and **Pass rate** on $N = 101$ tasks. For task i, let f_i and p_i denote the numbers of passing F2P and P2P test nodes, and let F_i and P_i denote their respective totals. All metrics are expressed as percentages:

$$\text{F2P} = 100\,\frac{\sum_{i=1}^{N} f_i}{\sum_{i=1}^{N} F_i}, \qquad \text{P2P} = 100\,\frac{\sum_{i=1}^{N} p_i}{\sum_{i=1}^{N} P_i},$$

$$\text{Pass rate} = \frac{100}{N}\sum_{i=1}^{N} \mathbf{1}[f_i = F_i \ \wedge \ p_i = P_i]\,.$$

F2P measures how many new requirements are satisfied, while P2P measures whether existing behavior is preserved. Both metrics aggregate test nodes across tasks, giving greater weight to tasks with more test nodes. Pass rate measures the fraction of tasks for which every F2P and P2P test node passes.

All models are evaluated against the full release contract: 2,699 F2P test nodes, 562 P2P test nodes, and 101 tasks. Only passes confirmed by the verifier count toward the numerators. For trajectory analysis, we use the fraction of F2P test nodes passed within each task to relate outcomes to agent behavior.

5.2 Experimental design

CoWork evaluation. We evaluate eleven models at max effort in the CoWork setting: Claude-Opus-5.5 (Anthropic, 2026), GPT-6 Astra (OpenAI, 2026), Seed 2.1 Pro (ByteDance Seed, 2026), ByteDance's Seed Evolving (Volcano Engine, 2026), Kimi K3 (Moonshot AI, 2026), DeepSeek V4 Pro (DeepSeek AI, 2026c), DeepSeek V4.1 Flash (DeepSeek AI, 2026b), GLM 5.3 (Z.ai, 2026b), GLM 5.2 (Z.ai, 2026a), and Qwen3.8-Max and Qwen3.8-Flash (Alibaba Cloud Model Studio, 2026).

GPT-6 Astra uses a Codex scaffold, and the other ten models use Claude Code scaffolds. In every CoWork run, the digital employees are backed by GPT-5.6 Terra at high reasoning effort, so all agents interact with the same colleague model. Each agent starts with a brief handoff and access to the organization. It identifies the requirements through workplace records and conversations with colleagues, implements the requested changes, and submits its work for F2P/P2P evaluation. Table 2 reports F2P, P2P, and Pass rate for all eleven models on the 101-task release contract.

Oracle-Spec comparison. We evaluate Seed Evolving and GPT-6 Astra under two conditions. In CoWork, the agent obtains requirements through workplace records and colleagues. In Oracle-Spec, the complete requirement set is provided at the start, and the agent has no colleagues to consult. Both conditions cover the same 101 tasks, with identical task definitions, repository states, and executable test contracts.

We compare F2P, P2P, and Pass rate across the paired runs to measure how access to complete requirements affects implementation outcomes.

Claude-Opus-5.5: trajectory analysis. We examine Claude-Opus-5.5's CoWork trajectories by combining programmatic disclosure logs with a rubric-based LLM trajectory reviewer. For each run, we trace which requirements reached the agent, which remained unresolved, how it followed up with colleagues, and how it verified and reported its work.

We select Claude-Opus-5.5 because it achieves the highest CoWork F2P score among the evaluated models. The analysis covers its recorded runs and examines the relationship between requirement acquisition, agent behavior, and implementation outcomes.

Supplementary diagnostics appear in Appendix F; Appendix H gives the measurement definitions, claim-classification rules, and evidence sources.

5.3 Supplementary analysis metrics

Model calls and token usage. We report the median per-run number of model calls, input tokens, output tokens, tool calls, and context compactions. Input usage accumulates tokens processed across calls within a run, including repeated context. We pair these usage medians with each model's F2P on the full release contract. Figures 12 and 13 in Appendix G.1 report these comparisons.

Workplace exploration and the first code edit. For eight models, we report per-run medians of workplace queries before the first code edit, the tool-call index of that edit, and workplace queries over the full run (Figure 6). The same comparison includes median messages sent to colleagues.

Requirement acquisition and collaboration. Each task's ORACLE-SPEC enumerates its requirements and identifies the responsible colleague or public record. Disclosure logs and saved tool replies establish which requirements were released by colleagues and which replies reached the agent.

We measure **ask rate**, the fraction of requirements held by colleagues that the agent raised with the responsible owner or obtained; **targeting**, the fraction of queried cards whose first question was directed to the accountable owner; **gain per question**, the number of disclosures per colleague message; **follow-through**, the fraction of raised requirements that were subsequently disclosed; **redundancy**, questions about cards whose requirements had already been fully disclosed; and **unread disclosures**, replies containing disclosed requirements that were absent from the agent's recorded observations.

6 Results

Across eleven models, requirement acquisition closely tracks delivery, while model-call and tool-call counts do not consistently track F2P. Supplying complete requirements improves partial delivery for Seed Evolving and GPT-6 Astra without increasing complete task success. Claude-Opus-5.5 trajectories locate further failures after initial contact: deferred questions remain unresolved, obtained requirements fail their tests, and handoff claims overstate completed work.

6.1 Results Across Models

Table 2 compares eleven models on the 101-task CoWork benchmark. Even the leading models rarely complete an entire assignment.

Model	F2P (%)	P2P (%)	Pass rate (%)
Claude-Opus-5.5	**63.84**	93.24	6.93
GPT-6 Astra	62.91	89.15	**7.92**
Seed Evolving	39.09	94.48	0.99
Qwen3.8-Max	35.70	91.80	0.00
GLM 5.3	31.27	87.72	0.99
Qwen3.8-Flash	30.70	93.40	0.00
DeepSeek V4.1 Flash	28.20	94.30	0.00
Kimi K3	27.08	84.88	0.99
GLM 5.2	24.40	**95.20**	0.00
Seed 2.1 Pro	21.01	81.85	0.00
DeepSeek V4 Pro	13.04	90.21	0.00

Table 2 CoWork evaluation on 101 tasks. F2P and P2P use the release totals of 2,699 and 562 test nodes, respectively. Pass rate requires every F2P and P2P test node to pass in the same run. All scores are percentages, with higher values indicating better outcomes. Shading marks the top three per metric, from darkest (best, bold) to lightest; ties share a shade. Rows are ordered by F2P.

Partial delivery versus complete task success. Claude-Opus-5.5 and GPT-6 Astra achieve F2P scores of 63.84% and 62.91%, respectively, yet their complete task pass rates are only 6.93% and 7.92%. The remaining nine models achieve at most 39.09% F2P and 0.99% complete task success. These results reveal a substantial gap between implementing individual required behaviors and satisfying the full assignment.

P2P scores are comparatively higher, ranging from 81.85% to 95.20%, with rankings that differ from F2P. This suggests that preserving existing behavior and delivering new requirements pose distinct challenges, with the latter remaining a major bottleneck in organizational software engineering.

Delivery tracks requirement acquisition. Figure 5 plots each model's F2P score against the share of requirements it obtained from colleagues and against its median number of tool calls per run. Table 9 in Appendix G lists all values, including colleague messages and wall-clock hours. The colleague rate counts the requirements held only by colleagues that a colleague disclosed during the run, as recorded by the disclosure events of the colleague service.

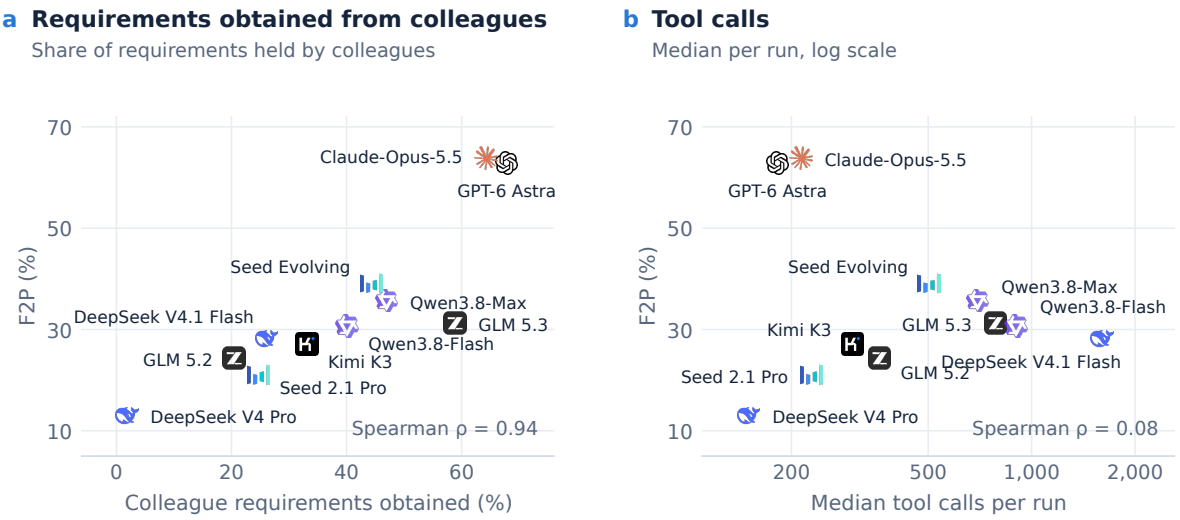


Figure 5 Requirement acquisition, tool calls, and delivery across eleven models. (a) F2P against the share of requirements held only by colleagues that a colleague disclosed during the run. (b) F2P against the median number of tool calls per run, on a logarithmic axis. Each model is shown by its developer's logo. Spearman ρ is computed over the eleven models.

Requirement acquisition is strongly associated with F2P (Spearman $\rho = 0.94$), whereas tool-call counts show little association ($\rho = 0.08$). GPT-6 Astra and Claude-Opus-5.5 obtain 67.8% and 64.2% of colleague-held requirements, respectively, and achieve the highest F2P scores. In contrast, DeepSeek V4 Pro obtains only 1.8% and reaches 13.04% F2P.

Greater interaction volume does not necessarily translate into better requirement acquisition or delivery. Some models exhaust the eight-hour execution budget with hundreds or thousands of tool calls, yet obtain fewer requirements than the leading models. These findings highlight the importance of acquiring relevant organizational information, while leaving open how effectively agents translate that information into working implementations.

Interaction volume and context usage. Across nine models with available usage telemetry, additional analysis of model calls, token usage, and context compactions shows no consistent relationship between interaction volume and delivery. In particular, large input-token totals may reflect repeated processing of retained context rather than more extensive workplace investigation. We provide the detailed usage comparisons in Appendix G.

High acquisition also leaves an implementation gap. We examine it with Oracle-Spec (Section 6.2) and requirement-level trajectories (Section 6.3).

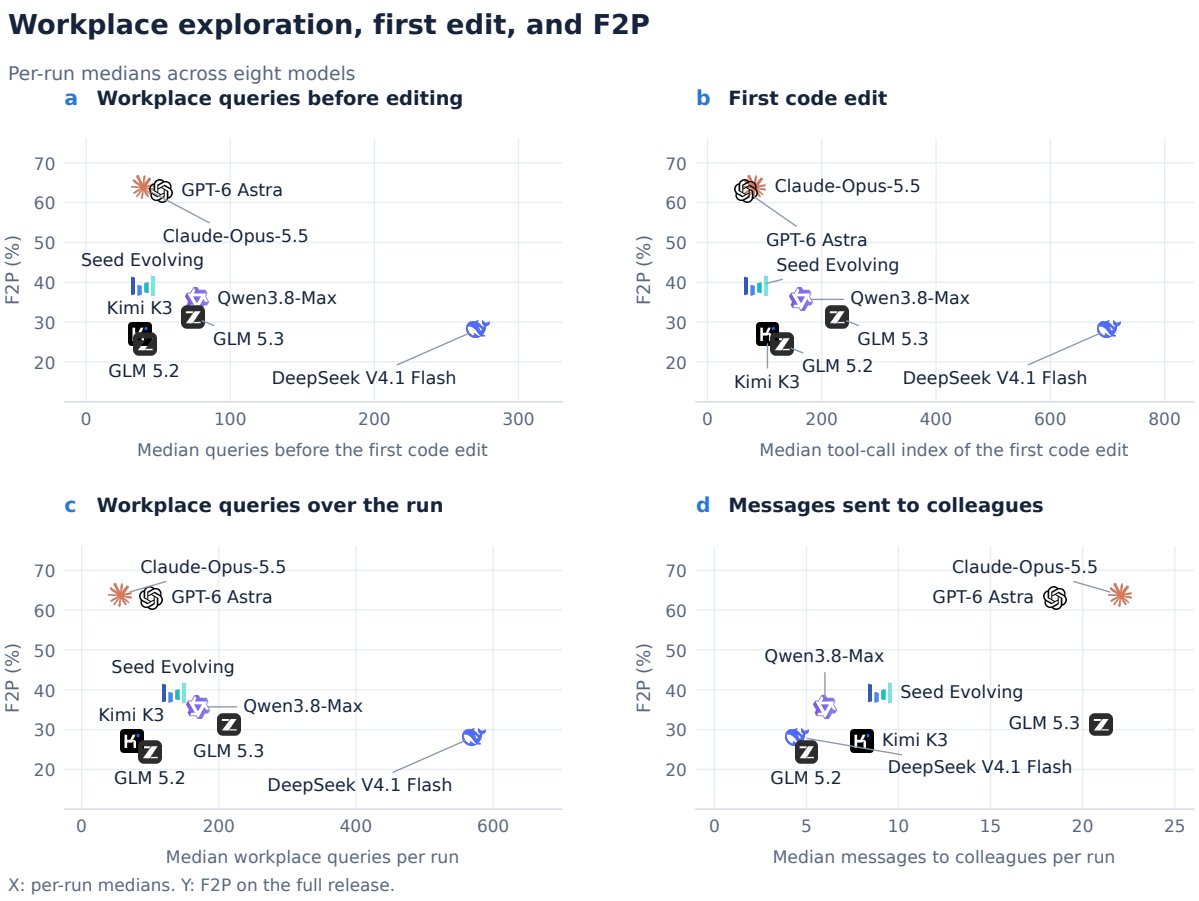


Figure 6 Workplace exploration, first code edit, and delivery. Full-release F2P against per-run medians of (a) workplace queries before the first code edit, (b) the tool-call index of that edit, (c) workplace queries over the full run, and (d) messages sent to colleagues. All panels cover the same eight models. All axes are linear. Each logo sits at the model's exact value; thin leaders link names to logos in crowded regions.

Workplace exploration and implementation. Figure 6 examines how agents explore organizational information and transition to code implementation. Across eight models, workplace queries before the first edit and total workplace queries show little rank association with F2P (Spearman $\rho = -0.10$ and $\rho = -0.17$, respectively). In contrast, a lower tool-call index for the first code edit ($\rho = -0.62$) and more frequent colleague communication ($\rho = 0.71$) are associated with higher F2P.

These patterns suggest that extensive investigation alone does not ensure successful delivery. Leading models tend to begin implementation after fewer tool calls while communicating more frequently with colleagues. These cross-model associations do not establish that earlier editing or more frequent messaging causes better outcomes. Together with the requirement acquisition results, they highlight the importance of integrating organizational information into the development process.

6.2 Effect of Supplying Complete Requirements

For Seed Evolving and GPT-6 Astra, supplying complete requirements up front raises partial delivery while leaving complete task success unchanged. The paired comparison holds the 101 tasks, repositories, executable contracts, scaffold and execution budget fixed.

Providing the specification raises Seed Evolving's F2P from 39.09% to 55.76%, a gain of 16.67 percentage points (Table 3). GPT-6 Astra's F2P rises by a similar amount, from 62.91% to 78.84%, a gain of 15.93 points. P2P moves in opposite directions: Seed Evolving's falls from 94.48% to 89.50%, while GPT-6 Astra's rises from 89.15% to 91.81%. With the complete specification, Seed Evolving implements more of the new behavior and preserves less of the existing behavior.

	Seed Evolving			GPT-6 Astra		
Metric	CoWork	Oracle	Gain	CoWork	Oracle	Gain
F2P	39.09	55.76	+16.67	62.91	78.84	+15.93
P2P	94.48	89.50	−4.98	89.15	91.81	+2.66
Pass rate	0.99	0.99	+0.00	7.92	7.92	+0.00

Table 3 CoWork vs. Oracle-Spec for Seed Evolving and GPT-6 Astra. Scores are percentages; gain is in percentage points. Pooled metrics use the complete 101-task release contract.

The roughly 16-point F2P gains reflect the combined benefit of direct access to requirements and the effort saved in searching for and confirming them. Complete task success nevertheless remains unchanged: Seed Evolving completes 0.99% of tasks and GPT-6 Astra 7.92% in both conditions. Even with every requirement in hand, Seed Evolving leaves 44% of F2P test nodes unsatisfied and GPT-6 Astra 21%. These remaining failures occur during implementation and verification, after requirements are available. The Claude-Opus-5.5 analysis below follows individual requirements through both stages.

6.3 Claude-Opus-5.5: Following Each Requirement to Delivery

We analyze 101 Claude-Opus-5.5 CoWork trajectories, linking colleague interactions, requirement disclosures, executable test outcomes, and final handoff reports. This analysis examines where requirements remain unresolved as agents move from clarification to implementation, verification, and delivery.

6.3.1 Clarification: reaching owners and raising requirements

Finding the right owner is not enough. Claude-Opus-5.5 directs its first question to the accountable owner for a mean of 93% of queried cards across runs (Table 7), yet obtains only 64% of the 2,176 colleague-held requirements (Figure 7). Of all colleague-held requirements, 16% are raised without obtaining disclosure, 17% remain unraised with owners already contacted about other cards, and 3% belong to owners never contacted. Thus, most missing requirements arise from incomplete clarification with already-contacted owners rather than failures to identify the right colleagues.

Acquired requirements remain difficult to deliver. F2P test nodes linked to disclosed colleague-held requirements pass at 68.8%, compared with 49.8% for undisclosed requirements. Within tasks containing both types, the mean difference is 21.3 percentage points. However, among failed F2P test nodes with linked requirements, 56% are assigned to *implement*: the requirement reached the agent, but the tested behavior remains unsatisfied. Another 42% are associated with requirements not successfully obtained through owner clarification (*ask*), and the remaining 2% are attributed to *discovery*. These results reveal gaps both in obtaining requirements and in translating available information into verified implementations.

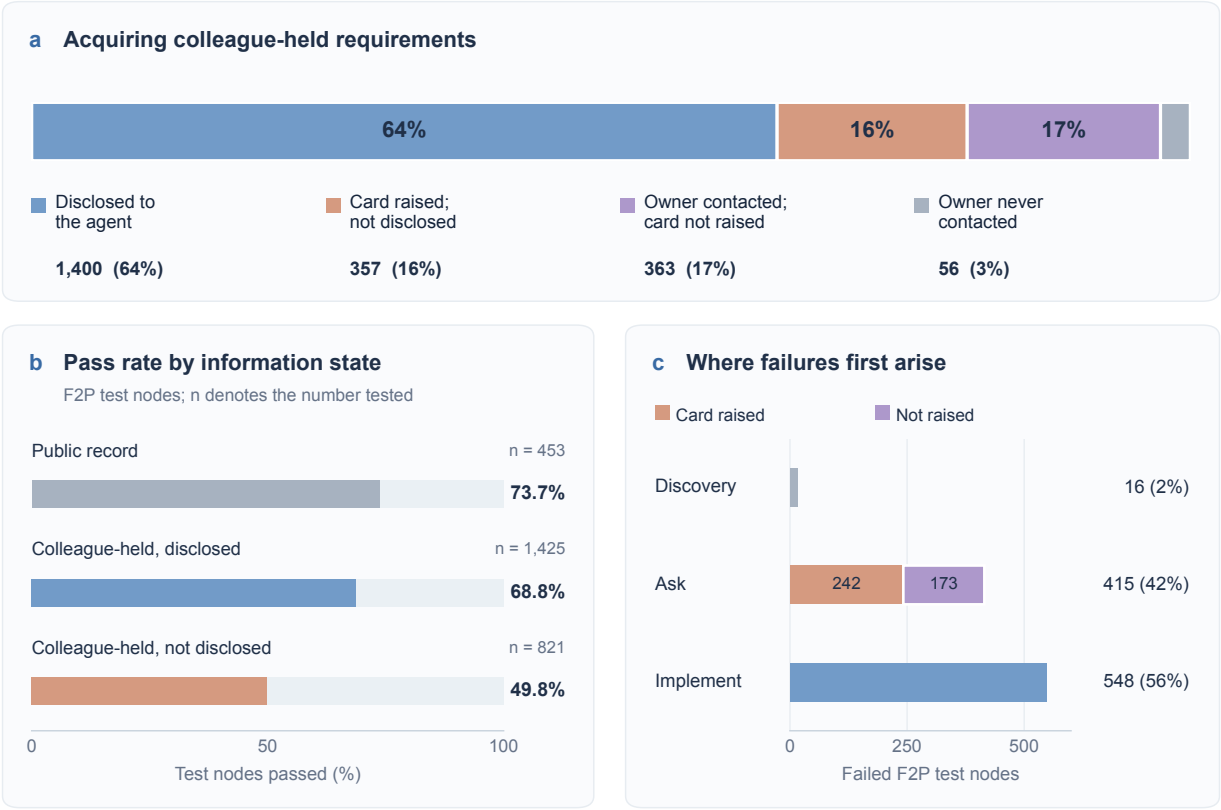


Figure 7 **Requirement acquisition and loss in Claude-Opus-5.5.** (a) Requirements held by colleagues, grouped by acquisition state. (b) F2P test-node pass rates against the full release contract, grouped by whether the corresponding requirement reached the agent. (c) Earliest observed loss stage (discovery, ask, or implement) for failed test nodes with linked requirements, evaluated against the full release contract. The *implement* category denotes unsatisfied behavior after the requirement reached the agent.

6.3.2 Clarification over time: following up deferred questions

Deferred questions remain unresolved despite continued activity. Among 450 owner–card threads in which a colleague initially defers an answer, Claude-Opus-5.5 follows up on 297, eventually obtaining disclosure in 205. The remaining 153 threads receive no follow-up (Figure 8b). At the initial deferral, these abandoned threads have a median of 7.1 hours of execution budget remaining, and the agent continues working for a median of 3.4 hours afterward.

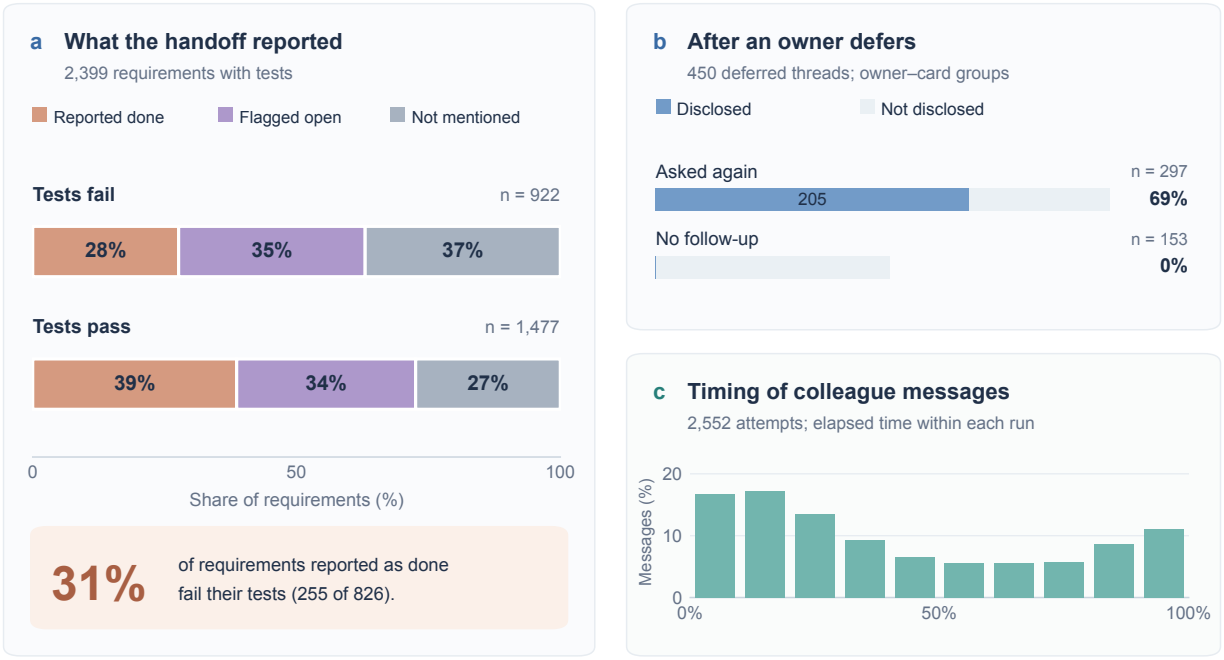


Figure 8 Follow-through and reporting in Claude-Opus-5.5. (a) Delivery claims compared with the test outcome of each requirement. (b) Deferred threads grouped by owner and card, subsequent follow-up and eventual disclosure. (c) Timing of colleague messages within a run. Message activity across the run can coexist with individual threads receiving no follow-up.

Communication nevertheless continues late in execution: of 2,552 colleague message attempts, 20% occur during the final fifth of execution (Figure 8c). These aggregate patterns show that sustained communication can coexist with individual questions left unresolved.

The PostCSS task (row 014) illustrates this difficulty. The agent raises card CSS-227 and follows up with additional evidence, but receives another deferral and eventually declares the streams complete without resolving the question. All 16 failed F2P test nodes in this run concern undisclosed colleague-held requirements. The case illustrates how unresolved clarifications can persist through delivery despite earlier attempts to obtain an answer.

6.3.3 Implementation and verification of obtained requirements

Requirement acquisition does not guarantee implementation. The carapace-bin task (row 001) illustrates this gap. Claude-Opus-5.5 obtains all 25 colleague-held requirements, yet fails 22 of 35 F2P test nodes. The trajectory reviewer identifies incorrect usages, flags, and candidate descriptions, along with an unchecked build failure. Despite acquiring the requirements, the agent leaves substantial parts of the executable contract unsatisfied.

Translating evidence into working code. Figure 9 examines exploratory associations between the trajectory reviewer's behavioral assessments and per-task F2P. Among the assessed dimensions, evidence-to-code translation shows the strongest association with F2P (Spearman $\rho = +0.36$), followed by causal debugging and verification design (both $\rho = +0.30$). The correlations for colleague-message count and tool-call count are $\rho = +0.21$ and $\rho = -0.21$, respectively.

These exploratory results suggest that effective use of acquired information matters beyond the amount of interaction. Agents may obtain the necessary requirements yet fail to incorporate them correctly into code or verify the resulting behavior.

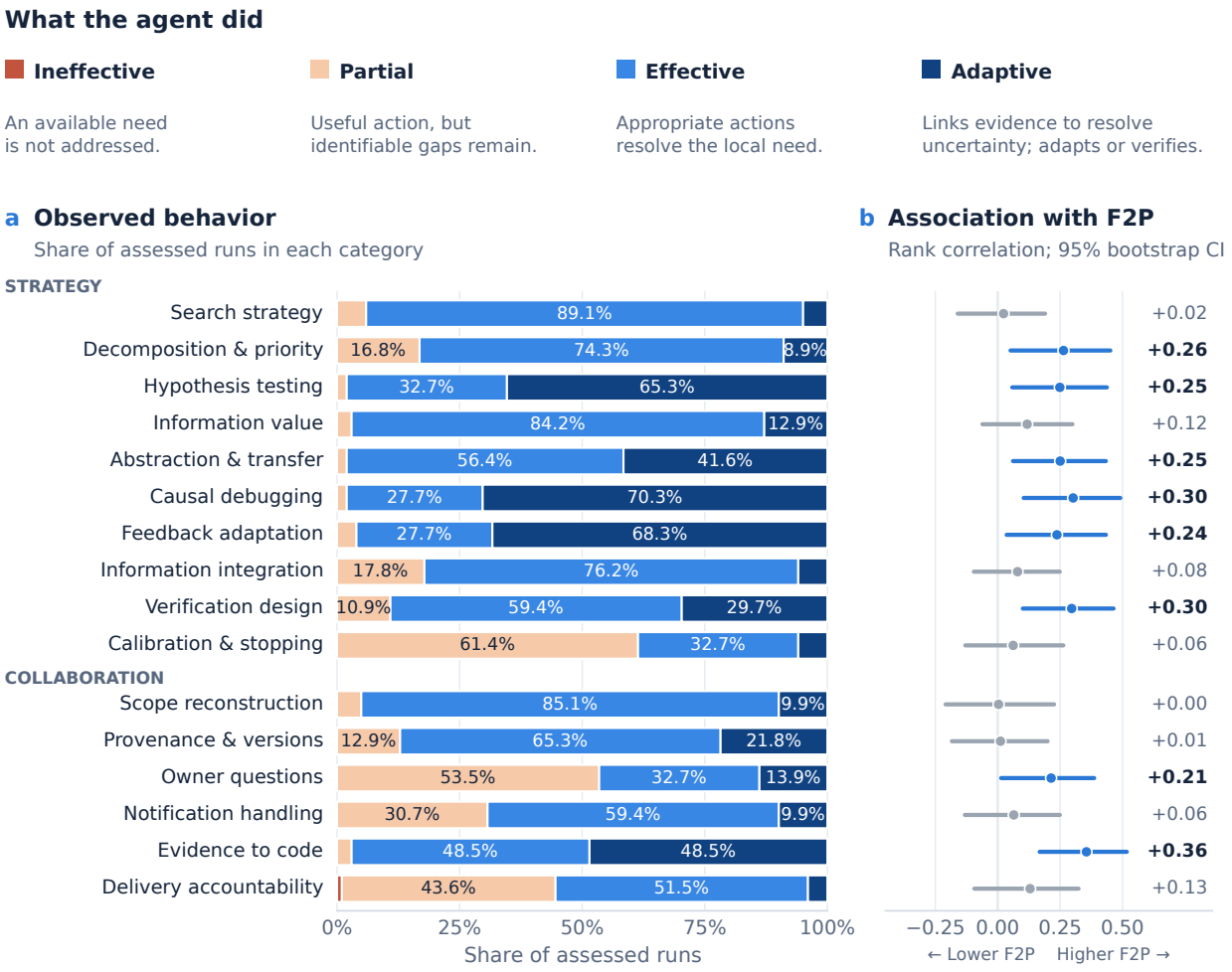


Figure 9 **Claude-Opus-5.5 behavior profile.** (a) Percentages of assessable runs by observed behavior. *Ineffective* actions fail to address an available need; *partial* handling leaves identifiable gaps; *effective* handling resolves the local need; and *adaptive* handling connects evidence across steps to resolve uncertainty and adapt or verify. Each bar is normalized over assessable cases for that dimension and evaluates behavior in its local interaction context. (b) Exploratory Spearman associations with per-task F2P, with 95% bootstrap intervals.

6.3.4 Handoff: stopping decisions and reported status

Stopping before the work is complete. The trajectory reviewer identifies 49 premature stops in which unresolved in-scope requirements still have actionable next steps (Figure 11e in Appendix F.1). Among the 48 autonomous stops in this group, the median remaining execution budget is 2.8 hours. These observations suggest that the agent may terminate assignments despite having both outstanding work and opportunities to continue.

Completion claims diverge from test outcomes. Among 826 requirements reported as completed at handoff, 255 (31%) fail their corresponding tests (Figure 8a). Conversely, many unsatisfied requirements are omitted from the final report rather than explicitly identified as outstanding.

The HotswapAgent task (row 021) illustrates this discrepancy: the agent declares the assignment complete while its own notes still list 8 pending requirements. Together, these observations reveal difficulties in determining when an assignment is complete and communicating its remaining obligations accurately.

6.4 Tracking Requirements Through an Assignment

Across the model comparisons, Oracle-Spec experiment, and trajectory analysis, requirements remain unsatisfied at multiple stages of an assignment. Supplying complete requirements improves partial delivery without ensuring whole-task success, while the trajectory analysis exposes gaps in implementation, verification, and handoff. Together, these findings suggest that reliable delivery requires maintaining the state of individual requirements throughout the engineering workflow.

One direction is to explicitly track each requirement's accountable owner, unresolved questions, implementation status, verification evidence, and handoff claims. Such tracking could help agents revisit outstanding dependencies and distinguish implemented changes from verified outcomes. Whether this approach improves end-to-end delivery remains an open question.

7 Conclusion

We introduced SWE-CoWork, a benchmark that evaluates coding agents working within software organizations, where requirements must be recovered from colleagues and workplace records before being implemented and verified. Our evaluation reveals a substantial gap between partial implementation and complete task delivery. Agents frequently leave clarifications unresolved and, after obtaining requirements, may still fail to satisfy executable tests or report completion inconsistently with verified outcomes.

These findings highlight sustained requirement tracking as a key challenge for autonomous software engineering. Future agents will need to maintain continuity across collaboration, implementation, verification, and handoff to support reliable delegation of engineering work. SWE-CoWork provides a setting for studying these capabilities and can be extended to evaluate teams of autonomous programming agents.

References

Alibaba Cloud Model Studio. Qwen3.8-Max and Qwen3.8-Flash model documentation, 2026. https://www.alibabacloud.com/help/en/model-studio/qwen3-8-max. For Qwen3.8-Flash, see https://www.alibabacloud.com/help/en/model-studio/qwen3-8-flash. Accessed November 13, 2026.

Anthropic. Introducing Claude Opus 5.5, 2026. https://www.anthropic.com/news/claude-opus-5-5. Official model announcement. Accessed October 19, 2026.

ByteDance Seed. Seed2.1 officially released: Advancing AI productivity, 2026. https://seed.bytedance.com/en/seed2_1. Official model page. Accessed October 19, 2026.

Cursor. CursorBench 4.0, September 2026. https://cursor.com/evals. Version 4.0, released September 10, 2026; accessed September 15, 2026.

Snagnik Das, Siddhant Paliwal, and Janak Sunil. Introducing Real-SWE. Specific Labs, September 2026. https://withspecific.com/blog/real-swe. Original release, September 2026; current page presents version 1.1. Accessed November 13, 2026.

Datacurve. DeepSWE v1.1, June 2026. https://deepswe.datacurve.ai/blog/deepswe-v1-1. Updated execution and node-id scoring release; accessed September 15, 2026.

DeepSeek AI. dsh-superagent: Towards a more autonomous agent system. https://github.com/deepseek-ai/dsh-superagent-prototype, 2026a. Research prototype.

DeepSeek AI. DeepSeek-V4.1-Flash release, September 2026b. https://api-docs.deepseek.com/news/news260910. Official release note, September 10, 2026.

DeepSeek AI. DeepSeek-V4-Pro GA release, August 2026c. https://api-docs.deepseek.com/news/news260813. Official release note, August 13, 2026.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can ai agents solve long-horizon software engineering tasks?, 2025. https://arxiv.org/abs/2509.16941.

Nicholas Edwards and Sebastian Schuster. Ask or assume? uncertainty-aware clarification-seeking in coding agents, 2026. https://arxiv.org/abs/2603.26233.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks, 2026. https://arxiv.org/abs/2607.07946.

Naman Jain. How we compare model quality in Cursor, March 2026. https://cursor.com/blog/cursorbench. March 11, 2026; accessed September 15, 2026.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations (ICLR)*, 2024. https://arxiv.org/abs/2310.06770.

Gyuhyeong Kim, Hyojung Gwon, Jeonghyeon Kim, Kyuhong Shim, and Sunjae Lee. RealSWE: A compositional evaluation of coding agents under realistic user requests, August 2026. https://arxiv.org/abs/2608.27831v2. Version 2, August 31, 2026.

Brendan King and Jeffrey Flanigan. Dialogue SWE-Bench: A benchmark for dialogue-driven coding agents, 2026. https://arxiv.org/abs/2606.13995.

Amy J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *29th International Conference on Software Engineering (ICSE)*, pages 344–353, 2007. doi: 10.1109/ICSE.2007.45.

Rishyanth Kondra, Sanket Mhatre, Akshit Kumar, Evan Chu, Bilal Bakht Ahmad, Ayush Nangia, Rajan Agarwal, Arpan Dasgupta, Animesh Sinha, Bhuvanesh Sridharan, Krupa Dave, Brendan Graham, Guanyu Song, Anirudh Rahul, Wei Hern Lim, Abishek Thangamuthu, Ramneet Singh, Danna Liu, Navid Pour, Calvin Chen, and Justus Mattern. FrontierSWE v2. Proximal Blog, 2026. https://www.frontierswe.com/blog/v2. Version 2, 34 tasks, accessed September 15, 2026.

Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, pages 492–501, 2006. doi: 10.1145/1134285.1134355.

Haobin Li, Ping Deng, Weizhong Qian, Liang Jiang, Zhenyu Huang, Mouxing Yang, and Xi Peng. Active-SWE: Benchmarking coding agents for proactive bug fixing without issue reports, 2026. https://arxiv.org/abs/2608.04682.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-Lancer: Can frontier LLMs earn $1 million from real-world freelance software engineering?, 2025. https://arxiv.org/abs/2502.12115.

Moonshot AI. Kimi K3 is here: Our most capable model, July 2026. https://forum.moonshot.ai/t/kimi-k3-is-here-our-most-capable-model/480. Official model announcement, July 22, 2026.

OpenAI. Introducing SWE-bench verified, 2024. https://openai.com/index/introducing-swe-bench-verified/.

OpenAI. GPT-6 Astra model, 2026. https://developers.openai.com/api/docs/models/gpt-6-astra. Official model documentation. Accessed October 8, 2026.

Zhongyuan Peng, Dan Huang, Chuyu Zhang, Caijun Xu, Changyi Xiao, Shibo Hong, David Lo, Lin Qiu, Xuezhi Cao, Jiyuan He, and Yixin Cao. ICAE-Bench: Evaluating coding agents as interactive project builders, 2026. https://arxiv.org/abs/2607.21217.

George Pu, Michael S. Lee, Udari Madhushani Sehwag, David J. Lee, Bryan Zhu, Yash Maurya, Mohit Raghavendra, Yuan Xue, and Samuel Marc Denton. LHAW: Controllable underspecification for long-horizon tasks, 2026. https://arxiv.org/abs/2602.10525.

Mohit Raghavendra, Soham Dan, Miguel Romero Calvo, Yannis Yiming He, Johannes Baptist Mols, Gautam Anand, Cole McCollum, Edgar Arakelyan, Vijay Bharadwaj, Andrew Park, Jeff Da, MohammadHossein Rezaei, Bing Liu, Brad Kenstler, and Yunzhong He. SWE Atlas: Benchmarking coding agents beyond issue resolution, 2026. https://arxiv.org/abs/2605.08366.

Preethi Seshadri, Samuel Cahyawijaya, Ayomide Odumakinde, Sameer Singh, and Seraphina Goldfarb-Tarrant. Lost in simulation: LLM-simulated users are unreliable proxies for human users in agentic evaluations, 2026. https://arxiv.org/abs/2601.17087.

Quan Shi, Keshav Dhandhania, Karthik Narasimhan, and Victor Barres. $\tau^{\intercal}$-bench: An environment for end-to-end, realistic agent construction, 2026. https://arxiv.org/abs/2609.04611.

Jonathan Sillito, Gail C. Murphy, and Kris De Volder. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34(4):434–451, 2008. doi: 10.1109/TSE.2008.26.

The Anthropic Institute. When AI builds itself: Our progress toward recursive self-improvement, and its implications. https://www.anthropic.com/institute/recursive-self-improvement, 2026. Official report. Accessed October 2026.

Volcano Engine. Model release announcements: Doubao-Seed-Evolving, 2026. https://docs.volcengine.com/docs/ark/model-release-announcement?lang=zh. Official release log with dated model updates. Accessed November 13, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025. https://arxiv.org/abs/2407.16741.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents, 2024. https://arxiv.org/abs/2407.01489.

Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44(10): 951–976, 2018. doi: 10.1109/TSE.2017.2734091.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025. https://arxiv.org/abs/2412.14161.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 50528–50652, 2024. https://arxiv.org/abs/2405.15793.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. SWE-bench multimodal: Do ai systems generalize to visual software domains? In *International Conference on Learning Representations (ICLR)*, 2025a. https://arxiv.org/abs/2410.03859.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents, 2025b. https://arxiv.org/abs/2504.21798. Introduces SWE-bench Multilingual (300 instances, 42 repositories, 9 languages).

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. https://arxiv.org/abs/2406.12045.

Z.ai. GLM-5.2 overview, 2026a. https://docs.z.ai/guides/llm/glm-5.2. Official developer documentation. Accessed October 19, 2026.

Z.ai. GLM-5.3 overview, 2026b. https://docs.z.ai/guides/llm/glm-5.3. Official developer documentation. Accessed October 19, 2026.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving, 2025. https://arxiv.org/abs/2504.02605.

Xin Zhou, Chun Yong Chong, Kisub Kim, Yun Peng, Rui Shu, Zihan Wu, Xu Han, Guowen Yuan, Zeyang Zhuang, Jounghoon Kim, Jeongjin Ju, Seongmin Ju, Taein Yoon, and David Lo. A unified issue resolution benchmark for requirement clarification, planning, and code generation for coding agents, 2026. https://arxiv.org/abs/2608.09072.

A Contributions

Leading Authors Xinping Lei

Core Contributors Minghao Liu, Ruijie Zhu, Xiaohui Hu

Contributors Xingqiao Li, Ao Zhang, Wenhao Chai, Han Li, Tao Wang, Jingyuan Zhang, Enduo Zhao, Yunfeng Shi, Yiming Liang, Zetao Yang, Xingjian Wang, Yifan Yao, Xinkai Ma, Yingru Li, Tianhan Yang, Huan Zhou, Weiran Shi, Shen Yan

Corresponding Authors Jiaheng Liu, Stephen Huang, Eli Cheung, Weihao Xuan

B Broader Benchmark Lineage

Table 4 compares repository and interactive benchmarks by their research focus, information sources and evaluation design. Table 1 provides a more detailed comparison of recent benchmarks in terms of task goals, initial inputs and information processes. SWE-bench (Jimenez et al., 2024) established patch evaluation in real repositories using F2P/P2P tests. SWE-bench Verified (OpenAI, 2024), SWE-bench Multimodal (Yang et al., 2025a), Multi-SWE-bench (Zan et al., 2025) and SWE-bench Multilingual (Yang et al., 2025b) extend validation quality, modalities and language coverage. SWE-bench Pro (Deng et al., 2025) evaluates larger changes, SWE-Lancer (Miserendino et al., 2025) draws on paid freelance assignments, and Active-SWE (Li et al., 2026) evaluates proactive bug discovery from repository state.

Benchmark	Focus	Information setting	Evaluation / control
SWE-bench (Jimenez et al., 2024)	Issue resolution	Issue and repository	F2P/P2P tests
SWE-bench Pro (Deng et al., 2025)	Complex software changes	Issue and repository	Repository tests
SWE-Lancer (Miserendino et al., 2025)	Paid freelance work	Task description	End-to-end tests for implementation tasks
Active-SWE (Li et al., 2026)	Proactive bug discovery and fixing	Repository without issue reports	Bug discovery and repair evaluation
TheAgentCompany (Xu et al., 2025)	Professional workflows, including SWE	Company services and multiple colleagues	Programmatic checkpoints; some model-based criteria
LHAW (Pu et al., 2026)	Controlled underspecification	Removed goals, constraints, inputs or context	Inherited evaluators and agent trials
Ask or Assume (Edwards and Schuster, 2026)	Clarification under uncertainty	Full issue held by a user simulator	Full, hidden and interactive conditions
Dialogue SWE-Bench (King and Flanigan, 2026)	Coding through dialogue	Persona-grounded user simulator	Task outcomes and dialogue quality
SWE-RPG (Zhou et al., 2026)	Requirement and planning diagnosis	Issue and repository	Executable patches and intermediate reference annotations
ICAE-Bench (Peng et al., 2026)	Interactive project building	Fuzzy requirements and a user simulator	Behavioral tests and multiple diagnostics
τ^{τ}-Bench (Shi et al., 2026)	Agent construction	Business artifacts and a client	Held-out user simulations; serving constraints
SWE-CoWork (ours)	Autonomous task completion	Distributed evidence, multiple owners and late notices	**Same-task oracle control**, F2P/P2P tests and recovery traces

Table 4 Comparison of repository and interactive benchmarks by research focus, information sources and evaluation design.

C Released Repositories

Table 5 lists the 101 upstream repositories represented in the 101 released tasks, grouped by application domain.

Domain	n	Upstream repositories
Web & Frontend	26	ada-url/ada, api-platform/core, collective/icalendar, emberjs/ember.js, feedbin/feedbin, fomantic/fomantic-ui, gioxx/marvelloussuspender, googleapis/google-api-php-client, hyperf/hyperf, ing-bank/lion, ionic-team/stencil, jasontaylordev/cleanarchitecture, jeremydaly/lambda-api, jnunemaker/httparty, jquery/jquery, kimai/kimai, mika-cn/maoxian-web-clipper, nolimits4web/swiper, python/pythondotorg, rubenlagus/telegrambots, saloonphp/saloon, shane32/qrcoder, tanstack/query, urllib3/urllib3, vacanza-/holidays, webrecorder/pywb
Dev Tools & Compilers	22	ansible/ansible-lint, carapace-sh/carapace-bin, cargo-generate/cargo-generate, cocoapods/xcodeproj, embarkstudios/cargo-deny, firebase/firebase-tools, git-lfs/git-lfs, hotswapprojects/hotswapagent, isomorphic-git/isomorphic-git, jctools/jctools, jest-community/jest-extended, junit-team/junit-examples, kpdecker/jsdiff, mdx-js/mdx, microsoft/vscode-js-debug, nedbat/coveragepy, pallets/click, postcss/postcss, psf/black, serde-rs/serde, stoplightio/elements, thought-machine/please
Data & Databases	11	alibaba/jetcache, clockworklabs/spacetimedb, dolthub/go-mysql-server, elastic/elasticsearch-rails, friendsofsymfony/foselasticabundle, kotlin-orm/ktorm, mongodb/laravel-mongodb, qishibo/anotherredisdesktopmanager, rubyzip/rubyzip, ruflin/elastica, webtorrent/webtorrent
Games & Graphics	10	caffeinemc/sodium, hubs-foundation/hubs, mezz/justenoughitems, mikke89/rmlui, multitheftauto/mtasa-blue, nasa-ammos/3dtilesrendererjs, sergio0694/computesharp, steamre/steamkit, tencent/puerts, underminer-steam/undertalemodtool
Desktop Applications	7	betterdiscord/betterdiscord, bobotig/python-mss, electronnet/electron.net, ipfs/ipfs-desktop, openutau/openutau, pimutils/khal, sozi-projects/sozi
Cloud & Infrastructure	5	boto/boto3, brightercommand/brighter, kubermatic/kubeone, robusta-dev/robusta, stackstorm/st2
Finance & Payments	5	btcsuite/btcd, ledgerhq/ledger-live, mempool/mempool, stripe/stripe-php, stripe/stripe-ruby
ML & AI Systems	4	huggingface/transformers.js, infiniflow/ragflow, lmcache/lmcache, onnx/onnx
Security & Identity	4	apache/caldera, bubka/2fauth, crytic/slither, pglombardo/passwordpusher
Mobile Applications	3	luoyesiqiu/dpt-shell, mygod/vpnhotspot, react-native-async-storage/async-storage
Observability	2	errbit/errbit, newrelic/newrelic-ruby-agent
Scientific Computing	2	angularsen/unitsnet, google-deepmind/mujoco

Table 5 The 101 released tasks by application domain and upstream repository.

D Evidence Case Examples

Figure 10 illustrates the judgments about document history, evidence provenance and scope discussed in Section 4.3. The examples use workplace records from Stencil, controller-runtime and Click. The Click records also appear in the opening comparison in Figure 1, where the issue identifies the output capture requirement in `DOC-11`.

Stencil: preserving an undefined map field. Card `AUR-202` leads to `DOC-13`, whose published correction supersedes an earlier proposal to fill in an empty map. When `map` is `undefined`, the chunk must still retain its other output information. The linked run excerpt reports 22 versus 21 output records but provides no record shapes. The public reply requests the corresponding input to determine whether the relevant map was

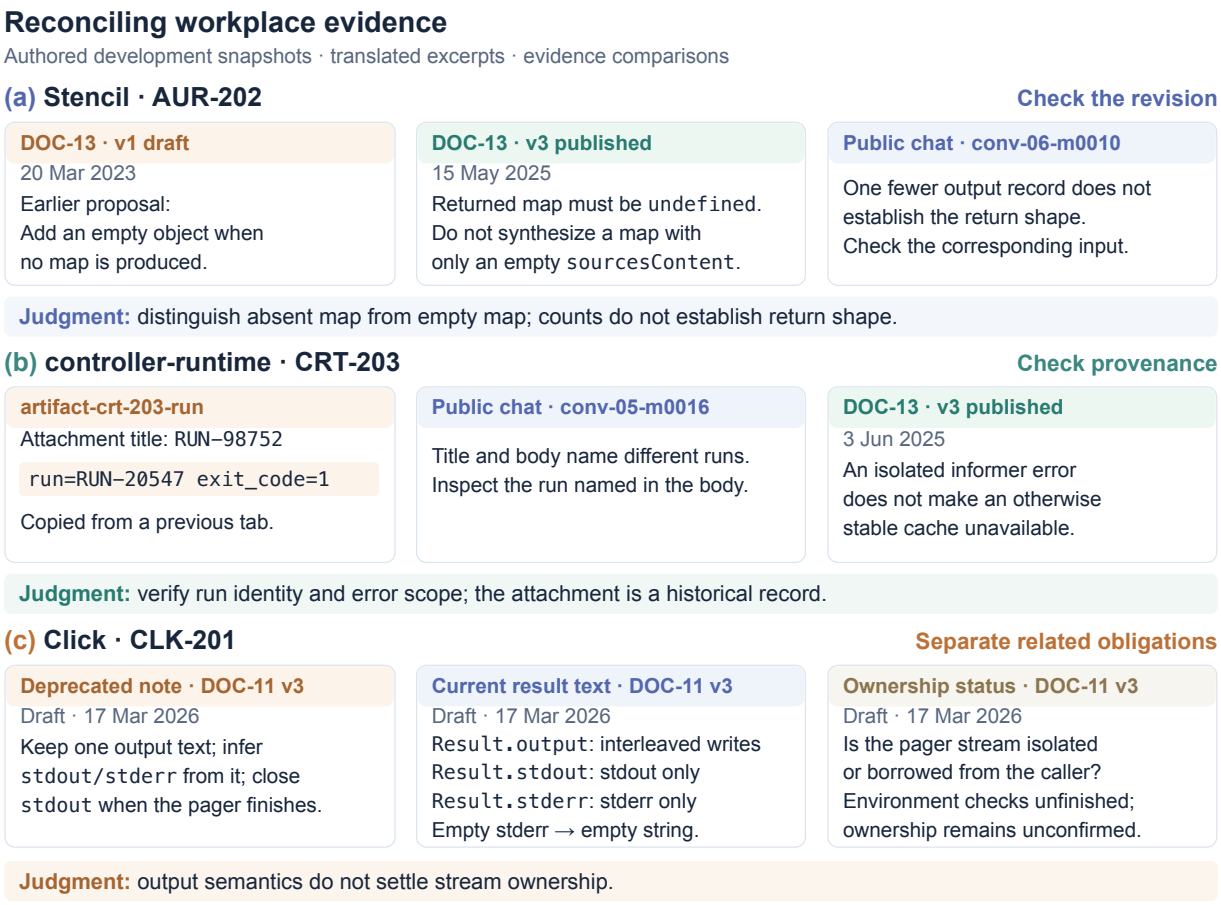


Figure 10 Examples of evidence assessment in workplace records, with source card, document, version and message identifiers. (a) A published correction supersedes an earlier return shape. (b) Run identifiers expose a discrepancy in copied terminal output. (c) Output semantics and an unresolved question about stream ownership.

absent or empty.

controller-runtime: verifying run identity and error scope. Around `CRT-203`, a public message identifies a mismatch between the attachment title and the run identified in copied terminal output. The linked `DOC-13` also corrects an earlier interpretation that an informer error affected the entire cache. The run identifiers establish which execution produced the observation. Assessing the error's scope requires separate evidence about the state of the otherwise stable cache.

Click: checking output semantics and stream ownership. The handoff on `CLK-201` points to `DOC-11`. Its third version, marked draft, distinguishes interleaved output from output attributed to individual sources and retains an earlier, deprecated simplification. An incomplete environment check leaves the pager's stream ownership unresolved. Resolving this question requires identifying the component that supplied the stream and whether the stream remains in use after return.

E Per-Task Statistics

Table 6 reports statistics on software changes, workplace records and information design across the released tasks.

Quantity per task	Mean	Median	P25	P75	Max
Code task					
Verifier test nodes	32.3	32	28	35	68
Fail-to-pass test nodes	26.7	27	23	31	67
Pass-to-pass test nodes	5.6	4	3	8	19
Production files in golden patch	97.5	60	37	133	524
Production lines added	4,590.3	2,955	1,687	5,645	26,605
Production lines deleted	1,752.0	1,069	468	2,217	13,232
Workplace					
Colleagues	26.0	26	26	26	26
Board cards	102.3	102	100	104	107
Card comments	62.2	63	58	66	80
Documents	50.5	50	48	52	56
Chat messages	404.3	405	399	410	417
Merge requests	11.8	12	11	13	14
MR review notes	27.8	28	26	30	35
Run artifacts	49.9	50	47	53	64
Information design					
Scored behavioral facts	26.9	28	24	30	32
Contract atoms pinned by tests	28.6	28	18	34	88
Static contract atoms	11.1	5	5	14	58
Facts held by an owner	22.1	23	19	25	28
Progress-gated deliveries	8.0	8	8	8	8

Table 6 Statistics across the 101 released tasks. Production lines and files count source code, excluding tests, documentation, configuration, generated files, fixtures, lockfiles and vendored code.

F Supplementary Claude-Opus-5.5 Diagnostics

The following tables provide additional results from the Claude-Opus-5.5 trajectory analysis in Section 6.3. They report information acquisition behavior and performance across task strata.

	Ask rate	Targeting	Gain/Q	Follow-thr.	Redundancy	Unread
Claude-Opus-5.5 (CoWork)	80.4	93.3	0.64	78.3	13.7	0.2
Spearman ρ with F2P	+0.11	+0.14	+0.04	+0.13	+0.14	+0.18

Table 7 Claude-Opus-5.5 acquisition behavior. The first row reports means across runs; the second reports Spearman correlations between each acquisition measure and task F2P. Ask rate, targeting, gain per question, follow-through, redundancy and unread disclosures are defined in Section 5. All means are percentages except gain per question.

Axis	Stratum	n	F2P (%)	Passed tasks	Disclosed (%)
Verifier test nodes	< 28	21	70.1	3	76.5
	28–35	55	62.5	2	61.3
	> 35	25	64.7	2	61.9
Edit size	$<$ 1k production lines	6	67.6	0	75.0
	1k–5k	58	66.0	5	65.4
	$>$ 5k	37	62.0	2	61.7
Language	Python	17	69.3	1	62.2
	TypeScript / JavaScript	32	57.3	1	64.1
	PHP / Ruby	19	68.4	1	68.2
	Go / Rust	11	56.5	0	72.6
	C++ / C# / Java / Kotlin	22	72.3	4	60.2
Domain	Web & frontend	26	75.8	4	72.2
	Dev tools & compilers	22	63.9	2	61.8
	Data & databases	11	70.2	1	67.6
	Games & graphics	10	48.9	0	58.5
	Other domains	32	58.9	0	61.4

Table 8 Stratified Claude-Opus-5.5 diagnostics. F2P is the mean of per-task F2P fractions within each stratum. Table 2 reports a pooled F2P score. Passed tasks counts tasks satisfying both F2P and P2P. Disclosed is the mean proportion of requirements held by colleagues that the agent obtained. Edit size counts production lines added by the reference change.

F.1 Run Activity, Stopping, and Assignment Constraints

Run activity. Figure 11 summarizes per-run tool calls, colleague messages, wall-clock usage, F2P outcomes, and stopping judgments by the trajectory reviewer. A median Claude-Opus-5.5 run contacts 6 colleagues and uses 5.0 hours of the 8-hour execution budget. Elapsed time shows little rank association with per-task F2P (Figure 11c).

Changes to existing tests. Among 94 assignment briefs that prohibit changes to existing tests, 15 submitted patches change or remove lines in existing test sources, violating the assignment constraints.

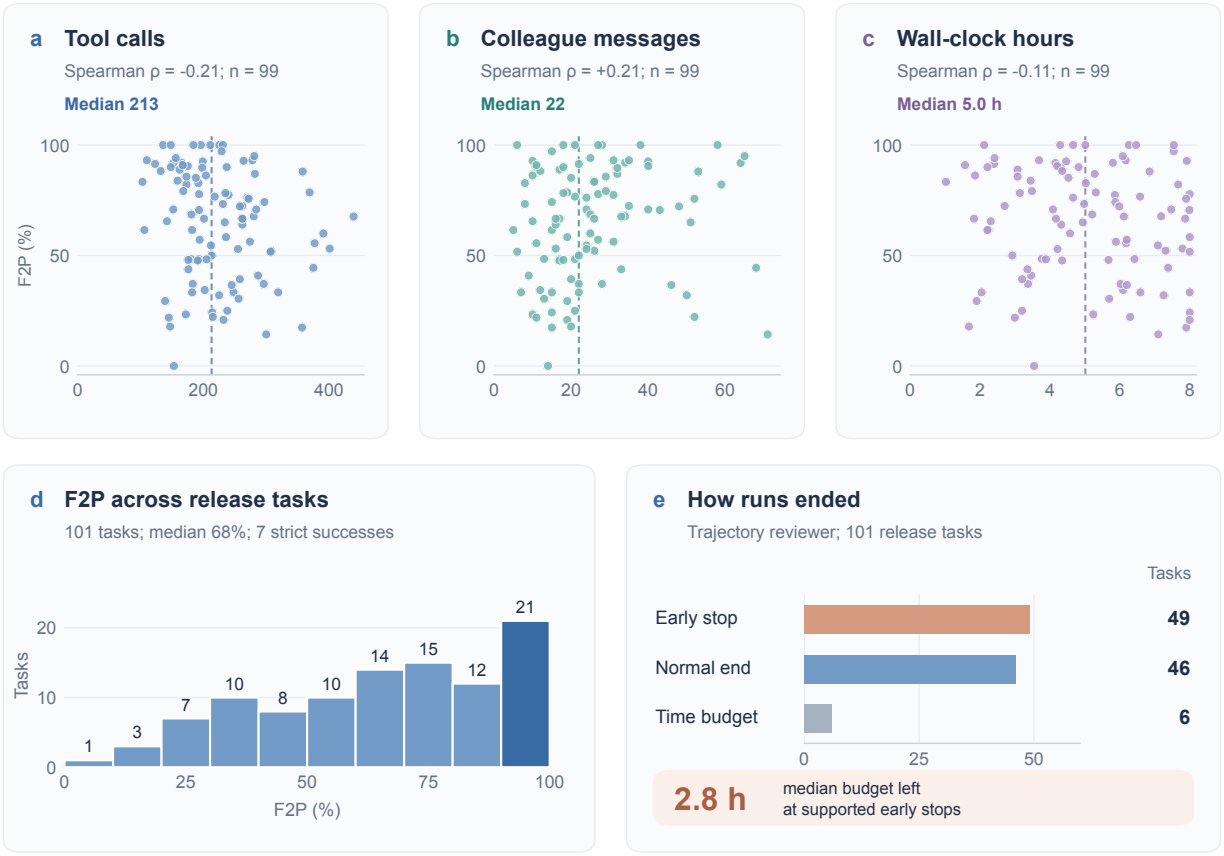


Figure 11 Effort, outcomes and stopping in Claude-Opus-5.5. (a–c) Tool calls, colleague messages and wall-clock hours against F2P fractions with per-run telemetry. (d) Distribution of F2P fractions over the 101-task release cohort. (e) Judgments by the trajectory reviewer of how runs ended, including stops with known, actionable gaps.

G Run Behavior and Requirement Acquisition

Table 9 lists the values plotted in Figure 5, together with the median number of colleague messages and wall-clock hours per run. The run median computes the colleague rate for each run and takes the median, so it can rank models differently from the pooled rate.

	Median per run			Requirements obtained (%)			
Model	Messages	Tool calls	Hours	Colleague	All	Run median	F2P (%)
GPT-6 Astra	18.5	182	3.00	67.8	75.3	71.6	62.91
Claude-Opus-5.5	22	213	5.01	64.2	70.7	72.0	63.84
GLM 5.3	21	785	2.46	58.8	66.3	60.9	31.27
Qwen3.8-Max	6	696	8.00	46.9	56.6	48.7	35.70
Seed Evolving	9	503	4.46	44.3	50.6	38.1	39.09
Qwen3.8-Flash	10	902	5.53	40.1	51.0	38.1	30.70
Kimi K3	8	301	1.41	33.1	45.2	27.3	27.08
DeepSeek V4.1 Flash	4.5	1603	8.00	26.1	39.5	16.0	28.20
Seed 2.1 Pro	8	229	2.50	24.7	36.7	16.3	21.01
GLM 5.2	5	361	2.06	20.4	35.0	12.5	24.40
DeepSeek V4 Pro	0	150	0.90	1.8	23.9	1.5	13.04

Table 9 Run behavior and requirement acquisition. Messages are sent to colleagues through `chat_send`; tool calls include terminal commands and workspace tools; hours are agent wall-clock time, capped at 8. Colleague: share of requirements held only by colleagues that were disclosed. All: disclosed colleague requirements plus public requirements. Run median: median of the per-run colleague rate. Rows are ordered by the colleague rate.

G.1 Interaction Volume and Context Usage

Figures 12 and 13 compare per-run medians of model calls, token usage, tool calls, and context compactions with full-release F2P across the nine models with available usage telemetry. Token-usage medians are unavailable for DeepSeek V4 Pro and Seed 2.1 Pro.

The models exhibit different interaction patterns. Some lower-scoring models terminate after relatively few calls, while others sustain extensive tool use without achieving correspondingly higher F2P scores. Claude-Opus-5.5 and GPT-6 Astra achieve the highest F2P scores despite making relatively few model and tool calls.

Input-token totals also depend on context management. Claude-Opus-5.5 processes a median of 47.57 million input tokens without context compaction, compared with 20.32 million for GPT-6 Astra. These totals include repeated processing of retained context and therefore do not directly measure the extent of workplace exploration.

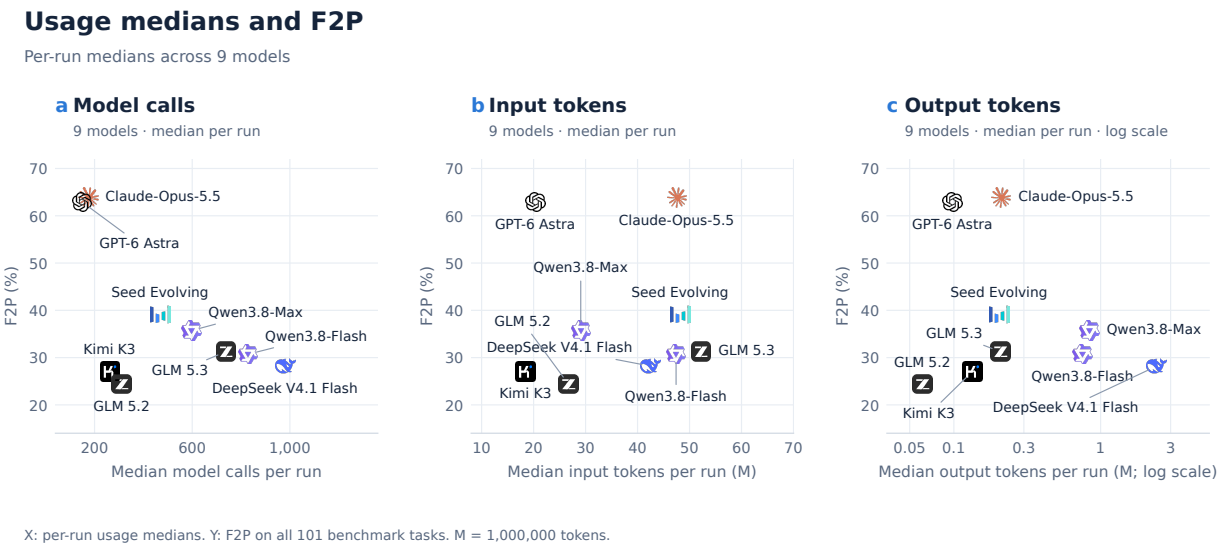


Figure 12 **Model calls, token usage, and delivery.** F2P on the full release against per-run medians of (a) model calls, (b) input tokens, and (c) output tokens. Each point is a model; token axes use millions (M), with a logarithmic scale in (c). Each logo sits at the model's exact value; thin leaders link names to logos in crowded regions.

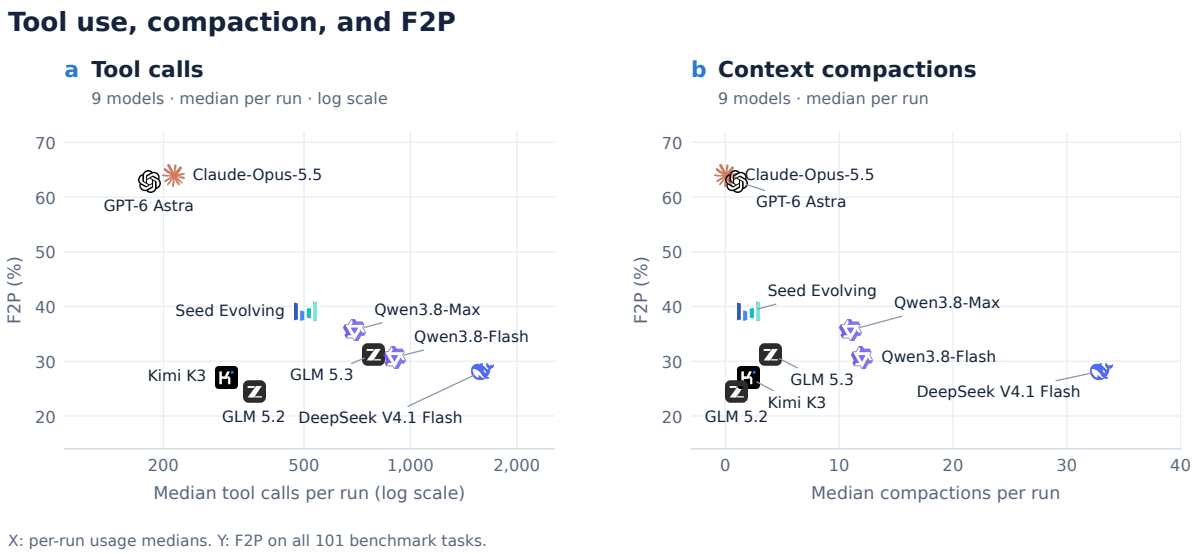


Figure 13 **Tool use, context compaction, and delivery.** Full-release F2P against per-run medians of (a) tool calls and (b) context compactions for nine models. Tool calls use a logarithmic axis; compactions use a linear axis to include zero. Each logo sits at the model's exact value; thin leaders link names to logos in crowded regions.

H Claude-Opus-5.5 Trajectories: Measurement Details

Trajectory archive. The analysis covers the 101-task release cohort. Behavioral measurements come from the available Claude-Opus-5.5 CoWork trajectories. Table 2 reports benchmark outcomes evaluated against the complete test contract. Each trajectory record includes its task version, package hash, run ID, requested models, scaffold version, runtime revision and budget. Runs resumed after a provider interruption are treated as single observations.

Linking tests to requirements. For each F2P test node represented in the trajectory archive, we map the tested requirements by comparing the per-node requirement files, test source code and upstream change titles with the Oracle-Spec for the same package.

Operational loss stages. We use card identifiers, document retrieval records and disclosure events to identify where requirements were missed or lost during information acquisition and use.

Disclosure and visibility. The workplace service logs each requirement disclosed to the agent and links the disclosure to the question that triggered it. A disclosure is considered visible when its reply body appears in a tool result saved in the trajectory. We associate questions with cards using the card identifiers mentioned in the questions. If the agent contacted a requirement's owner but never mentioned its card, we classify the requirement as *owner contacted, card not raised*.

Behavior annotation. The trajectory reviewer, GPT-6 Astra at xhigh reasoning effort, assesses observable actions and outcomes along ten strategy dimensions and six collaboration dimensions. It evaluates ordered trajectories before private grading outcomes are attached. Each dimension uses four ordered categories: *ineffective* (an identifiable need remains unaddressed), *partial* (useful actions leave identifiable gaps), *effective* (appropriate actions resolve the local need) and *adaptive* (the agent connects evidence across steps to resolve uncertainty and adjust or verify its actions).

The categories are encoded as 0–3 for rank analysis. Each dimension is assessed using its own behavioral criteria. When the trajectory provides insufficient evidence or no opportunity to exhibit a behavior, the dimension is marked *not assessable*.

Category percentages are calculated separately for each dimension using assessable runs. Spearman correlations use runs with both an assessable category and an observed per-task F2P score. A trajectory may contribute to one dimension or diagnostic even when another dimension lacks a completed assessment.

For owner-question follow-through, a relevant question that lacks necessary follow-up may be classified as partial. Following the question through to an actionable answer is effective. Resolving consequential ambiguity through linked questions and corroborating evidence is adaptive.

Dimension	Assessed n	I	P	E	A
Search strategy	101	0	6	90	5
Decomposition/prioritization	101	0	17	75	9
Hypothesis testing	101	0	2	33	66
Information value	101	0	3	85	13
Abstraction/transfer	101	0	2	57	42
Causal debugging	101	0	2	28	71
Feedback adaptation	101	0	4	28	69
Information integration	101	0	18	77	6
Verification design	101	0	11	60	30
Calibration/stopping	101	0	62	33	6
Scope reconstruction	101	0	5	86	10
Provenance/version reconciliation	101	0	13	66	22
Owner questions/follow-through	101	0	54	33	14
Notification handling	101	0	31	60	10
Evidence to implementation	101	0	3	49	49
Delivery accountability	101	1	44	52	4

Table 10 Behavior category counts for Claude-Opus-5.5 trajectories in the 101-task release cohort. I: ineffective; P: partial; E: effective; A: adaptive. Each dimension uses its own behavioral criteria. Counts are reported separately by dimension.

Deferrals and rules. A deferred thread is an owner–card pair for which the owner's first reply postpones the answer. A follow-up is recorded when the agent later messages the same owner about the same card, or when the environment records a qualifying follow-up for that thread.

Remaining budget is calculated from the eight-hour limit and elapsed time since the first saved action. Subsequent working time is measured through the final saved action.

For runs whose brief requires test preservation without a migration exception, we assess compliance by counting lines changed or removed from existing test source files in the submitted patch.