Agentic Coding Needs Proactivity, Not Just Autonomy

Nghi D. Q. Bui and Georgios Evangelopoulos
Google Labs

arXiv:2605.06717v1 [cs.SE] 7 May 2026

Coding agents are rapidly changing the landscape of software development, moving from inline completion to autonomous systems that edit repositories, open pull requests, respond to issues, and run scheduled or webhook triggered routines across the development life cycle. The next generation is increasingly described as proactive and long-horizon: agents should notice relevant changes before the developer asks, connect signals across tools, decide when to interrupt, and carry preferences across sessions. Yet the field still lacks a clear account of what proactivity means for software development, how it differs from autonomy, what acceptance criteria proactive long-horizon tasks should satisfy, and which metrics determine whether unsolicited agent behavior is useful rather than merely active. Proactive coding agents should be evaluated by the quality and improvement of their *insight policy*: the policy that decides what matters next, what evidence supports it, whether to show it, and how to adapt after feedback. This view is grounded in the principles of mixed initiative interaction. We propose a three level taxonomy of proactivity (Reactive, Scheduled, and Situation Aware), compare contemporary coding agents against five practical criteria, and sketch an active user simulation protocol with three evaluation targets: Insight Decision Quality (IDQ), Context Grounding Score (CGS), and Learning Lift (LL).

1. Introduction

Coding assistants have moved from inline completion to autonomous coding agents that participate in the full software development life cycle (SDLC) (Hassan et al., 2025; Jin et al., 2024; Li et al., 2025b; Tufano et al., 2024). The next shift is toward proactive agents that continuously absorb repository, toolchain, and workflow context, infer what matters from high-level developer needs, identify emerging problems or opportunities, and decide what to do next before a narrowly specified prompt arrives. This trajectory did not happen in one step: language model agents learned to combine reasoning with external actions (Yao et al., 2023) and executable code changes (Wang et al., 2024a); software engineering gave those agents repository interfaces, real issue-resolution benchmarks, and reusable platforms (Jimenez et al., 2024; Wang et al., 2024b; Yang et al., 2024a); and deployed tools moved them into the places where developers already work. Claude Code (Anthropic, 2025), OpenAI Codex (OpenAI, 2025a), Gemini CLI (Google, 2025a), Jules (Google, 2025e), Google Antigravity (Google, 2025c), and OpenDev (Bui, 2026) place agents in terminals, IDEs, repositories, pull requests, and remote virtual machines. The resulting systems can edit files, run commands, inspect failures, and produce patches; the open question is no longer whether an agent can act, but whether it can decide when action should start and which contribution matters next.

Recent products also move initiation away from the explicit prompt. Cursor Automations (Cursor, 2026; TechCrunch, 2026), Claude Code Routines (Anthropic, 2026b), and Jules Scheduled Tasks (Google, 2025d; Jules, 2026) let coding agents run from schedules, webhooks, GitHub events, integrations, or monitored repository state. These are important triggers, but not yet situation aware proactivity. Triggered runs still differ from agents that notice context shifts on their own, infer whether those shifts matter, and choose whether to notify, question, draft, or stay silent. One stream of work

Correspondence: {nghib, gevang}@google.com

treats proactivity as anticipating useful tasks from user activity and environment state (Lu et al., 2024; Yang et al., 2025a,b); another treats it as asking before acting on uncertain intent (Gan et al., 2024; Sun et al., 2025; Zhou et al., 2026). Both fit the mixed initiative framework of Horvitz (1999): reason about interruption cost, expected utility, and the benefit of leaving the user in control. Public documentation shows substantial autonomy and trigger coverage, but no clear evidence that production coding agents compute interruption cost, treat silence as an explicit action, or update what they show after developer feedback.

Proactive coding agents should be evaluated by the quality and improvement of their *insight policy*, not by autonomous task completion alone. We call the unit of proactive behavior an ***insight***: a context grounded, time sensitive hypothesis about what matters next for the developer, paired with a decision to *notify*, *question*, *draft*, or *stay silent*. This makes the central question sharper. A proactive coding agent is not merely an agent that can do more work. It is an agent that learns which potential contributions deserve the developer's attention, which evidence makes them credible, and which future decisions should change after feedback.

To make this gap precise we propose a three level taxonomy, drawn from Horvitz (1999) and Händler (2023). **Level 1, Reactive** agents run only when prompted. **Level 2, Scheduled** agents run from schedules or predefined triggers and may filter, batch, or rank outputs, but do not learn a cross context, per developer interruption policy. **Level 3, Situation Aware** agents monitor a continuous event stream, compare expected benefit with interruption cost, treat silence as an explicit action, and update a per developer model from feedback. Figure 1 expresses the taxonomy as numbered interaction flows to separate three properties that are easy to conflate: what initiates the agent, whether the developer may respond, and whether that response changes future policy. This distinction matters most for Level 2: scheduled automation may still produce artifacts that developers review, accept, dismiss, or edit, but that response is not equivalent to a learned policy over when to interrupt, ask, draft, or remain silent. The systems audited in Section 4 cluster around Level 2; the gap is judgment: when to interrupt, what to show, and when to remain silent.

Figure 2 sketches a Level 3 engine: context streams into an engine that maintains development state and a developer mental model, **emits *insights*** in the four actions used later (notify, question, draft, stay silent), and learns from response. A Level 3 engine is always deciding, but not always speaking; the quality of its insight policy is what a proactivity benchmark should measure.

The contribution is a framework for making proactive coding agents easier to define, compare, and evaluate. Rather than treating proactivity as a loose product label, the paper turns it into a set of design commitments and measurement targets centered on insights, interruption cost, silence, and learning from developer feedback. Concretely, the paper contributes:

- A three level taxonomy separating autonomy from proactivity in agentic coding (Section 3).
- A comparison of five contemporary coding agents against five practical criteria (Section 4).
- A prototype Level 3 engine organized around *insights* and the policy that selects, grounds, and updates them (Section 3).
- Three insight-stream metrics adapted from active user simulation (Nathani et al., 2026): Insight Decision Quality (IDQ), Context Grounding Score (CGS), and Learning Lift (LL) (Section 4).

2. Background and Related Work

2.1. Coding Agents

Coding agents are becoming a broader software engineering substrate rather than a narrow code-generation tool. After SWE-Bench (Jimenez et al., 2024), issue-resolution work diversified into iterative single-agent systems (Phan et al., 2024; Yang et al., 2024a; Zhang et al., 2024), multi-agent

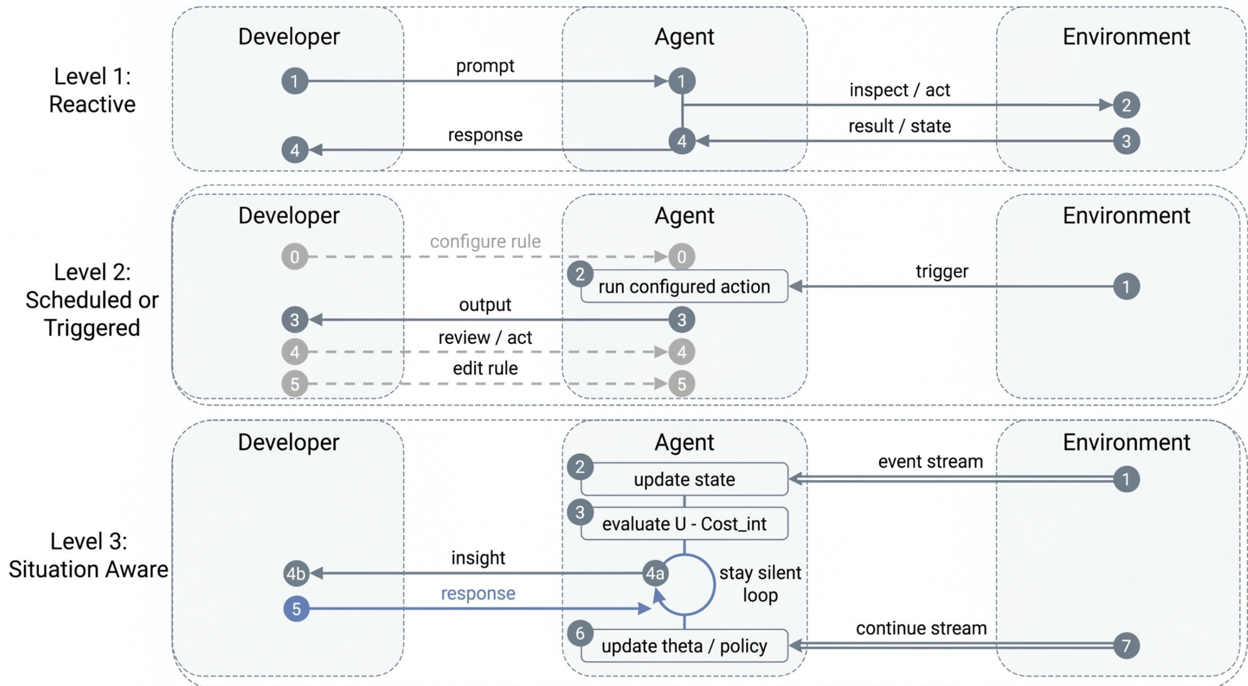

Figure 1 | Three levels of proactivity as numbered interaction flows. Level 1 begins with a developer prompt and ends after the agent response. Level 2 begins with a predefined schedule or event trigger; the developer may respond to the produced artifact, but this response does not automatically update a learned interruption policy. Level 3 monitors context continuously, chooses between staying silent and showing an insight, and uses developer feedback to update future timing, action selection, and framing.

decompositions (Chen et al., 2024; Tao et al., 2024), structured workflows (Xia et al., 2024), open platforms (Wang et al., 2024c), and bug-fix data synthesis for real-world resolution (Pham et al., 2025), as summarized by recent surveys (Hassan et al., 2025; Li et al., 2025a). Another line organizes agents around specifications or role-based collaboration rather than free-form prompts, as in Spec Kit (GitHub, 2025), OpenSpec (Fission AI, 2025), BMAD roles and story files (Blackington, 2025), AgileCoder (Nguyen et al., 2025), and Agentic Software Engineering with mentorship as code and life-cycle management (Hassan et al., 2025). At deployment scale, Li et al. (2025b) analyse 456,000 pull requests from Codex, Devin, Copilot, Cursor, and Claude Code across 61,000 repositories and 47,000 developers, finding faster submission but lower acceptance than human pull requests; the proactivity gap sits inside this real-world acceptance gap. These lines now feed directly into developer-facing tools. Claude Code (Anthropic, 2025), Cursor (Anysphere, 2025), Devin (Cognition, 2024), OpenAI Codex (OpenAI, 2025a), Jules (Google, 2025b), Gemini CLI (Google, 2025a), Goose (Block, 2025), OpenCode (OpenCode Contributors, 2025), Crush (Charm, 2025), Amazon Q Developer (Amazon Web Services, 2024), Cline (Cline, 2025), Aider (Gauthier, 2024), and OpenDev (Bui, 2026) place agents across developer workflows. They are the most relevant audit population here because they already support substantial autonomous work, yet public evidence remains thin on when they should initiate, interrupt, question, draft, or remain silent.

2.2. Defining Proactivity

Horvitz (1999) suggested that an interactive system should reason explicitly about three quantities: the expected benefit of acting for the user, the expected cost of interrupting the user, and the benefit of leaving the user in control. Following Händler (2023) we use *autonomy* for the property that the

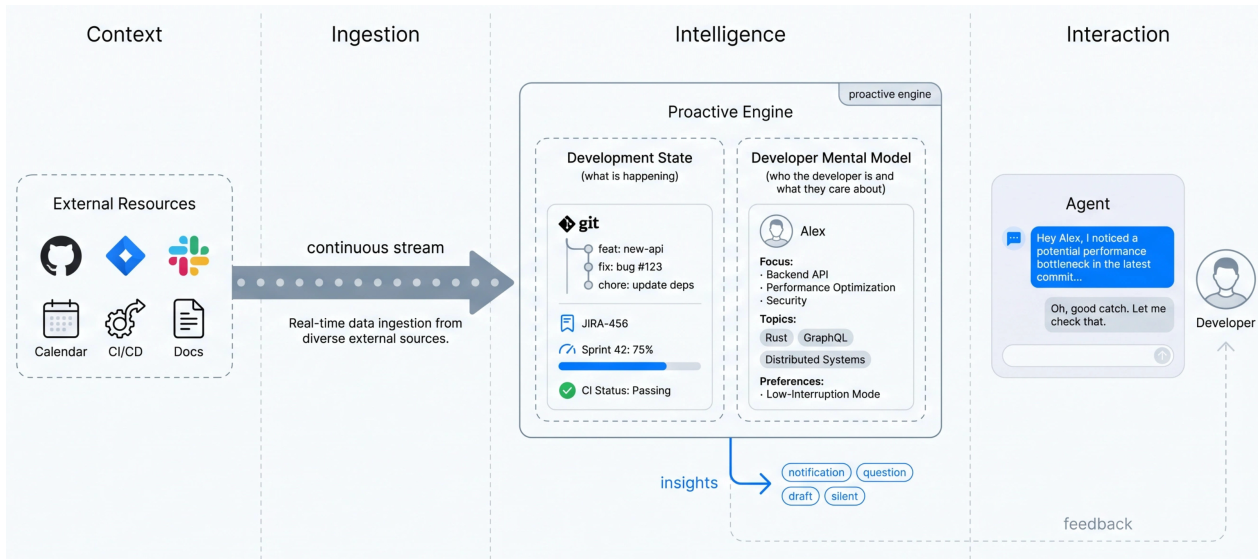

Figure 2 | Prototype proactive agentic coding engine. Context streams into an engine that maintains development state and a developer model, **emits *insights*** (notify, question, draft, stay silent), and learns from response.

agent can act without supervision, and *proactivity* for the property that the agent decides whether and when to act without an explicit prompt. Sapkota et al. (2025) note that recent work often blends the two; we keep them separate throughout.

The *suggestion* stream frames proactivity as anticipating useful tasks: ProactiveBench provides 6,790 human labeled events (Lu et al., 2024); ContextAgent extends to multimodal sensory input (Yang et al., 2025b); ProAgent adds on demand sensing and reports a 33.4 point prediction gain (Yang et al., 2025a); PROBE scores false interruptions and missed opportunities (Pasternak et al., 2025); and ProAgentBench uses 28,000 events from 500 hours of Microsoft 365 sessions with a When and How hierarchy (Tang et al., 2026). The *clarification* stream frames proactivity as asking before acting on uncertain intent: informative questions (Gan et al., 2024), persistent user modeling for stateful SWE Bench and a 12 developer deployment (Zhou et al., 2026), joint productivity-proactivity-personalization training (Sun et al., 2025), deciding when hidden user preferences are relevant (Kaur et al., 2026), and a POMDP rule based on Expected Value of Perfect Information (Suri et al., 2025). Proactive coding should handle both streams across the development life cycle, and long-horizon work has begun to combine them through intent-conditioned monitoring, event-triggered follow-up, and deliberate *stay silent* branches (Shi et al., 2026).

2.3. Reactive Substrates and Active User Simulation

The aspiration predates large language models: early agents targeted workload and information overload (Maes, 1994), direct manipulation concerns (Shneiderman and Maes, 1997), and continuously shown context (Rhodes and Maes, 2000; Rhodes and Starner, 1996; Rhodes, 1997). Current language model agents remain mostly reactive: ReAct loops (Yao et al., 2023), agent computer interfaces (Yang et al., 2024b), code action policies (Wang et al., 2024a), multi agent frameworks (Park et al., 2023; Wu et al., 2023), and task benchmarks such as SWE-Bench (Jimenez et al., 2024), τ (Yao et al., 2024), τ^2 (Barres et al., 2025), and ARE (Froger et al., 2025) all assume that the developer formulates a task rather than asking whether it should have been raised.

The closest prior work to our protocol is PARE, which models applications as finite state machines, simulates users with constrained navigation, and gives user and agent different interfaces under a

Stackelberg POMDP (Brero et al., 2024; Nathani et al., 2026); PARE Bench covers 143 communication, productivity, scheduling, and lifestyle tasks, where frontier models top out near 42 percent.

2.4. Interruption Cost in Software Development

Poorly timed notifications can be net negative (Mehrotra et al., 2016; Okoshi et al., 2015). For developers, on-screen interruptions degrade code comprehension, with recovery ranging from 10–15 minutes for bug fixes to 30–60 minutes for architecture and security tasks (Meyer et al., 2024); longitudinal evidence across 4,910 tasks and 17 developers further shows that self-interruptions can be more disruptive than external ones (Shakeri Hossein Abad et al., 2018). Relevance Theory (Sperber and Wilson, 1995), Grice's Maxims (Grice, 1975), Active Inference (Friston, 2010), and user intent or mental modeling methods (Arora et al., 2024; Berkovitch et al., 2025; Lin et al., 2022; Zhou et al., 2026) supply ingredients for relevance computation.

A state-aware IDE assistant reached 90 percent preference versus 47 percent for a persistent variant in a 65 participant study (Chen et al., 2025). Feedback-delayed LLM code suggestions lifted acceptance from 4.9 to 18.6 percent and cut wasted inference calls by 75 percent over two months (Al Awad et al., 2025). Mental-state inference also helps code understanding (Richards and Wessel, 2024), while CodingGenie implements a Level 2 fixed refresh policy and calls for further study (Zhao et al., 2025). Mozannar et al. (2024) ground both lines in when to show a suggestion. Their taxonomies classify topics; our action space $\mathcal{A}$ classifies intents.

Both streams often assume the agent is invoked for a task and then exits. Proactive coding instead needs a sustained partner that runs across the development life cycle, carries memory across sessions, integrates with the team toolchain, and earns trust over time. Sustained presence enables situation-aware judgment, but not constant interruption. Brady (2026) report a 23 day deployment with append-only memory and ambient self-perception across email and web channels; it illustrates the always-on, not-always-speaking idea in a working system.

3. A Three Level Taxonomy of Proactivity

We draw a three level taxonomy from mixed initiative theory (Horvitz, 1999) and autonomy taxonomies (Händler, 2023). It gives shared vocabulary for comparing agentic coding systems and evaluations of proactive agents.

3.1. A Decision Theoretic Formulation

Let s_t denote developer state at time t (open buffer, branch, recent commits, calendar, sprint deadlines, ticket status, communication context), and let $\mathcal{E}_t$ denote the cross context event stream since the last decision. We group events into *code*, *project*, *communication*, *infrastructure*, and *developer behavior*. Let $\mathcal{A} = \{notify, question, draft, stay\ silent\}$ be the insight action space. The first three actions show an insight; *stay silent* records a deliberate choice not to interrupt. The mixed initiative principles of Horvitz (1999) can be read as:

$$a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \mathbb{E}_{p(o|a,s_t,\mathcal{E}_t)}[U(o;\theta)] \ - \ \mathrm{Cost}_{\mathrm{int}}(s_t, a; \theta), \tag{1}$$

where o is the developer-observable outcome of action a (response, downstream code change, recovered context), $U(o;\theta)$ scores how useful that outcome is to the developer, with θ capturing what the agent has learned about this particular developer, and $\mathrm{Cost}_{\mathrm{int}}(s_t, a; \theta)$ is the cost of breaking flow. The agent should **pick the action whose expected payoff to the developer beats the cost of interrupting them right now**, and pick *stay silent* when no action clears that bar. Interruption cost varies by developer and workflow phase (Mehrotra et al., 2016; Meyer et al., 2024; Okoshi et al., 2015); coding examples include phase-aware gating from typing and error signals (Chen et al., 2025)

and feedback-updated delay policies for LLM code suggestions (Al Awad et al., 2025). A practical Cost_{int} proxy could combine IDE focus state, idle time, edit cadence, test or incident activity, calendar status, and recent dismiss or defer signals, with local scoping for sensitive telemetry. Equation 1 treats *stay silent* as an explicit option, allowing one rule to cover suggestion and clarification.

3.2. Three Levels

We characterize an agentic coding system by which terms of Equation 1 it actually takes into account. Figure 1 shows the levels, mapped into evaluation criteria O1 to O3 in Section 4.

- **Level 1, Reactive.** The agent runs only when the developer starts an interaction. Between requests it has no persistent environmental presence, time model, or candidate action.
- **Level 2, Scheduled.** The agent runs on a schedule or predefined event. It may filter, batch, rank, or summarize within that trigger, but does not learn a cross context, per developer interruption policy; silence is not a learned choice over the full event stream.
- **Level 3, Situation-Aware.** The agent monitors $\mathcal{E}_t$ continuously, computes utility and interruption cost over the full action space, including *stay silent*, and updates θ from developer responses. It decides what to show, when to show it, and whether showing anything is the right action.

3.3. Insights as the Unit of Agent Output

A Level 3 engine continuously evaluates Equation 1 but does not always emit a message. Its action space is: **notify** (state change), **question** (clarification), **draft** (pull request comment, patch, or review thread), and **stay silent** (a deliberate choice not to interrupt). An insight is a context grounded, time sensitive hypothesis about what matters next, paired with one action. An *insight policy* selects the action, chooses evidence, frames any message, and updates future decisions from feedback. The metrics in Section 4 score this stream rather than task completion. The suggestion stream (Lu et al., 2024; Yang et al., 2025a,b) mainly measures *notify* and *draft*; the clarification stream (Gan et al., 2024; Sun et al., 2025; Zhou et al., 2026) mainly measures *question* and *stay silent*. A proactive coding agent, in the spirit of Horvitz (1999), should handle all four.

Acceptance criteria. A candidate insight should satisfy four checks. First, it should be relevant now: the expected developer benefit should exceed interruption cost, and the policy should choose silence when the same fact can safely wait. Second, it should be grounded: a reviewer should be able to recover the files, diffs, tickets, logs, messages, or behavioral signals that support the claim. Third, it should be action matched: *notify* for awareness, *question* for missing intent, *draft* for low ambiguity work, and *stay silent* for low value or poorly timed items. Fourth, it should be learnable: acceptance, dismissal, deferral, edits, and later delegation should change future timing or framing. These checks turn proactivity from a product label into behavior that can be labeled and tested.

The Level 3 claim is meant to be tested. It would be revised by evidence that a coding agent computes a meaningful Cost_{int} from developer state, uses *stay silent* as an explicit action, and updates θ from per developer feedback (O1 - O3, Section 4).

4. Comparing the Landscape and Sketching an Evaluation Protocol

Section 4.1 compares five agentic coding systems against five practical criteria from Section 3. Section 4.2 adapts active user simulation (Nathani et al., 2026) to software development, and Section 4.3 proposes scoring the resulting insight stream rather than task completion. We present no measured results.

4.1. Side by Side Comparison of Five Contemporary Coding Agents

We compare five widely deployed agentic coding systems and one reference architecture in Table 1 using verifiable claims from technical disclosures or reputable reporting: cloud agents, issue-to-pull-

request agents, scheduled and suggested tasks, webhook and integration triggers, memory tools, SWE Bench Verified performance, and ambient-agent reference architectures (Anthropic, 2026a,b; Cursor, 2026; Google, 2025d,f; Jules, 2026; LangChain, 2025a,b; TechCrunch, 2026). We restrict this to software-development systems, while treating adjacent systems with feedback driven topic policy or desktop scheduling as comparators (Anthropic Claude Help Center, 2026; OpenAI, 2025b).

We use the following, marking ✓ for clear documentation, ∼ for partial evidence, and ✗ when no description is available. O1 to O3 are the testable conditions from Section 3.3; O4 and O5 are practical requirements for comparison.

- **O1, Cost of interruption is computed**. The system observes developer state and decides when to show a message.
- **O2, Stay silent is an explicit action** (the inclusion of *stay silent* in the action space $\mathcal{A}$). The system can detect a candidate action and still choose not to show it.
- **O3, Per developer feedback updates the policy**. The system records developer responses and changes future decisions about what to show and when.
- **O4, Cross context observation**. The system ingests events from at least three categories of context (Section 3.1).
- **O5, Initiation channel**. The system can show messages without requiring the developer to open a separate tool.

Table 1 | Comparison of five deployed coding agents and one reference architecture. ✓: clearly documented; ∼: partial; ✗: not found. [†]Conceptual reference architecture.

System	Level	O1	O2	O3	O4	O5
Cursor Background Agents (Cursor, 2026)	1	✗	✗	✗	∼	✓
GitHub Copilot Coding Agent	2	✗	✗	✗	∼	✓
Jules (Google, 2025d,f)	2	✗	✗	✗	∼	✓
Cursor Automations (Cursor, 2026; TechCrunch, 2026)	2	✗	✗	∼	✓	✓
Claude Code Routines (Anthropic, 2026b)	2	✗	✗	✗	∼	✓
LangChain Ambient Agents (LangChain, 2025b)	3[†]	∼	✓	✓	∼	✓

To the best of our finding in surveyed public materials, no deployed coding agent documents a meaningful interruption cost or explicit silence action. O3 varies only weakly: Cursor Automations exposes memory, but its effect on what is surfaced is unclear. Adjacent products show feedback driven and presence gated variants of O1 and O3 (Anthropic Claude Help Center, 2026; OpenAI, 2025b; Zheng et al., 2025), so the gap lies in adapting existing interaction patterns to the specificities of software development, not inventing entirely new or non-existent capabilities. O5 is widely satisfied; the frontier is to trigger breadth, not show judgment.

4.2. Adapting Active User Simulation to Software Development Workflows

Nathani et al. (2026) observe that proactivity cannot be measured on static traces because timing depends on user behavior. Their PARE framework simulates the user under a Stackelberg POMDP (Brero et al., 2024). A coding adaptation would expose the event categories, like the ones in Section 3.1, as state changes and condition a simulated developer on role, commitments, interruption tolerance, and workload.

ClawBench (Zhang et al., 2026) reinforces the need for realistic, multi-step evaluation with traceable trajectories. A coding focused long-horizon benchmark should keep rich event traces but score insight decisions rather than final task completion.

Long-horizon structure. In this setting, long-horizon should not mean that the agent simply runs for many steps. It should mean that useful evidence arrives over time and the right action can change as the situation develops. A scenario should include early weak signals, distractors that should be ignored, later confirming evidence, a moment where silence is correct, and a later response that should affect future behavior. This structure tests whether an agent can wait when evidence is thin, connect signals across tools when the case becomes stronger, and remember feedback without overreacting to one dismissal or acceptance.

In such a benchmark, each scenario would define states $\{s_t, \mathcal{E}_t\}_{t=1}^{T}$, a reference action $a_t^\star \in \mathcal{A}$, support facts $G_t^\star$, and feedback f_t for shown insights. The agent returns a_t and, for *notify*, *question*, or *draft*, a message m_t with recoverable grounding G_t. A seed suite across API breakage, dependency lifecycle, incident response, and routine background work should include silence, showing a message, and later feedback use.

Trace collection. Data should be collected as timestamped, replayable developer-workflow traces rather than as completed task records. Each trace should align repository diffs, issue and pull request events, CI logs, dependency or security notices, communication snippets, and privacy-scoped IDE state such as idle time, active file, edit cadence, and test cadence into a single event ledger. Benchmark creators would then sample decision points from both candidate alerts and quiet intervals, because the denominator for proactivity includes opportunities where the correct action is to remain silent. These traces can be synthesized from public repositories for seed scenarios, but deployment studies should log only scoped metadata or redacted excerpts needed to judge timing and grounding.

Example scenario. A payment provider announces that an older API version will be retired in two weeks. At first, the reference action is *stay silent*: the warning appears only in a release note, the repository has no failing tests, and the developer is working on an unrelated incident. Later, the agent observes that the developer's current branch touches the checkout flow, a Jira ticket in the sprint depends on the same endpoint, CI starts showing deprecation warnings, and a Slack thread confirms that the team plans to migrate this week. The reference action becomes *notify*, grounded in the release note, affected call sites, Jira ticket, CI warning, and Slack thread. If the developer asks the agent to handle routine migration work, a later state can make *draft* the correct action. If the developer dismisses low-priority dependency alerts during focus blocks, LL tests whether the agent learns to delay similar messages while still showing urgent migration risks.

4.3. Evaluating the Insight Stream

SWE Bench (Jimenez et al., 2024) scores what an agent does once a task is given. A Level 3 engine instead produces an insight stream that may be acted on, ignored, or never shown. We score whether that stream keeps the agent on the right path. Let t index a decision point, T the number of decision points in a scenario, s_t the current developer and project state, π the surfacing policy being evaluated, $\mathcal{A} = \{\textit{notify}, \textit{question}, \textit{draft}, \textit{stay silent}\}$ the action space, $a_t^\pi \in \mathcal{A}$ the action selected by that policy, and $a_t^\star$ the reference action.

- **Insight Decision Quality (IDQ)** scores whether the agent chose the right action at the right time:

$$\mathrm{IDQ}(\pi) = \frac{1}{T}\sum_{t=1}^{T} S_{\mathrm{dec}}(a_t^\pi, a_t^\star, s_t). \tag{2}$$

  Here $S_{\mathrm{dec}}(a_t^\pi, a_t^\star, s_t) \in [0, 1]$ is the decision score at time t. It gives full credit to the reference action, partial credit to reasonable but weaker alternatives, and penalties for false interruptions, missed opportunities, wrong action types, and bad timing. When the policy is clear from context, we write IDQ for $\mathrm{IDQ}(\pi)$. Unlike acceptance rate, IDQ scores both shown insights and deliberate silence.

- **Context Grounding Score (CGS)** asks whether a shown insight is supported by the right evidence:

$$\mathrm{CGS} = \frac{1}{|\mathcal{M}|} \sum_{t \in \mathcal{M}} F_1(G_t, G_t^\star) \cdot \mathbb{1}[\mathrm{faithful}(m_t, G_t)], \tag{3}$$

  where $\mathcal{M}$ is the set of times for which a_t^π is not the *stay silent* action, $|\mathcal{M}|$ is the number of shown insights, G_t and $G_t^\star$ are the agent and reference support facts, $F_1(G_t, G_t^\star)$ is the harmonic mean of evidence precision and recall, and $\mathbb{1}[\mathrm{faithful}(m_t, G_t)]$ equals 1 only when the factual claims in m_t are supported by G_t. If $\mathcal{M}$ is empty, CGS should be reported as not applicable rather than forced to zero.
- **Learning Lift (LL)** measures whether feedback improves later decisions:

$$\mathrm{LL} = \mathrm{IDQ}(\pi_{\mathrm{adapted}}) - \mathrm{IDQ}(\pi_{\mathrm{frozen}}). \tag{4}$$

  Here π_{adapted} may update from earlier developer feedback f_t, while π_{frozen} cannot. Both are evaluated on the same later decision points, so positive LL means feedback improved future timing or action choice rather than merely repeating the previous message.

Together, the metrics ask whether the agent chose the right action, used the right evidence, and improved after feedback. Repository outcomes remain useful secondary checks, but the primary unit should remain the insight decision.

Annotation protocol. A useful benchmark requires two labeling layers. *Action selection* labels the reference action $a_t^\star$ and acceptable alternatives at each decision point. *Grounding verification* labels support facts $G_t^\star$ and factual claims in m_t, making CGS checkable when an insight omits critical context or uses an alternate but sufficient evidence set. Both layers should allow multiple correct answers because developers differ in interruption tolerance and preference for detail.

Metric validity. IDQ, CGS, and LL should be validated as repeatable labeling protocols. LLM labelers can pre-segment traces, propose actions and support facts, and flag disagreements, but human reviewers should audit final labels and report agreement. Traces should include recurring scenarios so LL can compare adapted and frozen policies on held out later decisions.

Policy interpretation. Scores should be read together. High IDQ and low CGS means the agent has timing judgment but weak justification; high CGS and low IDQ means it can ground messages but surfaces them poorly. Positive LL shows useful adaptation, while negative LL reveals overfitting to feedback.

Validity limits. These metrics are evaluation targets rather than a completed benchmark. A full benchmark would need reference insight labels for IDQ, support facts for CGS, recurring but non-identical situations for LL, annotator agreement, and held out variants for measuring learning. The metrics should be reported together rather than collapsed into one ranking number.

5. Discussion

The framework above makes proactivity testable before a fully deployed Level 3 coding agent exists. The key question is not only whether an agent completes more tasks, but whether it notices the right situation, chooses an appropriate moment, explains its evidence, and learns from feedback. This shifts attention from task completion alone to the infrastructure needed to observe candidate insights, silent decisions, and later developer responses.

Timing and silence. Interruption cost is not a constant property of a notification; it depends on what the developer is doing, how urgent the event is, and how expensive recovery would be (Meyer et al., 2024). A proactive agent therefore needs a learned $\mathrm{Cost}_{\mathrm{int}}$ estimate that uses observable signals across

tools and avoids cold-start messages whose value is lower than their attention cost. A practical study could compare three policies: show every detected issue, show only after an idle threshold, and show only when learned Cost_{int} permits it. The important design choice is to log both shown messages and candidate messages that were withheld, since silence is meaningful only if the evaluator can see what the agent declined to surface. IDQ, CGS, and LL would then measure timing, grounding, and adaptation.

Communication. A shown insight also has to be framed at the right level of detail. Code-review automation suggests that comment quality matters more than volume, but proactive agents still need a measurable account of when evidence helps and when it increases reading burden. A benchmark could compare a headline, a short reason, a diff-grounded reason, and an interactive version that lets the developer request more context. IDQ, CGS, and time to decision would expose both under-explaining and over-explaining.

Benchmarking. Existing benchmarks provide important pieces, but none evaluates monitoring, relevance, timing, framing, and silence as one combined behavior. ProactiveBench (Lu et al., 2024), ContextAgent (Yang et al., 2025b), and ProAgent (Yang et al., 2025a) emphasize suggestion; Gan et al. (2024) and Zhou et al. (2026) emphasize clarification; PARE Bench (Nathani et al., 2026) studies general productivity; and ARE (Froger et al., 2025) and τ^2 (Barres et al., 2025) support reactive-agent evaluation. A coding benchmark should release replayable traces, IDQ, CGS, and LL labels, and comparable implementations where public APIs permit. The trace format matters: repository, issue, CI, communication, and privacy-scoped IDE signals should be aligned by time so that evaluators can judge what the agent knew before it acted. The result should support diagnosis rather than reduce developer experience to one leaderboard number.

Evaluation data collection. The most useful evaluation data would come from prospective logging rather than retrospective issue mining. During normal work, the system should record a bounded event ledger, run the insight policy in shadow mode, and store candidate decisions even when nothing is shown. For each sampled point, annotators should see only the evidence available before that time, then label the best action, acceptable alternatives, required support facts, and whether a developer response later changed the preferred policy. This design avoids two common shortcuts: treating every completed task as a missed proactive opportunity, and treating every alert dismissal as proof that the alert was wrong. It also gives LL a real denominator, because the benchmark can compare what the adapted policy would do after feedback with what a frozen policy would have done on related later events. For privacy, raw IDE snapshots and chat text need not be released; benchmarks can store hashed event identifiers, redacted evidence windows, and provenance for each visible excerpt. Evaluation logging can be broader than what the product surfaces, but labels must preserve temporal fidelity and use only information available before the decision.

Product design. A Level 3 coding agent should be an accountable decision surface, not merely a more persistent Level 2 automation. Its interface needs an inbox or agent-management view that makes shown and deferred insights inspectable, lets developers accept, edit, defer, or dismiss suggestions, and records those responses as feedback. The interface should also show enough evidence for the developer to understand why an insight appeared now, while keeping detailed behavioral telemetry out of team-visible channels. This requires a boundary between private developer state and shared project context: interruption tolerance and dismissal patterns should remain private, while evidence such as failing tests, affected files, and linked issues should remain auditable.

Reporting requirements. Future systems should report more than task success and trigger coverage. A credible proactivity claim should state which event streams were observed, how candidate insights were filtered, how often the system chose to *stay silent*, what feedback was collected, and whether feedback changed later surfacing decisions. Silence rates, delayed-message rates, feedback categories,

and learning lift are most useful with the denominator of candidate insights considered but not shown.

Limitations. This paper is a position and measurement proposal. The audit relies on public documentation, and O1 to O5 are practical criteria rather than formal guarantees. Simulated developers can diverge from real users (Sun et al., 2025; Zhou et al., 2026), and interruption findings from real-time work may not transfer cleanly to asynchronous agent messages (Meyer et al., 2024). These limits are reasons to report uncertainty and treat proactivity as a property involving real developers rather than a benchmark score alone.

6. Conclusion

Coding agents are maturing from tools that execute prompted tasks into partners that may notice drift, uncertainty, or emerging opportunities before being asked. Proactivity in agentic coding should be judged not by how often an agent acts, but by whether it surfaces the right insight at the right time, with enough evidence, and stays silent when intervention is unwarranted. Our three level taxonomy separates reactive execution, scheduled triggering, and situation-aware judgment; the Level 3 engine sketch centers the insight as the unit of proactive behavior; and IDQ, CGS, and LL make that behavior measurable. Together, these pieces frame proactive coding as accountable mixed initiative collaboration grounded in timing, relevance, evidence, and trust.

References

M. N. Al Awad, S. Ivanov, and O. Tikhonova. Optimizing LLM code suggestions: Feedback-driven timing with lightweight state bounds. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2025. doi: 10.1109/ASEW67777.2025.00049. URL https://arxiv.org/abs/2511.18842.

Amazon Web Services. Amazon q developer. https://aws.amazon.com/q/developer/, 2024.

Anthropic. Claude code: An agentic coding tool. https://docs.anthropic.com/en/docs/claude-code/overview, 2025.

Anthropic. 2026 agentic coding trends report, 2026a. URL https://resources.anthropic.com/2026-agentic-coding-trends-report.

Anthropic. Introducing routines in claude code, 2026b. URL https://claude.com/blog/introducing-routines-in-claude-code.

Anthropic Claude Help Center. Schedule recurring tasks in claude cowork, 2026. URL https://support.claude.com/en/articles/13854387-schedule-recurring-tasks-in-claude-cowork.

Anysphere. Cursor: The ai code editor. https://cursor.com, 2025.

G. Arora, S. Jain, and S. Merugu. Intent detection in the age of llms, 2024. URL https://arxiv.org/abs/2410.01627.

V. Barres, H. Dong, S. Ray, X. Si, and K. Narasimhan. tau2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

O. Berkovitch, S. Caduri, N. Kahlon, A. Efros, A. Caciularu, and I. Dagan. Identifying user goals from ui trajectories, 2025. URL https://arxiv.org/abs/2406.14314.

A. Blackington. BMAD-METHOD: Breakthrough method for agile AI-driven development. https://github.com/bmad-code-org/BMAD-METHOD, 2025.

Block. Goose: An open-source ai agent. https://github.com/block/goose, 2025.

S. Brady. Springdrift: An auditable persistent runtime for LLM agents with case-based memory, normative safety, and ambient self-perception, 2026. URL https://arxiv.org/abs/2604.04660.

G. Brero, A. Eden, D. Chakrabarti, M. Gerstgrasser, A. Greenwald, V. Li, and D. C. Parkes. Stackelberg pomdp: A reinforcement learning approach for economic design, 2024. URL https://arxiv.org/abs/2210.03852.

N. D. Bui. Building effective ai coding agents for the terminal: Scaffolding, harness, context engineering, and lessons learned, 2026. URL https://arxiv.org/abs/2603.05344.

Charm. Crush: An agentic coding tool for the terminal. https://github.com/charmbracelet/crush, 2025.

D. Chen et al. Coder: Issue resolving with multi-agent and task graphs. *arXiv preprint arXiv:2406.01304*, 2024.

V. Chen, A. Zhu, S. Zhao, H. Mozannar, D. Sontag, and A. Talwalkar. Need help? designing proactive ai assistants for programming. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2025.

Cline. Cline: Ai-powered coding in vs code. https://github.com/cline/cline, 2025.

Cognition. Introducing devin, the first ai software engineer. https://cognition.ai/blog/introducing-devin, 2024.

Cursor. Build agents that run automatically (automations), 2026. URL https://cursor.com/blog/automations.

Fission AI. OpenSpec: Spec-driven development framework. https://github.com/Fission-AI/OpenSpec, 2025.

K. Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2): 127–138, 2010.

R. Froger, P. Andrews, M. Bettini, et al. Are: Scaling up agent environments and evaluations, 2025. URL https://arxiv.org/abs/2509.17158.

Y. Gan, C. Li, J. Xie, L. Wen, M. Purver, and M. Poesio. Clarq-llm: A benchmark for models clarifying and requesting information in task-oriented dialog, 2024. URL https://arxiv.org/abs/2409.06097.

P. Gauthier. Aider: Ai pair programming in your terminal. https://aider.chat, 2024.

GitHub. Spec kit: Toolkit for spec-driven development. https://github.com/github/spec-kit, 2025.

Google. Gemini cli. https://github.com/google-gemini/gemini-cli, 2025a.

Google. Jules: Google's ai coding agent. https://blog.google/technology/google-labs/jules/, 2025b.

Google. Introducing Google Antigravity, a new era in AI-assisted software development, 2025c. URL https://antigravity.google/blog/introducing-google-antigravity.

Google. New updates make Jules a proactive AI partner, 2025d. URL https://blog.google/innovation-and-ai/models-and-research/google-labs/jules/.

Google. Build with Jules, your asynchronous coding agent, 2025e. URL https://blog.google/technology/developers/jules-proactive-updates/.

Google. Jules, 2025f. URL https://jules.google.com/.

H. P. Grice. Logic and conversation. In P. Cole and J. L. Morgan, editors, *Syntax and Semantics, Vol. 3: Speech Acts*, pages 41–58. Academic Press, 1975.

T. Händler. Balancing autonomy and alignment: A multi-dimensional taxonomy for autonomous llm-powered multi-agent architectures, 2023. URL https://arxiv.org/abs/2310.03659.

A. E. Hassan, H. Li, D. Lin, B. Adams, T.-H. Chen, Y. Kashiwa, and D. Qiu. Agentic software engineering, foundational pillars and a research roadmap. *arXiv preprint arXiv:2509.06216*, 2025.

E. Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 159–166. ACM Press, 1999.

C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen. From LLMs to LLM-based agents for software engineering: A survey of current, challenges and future, 2024. URL https://arxiv.org/abs/2408.02479.

G. Jules. Jules suggested tasks, 2026. URL https://jules.google/docs/suggested-tasks/.

K. Kaur, V. Gupta, A. Gupta, and C. Shah. The PROPER approach to proactivity: Benchmarking and advancing knowledge gap navigation, 2026. URL https://arxiv.org/abs/2601.09926.

LangChain. agents-from-scratch: Ambient agent email assistant, 2025a. URL https://github.com/langchain-ai/agents-from-scratch.

LangChain. Introducing ambient agents, 2025b. URL https://blog.langchain.com/introducing-ambient-agents/.

C. Li et al. Advances and frontiers of llm-based issue resolution in software engineering: A comprehensive survey. *arXiv preprint*, 2025a.

H. Li, H. Zhang, and A. E. Hassan. The rise of AI teammates in software engineering (SE 3.0): How autonomous coding agents are reshaping software engineering, 2025b. URL https://arxiv.org/abs/2507.15003.

J. Lin, D. Fried, D. Klein, and A. Dragan. Inferring rewards from language in context, 2022. URL https://arxiv.org/abs/2204.02515.

Y. Lu, S. Yang, C. Qian, G. Chen, Q. Luo, Y. Wu, H. Wang, X. Cong, Z. Zhang, Y. Lin, W. Liu, Y. Wang, Z. Liu, F. Liu, and M. Sun. Proactive agent: Shifting LLM agents from reactive responses to active assistance, 2024. URL https://arxiv.org/abs/2410.12361.

P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7): 30–40, 1994.

A. Mehrotra, V. Pejovic, J. Vermeulen, R. Hendley, and M. Musolesi. My phone and me: Understanding people's receptivity to mobile notifications. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1021–1032. ACM, 2016.

A. N. Meyer, L. E. Barton, G. C. Murphy, M. Züger, and T. Fritz. Breaking the flow: A study of interruptions during software engineering activities. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2024.

H. Mozannar, G. Bansal, A. Fourney, and E. Horvitz. When to show a suggestion? integrating human feedback in AI-assisted programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

D. Nathani, C. Zhang, C. Huan, J. Shan, Y. Yang, A. Patel, Z. Gan, W. Y. Wang, M. Saxon, and X. E. Wang. Proactive agent research environment: Simulating active users to evaluate proactive assistants. *arXiv preprint arXiv:2604.00842*, 2026.

M. H. Nguyen, T. P. Chau, P. X. Nguyen, and N. D. Bui. Agilecoder: Dynamic collaborative agents for software development based on agile methodology. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pages 156–167. IEEE, 2025.

T. Okoshi, J. Ramos, H. Nozaki, J. Nakazawa, A. K. Dey, and H. Tokuda. Reducing users' perceived mental effort due to interruptive notifications in multi-device mobile environments. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, pages 475–486. ACM, 2015.

OpenAI. Introducing codex. https://openai.com/index/introducing-codex/, 2025a.

OpenAI. Introducing chatgpt pulse, 2025b. URL https://openai.com/index/introducing-chatgpt-pulse/.

OpenCode Contributors. Opencode: Ai-powered terminal assistant. https://github.com/opencode-ai/opencode, 2025.

J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

G. Pasternak, D. Rajagopal, J. White, D. Atreja, M. Thomas, G. Hurn-Maloney, and A. Lewis. Beyond reactivity: Measuring proactive problem solving in LLM agents, 2025. URL https://arxiv.org/abs/2510.19771.

M. V. Pham, H. N. Phan, H. N. Phan, C. L. Chi, T. N. Nguyen, and N. D. Bui. Swe-synth: Synthesizing verifiable bug-fix data to enable large language models in resolving real-world bugs. *arXiv preprint arXiv:2504.14757*, 2025.

H. N. Phan, T. N. Nguyen, P. X. Nguyen, and N. D. Bui. Hyperagent: Generalist software engineering agents to solve coding tasks at scale. *arXiv preprint arXiv:2409.16299*, 2024.

B. Rhodes and P. Maes. Just-in-time information retrieval agents. *IBM Systems Journal*, 39:685–704, 2000.

B. Rhodes and T. Starner. Remembrance agent: A continuously running automated information retrieval system. In *Proceedings of the First International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM)*, 1996.

B. J. Rhodes. The wearable remembrance agent: a system for augmented memory. In *Digest of Papers. First International Symposium on Wearable Computers*, pages 123–128, 1997.

J. Richards and M. Wessel. What you need is what you get: Theory of mind for an LLM-based code understanding assistant, 2024. URL https://arxiv.org/abs/2408.04477.

R. Sapkota, K. I. Roumeliotis, and M. Karkee. AI agents vs. agentic AI: A conceptual taxonomy, applications and challenges. *Information Fusion*, 2025.

Z. Shakeri Hossein Abad, O. Karras, K. Schneider, K. Barker, and M. Bauer. Task interruption in software development projects: What makes some interruptions more disruptive than others?, 2018. URL https://arxiv.org/abs/1805.05508.

Q. Shi, D. Wang, H. Zhou, J. Li, J. Xu, J. Gao, J. Hao, and R. He. Long-term task-oriented agent: Proactive long-term intent maintenance in dynamic environments, 2026. URL https://arxiv.org/abs/2601.09382.

B. Shneiderman and P. Maes. Direct manipulation vs. interface agents. *Interactions*, 4(6):42–61, 1997.

D. Sperber and D. Wilson. *Relevance: Communication and Cognition*. Blackwell, 2nd edition, 1995.

W. Sun, X. Zhou, W. Du, X. Wang, S. Welleck, G. Neubig, M. Sap, and Y. Yang. Training proactive and personalized llm agents, 2025. URL https://arxiv.org/abs/2511.02208.

M. Suri, P. Mathur, N. Lipka, F. Dernoncourt, R. A. Rossi, and D. Manocha. Structured uncertainty guided clarification for LLM agents, 2025. URL https://arxiv.org/abs/2511.08798.

Y. Tang, H. Tang, T. Cao, L. Nguyen, A. Zhang, X. Cao, C. Liu, W. Ding, and Y. Li. ProAgentBench: Evaluating LLM agents for proactive assistance with real-world data, 2026. URL https://arxiv.org/abs/2602.04482.

W. Tao et al. Magis: Llm-based multi-agent framework for github issue resolution. *arXiv preprint arXiv:2403.17927*, 2024.

TechCrunch. Cursor is rolling out a new kind of agentic coding tool, 2026. URL https://techcrunch.com/2026/03/05/cursor-is-rolling-out-a-new-system-for-agentic-coding/.

M. Tufano, A. Agarwal, J. Jang, R. Zilouchian Moghaddam, and N. Sundaresan. Autodev: Automated AI-driven development, 2024. URL https://arxiv.org/abs/2403.08299.

X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji. Codeact: Your llm agent acts better when generating code. In *ICML*, 2024a. URL https://arxiv.org/abs/2402.01030.

X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Zhang, Y. Yang, S. Yao, B. Vasilescu, and G. Neubig. Openhands: An open platform for ai software developers as generalist agents, 2024b. URL https://arxiv.org/abs/2407.16741.

X. Wang et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024c.

Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

C. S. Xia et al. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

B. Yang, L. Xu, L. Zeng, Y. Guo, S. Jiang, W. Lu, K. Liu, H. Xiang, X. Jiang, G. Xing, and Z. Yan. ProAgent: Harnessing on-demand sensory contexts for proactive LLM agent systems, 2025a. URL https://arxiv.org/abs/2512.06721.

B. Yang, L. Xu, L. Zeng, K. Liu, S. Jiang, W. Lu, H. Chen, X. Jiang, G. Xing, and Z. Yan. ContextAgent: Context-aware proactive LLM agents with open-world sensory perceptions, 2025b. URL https://arxiv.org/abs/2505.14668.

J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024a. URL http://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering, 2024b. URL https://arxiv.org/abs/2405.15793.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=WE_vluYUL-X.

S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. tau-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1592–1604. ACM, 2024. doi: 10.1145/3650212.3680384.

Y. Zhang, Y. Wang, Y. Zhu, P. Du, J. Miao, X. Lu, W. Xu, Y. Hao, S. Cai, X. Wang, et al. Clawbench: Can ai agents complete everyday online tasks? *arXiv preprint arXiv:2604.08523*, 2026.

S. Zhao, A. Zhu, H. Mozannar, D. Sontag, A. Talwalkar, and V. Chen. Codinggenie: A proactive llm-powered programming assistant. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pages 1168–1172, 2025.

J. Zheng, H. Weng, X. Wang, C. Cui, S. Mayer, C.-l. Tai, and L.-H. Lee. PersoNo: Personalised notification urgency classifier in mixed reality, 2025. URL https://arxiv.org/abs/2508.19622.

X. Zhou, V. Chen, Z. Z. Wang, G. Neubig, M. Sap, and X. Wang. ToM-SWE: User mental modeling for software engineering agents, 2026. URL https://arxiv.org/abs/2510.21903.