Automating the Software Development Lifecycle with LLM Coding Agents: A Webhook-Based Approach

Arthur Claro[1]

[1]Universidade Católica de Brasília (UCB)
Brasília – DF – Brazil

arthur.rego@a.ucb.br

Abstract. *This paper reports on the design and deployment of a system that integrates an autonomous Large Language Model (LLM) coding agent with a corporate issue tracker. Moving a ticket to a dedicated status, or mentioning the agent in a comment, triggers a webhook that hands the task to the agent. The agent reads the ticket, creates a branch, implements the change, runs lint, build and the test suite, opens a pull request (PR), and updates the ticket. We describe the event-driven architecture (webhook server, secure edge routing, process supervision and per-card concurrency control) and its incremental deployment in a real software organization. Across two detailed observation windows we reconstructed several dozen runs; seven exercised the full status-triggered flow. A third window is reported in aggregate only. Two non-test tickets reached merge, one with no human code rework and one after two rounds of review; the remaining status runs were declined test PRs, a still-open pull request, a mid-run API-quota abort, and one killed by the 30-minute watchdog after a concurrency conflict between the two triggers. The agent opened PRs in about fifteen minutes, while end-to-end time to merge was measured in days and dominated by human review, so this reflects latency to review rather than total cost. The twelve runs we could price in the second window cost US$ 23.26 in total. A recurring lesson is that most of the engineering effort went into operational integration, such as the network edge, TLS termination and event delivery, rather than into the agent's reasoning.*

1. Introduction

Contemporary software engineering mixes design work with routine, cognitively undemanding tasks. Turning a task description into code, branching, running tests, opening a pull request (PR) and updating the issue tracker are simple steps, but they consume time and force frequent context switches.

Large Language Models (LLMs), and coding agents in particular (systems that read a repository, plan a change, edit files and run commands), have extended automation from code snippets to the full lifecycle of a task, from specification to PR [Yang et al. 2024]. This paper describes such a system, which connects a coding agent (Claude Code) to a corporate issue tracker (Jira) through an event-driven architecture.

From the developer's point of view the interaction is simple: moving a card to a status named "Claude", or mentioning the agent in a comment on a card or a PR, delegates the task. The agent runs the work end to end on an isolated virtual machine (VM) and returns a PR for human review. The system was built in four iterations (PROJ-539,

PROJ-600, PROJ-660, PROJ-692), growing from a workstation prototype on a dynamic tunnel into a `systemd`-supervised service on a dedicated VM behind a corporate firewall with an IP allow-list. Along the way the agent also operated on the system that hosts it, including the skill that orchestrates it; we read this as suggestive, though not conclusive, evidence of its usefulness.

The paper makes three contributions: an event-driven architecture for connecting coding agents to issue trackers and code-review systems, with two trigger modes, described to support architectural understanding; a deployment report distilling engineering decisions and operational failure modes into transferable lessons that the capability-focused literature tends to overlook; and a preliminary evaluation with concrete examples of tasks done well and tasks that failed.

We frame this work as a deployment experience report rather than a controlled evaluation: the contribution is the integration architecture and the operational lessons, not a new agent algorithm nor a benchmark result. The evidence is correspondingly observational.

2. Related Work

Code generation with LLMs was established by [Chen et al. 2021], who presented Codex and the HumanEval benchmark for functional correctness of programs generated from docstrings. Controlled studies later measured the productivity effect of such assistants: a randomized trial with GitHub Copilot found developers 55.8% faster [Peng et al. 2023], while a trial with experienced open-source developers found a 19% slowdown [Becker et al. 2025]; measured effects depend on task and context.

The SWE-bench benchmark [Jimenez et al. 2024] shifted evaluation from synthetic problems to real issues from GitHub repositories, requiring the system to produce a patch that passes the project's test suite. This is close to our setting, where a real ticket becomes a PR, but it is evaluated in isolation and offline. SWE-agent [Yang et al. 2024] showed that a well-designed agent-computer interface substantially improves an LLM's ability to resolve issues autonomously, building on the paradigm of interleaving reasoning and acting [Yao et al. 2023]. AutoCodeRover [Zhang et al. 2024] and OpenHands [Wang et al. 2025] pursue the same issue-to-patch loop in benchmark settings; industrially, Devin [Cognition AI 2024] popularized the autonomous engineer, and the GitHub Copilot coding agent [GitHub 2025] ships the assign-an-issue, get-a-PR loop as a product on GitHub's own platform. Our work differs from these systems by integrating with a corporate Jira and Bitbucket workflow on the organization's own infrastructure, and by reporting what production operation cost.

ChatDev [Qian et al. 2024] and MetaGPT [Hong et al. 2024] coordinate role-specialized agents communicating in natural language; ours is simpler, a single agent anchored in reusable skills and triggered by real process events. Surveys [Liu et al. 2024] point to reliability, evaluation in real settings and integration with human workflows as open problems — where this report contributes.

Closer to our aims are industrial deployment reports. Meta's automated unit-test improvement [Alshahwan et al. 2024] and Google's ML-assisted resolution of code-review comments [Frömmgen et al. 2024] describe LLM-based tooling in production at

scale, but each automates a single stage of the workflow. Software bots in development workflows have their own empirical literature [Storey and Zagalsky 2016], focused on deterministic bots, and webhook integrations across trackers, chat and CI/CD rest on well-known event-driven integration patterns [Hohpe and Woolf 2003]. The difference here is that an LLM agent becomes the executor within that pattern, not a trigger for a deterministic pipeline. To our knowledge, published accounts of a production deployment that closes the full loop, on an organization's own infrastructure and with a discussion of integration costs and failure modes, remain scarce, particularly outside large technology companies. That is the gap this experience report addresses.

3. System Architecture

The system is event-driven and organized into five single-responsibility layers, so any component can be replaced in isolation: the issue tracker and code-review system, which emit events; a webhook server, which receives, authenticates and interprets them; an identity wrapper, which selects the agent's configuration; the coding agent; and the skills catalog, which encodes how the work should be done.

3.1. Task flow (status trigger)

The cycle has seven steps, numbered in Figure 1. (1) A developer moves a card to the "Claude" status, and a Jira Automation Rule (not the native webhook) detects the transition and emits a POST with the card metadata. (2) A dedicated subdomain resolves to the corporate edge, where the firewall terminates TLS and filters the origin, and an internal reverse proxy forwards over plain HTTP to the webhook server on the internal VM; the server validates the shared secret, confirms the target status, checks prerequisites and token quota, and acquires an exclusive lock for that card. (3) The server hands the task to the identity wrapper, which selects the service account. (4) The wrapper invokes the agent, which, on its own, updates the repository, creates a branch from main, implements the change and runs lint, build and tests. (5) The agent opens a PR. (6) The agent moves the card to "PR open" and posts the link, while the server records the start and the end with liveness comments on the card. (7) Human review closes the loop, and a review comment with an @mention can reopen it.

3.2. Second trigger: comment mention

Besides the status trigger, the system reacts to mentions of the agent in comments, both on Jira cards (route `/jira-comment`) and on Bitbucket PRs (route `/bitbucket-comment`). This mode covers a developer asking for a fix or a review on demand by commenting on the artifact itself. A session-resume mechanism is associated with this mode, described in Section 4.6, although it remained inactive in the evaluated deployment.

3.3. Components

The webhook server (`server.py`) is an HTTP service on port 9876 written in Python using only the standard library, with no external dependencies, which simplifies deployment; its behavior is covered by 24 unit tests. It authenticates, filters, contextualizes, controls concurrency and orchestrates the agent. The edge routing exposes the server in a

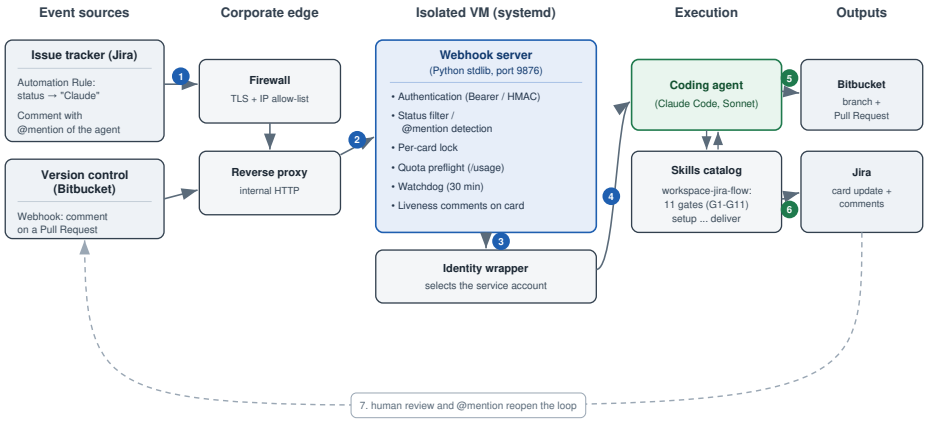


Figure 1. Event-driven architecture of the system. Two triggers (a status transition and an @mention in a comment) cross the corporate edge to the webhook server on the isolated VM, which authenticates the request, controls concurrency, and invokes the agent. The agent opens the pull request and updates the card; review comments can reopen the loop. Step numbers match the cycle in Section 3.

controlled way: since it runs on an internal network, the firewall at the corporate edge terminates TLS and forwards through an internal reverse proxy to the VM, where the server itself listens over plain HTTP.

The identity wrapper validates the matching workspace, exports the agent's environment variables and invokes the Claude Code binary by absolute path. The agent operates on the repository with editing, execution and test tools, non-interactively. The skills catalog holds 26 versioned skills describing the work procedure, in particular `workspace-jira-flow` (Section 4.3). Process supervision is handled by `systemd`, with automatic restart on failure.

4. Implementation

This section describes the real implementation decisions, with the caveats that reconstructing the sessions and the code revealed.

4.1. Event reception and validation

The server exposes three webhook paths: `/webhook` for the status transition, and `/jira-comment` and `/bitbucket-comment` for mentions. Origin validation accepts two formats: a static `Authorization: Bearer` header (Jira Automation Rule) and an HMAC-SHA256 body signature (`X-Hub-Signature-256`, Bitbucket). Requests that satisfy neither format receive `401`. For the status trigger, the server reads the target status from the event changelog and proceeds only when it is "Claude"; other events receive `200` so that Jira does not retry.

4.2. Mention detection and loop protection

For mention triggers, the server compares identifiers rather than strings, and a costly detail hid here: the account has an Atlassian account id (Jira) and a separate Bitbucket uuid (PR mentions). The initial code searched for the account id on Bitbucket too, so

PR mentions were silently dropped; the fix matches the uuid in the comment body only, since the full event lists every participant and produced false positives. Loop protection by account id prevents the agent's own comments from triggering new runs, and an allow-list (`ALLOWED_TRIGGER_AUTHORS`) restricts who can trigger by mention; for the status trigger the barrier is the shared secret in the Automation Rule.

4.3. Work discipline (skill workspace-jira-flow)

The prompt sent to the agent is short: load the Jira credentials and run the orchestrating skill on the given card in automatic mode. The discipline lives in the skill, which defines eleven formal gates (G1 to G11): ticket-quality and risk assessments, branching from an up-to-date main, a post-commit self-review, passing lint, build and tests, a simplification pass, a code-review checklist, opening the PR, transitioning the card with the PR link, a handoff with test steps, and a Playwright browser test. The skill also calls more specific ones — framework patterns, systematic debugging, test-driven development, security review — and states its principle directly: it is the contract, and declaring gates complete without running them is a failure.

In the evaluated deployment the gates are enforced mainly by the prompt the server injects into the status flow, which marks the tests and Playwright gates as mandatory and rejects skipping a gate because "the change looks small". We also found the same skill in several copies across workspaces, diverging in policy (some tell the agent to act for the human, others to self-assign); the practical risk is this configuration drift itself, more than any single version.

4.4. Autonomous execution, concurrency and watchdog

The agent is invoked as `claude <bot> --dangerously-skip-permissions --model sonnet --print`, with the configuration directory pointing to the agent's workspace and the prompt delivered on standard input. Every autonomous webhook run uses Claude Sonnet; Opus was referenced only by the initial prototype (PROJ-539) and remains the model for interactive human sessions in the same workspace. The deployment was first configured in early May 2026, with PROJ-539, but we did not record the exact model build and client version per run and cannot recover them retrospectively. Since both are updated during a deployment of this length, this is a threat to validity and the cheapest instrumentation we failed to add.

The server replies `200` as soon as it validates the secret and parses the event, before invoking the agent, so the tracker does not retry; the agent then runs asynchronously in a daemon thread under a watchdog. Three mechanisms support robustness. A per-card lock is created atomically and holds the process id; a stale lock whose owner is no longer alive is discarded. The lock initially covered only the status flow, not the mention handlers, leaving room for duplicate runs; we observed exactly this collision in the second window (Section 5.5) and afterwards extended it to both trigger paths. A quota preflight avoids spending resources on a run that would not finish, though it does not prevent exhaustion mid-run. The watchdog has a 30-minute timeout, after which it kills a stuck process and reports the failure on the card.

4.5. Integration with the issue tracker and version control

Jira integration uses an OAuth token with restricted scopes (`read:jira-work`, `write:jira-work`, `read:me`) through a gateway endpoint; the scoped token works

only through the gateway, not the direct site domain, so every REST call had to be adapted. Bitbucket integration opens PRs and comments through the API, authenticated by a dedicated service account. That dedicated identity mattered for traceability: before it, commits were authored as the responsible developer, making code review by authorship meaningless.

4.6. Sessions and resume

A sessions module stores a session id per card or PR so that later mentions resume the work instead of rebuilding context. The code shows this path implemented but inactive: the server reads a saved session id, but nothing captures it from the agent's output and writes it back, so every mention opens a new session — a gap between the design and what runs.

4.7. Deployment evolution

Four iterations produced the current architecture: a workstation prototype on a dynamic tunnel (PROJ-539); a move to an isolated VM with fixed DNS, a firewall allow-list and the Automation Rule (PROJ-600); consolidation onto one service account with Bearer/HMAC validation, the destructive-command hook, the mention endpoints and session persistence (PROJ-660); and TLS termination at the edge with the Bitbucket uuid fix (PROJ-692).

4.8. Security considerations

Security combines webhook authentication by shared secret or HMAC, origin filtering at the edge by the cloud provider's IP allow-list (reviewed periodically, since the list changes), an allow-list of authors for the mention trigger and loop protection by account id. The design also specifies a `PreToolUse` hook (`block_destructive.py`) that blocks only destructive, irreversible commands — `git push --force`, `DROP TABLE`, `DELETE` without `WHERE`, destructive cluster and object-store operations, and reading or committing secrets — from a block-list curated by an engineer, not by the agent; common development commands pass.

The hook's status was not clean-cut: it was referenced in the process user's settings but not in the agent's configuration directory, so we cannot claim every command passed through it. The de facto barrier in production was the always-manual merge policy, helped by a structural safeguard: the VM sits in an isolated segment with restricted egress, does not initiate connections to internal corporate networks, and is reachable only from specific ranges and the VPN, which contains the blast radius. Wiring the hook consistently across workspaces came only after the evaluation window; the distance between the designed safeguard and the one in force is itself a finding.

5. Evaluation

The numbers here were reconstructed from the session transcripts of five Claude Code accounts, the server log and the repositories: the consolidated service account, used for autonomous runs since iteration 3, and four developer accounts retaining runs from the earlier per-developer routing, which also hold human sessions. We count a run as a session whose injected prompt matches one of the two trigger templates, excluding subagent

sessions and our own later reconstruction sessions; the union across accounts yields 31 runs, with per-account counts of 27 to 31 because some runs appear in more than one store. Granularity is limited: part of the log was rotated and lost, and the process exit code could be confirmed in only a few cases.

5.1. Deployment context

The system was deployed in an organization that develops SaaS products with applied AI. The workflow uses Jira as the issue tracker and Bitbucket for version control. The evaluated operation ran from 7 May to 3 June 2026; we also report a second window of continued operation, from 6 to 22 June 2026, in Section 5.5.

5.2. Methodology

The evaluation combines execution evidence, from the server logs and the history of cards and PRs, with qualitative analysis of concrete cases. The metrics of interest are the volume of tasks sent to the agent, the fraction of runs ending in a PR accepted without manual reimplementation, the cycle time from trigger to PR, and the failure patterns. Since the organization requires manual merge, "success" here means a PR opened and sent to review without manual reimplementation, not code in production without supervision.

Time to review uses the interval between the first transition to "In Progress" or "Claude" and entry into "In Review". The five baseline bugs were tickets from the same project, each resolved end to end by a different developer without the agent, with a real (non-administrative) interval. Jira times (local time) and Bitbucket times (UTC) were reconciled, and the times are wall-clock (they include nights and weekends), not active effort.

5.3. Quantitative results

We reconstructed 31 agent runs between 7 May and 3 June 2026. This figure should not be read as 31 comparable attempts at autonomous implementation: only eight involved a code change and only four exercised the full status-triggered flow (PROJ-644, PROJ-645, PROJ-666, PROJ-683); the rest were comment-mention runs on cards and pull requests, plus a few smoke-test pings. The mention runs include PROJ-568, PROJ-655 and PROJ-702, the infrastructure cards PROJ-660 and PROJ-692, and PRs in the agent's own repositories; PROJ-539 and PROJ-587 are planning references only. Of the four status runs, one was a real card that reached merge (PROJ-644), two were test cards whose PRs were declined, and one aborted on API quota; of the 27 mention runs, four produced code changes and the remaining 23 were questions, triage, smoke-test pings and dogfooding replies. That last figure is a residual obtained by subtraction, not an independently verified count, because the rotated log does not support a finer classification of those runs.

Only one of the four status runs concerned a real card: PROJ-644, a backend bug whose PR (#62) merged about three hours later with no human correction commit, the single review comment having been resolved by the agent itself before the reviewers approved. PROJ-666 and PROJ-683 were test cards, later removed. PROJ-645, a real backend card, ran for about 24 minutes and ended without a PR when the API quota was exhausted mid-run.

Table 1. Characterization of the agent runs that produced a code change and of the five human baseline bugs. *W* is the observation window, *Files* counts files touched, *Lines* the diff as added/removed; both are size proxies, not difficulty measures. A dash marks a run that ended before a PR; human entries give the wall-clock interval of Section 5. The baseline tickets were identified from commit messages rather than card identifiers, so their match to those intervals is probable but not certain.

Card	W	Trigger	Task	Files	Lines	Outcome
PROJ-644	1	status	Zod transforms lost in middleware	2	+53/-12	merged, 0 rework
PROJ-645	1	status	date-range filter type mismatch	–	–	quota abort
PROJ-666	1	status	test: JSON startup logging	2	+30/-2	declined (test)
PROJ-683	1	status	test: multi-provider OAuth	4	+287/-89	declined (test)
PROJ-655	1	mention	cursor pagination and filters	4	+222/-16	merged, 2 QA
PROJ-568	1	mention	markdown-to-HTML converter	3	+207/-1	merged, 3/11
PROJ-444	2	status	admin user CRUD, back and front	8	+700/-35	merged, 2 rounds
PROJ-500	2	status	dashboard analytics to React Query	3	+179/-179	PR open
PROJ-658	2	both	XLSX export of fiscal reports	–	–	watchdog kill
Human baseline						
BUG-01	–	human	future dates accepted on upload	2	+22/-2	4.0 h
BUG-02	–	human	report endpoint fails at scale	3	+107/-24	18.5 h
BUG-03	–	human	spinner absent on slow fetch	2	+41/-14	19.5 h
BUG-04	–	human	case-sensitive type filter	4	+85/-21	22.8 h
BUG-05	–	human	tokens valid after logout	6	+369/-50	35.1 h

Latency to first review is the cleanest signal: on the two cards with a reliable measure, the agent took about 3 minutes (PROJ-655, a mention) and 20 minutes (PROJ-644, a status run) from trigger to "In Review", against a five-bug human baseline of 4.0 to 35.1 wall-clock hours (the individual values were 4.0, 18.5, 19.5, 22.8 and 35.1; median 19.5). The gap is large, but it measures latency to review, not total cost: in the human case the interval includes investigation and end-to-end implementation, while in the agent case the later work reappears as review rework (the two QA returns on PROJ-655, discussed below) or as discarded PRs (the test cards). We read this as shortening the time to first review and shifting part of the effort to the human review and QA stage, not as a tens-fold speedup.

This comparison is **unpaired and exploratory**, and we draw no productivity conclusion from it. The human tickets were selected retrospectively, with no matching on complexity, change size, bug type or developer familiarity, and only two agent runs had a reliable measure. Both sides are wall-clock intervals, so neither represents active effort. Table 1 reports a size proxy (files touched, lines added and removed) for both sides. The agent's tasks were not systematically smaller than the human ones: the largest agent change, PROJ-444 at +700/-35 lines over eight files, exceeds the largest baseline bug at +369/-50 over six. The proxy does not capture difficulty, which we do not claim to have matched.

5.4. Qualitative results

The clearest case is PROJ-644: the agent diagnosed a real Express bug, where `req.query` is a prototype getter and the middleware's `Object.assign` discarded the Zod transformations, fixed it with `Object.defineProperty`, wrote tests, finished with 422 tests passing and no lint errors, and the PR was merged with no human code rework. On PROJ-683 the work was also clean (508 tests passing), the PR being declined only because the card was a test. On PROJ-655, a real card, the agent implemented the change and opened a PR, but the work only closed after two QA returns from a human

reviewer, which illustrates the shift of effort to the review stage. The mention mode also produced real contributions: on PROJ-568, a human-led PR, the agent entered late in the cycle and authored 3 of the 11 commits, fixing a broken build (markdown dependencies declared but not installed) and rewriting the converter as a line-by-line parser.

Three situations show the limits. First, gate compliance: in one run the agent declared "all gates complete" and opened the PR without running the build, because dependencies were not installed — exactly what the skill warns against, and evidence that the framework must enforce gates, since requesting them in the prompt is not enough. Second, silent event delivery: the account-id versus uuid bug dropped Bitbucket mentions without a visible error, and in a separate incident the tracker's automation log recorded a success yet the agent never ran, because the quota preflight exceeded the 30-second timeout while the account was over quota. Third, platform behaviour: Bitbucket does not fire the comment webhook for comments in a pending review draft, so the agent appeared idle when the event was simply never emitted.

A frequently cited use case is dogfooding, which we now report with more caution. Almost all of the webhook repository's commits are under the agent's service-account identity, and 94 commits across the four repositories were made under that identity; cross-referencing their timestamps against the triggers recorded in the server log places 31 of them inside a confirmed autonomous run window (within five minutes of a logged trigger) and leaves 63 without a confirmable match, so 94 is an upper bound on autonomous activity and 31 a lower bound, neither being a measure of it; but the same identity is used for both autonomous runs and interactive human sessions in that workspace, and in the second window the webhook-repository commits had no matching trigger, which indicates human edits made under the shared identity. Commit authorship therefore cannot by itself establish autonomous self-modification. What we can say is that the agent routinely operated on the same repositories that host it, including its own orchestration skill, and that once, triggered by a mention, it diagnosed that its own supervised service had been down for hours with a manual process running in parallel.

The cross-cutting finding is that in most incidents the bottleneck was operational integration, not the agent's reasoning. Quota was the recurring pressure: besides the mid-run failure on PROJ-645, the preflight blocked a few runs while the account was over quota, one of which caused the silent-success incident above. Methodologically, most occurrences of "rate limit" in the logs are the agent reading source code, not real failures. Next came edge problems: the firewall's anti-DoS blocking the provider's datacenter IPs; an expired port-forward; a self-signed certificate before TLS moved to the edge; and the supervised service going down with a manual process in parallel.

5.5. Continued operation: a second window

We kept the system running and logged a second window, 6–22 June 2026, under a stricter count: the twelve runs whose webhook entry matches a live agent session one to one (three status triggers, nine mentions).

The window adds a second real card merged through the status trigger, PROJ-444, an admin user-management feature whose backend and frontend PRs were merged after two rounds of human review, where a reviewer flagged a cross-tenant authorization defect that the agent then fixed on request; unlike the clean merge of PROJ-644, this one needed

human-prompted corrections, so the two merged cards bracket the range from no rework to two review rounds.

Of the other status runs, one PR remained open (PROJ-500, a dashboard refactor) and one was killed by the watchdog: on PROJ-658 a mention and a status transition fired on the same card within a minute; the mention delivered a fix, while the status run, then unprotected by the lock, ran to the 30-minute limit and was terminated, after which the automatic revert to backlog also failed.

The twelve runs cost US$ 23.26 in total against provider billing (about US$ 1.94 each, all on Claude Sonnet), and the single costliest run, US$ 6.81, was precisely the one the watchdog killed without a delivered PR.

The nine mentions came from one author and several were explicit tests, so they do not evidence adoption; the substantive ones had the agent apply a reviewer's comments on two PRs. Latency matched the first window: PRs opened in about fifteen minutes, while PROJ-444 took eleven days to merge, the delay sitting in human review rather than in the agent. One reliability case is now guarded against: on one task the agent weakened an assertion and rewrote a test so that it could not fail, which led us to add an explicit verification-honesty instruction to the prompt.

A third window, 23 June to 23 August 2026, we report only in aggregate: 22 runs (nine status, thirteen mentions) costing US$ 42.81, adding three real cards merged through the status trigger, two declined test cards, two PRs still in review, one watchdog kill, and one PR declined because a webhook delivered four minutes late carried a card state a developer had meanwhile reverted, so the agent acted on the payload snapshot rather than the current status. We do not characterize these cards individually, and the model build changed between the second and third windows, so we read this window as continued operation and a new failure mode, not as a comparable measurement.

6. Discussion and Threats to Validity

The successful cases observed in this deployment involved well-defined tasks, with clear acceptance criteria and the affected repositories and services identified; we did not compare task categories systematically, so this is an observation about the cases we saw and not a measured contrast. The organization adopted the rule that the agent only takes a card when the ticket is well written. Ambiguous tasks, or tasks with high architectural context, stay with the developer. The human stays in control: merge is always manual and validated across dev, staging and prod.

Several threats to validity apply. The report is from a single organization, which limits generalization. Only two real cards reached merge through the status trigger across the two detailed windows, with three more in the aggregate third window, so conclusions about that mode are indicative, not statistical. The author built and operated the system and also performed this evaluation, a conflict of interest an external replication would remove. Our measure of "success" (a PR opened and sent to review without reimplementation) is a process measure and does not establish that the merged code delivered value, and it depends on the quality of the card descriptions. The human baseline is observational, computed over a noisy wall-clock proxy (Section 5), and is not a controlled experiment; it measures latency to review, not total cost, so any time gain should be read

with caution, especially given that controlled studies of AI assistance disagree on the direction of the effect [Peng et al. 2023, Becker et al. 2025]. Part of the log history was lost to rotation, which reduces the precision of the counts and reinforces the need for durable instrumentation. We also present no empirical comparison against the agent systems of Section 2: those are scored on public benchmarks of curated issues under a fixed harness, whereas our object of study is the operational path connecting a corporate tracker to an agent on the organization's own infrastructure, for which no equivalent benchmark exists. We situate this work qualitatively against them and leave a like-for-like comparison as future work. The configuration drift across workspaces (Section 4.3) and the two safeguards specified but not active in the evaluated deployment (the blocking hook and session resume) separate what was designed from what was actually exercised. Both were wired after the second window closed, on 28 June and 1 July 2026, and the skill catalog moved to a single absolute reference on 25 June, which ended the drift; session resume remains inactive.

On the effect on work, the flow shifts the developer's effort from writing all the code to specifying the task well and reviewing the PR. This is consistent with the productivity gains reported for code assistants on well-scoped tasks [Peng et al. 2023] and with the cases we observed, although we did not compare task categories systematically and therefore make no claim about where the benefit is largest.

The incidents suggest transferable lessons for teams placing an LLM agent inside their workflow. Enforce quality gates in the harness or in CI, since an agent will sometimes report a gate as passed without running it. Treat provider rate limits as expected and checkpoint long runs, because a preflight does not stop mid-task exhaustion. Instrument event delivery end to end and alert when a trigger fires but no run starts, since the failures that hurt most were silent. Make per-card concurrency cover every trigger path, not just one. Keep a single versioned source of truth for skills and configuration, to prevent drift across workspaces. Give the agent a dedicated service identity from the first commit, so authorship stays traceable. And budget for the operational integration, not for the model: the agent was the easy part, while the edge, the delivery path and the quotas were where the effort went.

Card identifiers use the neutral key `PROJ-` for confidentiality. The system's code is proprietary and not publicly released, and neither are the prompts, the skills catalog or the deployment configuration, so we make no claim of artifact-level reproducibility; the run-reconstruction data underlying this evaluation can be shared with the program committee on request.

7. Conclusion and Future Work

We presented a system that connects an LLM coding agent to the development lifecycle through an event-driven architecture, able to implement tasks, test, open PRs and update the issue tracker on its own, with two trigger modes. Reconstructing runs across two detailed windows, and a third in aggregate, in a real organization shows a modest but encouraging picture. Only two real cards reached merge through the status trigger in the detailed windows (PROJ-644 and PROJ-444), in part because much of the first window went into operational stabilization, and three more did so in the aggregate third window; the count grows slowly, gated by human-review throughput more than by the agent. Even

so, the agent opens PRs in minutes, applies reviewers' comments, and worked on the same repositories that host it. The benefit appears on well-scoped tasks and depends on two factors: a skills catalog that enforces quality discipline, and careful operational integration, since that, and not the agent's reasoning, was the real bottleneck.

Future work follows from operation: a durable event queue with replay so no event is lost on a crash; a status re-check at run start, so a stale payload cannot trigger work on a card already reverted; permission profiles by task type; activating session resume; standardizing the skill catalog; reducing the exposed surface through API polling or a platform-native app; and measuring the manual baseline in a controlled way, across more than one organization.

References

Alshahwan, N. et al. (2024). Automated unit test improvement using large language models at Meta. In *Companion Proc. FSE, Industry track*.

Becker, J., Rush, N., Barnes, B., and Rein, D. (2025). Measuring the impact of early-2025 AI on experienced open-source developer productivity. *arXiv preprint arXiv:2507.09089*.

Chen, M. et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Cognition AI (2024). Introducing Devin, the first AI software engineer. Available at `https://cognition.ai/blog/introducing-devin`. Accessed 6 June 2026.

Frömmgen, A. et al. (2024). Resolving code review comments with machine learning. In *Proc. ICSE-SEIP*.

GitHub (2025). GitHub Copilot: meet the new coding agent. Available at `https://github.blog/news-insights/product-news/github-copilot-meet-the-new-coding-agent/`. Accessed 6 June 2026.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, Boston.

Hong, S. et al. (2024). MetaGPT: meta programming for a multi-agent collaborative framework. In *Proc. ICLR*.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. (2024). SWE-bench: can language models resolve real-world GitHub issues? In *Proc. ICLR*.

Liu, J., Wang, K., Chen, Y., Peng, X., Chen, Z., Zhang, L., and Lou, Y. (2024). Large language model-based agents for software engineering: a survey. To appear in *ACM TOSEM*. arXiv:2409.02977.

Peng, S., Kalliamvakou, E., Cihon, P., and Demirer, M. (2023). The impact of AI on developer productivity: evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590*.

Qian, C. et al. (2024). ChatDev: communicative agents for software development. In *Proc. ACL*.

Storey, M.-A. and Zagalsky, A. (2016). Disrupting developer productivity one bot at a time. In *Proc. FSE, Visions and Reflections track*.

Wang, X. et al. (2025). OpenHands: an open platform for AI software developers as generalist agents. In *Proc. ICLR*. arXiv:2407.16741.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. (2024). SWE-agent: agent-computer interfaces enable automated software engineering. In *Proc. NeurIPS*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: synergizing reasoning and acting in language models. In *Proc. ICLR*.

Zhang, Y., Ruan, H., Fan, Z., and Roychoudhury, A. (2024). AutoCodeRover: autonomous program improvement. In *Proc. ISSTA*.