One Operator, Twenty Systems: An Eighteen-Month Field Study of Agent-Native Solo Software Development

Working Paper v0.2 (revised), 17 September 2026. Circulated for comment; not for citation.

Author: Klaus Arent (independent). Contact: klausarentde@gmail.com.

"Twenty Systems" is a working title; Section 4.1 reconciles the counts (128 repositories, 35 catalogued projects, 13 live systems). All quantitative figures come from an automated pipeline (`scripts/kennzahlen.mjs`, whitelist `scripts/whitelist.env`) at snapshot 2026-09-16; the automation-commit correction in Section 4.1 was computed by hand from commit subjects at the same snapshot and will move into the pipeline for v0.3. September 2026 is a partial month (1–16 September). Figures marked *operator report* are self-reported and not backed by a document. Sections marked *in progress* list what is missing. This version was reviewed blind by two independent model-based reviewers before circulation; Section 7 lists their main objections and what changed.

Abstract

Benchmarks measure what AI coding agents can do on isolated tasks. Population-level repository studies measure what happens to projects that adopt them. Neither shows what sustained agent-native work looks like from inside one working practice. We report a longitudinal single-operator field study of an online-marketing practitioner with twenty years in that field and no professional software background who, between April 2025 and September 2026, built and now operates a portfolio of production systems using LLM-based coding agents. The complete git record across the operator's machines and remotes covers 128 repositories and 9,593 own commits after author filtering; it contains one own commit before 2025. We pair this record with deployment logs from two hosting platforms, a dated corpus of 118 operational failure notes (42 coded in this version), agent session logs, and billing evidence. Four observations. (1) Recorded monthly output rose from 75 own commits per month in an AI-assisted, single-threaded phase (April–November 2025) to 874 per month after the switch to orchestrated agents in December 2025, a factor of 12 that ranges from 6 to 14 under alternative phase boundaries. Decomposed, the factor is mostly breadth: the number of repositories touched per month rose six-fold (3.4 to 21.3) while commits per touched repository roughly doubled. We do not observe working hours and make no per-hour productivity claim. (2) The share of commits carrying an agent co-author marker rose from 24% to a 66–72% plateau in June and July 2026 and then fell. Part of that fall is measurement: automation commits under the operator's identity account for 7 to 14 points; the remainder coincides with the adoption of agent tools that do not sign commits and cannot be attributed from git. (3) In the coded failure corpus the largest category is contact with the environment (platform contracts, toolchains, encodings), 15 of 42 notes; on our mapping, three of seven categories are candidates for extending the MAST taxonomy. (4) Out-of-pocket spend on the coding stack stayed between roughly 45 and 330 € per month; we report consumed token volumes descriptively and defer any list-price valuation. We discuss the shift of the operator's role toward decision bandwidth and argue that agent-era repository mining needs new controls for automation commits and unsigned agent work.

---

1 Introduction

Evaluations of AI coding agents are dominated by benchmarks such as SWE-bench [Jimenez 2024], by short-horizon controlled studies [Peng 2023; Becker 2025], and by surveys that capture attitudes rather than behaviour. A newer line of work mines repository populations for adoption effects [He 2026; Agarwal 2026; Robbes 2026] or analyses large samples of agent sessions [Tang 2026; Baumann 2026]. To our knowledge, from two literature passes in July 2026, what is still missing is the inside view: one person's complete portfolio, over more than a year, with commits, deployments, costs, and failures observed together.

This paper reports such a case. The subject is not a professional developer. He is an online-marketing practitioner with two decades of SEO, SEA, and e-commerce leadership work who, starting in April 2025, began producing software with AI assistance and, from December 2025, switched to orchestrating several coding agents in parallel. As of September 2026 he operates a portfolio of 35 catalogued projects, 13 of them classified as live, two of them processing payments through a live billing integration. His git history before April 2025 contains a single import commit. The record therefore does not capture a working developer getting faster. It captures a non-developer becoming the operator of a multi-system production portfolio through agents.

The case is informative for three reasons. Every decision, every accept, reject, and kill, was made by one person, so there is no team confound (other confounds remain; Section 6). The portfolio is diverse: consumer SaaS, e-commerce tooling, content systems, dashboards and marketing automation for clients, and internal developer tooling. And the failure record was written contemporaneously as operational memory for the agents themselves, not reconstructed for the study.

We ask six research questions.

- **RQ1 Output.** What recorded output does a solo operator with agents produce over eighteen months, measured in commits, repositories touched per month, deployments, and live systems?
- **RQ2 Delegation.** Which task classes go to which agents, what stays with the human, and how does that split move over time?
- **RQ3 Failure modes.** Where do agents fail repeatedly, and which countermeasures did the operator adopt?
- **RQ4 Economics.** What did agent-native operation cost the operator, and what inference volume did it consume?
- **RQ5 Evolution of the work system.** How did the tooling, memory, and coordination practices change?
- **RQ6 Cross-system observations.** What did parallel use of several agent frameworks show, based on documented usage rather than benchmarks?

Contributions of this version, with their evidence status:

- **C1** A longitudinal, author-filtered record of a solo operator's recorded output across an AI-assisted phase and an agent-native phase, with a reconciled inclusion ledger and sensitivity to phase boundaries (RQ1; complete).
- **C2** A decomposition of the output multiplier into breadth (repositories touched per month) and depth (commits per touched repository), showing that breadth dominates (RQ1; complete as a description of recorded activity, not as a productivity claim).

- **C3** A preliminary taxonomy of recurring agent failure modes and operator countermeasures from a dated operational-memory corpus, with a first mapping onto MAST [Cemri 2025] (RQ3; 42 of 118 notes, single coding pass).
- **C4** Out-of-pocket cost accounting of agent-native operation with documented anchors and a bounded estimate where records are fragmented, plus descriptive token volumes (RQ4; partial).
- **C5** Five measurement pitfalls specific to agent-era repository mining, each of which changed a headline number in this study: foreign upstream history, fabricated import dates, identity drift, automation commits under a human identity, and marker decay under tool heterogeneity (Section 3.4; complete).

Positioning: this is a single-case study in the sense of Flyvbjerg [2006], with autoethnographic elements [Ellis 2011], reported along the guidelines of Runeson and Höst [2009]. Section 3 defines the case, its embedded units, and the evidence chain. We claim transferable vocabulary and candidate mechanisms, not statistical generality and not causal identification.

---

2 Related Work

Productivity of AI coding assistance. Controlled studies of code completion report large speedups on bounded tasks: Peng et al. [2023] found 55.8% faster completion of a small web-server task with Copilot. Ziegler et al. [2022] showed that acceptance rate predicts perceived productivity. The METR randomized trial [Becker 2025] is the strongest counter-signal: 16 experienced open-source developers on 246 real tasks in their own large repositories were 19% slower with AI tools while believing they were 20% faster. Butler et al. [2024] combined a trial with a three-week diary in a corporate setting and found high enjoyment with unchanged trust. A 2025 review of 39 studies [Mohamed 2025] records the absence of longitudinal field data. Our record does not resolve the perception gap, because we do not observe time; it adds a logged output series where the literature has estimates.

Benchmarks and their validity. SWE-bench [Jimenez 2024] is the reference benchmark for agents. Its validity has been contested from several sides: solution leakage in 32.67% of successful patches [Aleithan 2024], memorization signatures [Liang 2025], the need for decontaminated and time-stamped variants [Badertdinov 2025], and OpenAI's own finding that 59.4% of one model's "failures" were test defects [OpenAI 2025]. These critiques motivate a field taxonomy: fixed test harnesses do not exercise contact with a live, changing environment.

Field and longitudinal studies of agents. He et al. [2026] use difference-in-differences over GitHub projects that adopted Cursor and find a transient velocity gain of one to two months with persistent increases in complexity and warnings. Agarwal et al. [2026] extend this with staggered adoption and report gains mainly where no AI IDE was used before, with static warnings up 18% and cognitive complexity up 39%. Tang et al. [2026] analyse 20,574 real agent sessions and locate failures primarily in requirements and context misunderstanding. Baumann et al. [2026] find that only 44% of agent-written code survives into user commits. Robbes et al. [2026] estimate that 22–29% of GitHub projects show agent traces. All of these are population-level or session-aggregate designs; none that we found is operator-centred, portfolio-wide, and paired with deployment and cost data.

The closest neighbour. Davis et al. [2026] report a twelve-week case study of one engineer working with agents on one system and propose "governance conversion": fast agent implementation is turned into durable control by engineering judgment, mostly after failures surface. We share the N=1 stance and the mechanism. We differ in horizon (an eighteen-month record, of which nine and a half months

are agent-native), breadth (a 35-project portfolio versus one system), the subject (a non-developer versus an engineer), and instrumentation (commit, deployment, cost, and failure mining versus a narrative account).

Control and the human role. Huang et al. [2025] observed 13 professional developers and surveyed 99 more: experienced developers keep active control over design and implementation. Chen et al. [2026] ran the first controlled comparison of a copilot workflow and an agent workflow and found that agents complete tasks participants would not otherwise finish. Both anchor our RQ2 account of what the operator prescribes as human-only.

Methods. Flyvbjerg [2006] defends the single case. Runeson and Höst [2009] give the reporting structure and validity categories we follow. Ellis et al. [2011] define autoethnography; Kaltenhauser et al. [2024] legitimize it in HCI. Kalliamvakou et al. [2014; 2016] catalogue the perils of mining GitHub, including bot and archive artefacts; we add agent-era perils. Wen et al. [2021] show how quick remedy commits distort metrics, which matters because agents commit in small steps. Hoess et al. [2025] show that the mining tool itself changes results. Goeminne and Mens [2013] and Bird et al. [2006] treat identity merging, which became a live problem for us twice.

Failure taxonomies. MAST [Cemri 2025] codes multi-agent failures into 14 modes in three classes (specification, inter-agent misalignment, task verification). Liu et al. [2025] code failures of automated issue solving into three phases and nine categories. We map our seven field categories onto MAST classes in Section 4.3 as a first, coarse mapping.

Gap statement. Existing agent-era field data is population-level and outcome-focused. Missing, as far as our two search passes show, is the operator-level longitudinal inside view with deployments, costs, and failure modes over a real multi-system portfolio. The one N=1 agent case study we found is short-horizon and single-system.

---

3 Study Design

3.1 The case

Bounded case. One operator's software production with AI tools, April 2025 to September 2026, across all repositories under his control on two machines and three remote accounts. *Embedded units.* Repositories (128), catalogued projects (35), months (18), and failure notes (118). *Observation.* All quantitative evidence is contemporaneous system-generated data (git, deployment APIs, session logs, receipts); the failure notes are contemporaneous operational documents; phase descriptions and a small number of flagged figures are retrospective operator report. *Researcher role.* The author is the subject. Mitigations: an automated pipeline with a versioned whitelist, published aggregates, blind review of each version by independent model-based reviewers, and the practice of reporting every pitfall that changed a headline number.

Subject. The author has worked in online marketing since 2004 in agency, in-house, and freelance roles, including head-of-SEO and e-commerce leadership positions. Programming was a self-taught interest, never a profession; he had not shipped production software before 2025. From 2022 he paid for AI tools for marketing content (Section 4.4). Between April 2025 and September 2026 he worked solo across his own products and a small number of client engagements, one of them a longer-running marketing-systems mandate for an industrial technology group. Client projects are anonymized throughout (Section 3.6).

3.2 Phases

Phase	Period	Mode of work	Evidence for the boundary
P0	before April 2025	marketing practice; AI spend on content tools (Jasper, Midjourney, ChatGPT); no own code under version control on any surveyed machine or remote	git (one 2023 import commit); bank export 2022–2024
P1 AI-assisted	April–November 2025	first own repositories; low-code platform exports (Lovable), editor agents (Cline), chat-assisted coding, first terminal-agent use in June 2025; single-threaded, sporadic	first own commit 6 April 2025; Cline checkpoint repositories April–August 2025
P2 agent-native	December 2025 to present	terminal agents as primary workers, later an orchestrator with sub-agents, peer CLIs, persistent memory, a task board, and a mobile channel; portfolio-parallel	second agent subscription 5 November 2025 (receipt); own-commit series (Table 1); operator report

Two cautions. First, P1 is not a no-AI baseline: 30% of June 2025 commits already carry an agent marker. The comparison in this paper is AI-assisted single-threaded work versus orchestrated agent work, not manual versus AI. Second, the P2 boundary was chosen where the own-commit series shows a level shift (398 commits in December 2025 after 8 in November), which is selection on the outcome. We therefore report the sensitivity of every ratio to the boundary (Section 4.1). An independent marker for the boundary exists (the second subscription receipt of 5 November 2025), but it precedes the level shift by a month.

P0 is empty in git. Across both machines and all remotes, the oldest genuine own commit is from 6 April 2025; before it lies one 2023 import. This does not prove that no software was produced before 2025 (unversioned or deleted work cannot be excluded), but it is consistent with the operator's account and with the 2022–2024 spend record, which contains content tools and no developer tools.

3.3 Data sources

Source	Coverage	RQs	Status
Local git history, laptop	128 own repositories, mined nightly	RQ1, RQ2	complete, automated
Local git history, second machine (PC)	72 curated repositories, 5,174 commits, of which 64 not present on the laptop	RQ1	merged, hash-deduplicated
GitHub API sweep	remote-only repositories	RQ1	merged
Cloudflare and Vercel deployment APIs	two Cloudflare accounts, one Vercel scope	RQ1	Feb–Jun 2026 complete; Jul and Aug partial; refresh pending
Operational memory corpus (`feedback_*` notes)	118 dated entries as of Sep 2026	RQ3, RQ5	42 coded (Jul 2026); 76 uncoded
Task board	379 tasks migrated Jul 2026, ongoing	RQ2, RQ5	not yet analysed
Agent CLI session logs	main account, one machine, retained 30 days; extract of 11 Jul 2026	RQ2, RQ4	token totals extracted; delegation micro-structure not yet coded
Billing evidence	e-mail receipts 2023–2026; bank export 2022–2024	RQ4	anchors extracted; 2025–26 bounded estimate
Portfolio inventory	35 projects with type, status, period	RQ1	classified Jul 2026; operator review pending
Operator's versioned instruction files	delegation rules, permission boundaries	RQ2, RQ5, RQ6	read; they document prescribed practice, not observed compliance

Commits are deduplicated across sources by hash. *Operator-controlled commits* are identified by an author whitelist: eleven operator e-mail identities plus two platform bots the operator drove (Lovable and gpt-engineer-app, 2025 exports). We use "own commits" for this set throughout; it is not a set of human-typed commits. *Agent marker*. A commit counts as agent-marked if it carries a `Co-Authored-By: Claude` trailer or has that agent as commit author. The marker is written by one vendor's terminal agent when it commits. False positives are unlikely (the operator does not write the trailer by hand); false negatives are the subject of Section 4.2. We call the resulting quantity *marker coverage*, not agent share.

3.4 Data cleaning: five agent-era pitfalls

Each of the following was found the hard way and changed a headline number. We report them as a methods contribution (C5).

1. **Foreign upstream history.** Cloned tool repositories (goose, nvm, a skill library, a code-navigation server fork) brought about 9,700 upstream commits with historical dates onto the machine. In raw data this produced a sharp "January 2025 transition" (358 and 372 commits in January and February 2025) that was entirely goose upstream. The clone dates of the same repositories are, however, valid evidence of tool adoption (Section 4.5). Rule: commit date is not work date for cloned repositories; foreign repositories are excluded by name, and foreign authors in forks are excluded by the whitelist.

2. **Fabricated import dates.** A low-code platform (Lovable) stamped its export commits with 2025-01-01. Two such commits remain in the series at their stamped date (Table 1, row 2025-01); they are excluded from the baseline window and are negligible for totals.
3. **Identity drift.** When the operator renamed his GitHub account in July 2026, the new noreply address was missing from the whitelist and 823 August commits silently disappeared from the pipeline, which still exited with status 0. The pipeline now warns on any unknown identity with 50 or more commits in own repositories. The first run of that warning, on 17 September 2026, surfaced a further operator identity (a brand mailbox used on the second machine) with 58 commits from June 2025; they are included in this version.
4. **Automation commits under the human identity.** From July 2026, two of the operator's own automations commit under his identity: a watchdog that mirrors a coordination board to a file every 15 minutes (206 commits in August and 243 in September to the snapshot; subject `chore(state): board.json vom Wachhund`) and a nightly knowledge-graph build (14 in July, 32 in August, 16 in September; subject prefix `chronik:`). These 511 commits are neither human nor agent authorship. They inflate the commit denominator and depress marker coverage. In this version they are removed commit by commit (by subject) in Table 1 and in every derived figure; a name-based bot filter would not have caught them.
5. **Marker decay under tool heterogeneity.** The co-author trailer is written reliably by one vendor's terminal agent. As the operator added other agent tools from August 2026 (Section 4.5), their commits carry no marker at all. Across the 73 repositories with activity since June 2026, only seven commits carry a non-Claude co-author trailer (two Codex, one instinct, three from a CMS bot, one external contributor). Marker coverage is therefore a lower bound on agent involvement whose tightness varies with the tool mix (Section 4.2).

3.5 Analysis

Quantitative. Monthly aggregation of own commits, agent-marked commits, and distinct repositories touched (at least one own commit in the month). The transition ratio compares the mean monthly own commits in P1 (April–November 2025, seven months with activity) with the mean in the complete P2 months (December 2025 to August 2026, nine months). We report the raw pipeline figure, the automation-filtered figure, and the sensitivity to (a) counting the zero-activity month August 2025, (b) moving the boundary to January 2026, and (c) restricting P2 to the months before the June 2026 tier upgrade and vacation sprint. Breadth is the mean number of repositories touched per month; depth is the mean over months of commits per touched repository. Deployments per month per platform and token volumes per month from session logs are reported descriptively. Changepoint detection on the weekly series is planned but not run in this version.

Qualitative. Open coding of the failure corpus by an agent (the operator's orchestrator, July 2026, prompted to propose one failure class and one countermeasure type per note), axial consolidation by the author into seven categories, and a coarse mapping onto MAST classes. The codebook, inclusion rules, and overlaps will be released with v0.3. Autoethnographic vignettes illustrate mechanisms and are marked as such.

3.6 Ethics and anonymization

Client projects appear only as sector-level descriptions. No revenue figures, transaction counts, or user counts are reported in this version. No client names, no personal data of third parties, and no code from private repositories appear in the paper or the released dataset. The operator's education history is not reported at his request; it is not needed for validity. The released dataset will contain aggregated

monthly series, repository-anonymized commit metadata, the coded failure taxonomy with codebook, and the analysis scripts.

---

4 Results

4.1 RQ1: Recorded output across phases

Inclusion ledger. Own commits after whitelist and deduplication: 9,593 in 128 repositories. Table 1 lists every month with at least one own commit; the rows sum to 9,593. The two months before P1 (one 2023 import, two Lovable-stamped commits) are shown for completeness and excluded from all ratios. Automation commits (Section 3.4, item 4) are shown in their own column and subtracted in the filtered columns.

Table 1. Own commits per month (snapshot 2026-09-16). Marker coverage = agent-marked commits divided by own commits. Filtered = without automation commits.

Month	Own commits	Automation	Filtered	Agent-marked	Coverage (filtered)	Repos touched	Commits per repo (filtered)
2023-08	1		1	0	0%	1	1.0
2025-01	2		2	0	0%	2	1.0
2025-04	50		50	0	0%	4	12.5
2025-05	87		87	0	0%	2	43.5
2025-06	292		292	87	30%	9	32.4
2025-07	1		1	0	0%	1	1.0
2025-09	52		52	0	0%	2	26.0
2025-10	37		37	0	0%	2	18.5
2025-11	8		8	3	38%	4	2.0
2025-12	**398**		398	97	24%	10	39.8
2026-01	551		551	205	37%	16	34.4
2026-02	624		624	232	37%	22	28.4
2026-03	315		315	225	71%	14	22.5
2026-04	145		145	73	50%	8	18.1
2026-05	777		777	329	42%	19	40.9
2026-06	1,404		1,404	925	66%	21	66.9
2026-07	2,074	14	2,060	1,480	72%	43	47.9
2026-08	1,834	238	1,596	886	56%	39	40.9
2026-09 (1–16)	941	259	682	344	50%	36	18.9

August 2025 had no own commit and is omitted from the table. The two repositories that hold the automations also hold ordinary development work, which is why the filter operates on commit subjects rather than on repositories.

The transition ratio and its sensitivity. P1 mean, seven active months: 75.3 own commits per month. P2 mean, nine complete months: 902.4 raw, 874.4 filtered. Ratio: 12.0 raw, **11.6 filtered**. Sensitivity: counting August 2025 as a zero month lowers the P1 mean to 65.9 and raises the ratio to 13.7; moving the boundary to January 2026 gives 8.4 (8.1 filtered), because December 2025 then sits in the baseline; restricting P2 to December–May, before the June tier upgrade and the July vacation sprint, gives 6.2. The multiplier is therefore between 6 and 14 depending on defensible choices, and we use the filtered 11.6 as the reference figure because it removes known non-authorship and uses the boundary with the clearest level shift.

Breadth and depth. Table 1 supports one decomposition of that multiplier. Repositories touched per month averaged 3.4 in P1 and 21.3 in P2, a factor of 6.3. Commits per touched repository averaged 19.4 in P1 (median 18.5) and 37.8 in P2 (median 39.8, filtered), a factor of 1.9. The product of the two, 12.2, matches the ratio of means within rounding. In words: most of the recorded multiplier is more repositories per month, and a smaller part is more commits per repository. Depth is not stable within P2; it ranges from 18.1 (April 2026) to 66.9 (June 2026). Two things this decomposition does not show: it does not show concurrency (one person can touch 43 repositories in a month sequentially), and it does not show productivity per hour, because hours are not logged. It shows that the recorded activity spread over many more units of work. Section 5 discusses what would be needed to say more.

Dips and the peak. April 2026 (145 commits, 8 repositories) is a real activity dip that coincides with a documented payment interruption on one agent subscription (paused 16 April, restored 20 April) and, per *operator report*, reduced working time; a fuller account is pending. July 2026 is the peak: the operator used a vacation period as an uninterrupted build sprint across 43 repositories. August and September are lower in total and, per *operator report*, reflect a deliberate reduction of total working hours and a concentration on fewer projects after the sprint. If that report is accurate, output per hour rose in those months; we cannot verify it.

Deployments. Deployment counts from the two hosting platforms are shown in Table 2 as a second activity series. They are not independent of commits (most deployments are triggered by pushes, and preview builds and retries are included), so they corroborate the shape of the commit series rather than validate it. Cloudflare history is available from February 2026; July covers 1–11 July; the August figure was counted once on 24 August. Only February to June are comparable months.

Table 2. Production and preview deployments per month.

Month	Cloudflare (2 accounts)	Vercel	Total
2026-02	177	138	315
2026-03	170	61	231
2026-04	56	41	97
2026-05	68	120	188
2026-06	533	317	850
2026-07 (1–11)	315	83	398
2026-08 (to 24th)	186	80	266

The April dip and the June peak appear in both series.

Systems. The title's "twenty systems" is a working title. The counts are: 128 repositories with own commits (many projects span several repositories, and the count includes tooling, experiments, and aban-

doned starts); 35 projects in the operator's portfolio inventory (July 2026 classification, operator review pending): 27 own, 7 client, 1 contribution to a third-party fork; by status 13 live, 10 active in development, 10 unclear, 1 paused, 1 retired. "Live" means deployed and reachable at the classification date; it does not establish use, and this version reports no user or transaction counts. Two own products process payments through a live billing integration. Client deliverables include analytics dashboards, first-party tracking infrastructure, landing pages, outreach automation, and an internal AI-training curriculum. Per-project marker coverage (July 2026) ranges from 29% for a data-pipeline product to 86% for a direct-to-production web application. Survival of committed changes (lines alive after 90 days) and maintenance load per live system are not measured in this version.

4.2 RQ2: Delegation *(largely in progress)*

What the marker series shows. Filtered marker coverage rose from 24% in December 2025 to a plateau of 66–72% in June and July 2026, with intermediate months between 37% and 71% (Table 1). It then fell to 56% in August and 50% in the first half of September. We investigated the fall because it contradicts the operator's account that no manual coding took place.

What we can attribute and what we cannot. Automation commits account for the difference between raw coverage (48% and 37%) and filtered coverage (56% and 50%) in August and September: 7 to 14 points. The remaining fall from 72% to 56% and 50% coincides with a change in the tool mix. At the repository level, marker coverage in August and September is concentrated at the extremes: 11 repositories with 0–9% coverage hold 873 commits, 14 repositories with 80–100% coverage hold 639 commits, and 24 repositories in between hold 1,263 commits (2,775 commits in total, matching Table 1). The 0–9% repositories are the projects on which the operator introduced, in late August 2026, a second vendor's terminal agent (which does not write the trailer) and a WhatsApp-based assistant that dispatches work through a GitHub-issue bridge (Section 4.5); one product repository has 105 August commits at 7% coverage and commit subjects that include hand-over documents from a "night shift" run by the new tool. This is consistent with unsigned agent work. It is not a demonstration: we have no commit-linked session records for these tools, no audited sample of unsigned commits, and no within-month timing that would separate the late-August adoption from the whole-month decline. The *operator report* is that the delegated share stayed constant or rose while total working hours fell. We therefore treat marker coverage after July 2026 as no longer measuring delegation, rather than as evidence that delegation fell, and we do not claim the reverse either. From v0.3 all agent tools are required to write a signed trailer; this fixes the instrument going forward and does not repair August and September.

Prescribed delegation architecture (September 2026). The operator's standing instructions to his agents, kept as a versioned file, prescribe the following division of labour. They document intent, not observed compliance; the same files contain rules that were written after a gate had been skipped.

- *Orchestrator* (one terminal agent, frontier model): plans, decomposes, synthesizes; instructed to keep its own reasoning effort moderate.
- *Deep reasoner* (sub-agent on a stronger model): architecture, algorithm design, and debugging analyses that can be packaged as a self-contained brief.
- *Fast worker* (sub-agent on a cheaper model): boilerplate, tests, formatting, mechanical edits; must report deviations from the plan.
- *Peers* (three separate vendor CLIs, each billed against its own subscription quota): independent second opinions, adversarial review, isolated patches. Rules forbid interactive debugging with peers (they lack the conversation context) and forbid two writers in one working tree.

- *Council* (since 29 August 2026): at two fixed moments, when a plan is ready and before a completion report, two peers are asked blind and in parallel for verdicts; disagreement between them is treated as the useful signal. A stop hook enforces the second moment once per session on large diffs. This paper's own review (Section 7) used that mechanism.
- *Mobile channel* (since 24 August 2026): a WhatsApp assistant without deployment capability writes contract-formatted GitHub issues; a watcher on the operator's machine executes them headlessly under a red-amber-green permission scheme.

Prescribed human-only decisions. Goals and priorities; accept, reject, and kill decisions on every piece of work; anything that costs money; anything published or sent to a real person in the operator's name (an explicit "go" is required); credentials, DNS, and mail; deletions; and disclosure decisions about client work. Whether these boundaries held in practice is exactly what the session-log and task-board coding planned for v0.3 will examine; the failure corpus already contains one case of an assistant changing a hosting environment variable on its own (Section 4.6).

4.3 RQ3: Failure-mode taxonomy *(preliminary)*

Corpus and unit. The operator's agents keep a memory directory. Each `feedback_*` file in it records one recurring agent failure and the rule adopted in response, written at the time of the incident because the agents read these files at the start of every session. The unit of analysis is therefore a *rule-worthy failure note*, not an independently sampled incident: a note can summarize several repeated events, notes can overlap, and documentation habits changed over the period (the corpus grew from 42 notes in July 2026 to 118 by 16 September 2026). One-off failures without recurrence risk are under-represented. This version reports the 42 notes coded in July 2026 in a single pass (Section 3.5); the remaining 76 will be coded, with a second coder and an agreement statistic, for v0.3.

Table 3. Seven categories of recurring agent failure (n=42 notes, first coding pass).

#	Category	n	Examples
1	Environment and platform traps	15	API contract differs from the model's prior (endpoint naming); build-tool parser quirks; UTF-8 on Windows; a CLI hanging on stdin for 340 minutes; platform timeouts returning HTML instead of JSON
2	Weak verification, premature success	4	"pushed" reported as "live" before the deploy finished, users saw 404s; image QA checked the `src` attribute while all 18 images were broken; TypeScript in an inline script failed silently after transpilation
3	Fabrication and stale or misjudged knowledge	3	a study author written from memory and wrong; the risk of inventing missing personal data; corrupted legacy metrics treated as valid
4	Objective and metric misalignment	4	optimizing for cost efficiency in a market the principal wanted to grow; measuring platform conversions instead of the business north-star metric
5	Autonomy miscalibration	5	build commands handed to the human as instructions instead of executed; unnecessary approval gates before merge or deploy; asking for feedback instead of starting the next step
6	Resource and cost misuse	5	API budget spent on work the subscription session could do; nine zombie dev servers at full CPU; 30,000 tokens per project entry spent on orientation; credentials in `.env` overlooked
7	Output-channel and compliance errors	4	deliverable produced on the wrong channel (a hosted artifact instead of a local file); staging links on a production page; a legally sensitive professional-status wording

Observations on the coded notes.

- *Countermeasure types.* Memory rule (about 18) > knowledge note (about 13) > process change (about 10) > tooling (about 0). The dominant recorded repair is a persistent rule that every future session reads. Whether rules prevent recurrence is not measured in this version; a growing corpus is equally consistent with growing exposure. Recurrence after rule introduction is a v0.3 analysis.
- *The largest category is environment contact.* 15 of 42 notes fall into category 1; categories 1 and 2 together hold 19 of 42. This says that, among failures the operator found worth a rule, environment behaviour and false success signals were the most frequent; it does not say that reasoning failures were rare in operation, since those may be repaired without a rule.
- *Intent contact.* Categories 4 and 5 (9 of 42) are failures at the contact between the model and the principal's goals or mandate, in line with Tang et al. [2026].
- *Autonomy notes point one way.* All five autonomy notes record agents doing too little. The corpus contains no note of an agent doing too much; over-autonomy incidents may have been handled by rollback or permission changes rather than by a rule, so absence from the corpus is not absence in practice.

Mapping onto MAST. Table 4 gives a coarse mapping of our categories onto the three MAST classes [Cemri 2025]. A category-to-class mapping cannot establish that a category is absent from MAST's 14 individual modes; case-level mapping against the full mode definitions, with multiple assignments allowed, is planned. On the present mapping, categories 1, 6, and 7 (24 of 42 notes) are candidates for extending a benchmark-derived taxonomy with field-specific classes.

Table 4. Field categories versus MAST classes (coarse mapping).

Field category	MAST class	Note
1 Environment and platform traps	no direct class	candidate extension
2 Weak verification	FC3 task verification	congruent
3 Fabrication, stale knowledge	FC1 (partly)	MAST has no hallucination class
4 Objective and metric misalignment	FC1 specification	here: principal intent rather than prompt specification
5 Autonomy miscalibration	FC2 (distantly)	MAST: agent-agent; here: agent-human mandate
6 Resource and cost misuse	no direct class	candidate extension
7 Output channel and compliance	no direct class	candidate extension

4.4 RQ4: Economics *(partial)*

Two cost zones. We separate the coding and agent stack (Zone A) from the wider AI toolkit for design, image, video, and social media (Zone B), which is reported as context.

Documented anchors (receipts). ChatGPT Plus since March 2023. Claude Pro since April 2024 (receipt of 1 April 2024). A second Claude subscription from November 2025 (50 USD receipt on 5 November 2025; tier to be confirmed). A Kimi subscription from February to April 2026 (two accounts, one refunded within the month). A Claude Max tier from June 2026 (200 USD); a proration receipt of 98.72 € on 8 July 2026 indicates a further upgrade. Payment interruptions on one account are documented (paused 16 April 2026, restored 20 April; paused 20 May). Manus receipts exist but are in a mailbox not yet searched.

Pre-agent AI spend (bank export 2022–2024, one account). Jasper 1,357.81 € in 2022 and 302.60 € in 2023; OpenAI 111.80 € in 2023; Midjourney 40.76 € (2023) and 71.66 € (2024); Anthropic 261.94 € in 2024; AppSumo lifetime deals for AI applications 7,967.29 € in 2024 (Zone B). AI spend is not a 2025 phenomenon for this operator; its structure moved from content tools to coding agents.

Bounded estimate for Zone A, 2025 to July 2026. From 2025 the operator's payments were spread across several private and family accounts, so an exact monthly series is not reconstructable from one source. We report a low–high band built from the receipts above and *operator report*.

Period	Stack	€ per month (low–high)
Jan–Oct 2025	Claude Pro, ChatGPT Plus, Gemini Pro (+ Manus months)	45–90
Nov–Dec 2025	+ second Claude subscription	65–140
Jan–May 2026	+ Kimi (Feb–Apr)	85–160
Jun–Jul 2026	+ Claude Max	250–330

Current state (July 2026, *operator report*): three Claude subscriptions (20, 20, and 200 USD) and one OpenAI subscription (20 USD), about 260 USD per month. Year totals: about 600–1,200 € for 2025 and 1,000–1,500 € for January to July 2026. These are the operator's out-of-pocket costs. They understate resources consumed: inference credits provided under a client engagement, which would otherwise have expired, were also used, in unknown volume; and some accounts are held by family members. A full resource-cost account is not possible from the records we have, and we say so rather than estimate it.

Consumed inference (descriptive). The terminal agent keeps local session logs that are retained for 30 days by default. The extract was taken on 11 July 2026, so only roughly 11 June to 11 July is fully covered; May and early June are truncated, which explains the jump between the first two rows of Table 5. Coverage is one account on one machine and one tool. "Logged messages" are entries in the session log, including tool results; sub-agent sessions are separate files and may overlap with their parent's totals. Cache-read tokens are a distinct, cheaper class from input tokens.

Table 5. Token volume from session logs, one account, one machine, terminal agent only.

Month	Logged messages	Input tokens	Output tokens	Cache read	Cache write
2026-05 (truncated)	4,226	0.6 M	1.8 M	564 M	13 M
2026-06 (from ~11 Jun)	366,737	54.0 M	101.5 M	23,641 M	1,138 M
2026-07 (1–11)	65,571	32.1 M	52.6 M	10,123 M	570 M

A list-price valuation of these volumes is deferred. It would require validated unique-usage units, the model mix, contemporaneous per-class prices, and matched account periods; and even then it would measure the vendor's pricing structure, not savings or the cost of equivalent development. The operator's earlier back-of-envelope figure (about 63,000 USD for a six-week window) is withdrawn from this version pending that work.

Quota as an observed constraint. The record shows quota management: three subscriptions with one vendor, load spread across private, family, and client-provided accounts, role assignment partly by quota characteristics (the OpenAI subscription is used for "controlling and sparring, because its tokens are gone in ten minutes otherwise"), and rapid tool adoption and abandonment (Kimi for three months; a year-long Manus subscription left unused). Whether quota, money, or the operator's attention was the

binding constraint at any point is not established; the late-August adoption of a cheaper vendor's agent because it was cheaper shows that price mattered too. Dated rate-limit events and blocked work are not logged and are on the v0.3 list.

4.5 RQ5: Evolution of the work system

The record allows a dated timeline (Table 6). Adoption dates come from receipts, clone or install timestamps, database creation dates, and the operator's versioned instruction files; entries marked † rest on operator report.

Table 6. Tool and practice adoption timeline.

Date	Event	Evidence
Mar 2023	ChatGPT Plus	receipt
Apr 2024	Claude Pro	receipt
Apr 2025	first own repositories; low-code platform exports	git
Apr–Aug 2025	editor agent (Cline) checkpoint repositories on second machine	PC mining
Jun 2025	first agent-marked commits	git
~Nov 2025	openclaw (multi-assistant framework) installed	install directory †
Nov 2025	second Claude subscription	receipt
Dec 2025	level shift in own-commit series; terminal agent as primary worker †	Table 1
26 Jan 2026	goose cloned	clone date
30 Jan 2026	task board created	database creation date
Feb 2026	Gemini CLI histories; Kimi subscription	PC mining; receipt
4 May 2026	skill library (superpowers) adopted	clone date
Jun 2026	Claude Max tier	receipt
8 Jul 2026	fork of a code-navigation server; contribution upstream	clone date, PR
12 Jul 2026	task board rebuilt inside the operator's own workspace app	commits
30 Jul 2026	third peer CLI (Grok Build) installed	instruction file
2 Aug 2026	operator's own orchestration tool (kaprek) published on npm; council feature built	npm, commits
Aug 2026	nightly knowledge-graph build of all operator sources as agent memory	commits from July
17 Aug 2026	peer CLI migration (kimi-code)	instruction file
20–24 Aug 2026	WhatsApp assistant (instinct) adopted; GitHub-issue bridge with contract and watchdog live	memory files, commits
29 Aug 2026	council standard: two fixed review gates, enforced by hook	instruction file
late Aug 2026	second vendor's terminal agent (Z.ai ZCode on GLM 5.3 Flash) in daily use, chosen for price	knowledge-graph entries from 30 Aug †

Stages. The timeline shows six stages: chat copy-paste; low-code export and editor agents; one terminal agent; an orchestrator with typed sub-agents; peers from other vendors with a formal council; and a

mobile channel plus a nightly-built knowledge graph as shared memory. The operator's account (†) is that each stage was added when the previous one hit a specific bottleneck: the terminal agent replaced copy-paste when review of pasted code became the constraint; sub-agents were added when the main context filled with orientation (category 6); peers were added for second opinions after the operator stopped reading every diff; the council was formalized after review gates were skipped; the mobile channel was built so that the system runs while the operator is away with his family. These bottleneck attributions are retrospective and will be checked against dated notes for v0.3.

The substrate. The one component present in every stage after December 2025 is externalized memory: dated rule files, later a router file with topic files, later a nightly graph. Section 4.3 shows that memory rules are the dominant recorded countermeasure. We hypothesize in Section 5 that this substrate is what makes portfolio-scale work survivable for one person; the hypothesis is not tested here.

4.6 RQ6: Cross-system observations *(in progress)*

This version names the tools because the scientific context differs from the operator's public no-brands rule; the decision is confirmed with the author for v0.3. Documented usage, not benchmarks, is the evidence base, and the observations below are single instances from instruction files and memory entries, not a comparison.

- Peers are useful when given a package and a question, and not when given the conversation. Rules in the instruction files forbid interactive debugging with peers because they "do not see the main context and the investigation thread is lost".
- The council's value lies in disagreement. The standard instructs the orchestrator to name where peers agree and where they do not; a first review of the operator's own tool found the same most severe defect (a store that could write anywhere on disk) from two peers independently.
- Findings from a second vendor's model were re-verified before acting. Of one GLM review's points, several were exaggerated or wrong (a claimed keyword collision did not exist).
- Cheaper models moved routine work: the operator reports that the second vendor's agent does "an extreme amount of work" because it is inexpensive and good enough, at the cost of unsigned commits (Section 4.2).
- A tool that cannot deploy found a three-week outage: the WhatsApp assistant discovered a broken payment integration on a live product and changed the environment variable through the hosting dashboard on its own, while deployment remained the terminal agent's job. This is also the one recorded instance of an agent acting beyond the prescribed red list (Section 4.2).

A systematic comparison (task classes per tool, acceptance rates, cost per accepted change) requires the session-log coding planned for v0.3.

---

5 Discussion

What the output record does and does not show. The recorded multiplier of about 12 is robust to the identity and automation corrections and sits between 6 and 14 under alternative boundaries. Its decomposition is mostly breadth: six times as many repositories touched per month, twice as many commits per touched repository. We read this as a description of how one operator's recorded activity changed, not as a productivity measurement. To turn it into one would require hours (not logged), task durations and overlap from session timestamps (available, not yet coded), change survival (planned),

and a control for the shift toward smaller agent commits [Wen 2021]. The operator's account that hours fell after July while output stayed high, if it holds up, would mean output per hour rose; the record cannot confirm it.

Relation to the METR result. Becker et al. [2025] found experienced developers slower with AI on real tasks in large, mature open-source repositories they knew well. Our operator's recorded output multiplied. The two results are not comparable: METR measured task time under a controlled assignment; we measured recorded activity without time. What the two settings suggest, as a hypothesis, is that the effect of agents depends on the work system around them: METR's developers worked single-threaded under external quality bars; our operator worked greenfield on systems he owns, with no external reviewer, cheap kills, and, per his account, in parallel. A non-developer, who has no manual speed to lose, is the extreme case of that regime.

Decision bandwidth as the candidate bottleneck. Davis et al. [2026] describe governance conversion on one system over twelve weeks. Our record is consistent with the same mechanism at portfolio scale: the failure corpus is the visible form of the conversion, and the work system's stages (Section 4.5) were each added, by the operator's account, when a human bottleneck appeared. Whether the bottleneck is decision bandwidth, verification trust, or something else is not identified here; the task-board and session-log analyses are designed to test it.

The failure profile. Among rule-worthy failures, environment contact and false success signals dominate. If that pattern survives coding of the full corpus with a second coder, it points tool builders toward environment instrumentation (live checks, deploy verification, encoding discipline) rather than reasoning benchmarks alone. The categories without a direct MAST class are candidate extensions, to be tested case by case against MAST's mode definitions.

Under-autonomy. The absence of over-autonomy notes in a solo, high-trust setting is a property of this operator's configuration (he instructs agents to act without asking) and of his documentation habits, not of agents in general. The one recorded boundary crossing (Section 4.6) was not written up as a rule, which illustrates the corpus's bias.

Cost and quota. Subscription pricing decouples the operator's out-of-pocket cost from consumed inference, and the record shows active quota management. We do not claim that quota rather than price is the binding constraint; the adoption of a cheaper vendor for volume work shows price at work, and pooling of family and client accounts means that "the operator's cost" is a lower bound on resources consumed. Studies that value agent output at list price should say what they are measuring.

Agent-era mining needs new controls. Kalliamvakou et al. [2014] warned about bots and archives. Our five pitfalls (Section 3.4) extend the list. Two are novel in kind, as far as we know. Automation commits under a human identity arise because operators now build their own automations that commit as themselves; a bot filter on author name does not catch them, and repository-level exclusion removes real work. Marker decay arises because signing conventions are vendor-specific; a coverage figure computed from one vendor's trailer becomes less tight the more vendors the operator uses, and a falling figure can mean more delegation, not less. Population studies that count trailers [Robbes 2026] inherit both problems.

Questions this raises rather than answers. For individuals: whether decomposition, specification, verification design, and kill decisions are the entry skills, given that a non-developer runs 13 deployed systems. For teams: whether "10×" claims conflate breadth and depth as this record does. For tool builders: whether the taxonomy holds beyond one operator, and how much measurement debt unsigned commits are creating for future studies.

6 Threats to Validity

Construct. Commits are recorded activity, not value and not productivity. Baumann et al. [2026] find that only 44% of agent code survives into commits; the fraction of committed code that survives in production is unknown here. Deployments are not independent of commits. "Live" does not establish use. Marker coverage is a lower bound on agent involvement whose tightness changed in August 2026. Commit granularity shifts with tooling. "Own commits" includes platform-bot commits the operator drove.

Internal. No counterfactual exists, and the P2 boundary was chosen on the outcome series; we report sensitivity but cannot remove the selection. Rival explanations for the multiplier that the record cannot separate: more working hours; smaller commits; more projects started rather than more work done; learning; model improvements over the period; changing tool access and quota; the consulting wind-down in February 2026 that freed time; and changing client demand. The April 2026 dip has a documented payment interruption but no full account. The post-July marker series is treated as unmeasured.

External. N=1. The operator is unusual in three ways: no professional software background, twenty years of marketing and automation practice, and full ownership of every system. Transfer to teams, to legacy codebases, or to developers with high manual speed is not claimed.

Reliability. The failure corpus was coded once, by an agent with author consolidation, on a subset. The corpus records only rule-worthy failures. Retrospective operator report carries RQ2, RQ5 (bottleneck attributions), and parts of RQ4 and RQ6, and is marked wherever it does. The author is the subject; the pipeline, the whitelist, the released aggregates, and blind review of each version are the mitigations.

Completeness. Absence of pre-2025 commits on the surveyed machines and remotes does not prove absence of earlier software work. Session logs are retained 30 days and cover one account. Receipts cover one mailbox. Deployment history covers February 2026 onward.

7 Status of This Draft, Peer Review, and Next Steps

Blind review of this version. Before circulation, the draft was reviewed by two independent model-based reviewers (OpenAI Codex and xAI Grok Build), asked blind and in parallel for an adversarial ICSE SEIP review. Both returned "disagree" and recommended rejection of the earlier text in its submitted form; neither cited N=1 as the reason. Their shared objections and what changed:

1. The baseline arithmetic was not reproducible from Table 1 and the "conservative" claim was backwards. Changed: the baseline is now Phase P1 with an explicit ledger; the ratio fell from 17.2 to 11.6 (filtered) and its sensitivity is reported.
2. "Commits per repository stayed in a narrow band" rested on three selected months. Changed: replaced by the full-series breadth and depth decomposition, with the range of depth stated.
3. Automation commits were removed by repository, not by commit, and only for one series. Changed: removed by subject in every figure, with monthly counts.
4. "Mostly instrument decay" overclaimed a residual of more than ten points. Changed: automation share attributed, remainder stated as unattributed and consistent with unsigned tools.

5. The list-price comparison was unverified and did not belong in the abstract. Changed: withdrawn; token volumes reported descriptively with retention and coverage caveats.
6. Instruction files document prescribed, not observed, delegation; MAST "no counterpart" was a mapping claim, not a gap test; "each failure is paid for once" was an effectiveness claim without recurrence data. Changed: reworded throughout as prescribed practice, candidate extensions, and hypotheses.

Points not yet addressed: session-log and task-board coding for observed delegation (v0.3), second coder for the taxonomy (v0.3), change survival (v0.3), and a case-level MAST mapping (v0.3). The full review texts are in the repository.

Item	Status in v0.2	Planned for v0.3 (by 10 October 2026)
RQ1 output	complete; ledger, sensitivity, decomposition	automation filter in the pipeline; changepoint test; deploy refresh Aug–Sep; operator review of inventory; change survival
RQ2 delegation	marker series with attribution limits; prescribed architecture	session-log coding; task-board analysis; signed trailers for all tools
RQ3 failures	42 of 118 coded, single pass	code remaining 76; second coder; agreement statistic; codebook; case-level MAST mapping; recurrence after rule
RQ4 economics	anchors, bounded out-of-pocket estimate, descriptive tokens	receipt tabulation; token-unit validation; rate-limit event log
RQ5 evolution	dated timeline	bottleneck attributions checked against dated notes; two vignettes
RQ6 systems	single documented instances	tool-level task and acceptance analysis
Framing	tools named; working title	confirm naming; native-English pass
Venue	ICSE 2027 SEIP, deadline 23 October 2026; fallback FSE 2027 Industry, 23 January 2027	Zenodo preprint with dataset before submission; arXiv endorsement

We would particularly welcome comment on: the case definition and evidence chain in Section 3.1 for an information-systems audience; whether the breadth-and-depth decomposition is the right way to present Table 1; the treatment of marker coverage after July; and whether an out-of-pocket cost account without a valuation of consumed inference is sufficient for RQ4.

Data Availability

An aggregated dataset (monthly series, repository-anonymized commit metadata, deployment counts, the coded failure taxonomy with codebook) and the analysis scripts will be released with a DOI on Zenodo before submission. Raw repositories, session logs, and receipts remain private.

References

Full BibTeX in `references.bib` (30 entries, all verified against arXiv or DOI on 18 July 2026).

- Agarwal, He, Vasilescu (2026). AI IDEs or Autonomous Agents? arXiv:2601.13597.
- Aleithan et al. (2024). SWE-Bench+. arXiv:2410.06992.
- Badertdinov et al. (2025). SWE-rebench. NeurIPS 2025. arXiv:2505.20411.
- Baumann et al. (2026). SWE-chat. arXiv:2604.20779.
- Becker et al. (2025). Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. METR. arXiv:2507.09089.
- Bird et al. (2006). Mining email social networks. MSR 2006.
- Butler et al. (2024). Dear Diary. arXiv:2410.18334.
- Cemri et al. (2025). Why Do Multi-Agent LLM Systems Fail? NeurIPS D&B 2025. arXiv:2503.13657.
- Chen, Talwalkar, Brennan, Neubig (2026). Code with Me or for Me? CHI 2026. doi:10.1145/3772318.3790850.
- Davis et al. (2026). Cheap Code, Costly Judgment. arXiv:2607.01087.
- Ellis, Adams, Bochner (2011). Autoethnography: An Overview. FQS 12(1).
- Flyvbjerg (2006). Five Misunderstandings About Case-Study Research. Qualitative Inquiry 12(2).
- Goeminne, Mens (2013). A comparison of identity merge algorithms. SCP 78(8).
- He, Miller, Agarwal, Kästner, Vasilescu (2026). Speed at the Cost of Quality? MSR 2026. arXiv:2511.04427.
- Hoess et al. (2025). Does the Tool Matter? arXiv:2501.15114.
- Huang et al. (2025). Professional Software Developers Don't Vibe, They Control. arXiv:2512.14012.
- Jimenez et al. (2024). SWE-bench. ICLR 2024. arXiv:2310.06770.
- Kalliamvakou et al. (2014). The Promises and Perils of Mining GitHub. MSR 2014.
- Kalliamvakou et al. (2016). An in-depth study of the promises and perils of mining GitHub. EMSE 21(5).
- Kaltenhauser et al. (2024). Playing with Perspectives and Unveiling the Autoethnographic Kaleidoscope in HCI. CHI 2024. doi:10.1145/3613904.3642355.
- Liang et al. (2025). The SWE-Bench Illusion. arXiv:2506.12286.
- Liu et al. (2025). An Empirical Study on Failures in Automated Issue Solving. arXiv:2509.13941.
- Mohamed, Assi, Guizani (2025). Systematic review of AI coding assistants (39 studies). arXiv:2507.03156.
- OpenAI (2025). Why we no longer evaluate SWE-bench Verified.
- Peng et al. (2023). The Impact of AI on Developer Productivity. arXiv:2302.06590.
- Robbes et al. (2026). Agentic Much? arXiv:2601.18341.
- Runeson, Höst (2009). Guidelines for conducting and reporting case study research in software engineering. EMSE 14(2).
- Tang et al. (2026). How Coding Agents Fail Their Users. arXiv:2605.29442.
- Wen, Nagy, Lanza, Bavota (2021/2022). Quick Remedy Commits. EMSE.
- Ziegler et al. (2022). Productivity Assessment of Neural Code Completion. MAPS 2022. arXiv:2205.06537.