Technical report v1.0 2 November 2026 27 min read

Green but Wrong: An Experience Report on Defects in AI-Coding-Agent Output Across Eight Products, and the Revert-the-Fix Protocol

Yousof Selim
Sunway University (Lancaster University dual degree), Malaysia

67% of serious bugs were found by adversarial review

1. Dedicated adversarial review found 24 of the 36 high-severity defects (67%) in the record.
2. 21 of 183 defects (11%) were present while a test, gate or written claim covered them.
3. 65 of 182 fixes (36%) record a revert or failing-first check, as reported by the agents themselves.
4. Using the apps on a device found 20 defects, mostly visual and platform, and none security-related.

Limits. One developer's record, coded by one AI coder, with red evidence self-reported by the agents and a dataset that favours defects someone documented.

Abstract

The problem	AI coding agents write code that passes its own tests. How often does that green result hide a real defect, and which practices actually catch it?
What I did	I mined the git history and project records of eight products I built with Claude Code and Codex (1,721 commits, June to November 2026) and coded 183 defects by category, severity, how each was found and whether its fix was shown to fail without it.

What I found	**Dedicated adversarial review found 24 of the 36 high-severity defects (67%). 21 defects (11%) sat under a passing test, gate or written claim. 65 of 182 fixes (36%) record a revert or failing-first check: 55% of review-found fixes against 10% of those found by using the app.**
Why it matters	A green suite on agent-written code is a claim, not proof. The report sets out a revert-the-fix protocol with adversarial review, and where it fell short. It is one developer's record, coded by an AI agent, and the red evidence is the agents' own report.

Contributions

- A dataset of 183 defects from eight agent-built products, each tied to commit hashes and a primary source, with the read-only mining scripts.
- A sixteen-category taxonomy for defects in multi-component, agent-built product code, with the tests themselves as a defect locus.
- A descriptive comparison showing that, in this record, adversarial review, the implementer's own checks and the owner's device use found largely different kinds of defect, with very different rates of recorded red evidence.
- A written revert-the-fix and adversarial-review procedure, together with the failures of that procedure the record shows.

Keywords. AI coding agents · test adequacy · mutation testing · adversarial review · defect taxonomy · experience report · row-level security · Claude Code · Codex

1 Introduction

A passing test suite is a claim: that the behaviour it checks is the behaviour the product has. This report studies cases in which that claim was false, in software written mostly by AI coding agents. Between June and November 2026 one developer, Yousof Selim, directed Claude Code and OpenAI Codex agents to build eight products across mobile, web, desktop and database back ends. The repositories hold 1,721 commits; 1,419 carry a `Co-`

`Authored-By: Claude` trailer and a further 123 sit on Codex branches without one. The commit messages and project documents record many of the defects found along the way in unusual detail: what was wrong, how it was found, and whether the fixing test was ever seen to fail.

Several records describe the same pattern. In Bupples, a bill-splitting app, 2,582 tests passed while a user's profile bio silently never saved, because the test that covered it asserted a method with no production callers. In WayClub, a club-management platform, a shipped test asserted that an advisor's approval survives an edit to the event, which is the approval-routing defect an adversarial review later demonstrated. In CareerOS, the publication gate's evaluation reported 14 of 14 caught until eight adversarial cases were added, after which the old gates caught 14 of 22.

Controlled studies have shown that AI assistance can make code less secure while making its authors more confident [1], that suggested code often contains known weakness classes [2], and that agent patches which pass a benchmark's tests are not always correct [3][4][5]. Less is known about the defects that reach a multi-product codebase built by agents over months, and about which working practices caught them. This report examines that record through three research questions:

1. **RQ1.** What kinds of defects appear in agent-built product code once it is reviewed, used and tested, and how severe are they?
2. **RQ2.** Which practices detected the defects, and which kinds of defect did each practice find?
3. **RQ3.** How often did a passing check cover a defect while it was present, and how often was the fix shown to fail without it?

The report makes four contributions:

- a dataset of 183 defects from eight products, each tied to commit hashes and a primary source and coded for category, severity, detection, attribution and strength of red evidence, with read-only mining scripts (Section 4 and the artifact);
- a sixteen-category taxonomy for defects in multi-component, agent-built product code, including a category for defective tests (Section 5.1);
- a descriptive comparison of detection practices: review found 87 of the 183 defects and 24 of the 36 high-severity ones, and its fixes carried recorded red evidence in 55% of cases, against 10% for defects the owner found by using the product (Sections 5.2 and 5.4);
- a written procedure, the revert-the-fix protocol with adversarial review, distilled from what the repositories show working, with the ways it was observed to fail (Section 6).

This is an experience report on one developer's practice. It is not peer reviewed, and the threats in Section 8 and limitations in Section 9 are substantial. The counts describe this record, not AI agents in general, and none of the comparisons is causal.

---

2 Related work

2.1 Security and correctness of AI-generated code

Pearce et al. prompted GitHub Copilot with 89 scenarios built around high-risk CWE classes and found that roughly 40% of the 1,689 generated programs were vulnerable [2]. Perry et al. ran a user study in which participants with an AI assistant wrote significantly less secure code than a control group and were more likely to believe their code was secure [1]. That second finding matters here: the risk is not only defective code but misplaced confidence in it. Tambon et al. examined 333 bugs in code from three LLMs and derived ten bug patterns, among them misinterpretation of the prompt, missing corner cases, hallucinated objects and wrong attributes [6]. Their taxonomy describes function-level generation; the taxonomy here describes defects in multi-component products, where many of the most serious failures sit between components rather than inside one function.

2.2 When passing tests are not enough

SWE-bench measures whether a model can resolve real GitHub issues, judged by whether the repository's tests pass after the patch [3]. Wang, Pradel and Liu studied patches that pass that validation and found that 7.8% of patches counted as correct fail the developer-written test suite, and estimated from a manual sample that about 11% of plausible patches are incorrect [4]. Liu et al. showed with EvalPlus that adding generated tests to HumanEval changed both measured correctness and model rankings [5]. All three point at the same weakness this report meets in practice: a test oracle that is too weak makes wrong code look right. The difference is scale and setting. Those studies examine benchmark patches judged against fixed tests; this one examines defects that a developer's own, evolving product suites passed over, and asks which practices exposed them.

2.3 Mutation testing and test adequacy

Mutation analysis asks whether a suite can tell the program from small deliberate faults in it [7]; Papadakis et al. survey four decades of the technique [8]. Just et al. found that mutant detection correlates with real-fault detection even after controlling for coverage,

while also finding real faults that no mutant coupled to [9]. The revert-the-fix protocol in Section 6 is a narrow, manual mutation analysis: the "mutant" is the absence of one specific fix, which is by construction coupled to a real fault. Test-driven development's red-green cycle [10] asks for the same observation before the fix is written; the protocol here asks for it after, which is what an agent fixing an inherited defect can actually do.

2.4 Test reliability

Luo et al.'s analysis of flaky tests in 51 Apache projects classified root causes such as asynchronous waits, concurrency and test-order dependency [11]. Flakiness is the mirror image of the problem here: a flaky test fails when the code is right, whereas a vacuous test passes when the code is wrong. Two records in this dataset sit on the boundary, a revocation test that needed a 1.1 second sleep to pass and an audit-chain test that passed only because the suite ran serially.

2.5 Position of this report

Prior work measures defects in generated code at function or patch level, under controlled prompts or benchmark tests. This report differs in three ways: its unit is a defect in a shipped or near-shipped multi-component product; it records how each defect was detected, not only that it existed; and it codes whether the fix was ever observed to fail without it, a field that benchmark studies take for granted and that product records rarely state. The cost is control: the data are observational, from one developer, and depend on what the agents chose to write down.

---

3 Context: one developer, many agents

All eight products were built by agents working under the direction of one developer, who wrote specifications and feedback, reviewed results, tested on devices and decided what shipped. The agents were Claude Code sessions (model names in trailers include Claude Opus 4.8, Opus 5, Opus 5.5, Fable 5 and Fable 5.1) and OpenAI Codex sessions. The two were often alternated on the same repository, and a standing instruction was that an agent inheriting another's work should audit it before building on it. Table 1 lists the products and their state on 1 November 2026.

Table 1. Products, repositories and state. Commits are all commits on all branches at the census, 1 to 2 November 2026.

Product	What it is	Stack	Commits	State
Bupples	Group bill splitting and settling	Flutter, Firebase	1,250	Shipped mobile app; release builds through 1.4.6 (September 2026); design rework in progress
CareerOS	Evidence-backed career record and publishing	Next.js, Postgres	89	Live pilot on Google Cloud Run since 21 October 2026
WayClub	Group management for clubs and teams	NestJS, Next.js, Postgres RLS, Flutter	106	In development, pre-launch; an August 2026 deployment was retired
Photoshoot	Photo booth for desktop and web	Electron, WebGL	78	Released (desktop 1.5.1, web app live)
CodeAtlas	Evidence-bound code-change analysis	TypeScript	65	Local implementation verified in CI; not deployed
Adelante	Quotes, widgets and sharing	Flutter, Firebase	40	iOS build 21 prepared for App Review (October 2026); design rework in progress
Planket	Shared calendar for friends	Next.js, Capacitor, Supabase	46	In development; TestFlight builds
HACKFAM	Hackathon teammate matching	Supabase, Flutter, web	47	In development

The repositories differ in how much they document, which shapes the dataset (Section 8.2). Bupples carries eleven read-only performance audits, a consolidated security audit and a quality audit under `docs/audits/`. WayClub keeps `docs/SECURITY.md`, `docs/TESTING.md` and architecture decision records that narrate defects with measurements. CareerOS keeps a session-by-session `docs/PROGRESS.md`. HACKFAM keeps verification records under `docs/verification/`. Photoshoot keeps

`docs/HANDOFF.md`. CodeAtlas, Adelante and Planket rely mostly on commit messages, and CodeAtlas's Codex commits are one line each.

---

4 Method

4.1 Data sources

Two read-only passes were made over the eight repositories on 1 and 2 November 2026. No repository was modified; the scripts run only `git log`, `git rev-list`, `git for-each-ref` and `git show -s`, and documents were read from the working tree.

- **Commit census.** `mine_commits.py` counts every commit on every branch and attributes each to an author kind (Section 4.4). It also extracts every commit whose message matches `fix(es|ed)?|bug|regress(ion)?|revert(ed)?|root cause|defect` (531 candidates, 433 of them in Bupples).
- **Documents.** The audit, security, testing, progress, handoff, verification and decision documents named in Section 3.

4.2 Inclusion rule

A defect is included when a primary source in the repository (a commit message or a document) (a) states that previously committed code, tests or documents behaved wrongly, (b) describes the wrong behaviour concretely enough to classify, and (c) records a fix or an explicit decision about it. Design feedback, missing features, commits that name no behaviour (for example a one-line "fix: serialize launch prompts after setup"), research text that never became code, and fixes to flaky test selectors are excluded. A commit or document that records several distinct defects yields several rows; a defect recorded in both a document and its fix commit yields one.

Screening covered every fix-pattern commit in the seven smaller repositories and, for Bupples, a random sample of 40 of its 433 candidates drawn with a fixed seed (20261101), plus the documents. Fifteen of the 40 sampled Bupples commits met the rule (37.5%; Wilson 95% interval 24% to 53%), which suggests that roughly 105 to 229 of Bupples' 433 fix-pattern commits record a qualifying defect. The dataset therefore holds all documented Bupples defects but only a sample of the undocumented ones. The result is 183 defects: 104 from documents, 64 from screened commits and 15 from the Bupples sample (Table 2).

Table 2. Defects by product and inclusion path (D documents, C commit screening, S Bupples sample).

Product	D	C	S	Total
Bupples	39	0	15	54
CareerOS	21	14	0	35
WayClub	24	12	0	36
Photoshoot	6	11	0	17
CodeAtlas	1	15	0	16
Adelante	2	6	0	8
Planket	2	6	0	8
HACKFAM	9	0	0	9
Total	104	64	15	183

4.3 Coding

Each defect was coded on six fields from its source text.

- **Category**, from a taxonomy derived bottom-up while reading the sources and then fixed: authorisation and tenant isolation; privacy and data exposure; business-rule and workflow logic; concurrency, locking and idempotency; stale state and caching; UI state, visibility and layout; abuse, injection and resource limits; performance and resource lifecycle; cross-component contract mismatch; test adequacy (the test or verification itself was wrong); money arithmetic and ledger; platform-specific runtime; error handling and honest states; units, frames and precision; build, configuration and operations; overclaiming or stale documentation. Full definitions are in the artifact.
- **Severity.** High: a security or privacy breach, data loss, wrong money, or the product's core action unusable. Medium: a feature wrong or unusable on a path or platform, or a user-visible loss of availability or performance. Low: cosmetic, minor or latent. Where a source audit states a severity (the Bupples security and quality audits) it is used.
- **Detection**: a dedicated adversarial or audit review, run read-only by one or more agents; the implementing agent's own verification (re-running suites, mutation checks, writing proof tests, driving the running product); the owner using the product or testing on a

device; an automated gate already in place (CI, an existing test, an axe accessibility check, an evaluation set); or not recorded.

- **Red evidence**, an ordinal scale for how strongly the source shows that the fixing test fails without the fix: revert or mutation observed red (RV); test or exploit observed failing before the fix (RF); test commit precedes fix commit, inferred but not observed (RO); failure reproduced by hand before and after, no committed test (RM); test added but red not recorded (T); no test recorded (U); no test, stated or evident (N); not fixed (X).
- **Green over the defect**: whether the source documents a passing test, suite, gate or written claim that covered the defect while it was present.
- **Author kind** and attribution tier (Section 4.4).

4.4 Attribution rule

The developer is the committer of record on every commit, so authorship of record says nothing about who wrote the change. The census applies a mechanical rule per commit: a `Co-Authored-By: Claude` trailer means Claude (or "Codex work committed by Claude" when the message also names Codex); otherwise a commit reachable only through a `codex/*` branch means Codex; otherwise unknown. For defects the rule is tiered. Tier A: the introducing commit was identified with `git log -S` or `--diff-filter=A` and carries a trailer (21 defects, all Claude). Tier B: the repository documents the provenance, for example "a read-only adversarial review proved 3 HIGH and 5 MEDIUM defects" in code a progress log attributes to a Codex session, or a Codex branch without trailers (39 defects, all Codex). Tier C: the introducing commit was not identified, and the defect is attributed from the period, as Claude when every commit in that period carries a Claude trailer and otherwise as "agent" without a vendor (117 defects: 56 Claude, 61 agent). Tier D: unknown (6, all in Adelante, whose early commits carry no trailer and whose defects could not be traced to an attributed commit).

The rule exposed one conflict worth reporting. Twelve CareerOS commits dated 17 October 2026 carry Claude trailers, yet the project's progress log records that session as a Codex continuation, and the first of them describes itself as continuing "from the existing scaffold". The defects introduced there are attributed by the document (tier B), and the census figures should be read with that in mind: a trailer records who committed, which in a two-agent workflow is not always who wrote.

4.5 Analysis

All analysis is descriptive: counts, cross-tabulations and proportions, produced by `analyse.py` from `defects.csv`. Proportions are reported with Wilson 95% intervals, and two contrasts with Fisher's exact test. Because the 183 defects are not a random sample

of anything, the intervals indicate how much a proportion could move with a handful of different cases; they are not estimates for a wider population.

4.6 Second-pass consistency check

All coding was done by one coder: an AI agent (Claude) working under the developer's direction, the same arrangement that produced much of the code. Independent human double-coding was not available. As a weaker substitute, a deterministic keyword coder was written against the codebook and applied to the short defect titles alone, without access to the first-pass codes, and compared on a seeded random sample of 40 (seed 20261102). It agreed on category for 32 of 40 (80%; Cohen's $\kappa = 0.77$), and for 83% of all 183 ($\kappa = 0.81$). The eight sample disagreements were mostly boundary cases between concurrency, logic and money (for instance an audit-chain fork that the keyword coder filed under money because the word "concurrency" contains the substring "currency"). This is a check that the codebook can be applied reproducibly, not an inter-rater reliability measure: the keyword rules were written by the same coder after the first pass, which inflates agreement.

---

5 Results

5.1 RQ1: kinds and severity of defects

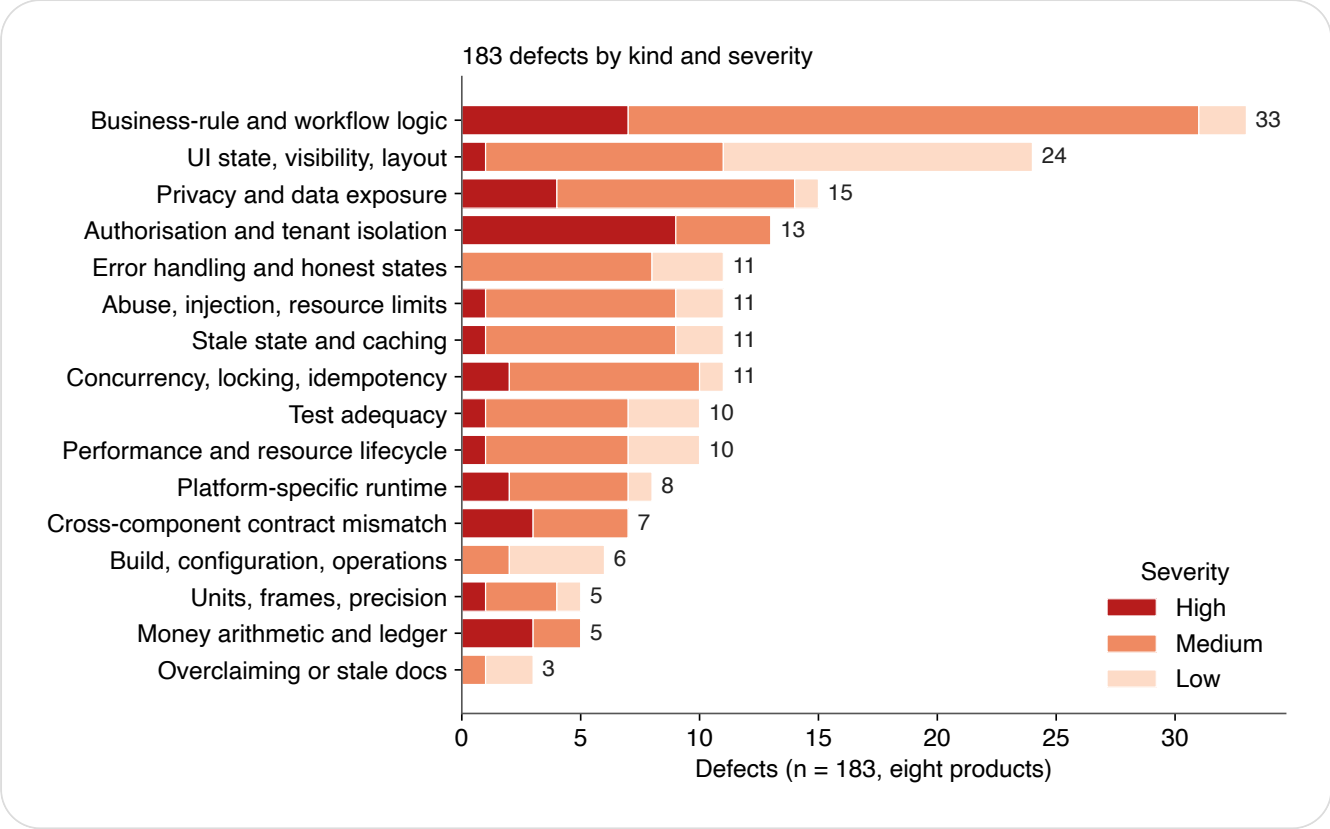


Figure 1. The 183 defects by category and severity.

Figure 1 and Table 3 give the distribution. Of the 183 defects, 36 (20%) are high severity, 109 (60%) medium and 38 (21%) low. The largest category is business-rule and workflow logic (33), followed by UI state (24), privacy (15) and authorisation (13). The three security-relevant categories (authorisation, privacy, and abuse and resource limits) together hold 39 defects, 14 of them high severity.

Table 3. Category by severity.

Category	High	Medium	Low	Total
Business-rule and workflow logic	7	24	2	33
UI state, visibility, layout	1	10	13	24
Privacy and data exposure	4	10	1	15
Authorisation and tenant isolation	9	4	0	13
Concurrency, locking, idempotency	2	8	1	11
Stale state and caching	1	8	2	11
Abuse, injection, resource limits	1	8	2	11
Error handling and honest states	0	8	3	11
Performance and resource lifecycle	1	6	3	10
Test adequacy	1	6	3	10
Platform-specific runtime	2	5	1	8
Cross-component contract mismatch	3	4	0	7
Build, configuration, operations	0	2	4	6
Money arithmetic and ledger	3	2	0	5
Units, frames, precision	1	3	1	5
Overclaiming or stale docs	0	1	2	3

Four shapes recur across products. Each looks correct when its parts are inspected one at a time.

Controls that are right about what, but wrong about whom. Nine of the 13 authorisation defects are high severity. In Bupples, the Firestore rule that let a member leave a hangout checked that the caller had been a member, was now absent, and that the member list shrank by exactly one; it never checked that the new list was a subset of the old, so a "leave" could evict the host and add anyone (B01, introduced in `29796ba7`, fixed in `a6a5d9a7`). In WayClub, `app.write_audit` accepted a row naming another tenant (W09) and waitlist promotion was authorised on tenancy alone (W11). In HACKFAM, a migration

revoked execute rights with a per-schema default that cannot override PostgreSQL's global PUBLIC grant, so every private helper, including one returning another person's profile strength, stayed callable (H01). WayClub's security document names the shape directly: each was "a control that looked correct until somebody asked who exactly it applied to".

Contracts split across two components. Seven defects were mismatches between two parts that each passed their own tests. Bupples' profile bio was written to one document, read from another, and copied between them by a server function whose allowlist omitted the field (B14). WayClub's web questionnaire and server derivation used disjoint answer keys, so a high-risk event submitted from the web was classified as low risk (W04), and the UI sent no body to an endpoint that required one, making the product's core action a guaranteed 400 (W05).

Wrong frames and units. Photoshoot uploaded face landmarks top-down to shaders that sample bottom-up, and mirrored x twice, so every face effect landed below or beside the face (P08). WayClub revoked sessions by comparing a timestamp against the JWT `iat` claim, which has whole-second resolution, leaving a one-second window in which a revoked token survived (W12). CareerOS's numeric gate tokenised ISO dates as year-month, so its date rule never matched (C12).

Platform behaviour invisible on the development machine. Photoshoot requested a WebGL context with the `desynchronized` hint. Chromium ignores that hint on macOS, where the app was developed, and honours it on Windows, where on Direct3D 11 it presented a black buffer while the app reported "Camera live" (P01). The fix commit records 0 of 256 sampled pixels lit on a Windows runner with the hint and 256 of 256 without it. Planket's iOS scene delegate used the plain bridge controller, so a build announced as carrying native fixes never ran them (K01).

Ten defects sit in the tests themselves (the test-adequacy category). They include a revocation test that waited 1.1 seconds and documented the window as acceptable (part of W12), an audit-chain verifier that walked the wrong column and passed only because the suite never overlapped two transactions (W21), a tenant-isolation coverage query that silently excluded every table added after migration 017 (W22), a HACKFAM badge test that exercised only a closed match and so could not fail (H06), and a mutation run in which 4 of 36 mutants survived and exposed weak tests (H08).

5.2 RQ2: how defects were detected

Table 4. Detection by severity.

Detection	High	Medium	Low	Total
Adversarial or audit review	24	48	15	87
Implementer's own verification	4	28	10	42
Owner use or device testing	4	12	4	20
Automated gate (CI, test, axe)	1	1	4	6
Not recorded	3	20	5	28

Dedicated review found 87 of the 183 defects (48%; 95% CI 40% to 55%) and 24 of the 36 high-severity ones (67%; 50% to 80%) (Table 4). Part of that share reflects selection, because reviews leave documents and documented defects were easier to include (Section 8.2). These reviews took several forms: Bupples' eight-domain security audit and eleven performance audits, WayClub's three-lens adversarial review (tenancy and RLS, API and workflow, web and accessibility), CareerOS's review of inherited Codex code and its later security review, HACKFAM's critical review of Codex's specifications, and the "fresh agent" reviews in Photoshoot. In every case they were run by agents, read-only, with an instruction to prove findings with reproducible output. They were independent of the implementing session, not of the agent families: the reviewers were themselves Claude or Codex sessions. Their findings were not always accepted at face value: WayClub records that all eighteen review defects were "re-verified by replaying the reviewer's own exploit rather than by trusting a fix report".

The implementing agent's own verification found 42, the owner 20 and pre-existing automated gates only 6. The gate figure is low partly by definition, since a defect a gate catches before commit leaves no trace in history, but the six that did leave a trace are instructive: two axe contrast failures, a typecheck, a debug assertion, a pgTAP test that went red on inherited Codex SQL (H01), and CodeAtlas's continuous integration run 37443139235, which passed 242 of 243 tests and exposed a file-ownership race in artefact cleanup (G09; the merge run 37453388832 passed all 247).

Review also surfaced its own blind spots. WayClub's security document records two defects "found during verification, not by the review": the one-second revocation window, which sat inside the review's own fix, and a policy that refused the one person a handover names to confirm it (W12, W20). In Bupples, two source files containing NUL bytes were invisible to `grep` and had "misled an auditor into calling live code dead" (B20).

Which practices found which kinds of defect

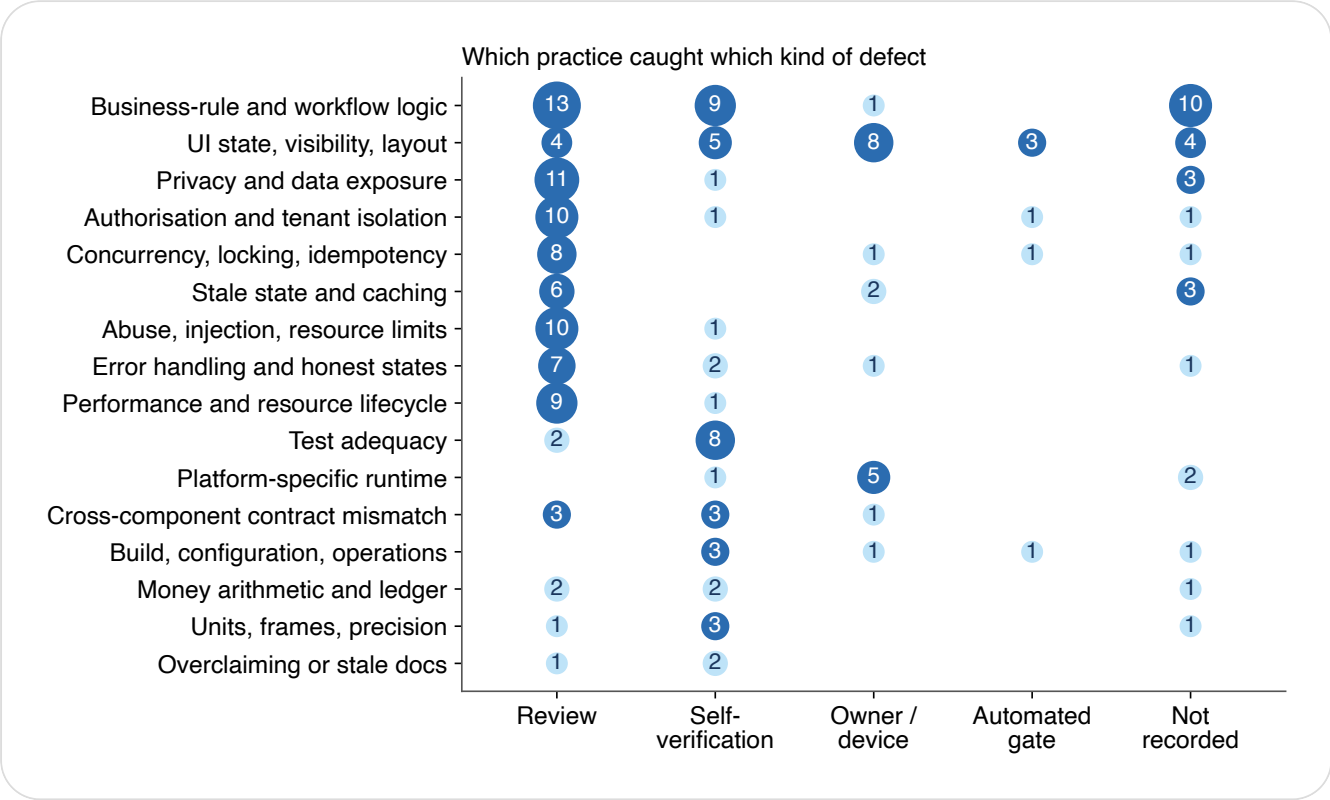


Figure 2. Which practice caught which kind of defect.

In this record the practices found largely different kinds of defect (Figure 2).

- **Adversarial review** found 31 of the 39 security-relevant defects (10 of 13 authorisation, 11 of 15 privacy, 10 of 11 abuse), 8 of 11 concurrency defects and 9 of 10 performance defects. These defects require reasoning about an adversary, an interleaving or a lifecycle, which a feature-oriented test seldom attempts.
- **The implementer's own verification**, especially mutation checks, found 8 of the 10 test-adequacy defects. A suite's weakness became visible when someone tried to make it fail.
- **The owner on a device** found 8 of the 24 UI defects and 5 of the 8 platform defects; 16 of the 20 owner-found defects are UI, platform, stale-state or error-handling defects. None of the security defects came from the owner.
- **Existing automated gates** caught six defects of mixed kinds; the pgTAP suite that caught HACKFAM's privilege leak (H01) is the only gate catch of a high-severity defect.

By author kind, the 39 Codex-attributed defects are dominated by business-rule and evidence-integrity logic (18), reflecting CodeAtlas's history; the 77 Claude-attributed defects are spread more widely, led by UI (13) and logic (11). These distributions follow which products each agent worked on more than any property of the agents, and with tier

C attribution resting on the period rather than the line, the dataset does not support a comparison of vendors.

5.3 RQ3a: defects under a passing check

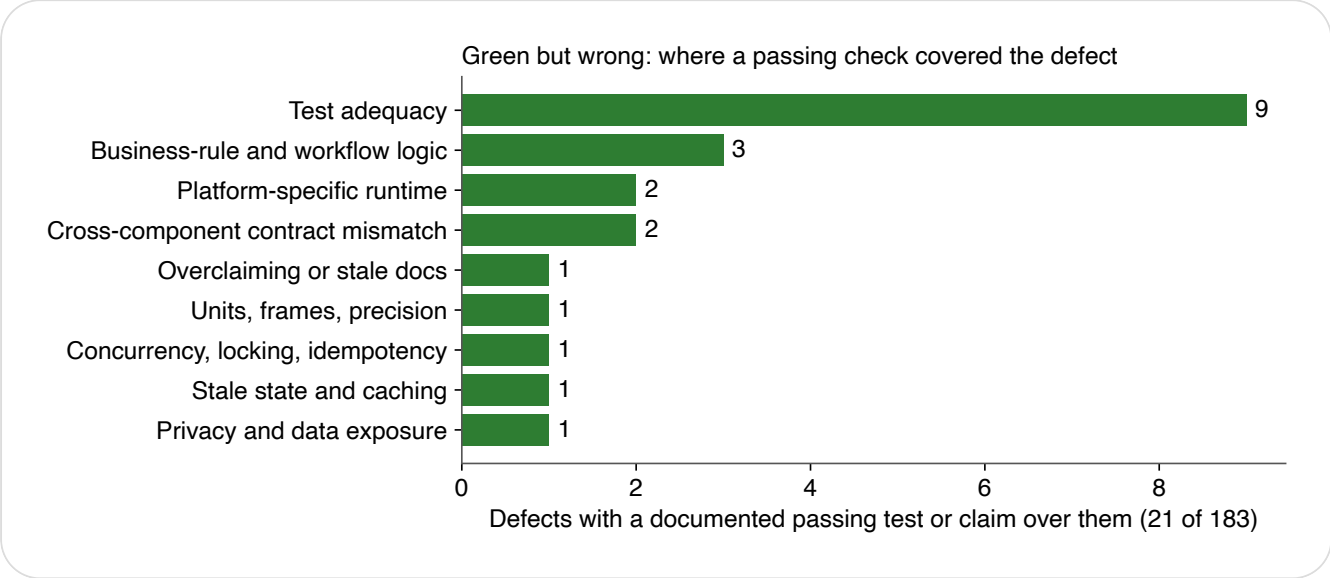


Figure 3. The 21 defects that a passing check or written claim covered while they were present.

For 21 of the 183 defects (11%; 95% CI 8% to 17%) the source documents that something green covered the defect while it was present (Figure 3); for 150 it records no such check, and for 12 the record is silent. Nine of the 21 are in the test-adequacy category. The cases fall into five shapes:

1. **The test asserts something other than the production path.** Bupples' bio test checked a client-side projection method with zero production callers, while the server owned the real field list (B14).
2. **The test overrides the thing under test.** Bupples' how-to-pay test overrode the very provider whose stale cache was the defect (B15).
3. **The test encodes the defect.** WayClub shipped a test asserting that an advisor stays approved after the event changes (W01), and two tests asserting the transition defects the review later demonstrated (W10).
4. **The test cannot exercise the failure.** The revocation test slept past the window (W12); the chain test never overlapped transactions (W21); CareerOS's adapter test for a Next.js request bug "does not reproduce the Next-runtime error, which only the live check catches" (C17).
5. **A written claim stands in for a test.** The Bupples commit that relaxed the direct-message photo rule stated that the change was safe because the URL lived only in a participant-gated message; the later audit found every direct-message photo readable by any

signed-in user through a predictable path (B02). WayClub's security, testing and workflow documents "overstated the code" until corrected (W36). Planket's "native fixes" never ran (K01).

The CareerOS publication gate (C19) shows the evaluation-set version of the same problem. In session 5 its evaluation reported a gate catch of 14 of 14. When eight adversarial cases were added, the old gates caught 14 of 22 (0.64), letting private repository names and links through to public drafts; twelve of fourteen new unit tests failed on the old gates, and after the fix the evaluation caught 22 of 22 (commit `70b84e6`).

5.4 RQ3b: how often fixes were proved red

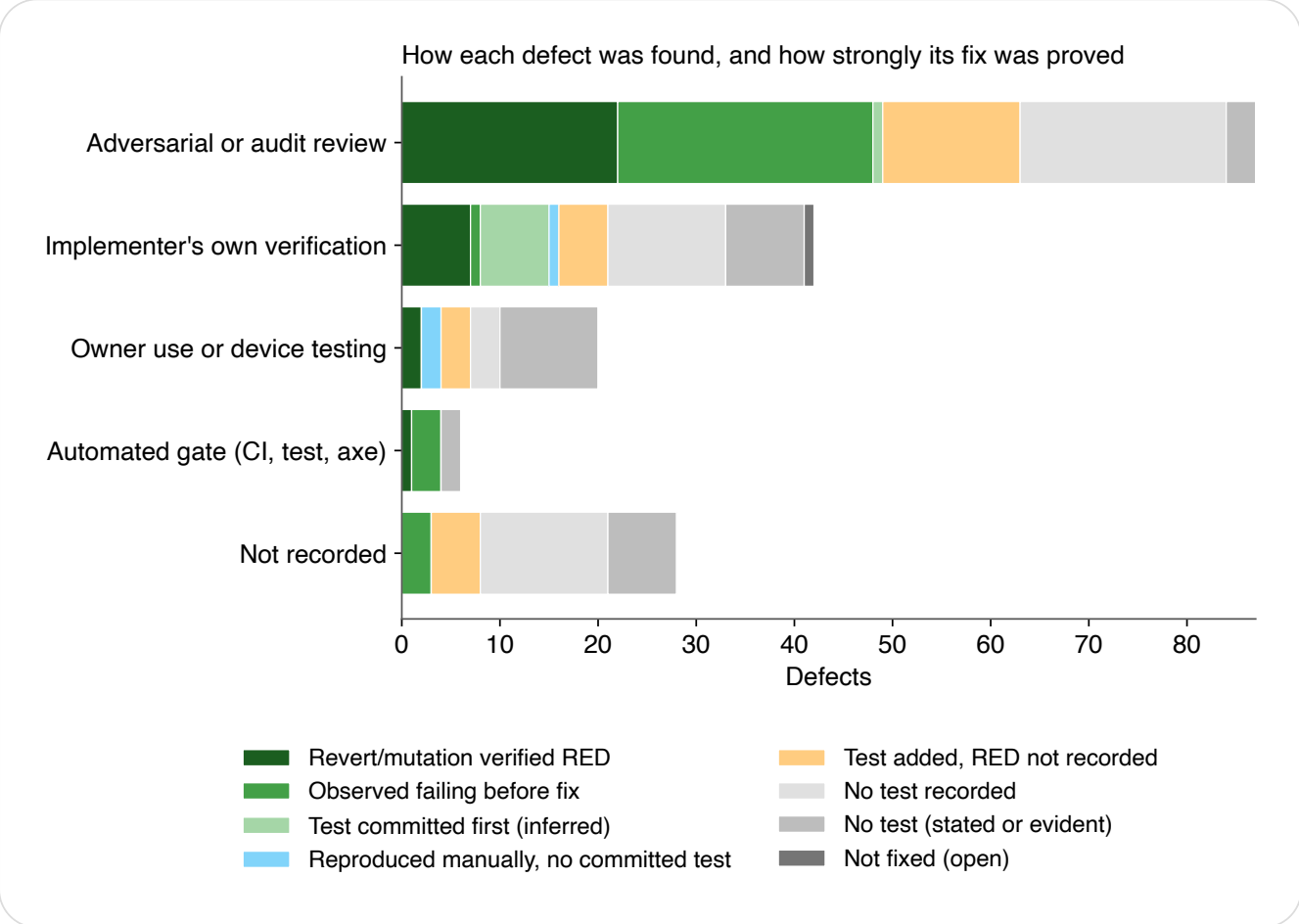


Figure 4. How each defect was found, and how strongly its fix was proved red.

One defect (H09) is still open. Of the 182 fixed defects, 65 (36%; 95% CI 29% to 43%) carry strong red evidence: 32 with a revert or mutation of the fix observed to turn the test red and 33 with a test or exploit observed failing before the fix. Eleven more carry weaker evidence (8 inferred from commit order, 3 reproduced by hand), for 76 (42%) in all. Twenty-seven have a test with no recorded red, 49 have no test recorded, and 30 have no

test at all. For the 36 high-severity defects the strong-evidence share is 19 of 36 (53%; 37% to 68%).

The share depends heavily on how the defect was found (Figure 4). Defects found by dedicated review carry strong red evidence in 48 of 87 cases (55%; 45% to 65%); fixed defects found by the implementer's own verification in 8 of 41 (20%; 10% to 34%); those found by the owner in 2 of 20 (10%; 3% to 30%). The review-versus-owner and review-versus-self contrasts are both unlikely to be chance alone (Fisher's exact test, $p < 0.001$ in each case), but they are confounded: review findings were mostly security and logic defects in products whose sessions logged mutations, while owner findings were mostly visual defects with no test harness. Half of the owner-found defects (10 of 20) were fixed with no test at all. These were mostly visual and platform defects (a blurred cover leaking a coloured bar, a decorative tick, buttons swallowing first taps on iOS) for which the repositories had no rendering or device test harness. The two owner-found defects that were proved red are both Photoshoot camera faults, and in both the agent built the harness the bug required: a Windows Direct3D 11 Playwright project that reads real pixels (P01) and an Electron test of grid visibility (P06).

By product the picture is uneven. CareerOS records strong red evidence for 27 of 35 defects, because its sessions logged each mutation and its result; the session-7 audit of the acceptance suite lists, criterion by criterion, which guard was broken and which test went red. Bupples' 21 revert-verified defects rest mostly on one aggregate statement in its performance completion report ("every behavioral fix carries a test that was verified RED against the pre-fix code by stash-reverting `lib/`") and on individual commit messages that say the same; Planket records none. CodeAtlas's Codex history never states red results, but in eight cases a commit adding tests immediately precedes the fix, for example "test: capture runner provenance boundary gaps" before "fix: bind local runner evidence to execution", which is consistent with red-first work but is not an observation.

---

6 The revert-the-fix protocol and adversarial review

The practices that produced strong red evidence in the record can be written as one procedure. It is a description of what produced evidence here, with the failures the record shows, not a validated method; no comparison with a control workflow was made.

6.1 Adversarial review before trust

1. **Review inherited or finished work read-only, in lenses.** Give each reviewer one lens (tenancy and authorisation; API and workflow; client, accessibility and platform; performance and lifecycle) and forbid edits. Bupples' audits required a file and line for every finding; WayClub's required reproducible output.
2. **Demand proof, not opinion.** A finding counts when the reviewer shows the exploit or the failing behaviour: a forged row accepted, a cross-tenant read returned, 4,780 ms against 43 ms. WayClub's review also recorded what it refuted (cross-tenant sweeps that leaked nothing, rejected `alg:none` tokens), which keeps a review honest about coverage.
3. **Ask "applies to whom?" of every control.** The most serious defects were controls correct in what they checked and wrong in whom they bound: the leaving member, the caller's tenant, PUBLIC, the person named on a handover.
4. **Re-verify fixes by replaying the reviewer's exploit**, not by reading the fix report.

6.2 Revert the fix

1. **Write the test against the production path**, the real seam, not a projection, a fake that cannot model the defect, or an override of the provider under test.
2. **Revert the fix and run the test.** In Bupples this was `git stash` of `lib/`; in CareerOS a mutation of the guard restored with `git checkout`; in HACKFAM a scripted mutation inside a rolled-back transaction, or a file restored byte-identically and checked by SHA-256. The test must fail, and fail on its named assertion.
3. **If it passes, the test is pointed at the wrong thing.** Fix the test, not the threshold. HACKFAM's first mutation run turned 32 of 36 mutants red; the four survivors exposed weak tests, which were strengthened until the next run turned all 41 red.
4. **Restore and re-run green**, and record the command and the red output in the commit message or progress log. The record is what turns "tested" into evidence a later reader can check.
5. **Prefer catalogue queries to lists.** WayClub's isolation tests read `pg_class` and `pg_policy` rather than a hand-written table list, so that a new table without row-level security fails the suite instead of passing unnoticed. The coverage gap at migration 017 (W22) came from a query that still keyed on the wrong column, which is why the catalogue query also needs a positive control.
6. **Treat a sleep, a serial suite or a constant assertion as a smell.** Each hid a real defect here (W12, W21).

6.3 Where the protocol did not reach

The protocol depends on a harness that can express the failure. Where none existed, as for most visual, device and platform defects, the agents either built one (Photoshoot's Direct3D 11 pixel probe) or shipped without a test. Some failures need the real runtime: CareerOS's NextRequest defect appeared only in the Next.js runtime and was verified by a live check. And the evidence itself is self-reported. Every "verified red" in this dataset was written by an agent about its own work; the WayClub practice of independent replay is the only cross-check the record shows.

---

7 Discussion and practical implications

Answers to the research questions. RQ1: business-rule logic and UI state were the most common defects, but authorisation held the most high-severity ones (9 of 13); four recurring shapes (wrong party, split contracts, wrong frames, invisible platform behaviour) cut across products. RQ2: adversarial review found 87 of 183 defects and 24 of 36 high-severity ones, the implementer's checks found most defective tests, and the owner found mostly visual and platform defects. RQ3: 21 defects (11%) sat under a passing check, and 65 of 182 fixes (36%) record red evidence, with the share ranging from 55% for review-found defects to 10% for owner-found ones.

For developers directing agents. A green suite on agent-written code is best read as a hypothesis. In this record the most direct test of that hypothesis was to revert the fix and watch the test fail, and the largest source of high-severity defects was a separate, read-only adversarial review that had to demonstrate its findings. The two found different things, and the owner's own use of the product found a third set, almost all visual or platform defects, that neither reached.

For agent workflows. The strongest records came from sessions that wrote the red output into the commit or progress log. Making that a required field of a fix, alongside the test name, would turn a self-report into something a later agent or person can check. An agent inheriting another agent's code is well advised to start with a read-only review, as the CareerOS and HACKFAM rework sessions did, because the inherited suite is exactly the evidence that may be wrong.

For evaluation and research. The benchmark findings that plausible patches can be wrong are visible here at product scale: in the evaluation set that reported 14 of 14, in the test that encoded the defect, and in tests that could not express the failure. Studies of

agent output could record, per fix, whether the test was ever seen to fail without it; in this dataset, a written record of that observation is what made a fix checkable after the fact.

---

8 Threats to validity

8.1 Construct validity

"Defect" here means what the repository record says was wrong. Severity was coded by one coder against a short rubric, and differs in places from the sources: for example, coding the thirteen CareerOS findings recorded from the inherited-code review yields four high-severity defects where the progress log counts three HIGH and five MEDIUM. Red evidence is taken from commit messages and documents written by the agents themselves; a claim of a revert check was not re-run for this study, and Bupples' 21 revert-verified codes rest largely on one aggregate claim in its completion report. The codes measure what was recorded, which is not the same as what was done.

8.2 Internal validity and selection

The dataset over-represents defects that someone wrote down. Reviews produce documents, so review-found defects are more likely to enter the dataset than defects fixed quietly in a one-line commit; the 48% share for review is partly an artefact of that. Bupples, the largest repository, contributes all of its documented defects but only a 40-commit sample of its undocumented ones. Defects never found are absent by construction, and defects caught before commit by a gate leave no trace. Detection was not recorded for 28 defects. The associations between detection practice and red evidence are confounded with product, defect category and session style, so they do not show that one practice causes better evidence.

8.3 Attribution

Only 21 defects have an identified introducing commit. Attribution elsewhere rests on documents or on the period, and the CareerOS case shows trailers recording the committing agent rather than the writing one. The study therefore makes no claim about which agent produces fewer defects.

8.4 Reliability of coding

One AI coder did all coding, and the reviewers whose findings dominate the dataset were also AI agents, so neither the codes nor the findings are independent human judgements.

The consistency check in Section 4.6 shows that the codebook can be applied reproducibly to short titles, not that a second person would agree.

8.5 External validity

This is one developer's practice over five months, with two agent families, on eight products he directed. The owner's standing instructions (audit inherited work; revert the fix and watch it fail; read the catalogue, not a list) shaped both the defects that were found and the evidence recorded, so the figures describe a workflow that already valued red evidence.

8.6 Conclusion validity

The intervals and tests in Section 5 treat the 183 defects as if they were a sample, which they are not. They show how fragile each proportion is to a few cases, not how agents behave elsewhere.

---

9 Limitations

- **Scope.** One developer, eight products, five months and two agent families. Nothing here estimates defect rates for AI agents in general or compares Claude with Codex.
- **Self-reported evidence.** Every revert or failing-first result was written by an agent about its own work and was not re-run for this study.
- **Single AI coder.** Category, severity and red-evidence codes come from one AI coder; the consistency check is not an inter-rater reliability measure.
- **Incomplete sampling.** Bupples' undocumented defects are represented by a 40-commit sample, and defects never found, or caught before commit, are absent.
- **Private sources.** The repositories are private. The artifact gives hashes, paths and short descriptions, so the owner, or anyone he grants access, can check each row, but readers cannot.
- **No control workflow.** The protocol in Section 6 is distilled from practice and has not been compared with an alternative.

10 Conclusion

Across 183 defects mined from eight agent-built products, the most serious were controls that checked the right thing about the wrong party, contracts split between components that each passed their own tests, and platform behaviour that the development machine never showed. Twenty-one sat under a passing test, gate or written claim. In this record, dedicated adversarial review found most of the high-severity defects (24 of 36) and most often recorded red evidence for its fixes; the owner's use of the product found a different set, mostly visual and platform defects, which seldom ended in a test. The revert-the-fix protocol is a simple discipline, and the record shows both its value (four surviving mutants exposing weak tests; a gate that caught 14 of 22) and its limits (harnesses that did not exist, and red evidence that is still self-reported). These are observations from one developer's practice, not general findings about AI agents; the dataset and scripts are published so that the counts can be checked and the coding repeated.

Artifact

The artifact at /research/green-but-wrong/artifact/ contains the 183-row defect dataset (CSV and JSON) with commit hashes, repository, category, severity, detection method, red-evidence level, attribution tier and source; the commit census for all 1,721 commits; the 531 fix-pattern candidates; the screened Bupples sample; the consistency-check sample; and three scripts (`mine_commits.py`, read-only git mining; `build_dataset.py`, the coded dataset; `analyse.py`, tables, statistics and figures). No source code from the private repositories is included.

References

1. N. Perry, M. Srivastava, D. Kumar and D. Boneh. Do users write more insecure code with AI assistants? In Proceedings of the 2023 ACM SIGSAC Conference on Computer and

Communications Security (CCS '23), pp. 2785–2799, 2023. doi.org/10.1145/3576915.3623157

2. H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt and R. Karri. Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In 2022 IEEE Symposium on Security and Privacy (SP), pp. 754–768, 2022. doi.org/10.1109/SP46214.2022.9833571
3. C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press and K. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In The Twelfth International Conference on Learning Representations (ICLR), 2024. openreview.net/forum?id=VTF8yNQM66
4. Y. Wang, M. Pradel and Z. Liu. Are "solved issues" in SWE-bench really solved correctly? An empirical study. In Proceedings of the IEEE/ACM 48th International Conference on Software Engineering (ICSE '26), pp. 169–181, 2026. doi.org/10.1145/3744916.3764576
5. J. Liu, C. S. Xia, Y. Wang and L. Zhang. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In Advances in Neural Information Processing Systems 36 (NeurIPS), 2023. openreview.net/forum?id=1qvx610Cu7
6. F. Tambon, A. Moradi-Dakhel, A. Nikanjam, F. Khomh, M. C. Desmarais and G. Antoniol. Bugs in large language models generated code: an empirical study. Empirical Software Engineering 30(3), article 65, 2025. doi.org/10.1007/s10664-025-10614-4
7. R. A. DeMillo, R. J. Lipton and F. G. Sayward. Hints on test data selection: help for the practicing programmer. Computer 11(4), pp. 34–41, 1978. doi.org/10.1109/C-M.1978.218136
8. M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon and M. Harman. Mutation testing advances: an analysis and survey. Advances in Computers 112, pp. 275–378, 2019. doi.org/10.1016/bs.adcom.2018.03.015
9. R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes and G. Fraser. Are mutants a valid substitute for real faults in software testing? In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014), pp. 654–665, 2014. doi.org/10.1145/2635868.2635929
10. K. Beck. Test-Driven Development: By Example. Addison-Wesley, 2003.
11. Q. Luo, F. Hariri, L. Eloussi and D. Marinov. An empirical analysis of flaky tests. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014), pp. 643–653, 2014. doi.org/10.1145/2635868.2635920

Cite and reuse

Authorship and AI assistance. Yousof Selim directed the products and the review practice studied here. Much of the code in these repositories was written by AI coding agents (Claude Code, Codex) under his direction and review. The defect coding, the scripts and the draft of this report were produced by Claude under his direction; he reviewed and approved the final text. The dataset and scripts are in the artifact.

```
@techreport{selim2026green,
  author      = {Yousof Selim},
  title       = {Green but Wrong: An Experience Report on Defects in AI-
  institution = {Sunway University (Lancaster University dual degree), M
  type        = {Technical report},
  number      = {v1.0},
  year        = {2026},
  url         = {https://yousofselim.com/research/green-but-wrong/}
}
```