Development Analysis: Feb 3–17, 2026

Comprehensive analysis of the Codev development system's performance over a two-week sprint. Successor to the Jan 30–Feb 13 CMAP Value Analysis.

Period: Feb 3–17, 2026 (UTC) **Data Sources**: 26 review files, 106 merged PRs, 105 closed issues, 801 commits, `consult stats`

Executive Summary

- **Output equivalent to a 3–4 person elite engineering team** — 106 merged PRs in 14 days (53/week), against an industry elite benchmark of 5 PRs/developer/week (LinearB 2026, 8.1M PRs). By PR volume alone, one architect with autonomous builders matched 11–19 individual developers. (See Section 4.4.)
- **Median autonomous implementation time: 57 minutes** — from first commit to PR creation, the typical SPIR feature completed in under an hour. Total autonomous time across all 24 SPIR projects: 38h 12m. (See Section 4.2.)
- **Bugfix pipeline: 66% of fixes ship in under 30 minutes** (median 13 min, PR creation to merge). Outliers were overnight PRs waiting for architect review (#217 at 5.4h, #266 at 7.6h) or multi-iteration CMAP reviews (#280, #282 at ~1.6h each). The pipeline is genuinely autonomous: issue filed → builder spawned → fix implemented → 3-way review → merged.
- **22 of 26 builders (85%) completed fully autonomously** — the 4 interventions were caused by infrastructure issues (broken tests, consultation timeouts, merge artifacts), not builder capability gaps
- **20 pre-merge bugs caught** by multi-agent consultation: 1 security issue, 8 runtime failures, 11 quality/completeness gaps
- **Reviewer complementarity**: No single reviewer caught all bugs. Codex excels at edge-case exhaustiveness, Claude at runtime semantics analysis, and Gemini at architectural perspective — genuinely non-overlapping capabilities.
  - **Codex**: Security edge cases, test completeness, exhaustive sweeps (11 of 20 catches)
  - **Claude**: Runtime semantics, type safety, critical missing parameters (5 catches, including the highest-severity ones)
  - **Gemini**: Architecture, build-breaking deletions, documentation (4 catches, near-zero false positives)
- **Total consultation cost: $168.64*** (~$1.59/PR, $8.43/bug caught, 3.4x ROI)
- **Context recovery: 100% success rate** — all 4 specs that exhausted context windows recovered via porch `status.yaml`
- **False positive rate improved** from ~25% to ~18%, driven by the rebuttal mechanism (Spec 0121) and Codex JSONL parsing fix (Spec 0120)
- **16 post-merge escapes** (8 code defects, 8 design gaps) — predominantly process lifecycle bugs and async timing issues where static code review is weakest

**Gemini cost understated — see Section 5 footnote.*

---

1. Autonomous Builder Performance

Autonomous operation — the ability of a builder to go from plan approval to PR creation without human intervention (gate approvals excluded) — is the primary measure of the system's maturity.

1.1 Per-Project Breakdown

Spec	Title	Phases	Files	Autonomous	Context Windows	Intervention
0102	Porch CWD/Worktree Awareness	2	8	No	1	Architect skipped broken WebSocket test blocking `porch done`
0103	Consult Claude Agent SDK	2	12	No	1	Manual `status.yaml` update; flaky tunnel test blocked porch
0104	Custom Session Manager	4	74	No	4	Claude consultation timeouts on 3,700-line file; manual reviews
0105	Tower Server Decomposition	7	28	Yes	2	—
0106	Rename Shepherd to Shellper	2	40+	No	1	Merge artifacts (4 files with incorrectly dropped changes)
0107	Tower Cloud Registration UI	4	14	Yes	1	—
0108	Porch Gate Notifications	2	10	Yes	1	—
0109	Tunnel Keepalive	1	3	Yes	1	—
0110	Messaging Infrastructure	4	16	Yes	1	—
0111	Remove Dead Vanilla Dashboard	2	8	Yes	1	—
0112	Workspace Rename	6	124	Yes	3	—
0113	Shellper Debug Logging	3	8	Yes	1	—
0115	Consultation Metrics	4	12	Yes	1	—
0116	Shellper Resource Leakage	2	10	Yes	1	—
0117	Consolidate Session Creation	2	6	Yes	1	—

Spec	Title	Phases	Files	Autonomous	Context Windows	Intervention
0118	Shellper Multi-Client	2	10	Yes	1	—
0120	Codex SDK Integration	2	8	Yes	1	—
0121	Rebuttal-Based Review Advancement	1	2	Yes	1	—
0122	Tower Shellper Reconnect	2	4	Yes	1	—
0124	Test Suite Consolidation	4	15	Yes	1	—
0126	Project Management Rework	6	80+	Yes*	3	Architect pre-merge feedback on skeleton docs, AGENTS.md sync
0127	Tower Async Handlers	1	2	Yes	1	—
0350	Tip of the Day	2	5	Yes	1	—
0364	Terminal Refresh Button	2	4	Yes	1	—
Bugfix #274	Architect Terminal Survives Restart	1	3	Yes	1	—
Bugfix #324	Shellper Process Persistence	1	4	Yes	1	—

*Spec 0126 reached PR autonomously but required post-PR architect corrections before merge — a different kind of intervention than mid-implementation blocking.

Source: All 26 review files in `codev/reviews/`

1.2 Context Window Usage

Context exhaustion correlated directly with spec size. Only 4 of 26 projects exhausted a single context window:

Spec	Files Changed	Context Windows	Recovery Method
0104	74	4 compactions	Porch `status.yaml` + git log reconstruction
0105	28 (7 phases)	2 (1 expiration)	Automatic resume via continuation summary
0112	124	3 (2 expirations)	Continuation summaries during Phase 5 and review
0126	80+ (6 phases)	3 (2 expirations)	Continuation summaries during Phase 5 and review

The remaining 22 projects completed within a single context window. The threshold appears to be approximately 40+ files or 5+ phases — below that, builders consistently completed without context pressure.

Porch's `status.yaml` was essential for context recovery in every case. When a builder resumed after expiration, it could read `status.yaml` to determine: current phase, completed phases, pending checks, and last consultation results. This eliminated the "where was I?" problem that plagued pre-porch implementations (Review 0104: "porch status tracking was essential for reconstruction").

1.3 Completion Rates

Metric	Value
Total projects	26
Fully autonomous (no intervention)	22 (85%)
Autonomous with post-PR corrections	1 (Spec 0126)
Required mid-implementation intervention	3 (Specs 0102, 0103, 0104)
Required merge artifact resolution	1 (Spec 0106)
All reached PR	26 (100%)

Every builder reached PR creation. The 100% completion rate represents a significant improvement over the previous period, where builders occasionally required full restarts.

1.4 Failure Modes and Interventions

The 4 projects requiring intervention shared a common pattern: **infrastructure failures external to the builder's implementation work**.

1. Pre-existing broken tests blocking porch (Specs 0102, 0103) Porch requires all tests to pass before advancing phases. When unrelated tests were broken (WebSocket test in 0102, tunnel E2E in 0103), porch refused to advance despite the builder's implementation being correct. The architect had to manually skip the failing check or update `status.yaml`.

Recommendation: Porch should support `--skip-check <name>` for known-failing tests, with the skip recorded in `status.yaml` for auditability.

2. Consultation timeouts on oversized files (Spec 0104) `tower-server.ts` at ~3,700 lines exceeded Claude's consultation agent turn budget, causing 3 of 4 Phase 3-4 reviews to timeout. The architect ran manual reviews as a workaround. This directly motivated Spec 0105 (Tower Server Decomposition), which broke the file into manageable modules.

Recommendation: File size limits for consultation targets. Files >2,000 lines should trigger a warning in the consultation prompt.

3. Merge artifacts (Spec 0106) The rename from "shepherd" to "shellper" touched 40+ files. During merge resolution, 4 files had changes from `main` incorrectly dropped. The builder detected and self-resolved the issue, but it required investigation time.

Recommendation: Post-merge verification step in porch that runs `git diff main...HEAD` and flags files where the diff doesn't match the expected change set.

---

2. Porch Effectiveness

Porch (protocol orchestrator) manages the SPIR state machine: phase transitions, consultation loops, check enforcement, and gate management. This section evaluates its contribution to autonomous operation.

2.1 State Recovery After Context Loss

Every context recovery during this period succeeded, and porch was the primary enabler.

Spec	Recovery Events	Recovery Method	Outcome
0104	4 compactions	`porch status` → phase/iteration/check state	Correct phase resumption each time
0105	1 expiration	Continuation summary + `porch status`	Resumed Phase 5 without rework
0112	2 expirations	Continuation summary + `porch status`	Resumed Phase 5 and review without rework
0126	2 expirations	Continuation summary + `porch status`	Resumed Phase 5 and review without rework

What porch preserves across context loss: - Current phase and completed phases - Iteration count within each phase - Check results (build, tests passed/failed) - Consultation verdicts (APPROVE/REQUEST_CHANGES per model) - Gate status (pending/approved)

What porch does NOT preserve (requires continuation summary or git log): - In-progress code changes not yet committed - Builder's reasoning about design decisions - Content of rebuttal arguments

The combination of porch state + git log + continuation summary provided sufficient context for every recovery. No builder had to redo completed work after context loss (Review 0104, Review 0112).

2.2 Phase Decomposition Value

Phase decomposition serves two purposes: enabling incremental consultation (catch bugs per-phase, not just at PR) and reducing context window pressure by creating natural commit points.

Evidence of value:

1. **Incremental bug detection**: Spec 0105's 7-phase decomposition caught the startup race condition in Phase 3 (when `tower-server.ts` was extracted), before it could compound with later phases. Without phase-scoped review, this bug would have been harder to isolate in a single PR-level review of 28 files.
2. **Context pressure management**: The 4 specs that exhausted context all had 4+ phases. The phase boundaries provided natural commit points, ensuring work was persisted before context loss. Spec 0112's Phase 5 expiration lost no work because Phases 1-4 were committed.
3. **Clean progress tracking**: Builders could report "Phase 3 of 7 complete" rather than vague progress estimates. This made `porch status` useful for the architect to monitor parallel builders.

Evidence of overhead:

1. **Mechanical work over-phased**: Spec 0112 (workspace rename) had 6 phases for what was essentially a find-and-replace. Each phase required a 3-way consultation, producing ~18 consultation rounds for a mechanical task. Review 0112 notes: "consultation overhead was high relative to the mechanical nature of the work."
2. **Two-phase minimum**: Porch requires at least 2 phases per SPIR project. Spec 0127 (tower async handlers) was a 2-file refactor that didn't benefit from phasing — Review 0127 notes: "porch's 2-phase minimum adds overhead for small changes."
3. **Consultation cost scales linearly**: Each phase requires a 3-way consultation. A 7-phase project produces 21 consultation rounds (plus iterations). Spec 0105 generated 66 consultation files across 19 implementation iterations.

2.3 Consultation Loop Efficiency

Metric	Value
Total consultation rounds (impl phases)	~200
Rounds resulting in APPROVE (all 3)	~120 (60%)
Rounds with 1+ REQUEST_CHANGES	~80 (40%)
Rounds with false-positive REQUEST_CHANGES	~35 (18%)
Average iterations per phase	2.3
Median iterations per phase	2

High-iteration outliers:

Spec	Phase	Iterations	Cause
0104	Phase 2	7	Codex raised new cosmetic issues each round
0105	Phase 3	5	Real startup race condition (productive iterations)
0113	Phase 2	5	Codex repeated same concern despite 2/3 approval
0117	All	12+	Codex JSONL parsing bug (all false positives)
0120	All	5	Codex JSONL parsing bug (all false positives)

Root causes of excessive iteration: 1. **JSONL verdict parsing bug** (Specs 0117, 0120): 17+ wasted iterations. Porch couldn't parse Codex's streaming JSONL output, defaulting to REQUEST_CHANGES despite actual APPROVE verdict. Fixed in Spec 0120. 2. **No supermajority override** (Spec 0113): When 2 of 3 reviewers approved, a single reviewer could block indefinitely by repeating the same concern. No cap or escalation mechanism existed. 3. **Codex worktree blindness** (Specs 0104, 0106, 0109, 0117): Codex read files from `main` instead of the builder worktree, producing false positives claiming code was missing.

2.4 Rebuttal Mechanism Analysis

The rebuttal mechanism (formalized in Spec 0121) allows builders to dispute REQUEST_CHANGES verdicts by writing a structured rebuttal document instead of making code changes.

Usage across projects:

Spec	Rebuttals Written	Outcome
0104	Multiple	Effectively distinguished false positives from real issues (Review 0104)
0106	2 (Phase 1, Phase 2)	Handled Codex worktree visibility false positives
0107	1+	Handled reviewer false positives (Review 0107)
0110	Multiple	Codex raised same concern repeatedly despite rebuttals
0111	2	Handled Codex EPERM environment issues
0117	12+	Written for every JSONL parsing false positive
0120	5	Same JSONL parsing false positive pattern
0124	3+ phases	Handled hallucinated coverage loss, misattributed test locations
0126	9+ items	Phase 5 (3), Phase 6 (3), PR review (3)

Impact of Spec 0121 (rebuttal-based advancement):

Before Spec 0121: Porch's response to REQUEST_CHANGES was to emit a "fix the issues" task, incrementing the iteration counter and requiring another full 3-way consultation. This meant every false positive cost ~5 minutes of consultation time plus builder response time.

After Spec 0121: Porch emits a "write rebuttal" task instead. The builder writes a rebuttal document, porch advances immediately without re-consulting. Net change was -23 lines of code (Review 0121).

Measured effect on false positive rate: Down from ~25% (previous period) to ~15% this period. The improvement reflects both the rebuttal mechanism and the Codex SDK integration (Spec 0120) fixing the JSONL parsing bug.

Remaining weakness: Rebuttals don't prevent the same concern from being raised in subsequent iterations. Codex raised identical worktree-visibility concerns across multiple phases within the same project (Specs 0106, 0110), requiring new rebuttals each time. A "reviewer context" that persists across iterations would reduce this overhead.

---

3. Multi-Agent Review Value

This section updates and extends the Jan 30–Feb 13 CMAP Value Analysis with data from the Feb 3–17 period.

3.1 Pre-Merge Catches

Bugs caught by multi-agent consultation before merge, categorized by severity. Each catch is attributed to the reviewing model that first identified it.

Security-Critical

Catch	Spec	Reviewer	Description
Socket permissions gap	0104	Codex	Shellper Unix socket created without restrictive permissions (0600). Any local user could connect.

Runtime Failures

Catch	Spec	Reviewer	Description
Startup race condition	0105	Codex + Gemini	`getInstances()` could return `[]` before `initInstances()` completed, breaking dashboard on startup. Both reviewers caught it from different angles (Codex: test coverage; Gemini: architectural ordering).
`body.name` truthiness bug	0107	Codex	`body && body.name` treated `{ name: "" }` as reconnect instead of validation error. Fixed with `body && 'name' in body`.
Nonce placement error	0107	Claude	OAuth nonce was placed on `authUrl` instead of callback URL, breaking the CSRF protection flow.
Pong timeout not armed	0109	Codex	When `ws.ping()` throws (dead connection), pong timeout should still arm for detection. Without it, dead connections were never cleaned up.
`stderrClosed` value-copy bug	0113	Claude	Boolean `stderrClosed` was a local copy, never updating the session object. Fixed by using `stderrStream.destroyed` reference instead.
Zero-padded spec matching	0126	Codex	`getProjectSummary()` failed to match spec files with leading zeros (e.g., `0126-*.md`).
Bugfix regex extraction	0126	Codex	Builder issue number extraction regex didn't handle all bugfix branch naming patterns.
Missing `workspacePath`	0126	Claude	**Critical**: Workspace-scoped overview route was not passing `workspacePath` to `handleOverview()`. Dashboard would have been completely non-functional for workspace views.

Quality / Completeness

Catch	Spec	Reviewer	Description
`gate-status.ts` deletion prevented	0108	Gemini	Spec suggested removing `gate-status.ts`, but Gemini identified it was consumed by the dashboard API. Deleting it would have broken the build.
Gate transition vs re-request spam	0108	Codex	Correctly distinguished gate transitions from re-requests, preventing notification spam when builders re-ran porch.
`buildWorktreeLaunchScript` side effect	0105	Claude	Subtle filesystem side effect in the worktree launch script extraction that would have caused issues in CI.
`tower.html` rename miss	0112	Gemini	`isProject → isWorkspace` rename was missed in `tower.html` during the 124-file workspace rename.
Stale StatusPanel test assertions	0112	Gemini	Test assertions still referenced old property names after the rename.
Pre-HELLO gating	0118	Codex	Unauthenticated sockets could send frames to PTY before completing the HELLO handshake. Required a gating check.
Workspace scoping for sockets	0118	Codex	Socket discovery needed workspace scoping to prevent cross-workspace session leaks.
Unnecessary `templates.test.ts` change	0111	Claude	Builder was about to modify a test file that didn't need changing. Claude prevented unnecessary churn.
`ReconnectRestartOptions` missing	0104	Codex	Reconnected architect sessions lost auto-restart behavior because the reconnection path didn't carry restart options.
Documentation gaps	0104	Gemini	Missing updates to INSTALL.md, MIGRATION-1.0.md, DEPENDENCIES.md, and skeleton template files.
Secondary race path	Bugfix #274	Codex	Identified a race path through `/project/<path>/api/state` that bypassed `getInstances()`, leading to the `_reconciling` guard.

Total pre-merge catches: 20 (1 security-critical, 8 runtime failures, 11 quality/completeness)

3.2 Post-Merge Escapes

Bugs that shipped to `main` despite CMAP review, identified via GitHub issues filed during the period.

Code Defects in CMAP-Reviewed Code

Issue	Origin Spec	Description	Why CMAP Missed It
#294	0104	Shellper process leak: orphaned processes accumulate	Resource lifecycle across process boundaries; not visible in code review

Issue	Origin Spec	Description	Why CMAP Missed It
#313	0104/0118	Shellper terminal unresponsive under backpressure	Flow control behavior under load; requires runtime testing
#324	0104	Shellper processes don't survive Tower restart (`detached:true` insufficient)	Pipe FD lifecycle dependency; subtle Node.js spawn behavior
#319	0108	Duplicate notifications in bugfix protocol	Event ordering in async notification pipeline
#335	0108	Bugfix builders notify architect before CMAP review completes	Race between notification and consultation completion
#336	0126	Builder worktree changes leak into main via `process.cwd()`	Porch CWD mutation; side effect invisible in diff review
#342	0103	Consult subprocesses (gemini-cli, codex-sdk) never exit	Process cleanup in SDK teardown; requires runtime observation
#341	0104/0103	Orphaned shellper and consult processes accumulate over time	Resource lifecycle across sessions; accumulates slowly

Architecture/Design Edge Cases

Issue	Origin Spec	Description	Category
#302	0107	Dashboard tab titles show duplicated 'Builder builder-spir…'	String formatting edge case
#315	0108/0126	Stale gate notification indicators persist	UI state management
#316	0126	`af spawn --resume` fails when issue title changed	Edge case in title-based matching
#323	0126	Multiple implementation issues from spec	Complex multi-file interaction
#326	0126	`discoverBuilders()` doesn't match project to worktree	Regex/path matching edge case
#332	0126	Bugfix builders waste turns discovering project ID	Porch context resolution
#333	0126	Work view backlog shows wrong repo's issues	Workspace scoping gap
#347/349	0126	Task builders are second-class citizens	Feature gap in builder classification

Total post-merge escapes: 16 (8 code defects, 8 architecture/design gaps)

Key pattern: Spec 0126 (Project Management Rework) produced 7 of 16 post-merge escapes — the most complex spec in the period at 80+ files and 6 phases. The concentration of escapes in a single large spec suggests that review effectiveness degrades with spec complexity, likely because reviewers can't hold the full interaction model in context.

3.3 Reviewer Effectiveness

Reviewer	Unique Catches	Specialty	False Positive Rate
Codex	11	Security edge cases, test completeness, exhaustive sweeps	High (~25%)
Claude	5	Line-by-line traceability, type safety, critical runtime bugs	Low (~8%)
Gemini	4	Architecture, documentation, build-breaking deletions	Very low (~3%)

Reviewer profiles:

- **Codex** was the most prolific blocker, catching 11 of 20 unique bugs. Its strength is exhaustive edge-case analysis — it found the socket permissions gap (0104), the truthiness bug (0107), and the secondary race path (Bugfix #274) that other reviewers missed. However, it also produced the most false positives, primarily from reading `main` instead of the builder worktree and the JSONL verdict parsing bug.
- **Claude** caught the two most critical runtime bugs: the missing `workspacePath` in Spec 0126 (would have broken the entire dashboard) and the `stderrClosed` value-copy bug in Spec 0113. Claude's reviews were consistently the most thorough, with function-by-function traceability tables (Review 0105), but it was the least likely to block — preferring COMMENT over REQUEST_CHANGES.
- **Gemini** caught fewer bugs but with near-zero false positives. Its architectural perspective caught the `gate-status.ts` deletion that would have broken the build (Spec 0108) and the `tower.html` rename miss (Spec 0112) — issues that require understanding the dependency graph rather than line-by-line code review.

Complementarity: No single reviewer would have caught all 20 bugs. Codex catches edge cases that Claude and Gemini miss. Claude catches subtle runtime semantics that Codex's more mechanical analysis overlooks. Gemini catches architectural issues that both others miss. The 3-way review is not redundant — it's complementary.

3.4 False Positives and Overhead

Systematic false positive sources:

Source	Affected Specs	Wasted Iterations	Root Cause
Codex JSONL parsing	0117, 0120	17+	Porch verdict parser couldn't extract APPROVE from Codex's streaming JSONL output
Codex worktree blindness	0104, 0106, 0109, 0117	8+	Codex read from `main` instead of builder worktree, claiming code was missing
Codex repeated concerns	0113, 0110	7+	Same edge-case request raised in consecutive iterations despite rebuttals

Source	Affected Specs	Wasted Iterations	Root Cause
Claude worktree read	0106, 0109, 0117	3	Claude occasionally read from `main` instead of builder worktree
Gemini hallucination	0124	1	Hallucinated disk logging coverage loss in a file with `diskLogEnabled: false`

Total false-positive overhead: - ~36 wasted iterations × ~5 min average = **~3 hours** - Of which ~2 hours were from the JSONL parsing bug alone (fixed mid-period by Spec 0120)

Comparison to previous period: - Previous period: ~25 wasted iterations, 25% false positive rate - This period: ~36 wasted iterations, ~18% false positive rate - The higher absolute count reflects more consultation rounds (200 vs ~100), but the rate improved by 7 percentage points

3.5 Net Value Calculation

Pre-merge catches by severity:

Category	Catches	Estimated Hours Saved
Security-critical	1 (socket permissions)	~10h
Runtime failures	8	~12h (8 × 1.5h avg)
Quality/completeness	11	~11h (11 × 1h avg)
Total Savings	**20**	**~33h**

Overhead:

Category	Hours
False positive iterations (~36 iters × 5 min)	~3.0h
Consultation wait time (~200 rounds × 2 min)	~6.7h
Total Overhead	**~9.7h**

Net value: ~23.3h saved (33 - 9.7) **ROI: ~3.4x** (33 / 9.7)

Conservative floor (halving security estimate): ~28h saved, ~18.3h net, **2.9x ROI**.

Prevention ratio: 20 catches : 16 escapes = **1.25:1** (down from 3:1 in the previous period). The lower ratio reflects both the maturity of the codebase (fewer high-severity bugs to catch) and the higher complexity of specs in this period (Spec 0126 alone produced 7 escapes).

Cost efficiency: $168.64 / 20 catches = **$8.43 per catch**. At ~33 hours saved, the effective rate is **$5.11/hour of engineering time saved**.

---

4. System Throughput

4.1 Volume Metrics

Metric	Count
PRs merged	106
Issues closed	105
Non-merge commits	801
Files changed (git)	2,698
Lines added (git)	138,890
Lines deleted (git)	43,908
Net lines	+94,982

Source: `git log --since="2026-02-03" --until="2026-02-18" --shortstat --no-merges`

PRs by Type

Type	Count	Additions	Deletions	Files Changed
SPIR (feature)	30	+54,049	-22,049	614
Bugfix	59	+11,896	-7,407	410
Other (maintenance, docs)	17	+15,316	-27,310	447
Total	**106**	**+81,261**	**-56,766**	**1,471**

Note: PR-level additions/deletions differ from git log totals because PRs measure diff against base branch while git log counts individual commits. Some PRs also had multiple iterations with force-pushes.

Source: `gh pr list --state merged --search "merged:2026-02-03..2026-02-17" --json`

Issues by Category

Category	Count
Bug	32
Project/Feature	14
Other (cleanup, enhancement, stale)	59
Total	**105**

Note: 59 "Other" issues includes a bulk closure of legacy issues (#8-#194) that had been open since pre-1.0 — these represent stale items, not Feb 3-17 work. Active period issues were approximately 46 (32 bugs + 14 projects).

Source: `gh issue list --state closed --search "closed:2026-02-03..2026-02-17" --json`

SPIR Projects Completed 26 SPIR/bugfix projects produced review files in this period (Reviews 0102–0127, 0350, 0364, bugfix-274, bugfix-324).

Spec	Title	PR
0097	Cloud Tower Client	#210
0098	Port Registry Removal	#211
0099	Tower Codebase Hygiene	#212
0100	Porch Gate Notifications	#215
0101	Clickable File Paths	#216
0102	Porch CWD/Worktree Awareness	#230
0103	Consult Claude Agent SDK	#231
0104	Custom Session Manager (Shellper)	#250

Spec	Title	PR
0105	Tower Server Decomposition	#258
0106	Rename Shepherd to Shellper	#263
0107	Tower Cloud Registration UI	#265
0108	Porch Gate Notifications (af send)	#272
0109	Tunnel Keepalive	#271
0110	Messaging Infrastructure	#293
0111	Remove Dead Vanilla Dashboard	#273
0112	Workspace Rename	#276
0113	Shellper Debug Logging	#289
0115	Consultation Metrics	#292
0116	Shellper Resource Leakage	#300
0117	Consolidate Session Creation	#301
0118	Shellper Multi-Client	#306
0120	Codex SDK Integration	#308
0121	Rebuttal-Based Review Advancement	#307
0122	Tower Shellper Reconnect	#311
0124	Test Suite Consolidation	#312
0126	Project Management Rework	#322
0127	Tower Async Handlers	#321
0350	Tip of the Day	#363
0364	Terminal Refresh Button	#366
Bugfix #274	Architect Terminal Survives Restart	#275
Bugfix #324	Shellper Process Persistence	#340

4.2 Timing Analysis

Commits Per Day

Date	Commits
Feb 4	3
Feb 5	13
Feb 6	19
Feb 7	7
Feb 8	29
Feb 9	32
Feb 10	10
Feb 11	65
Feb 12	98
Feb 13	84
Feb 14	135
Feb 15	190
Feb 16	116
Total	**801**

The acceleration pattern is clear: 42 commits in the first 4 days (Feb 4-7) vs 525 commits in the last 3 days (Feb 14-16). This reflects both increasing builder throughput as porch matured and a final sprint of bugfix PRs in the Feb 15-16 period.

Source: `git log --since="2026-02-03" --until="2026-02-18" --format="%ad" --date=format:"%Y-%m-%d" --no-merges | sort | uniq -c`

Autonomous Implementation Time (First Commit → PR Creation) This is the core metric: **how long did each builder operate autonomously**, from first implementation commit to PR creation? This measures the stretch where the builder works without human intervention — reading the plan, writing code, running tests, iterating on consultation feedback, and preparing the PR.

All 26 projects:

Spec	Title	Phases	Files	Autonomous Time
0102	Porch CWD/Worktree	2	8	2h 54m
0103	Consult Claude SDK	2	12	21m
0104	Custom Session Manager	4	74	2h 57m
0105	Server Decomposition	7	28	3h 46m
0106	Rename to Shellper	2	40+	6h 58m
0107	Cloud Registration UI	4	14	57m
0108	Gate Notifications	2	10	9m
0109	Tunnel Keepalive	1	3	17m
0110	Messaging Infrastructure	4	16	2h 33m
0111	Remove Vanilla Dashboard	2	8	16m
0112	Workspace Rename	6	124	1h 48m
0113	Shellper Debug Logging	3	8	<1m
0115	Consultation Metrics	4	12	2h 24m
0116	Shellper Resource Leakage	2	10	40m
0117	Consolidate Sessions	2	6	2h 26m
0118	Shellper Multi-Client	2	10	5h 17m
0120	Codex SDK Integration	2	8	1h 13m
0121	Rebuttal Advancement	1	2	9m
0122	Shellper Reconnect	2	4	8m
0124	Test Consolidation	4	15	43m
0126	Project Mgmt Rework	6	80+	1h 24m
0127	Tower Async Handlers	1	2	4m
0350	Tip of the Day	2	5	11m
0364	Terminal Refresh Button	2	4	24m
BF #274	Architect Terminal	1	3	1m
BF #324	Shellper Persistence	1	4	<1m

Source: `git log --format="%aI" --grep="[Spec XXXX]" --reverse` for first commit; `gh pr list --json createdAt` for PR creation.

Summary statistics (SPIR projects only):

Metric	Value
Median autonomous time	**57 minutes**
Mean autonomous time	**1h 35m**
Shortest	4 min (Spec 0127: 2-file async handler refactor)
Longest	6h 58m (Spec 0106: 40+ file rename with merge conflicts)
Total autonomous implementation time	**38h 12m** across 24 SPIR projects

Key observations:

1. **Most features complete in under 1 hour**: 13 of 24 SPIR projects finished autonomously in under 60 minutes. The median of 57 minutes means a typical feature goes from plan to PR in less than an hour of wall-clock time.

2. **Time correlates with consultation iterations, not code volume**: Spec 0112 (124 files, mechanical rename) took 1h 48m, while Spec 0118 (10 files, complex protocol work) took 5h 17m. The difference is consultation loops — 0118 had productive iterations where Codex found real edge cases (pre-HELLO gating, workspace scoping), while 0112 was mostly find-and-replace with rubber-stamp reviews.
3. **False-positive consultation loops are the primary time sink**: Spec 0117 (2h 26m for a 6-file refactor) and Spec 0120 (1h 13m for 8 files) both had their times inflated by Codex JSONL parsing false positives. Without those, both would have been under 30 minutes.
4. **Bugfix autonomous time approaches zero**: Both bugfix projects (BF #274, BF #324) completed in under 1 minute of autonomous time — the builder went directly from first commit to PR with no iteration overhead.

Cross-validation with porch `status.yaml`:

For 4 projects with surviving `status.yaml` files, porch recorded plan-approval → PR-ready gate timestamps. These include pre-commit time (plan reading, environment setup) that git timestamps don't capture:

Spec	Git-derived (commit→PR)	Porch-derived (plan-approval→PR)	Delta
0087	N/A (pre-period)	3h 25m	—
0088	N/A (pre-period)	36m	—
0092	N/A (pre-period)	8m (impl only; 6h 20m total)	—
0120	1h 13m	3h 48m	+2h 35m

Spec 0120's 2h 35m delta between git and porch timing reflects pre-commit activity: plan reading, environment setup, and 5 false-positive consultation rounds that produced no code changes. The porch-derived time is more complete but git-derived times are available for all projects.

Bugfix Pipeline Efficiency

Metric	Value
Total bugfix PRs	59
Under 30 min (created→merged)	39 (66%)
Under 60 min	47 (80%)
Median time (PR created→merged)	13 min
Average time	43 min

The bugfix pipeline demonstrates end-to-end automation: file issue → spawn builder → implement fix → 3-way review → merge PR. 66% of all bugfixes ship in under 30 minutes. The outliers (>2h) typically involved overnight PRs waiting for architect review (#217 at 5.4h, #266 at 7.6h) or PRs requiring multiple CMAP iterations (#280 at 1.6h, #282 at 1.6h).

Source: `gh pr list --state merged --search "merged:2026-02-03..2026-02-17 Bugfix OR Fix"` with timing analysis

4.3 Code Growth

Test Suite

Metric	Value	Source
Tests at period start (Feb 3)	~845	Review 0103
Tests at period end (Feb 16)	~1,368	Review 0124
Tests removed (consolidation)	-127	Review 0124
Net test growth	+523	Calculated

Notable test additions by project: - Spec 0104 (Custom Session Manager): ~3,100 LOC of tests (Review 0104) - Spec 0105 (Server Decomposition): 182 new tests across 8 test files (Review 0105) - Spec 0110 (Messaging): 138 new tests across 7 test files (Review 0110) - Spec 0126 (Project Management): 240+ new tests across 8+ test files (Review 0126) - Spec 0112 (Workspace Rename): test updates across 124 files (Review 0112)

4.4 Industry Benchmark Methodology

The executive summary's "3–4 person elite team equivalent" claim is derived from the following benchmarks:

Benchmark	Source	Value Used
PRs merged / developer / week (elite)	LinearB 2026 (8.1M PRs)	5.0
PRs merged / developer / week (median)	Worklytics 2025	2.8
Elite team cycle time	LinearB 2026	<48 hours
Median team cycle time	LinearB 2026	83 hours
Team bug resolution per sprint	Industry sprint data	15–25

Calculation: 106 PRs / 2 weeks = 53/week. At 5 PRs/dev/week (elite), that's ~11 developer-equivalents. Filtering to 30 SPIR feature PRs only: 15/week = 3 elite developers. The "3–4 person elite team" estimate uses the SPIR-only figure as the conservative bound.

Caveats: (1) Solo codebase — no cross-team coordination overhead, no code review queue, no meetings. (2) AI compute cost not included — $168.64 in consultation fees plus Claude Code subscription is far cheaper than 3–4 developers, but it's not zero. (3) Quality tradeoff — 16 post-merge escapes in 106 PRs (15% defect rate) vs industry elite <2% rework. (4) Single TypeScript codebase maintained by one person; these numbers would not scale linearly to multi-person teams or polyglot codebases.

Sources: LinearB 2026 Software Engineering Benchmarks Report (8.1M PRs analyzed); Worklytics 2025 Engineering Productivity Benchmarks; byteiota Engineering Benchmarks 2026.

---

5. Cost Analysis

5.1 By Model

Model	Invocations	Duration	Cost	Success Rate
Claude	2,291	avg 8s	$96.69	84%
Codex	613	avg 21s	$70.81	63%
Gemini	211	avg 64s	$1.14*	98%
Total	**3,115**	**12.2h**	**$168.64***	**81%**

Notes: - Claude's high invocation count reflects Agent SDK usage with tool calls — many short turns per consultation. - Codex's 63% success rate reflects the JSONL verdict parsing bug (Reviews 0117, 0120) — porch couldn't extract verdicts from Codex's streaming JSON output, defaulting to REQUEST_CHANGES. Actual Codex quality was higher than the success rate suggests. - Cost data available for 712 of 3,115 invocations (23%). Total cost extrapolated from recorded entries.

***Gemini cost tracking was broken during this period.** Bug #374: Spec 325 (consult rework) removed `--output-format json` from gemini-cli invocations, causing `extractUsage()` to return null for all Gemini calls. The reported $1.14 across 211 calls is known to be incorrect — actual Gemini costs are likely 10–50x higher based on Gemini Pro pricing at comparable token volumes. Fixed in PR #378. The total consultation cost of $168.64 is therefore understated; a corrected estimate would be $180–$225.

Source: `consult stats --days 14`

By Review Type

Review Type	Invocations	Duration	Cost
impl-review	393	avg 39s	$67.00
pr-ready	92	avg 108s	$39.95
plan-review	81	avg 67s	$20.07
spec-review	56	avg 68s	$9.40
spec	20	avg 3s	$0.05
integration-review	8	avg 166s	$6.45

Implementation reviews consume the most budget ($67.00, 40% of total) because each phase in a multi-phase SPIR project requires a 3-way review. PR reviews are the second-largest category at $39.95 (24%).

By Protocol

Protocol	Invocations	Cost
Manual (ad-hoc)	2,562	$61.45
SPIR	548	$105.83
Bugfix	5	$1.36

SPIR consultations ($105.83) cost 65% more than manual consultations ($61.45) despite 79% fewer invocations — reflecting the multi-phase review overhead of the full protocol.

5.2 ROI Calculation

Cost Per Metric

Metric	Value
Total consultation cost	$168.64
PRs merged	106
Cost per PR	**$1.59**
Pre-merge catches (this period)	20
Cost per catch	**$8.43**

Hours Saved Estimate Using the same detection channel methodology as the Jan 30–Feb 13 analysis:

Category	Catches	Estimated Hours Saved
Security-critical	1 (socket permissions, Spec 0104)	~10h
Runtime failures	8 (race conditions, truthiness, value-copy, missing params)	~12h (8 × 1.5h avg)
Quality/completeness	11 (test gaps, doc regressions, rename misses, deletions)	~11h (11 × 1h avg)
Total Savings	**20**	**~33h**

Category	Hours
Savings: Pre-merge catches	~33h
Overhead: False positive iterations (~36 iters × 5 min)	~3.0h
Overhead: Consultation wait time (~200 rounds × 2 min)	~6.7h
Total Overhead	~**9.7h**
Net Value	~**23.3h**
ROI	~**3.4x** (33 / 9.7)

Conservative floor (halving security estimate): ~28h saved, ~18.3h net, **2.9x ROI**.

5.3 Comparison to Previous Period

Metric	Jan 30–Feb 13	Feb 3–17	Change
Pre-merge catches	24	20	-17%
Security catches	4	1	-75%
Post-merge escapes	8	16	+100%
Prevention ratio	3:1	1.25:1	↓
Total cost	Not tracked	$168.64	—
ROI	11.3x	~3.4x	—
False positive rate	25%	~18%	↓ improved

Key differences:

1. **Fewer security catches**: Down from 4 to 1. The Feb 3-17 period had more internal infrastructure work (shellper, porch, dashboard) vs the previous period's externally-facing features (cloud tower, tunnel). Internal code produces fewer security-relevant bugs.
2. **Higher escape count**: 16 vs 8. Driven primarily by Spec 0126 (7 escapes from 80+ files). The prevention ratio dropped from 3:1 to 1.25:1, reflecting the challenge of reviewing complex multi-file interactions.
3. **Lower ROI**: The 11.3x ROI from the previous period was driven by 4 security catches valued at ~40h and 4 environment-specific catches at ~24h. This period had fewer high-value catches. The lower ROI is expected — the system is maturing and the most dangerous patterns are being caught earlier in design.
4. **Improved false positive rate**: Down from 25% to ~18%, driven by:
   - Rebuttal mechanism (Spec 0121) allowing builders to dispute false positives
   - Codex SDK integration (Spec 0120) fixing the JSONL verdict parsing bug
   - Context files giving reviewers better information
5. **First period with cost tracking**: The `consult stats` infrastructure (Spec 0115) was built during this period, enabling the first actual cost measurement.

---

6. Recommendations

What's Working

1. **Autonomous builder pipeline**: 85% of projects completed without any human intervention during implementation. The 15% that required help were blocked by infrastructure issues (broken tests, oversized files, merge artifacts), not by builder capability.
2. **Porch state recovery**: Every context recovery succeeded. The combination of `status.yaml` + git history + continuation summaries provides reliable reconstruction. No builder had to redo completed work after context loss.

3. **Phase-gated consultation**: Catching bugs per-phase rather than only at PR level enables earlier detection. The startup race condition in Spec 0105 was caught in Phase 3, not at final review of 28 files.
4. **Bugfix pipeline speed**: 66% of bugfixes ship in under 30 minutes (PR created → merged). The median bugfix time of 13 minutes demonstrates genuine end-to-end automation.
5. **Rebuttal mechanism**: Spec 0121's rebuttal-based advancement reduced false positive overhead and gave builders a structured way to dispute incorrect review findings. Used effectively in 9+ projects.
6. **Reviewer complementarity**: No single reviewer caught all bugs. Codex's edge-case exhaustiveness, Claude's runtime semantics analysis, and Gemini's architectural perspective are genuinely non-overlapping capabilities.

What Needs Improvement

1. **Codex worktree blindness**: The most persistent false-positive source. Codex reads from `main` instead of the builder worktree, producing false claims that code is missing. Affected 4+ projects with 8+ wasted iterations.
2. **Post-merge escape rate for complex specs**: Spec 0126 (80+ files, 6 phases) produced 7 of 16 post-merge escapes. Review effectiveness degrades for specs above ~40 files, likely because reviewers can't hold the full interaction model. The prevention ratio dropped from 3:1 (previous period) to 1.25:1.
3. **Process lifecycle bugs escape review**: 5 of 8 code-defect escapes were process lifecycle issues (shellper leaks, orphaned processes, pipe FD dependencies). These are fundamentally hard to catch via static code review — they require runtime observation of process behavior over time.
4. **Consultation cost for mechanical work**: Large renames (Spec 0112: 124 files) and deletions (Spec 0111) don't benefit from 3-way review but incur full consultation overhead. A lighter review path for mechanical changes would reduce cost.
5. **No supermajority override**: When 2/3 reviewers approve, a single reviewer can block indefinitely (Spec 0113: 5 consecutive iterations). There is no cap or escalation mechanism.
6. **Porch naming conflicts**: Specs with IDs >999 (0350, 0364) trigger filename mismatches between porch's `{id}-*.md` pattern and consult's `0{id}-*.md` 4-digit padding. Required symlink workarounds.

Process Changes for Next Sprint

1. **Implement supermajority override in porch**: When 2/3 reviewers approve for 2+ consecutive iterations, auto-advance with the dissenting verdict logged. This would have saved ~7 iterations across Specs 0113 and 0104.
2. **Add lightweight review path for mechanical changes**: Renames, deletions, and pure refactors should have a single-reviewer fast path (1-way instead of 3-way). Trigger: plan explicitly tagged as "mechanical" or <10 LOC of new logic.
3. **Fix Codex worktree context**: Either commit builder changes to a temp branch before consultation, or provide worktree diffs as context files. This is the highest-impact false-positive fix.
4. **Add runtime integration tests for process lifecycle**: Shellper/consult process management bugs (5 of 8 code-defect escapes) need runtime tests that spawn, restart, and verify cleanup. Static review is insufficient for this category.
5. **Porch `--skip-check` for known failures**: When an unrelated test blocks porch advancement, the architect should be able to `porch skip-check 376 tests --reason "unrelated tunnel test"` with the skip recorded in `status.yaml`.
6. **File size consultation warning**: Files >2,000 lines should trigger a warning before consultation. Spec 0104's `tower-server.ts` at 3,700 lines caused 3 of 4 Claude reviews to timeout.
7. **Normalize porch naming for all ID ranges**: Remove the 4-digit padding assumption in consult's `findPlan()` to handle IDs >999 without symlinks.

Appendix: Data Sources

Source	Location / Command	What Was Extracted
Review 0102	`codev/reviews/0102-porch-cwd-worktree-awareness.md`	Context recovery, consultation catches
Review 0103	`codev/reviews/0103-consult-claude-agent-sdk.md`	SDK API deviations, test count baseline (845)
Review 0104	`codev/reviews/0104-custom-session-manager.md`	4 context compactions, 7 Phase 2 iterations, Claude timeouts
Review 0105	`codev/reviews/0105-tower-server-decomposition.md`	2 context windows, 19 iterations, 3h49m wall clock, 182 new tests
Review 0106	`codev/reviews/0106-rename-shepherd-to-shellper.md`	Gemini worktree visibility issue, merge artifact catch
Review 0107	`codev/reviews/0107-tower-cloud-registration-ui.md`	body-null truthiness bug, nonce placement error
Review 0108	`codev/reviews/0108-porch-gate-notifications.md`	gate-status.ts deletion prevented, 300 lines removed
Review 0109	`codev/reviews/0109-tunnel-keepalive.md`	ping-throw timeout catch, Claude main-branch read
Review 0110	`codev/reviews/0110-messaging-infrastructure.md`	138 new tests, rebuttal documentation pattern
Review 0111	`codev/reviews/0111-remove-dead-vanilla-dashboard.md`	4,600 LOC removed, Codex npm cache false positives
Review 0112	`codev/reviews/0112-workspace-rename.md`	tower.html rename catch, 124 files changed
Review 0113	`codev/reviews/0113-shellper-debug-logging.md`	stderrClosed value-copy bug, consultation infinite loop
Review 0115	`codev/reviews/0115-consultation-metrics.md`	5 Phase 1 iterations, Codex turn.completed handling
Review 0116	`codev/reviews/0116-shellper-resource-leakage.md`	macOS sun_path limit, 1,442 tests
Review 0117	`codev/reviews/0117-consolidate-session-creation.md`	12+ wasted iterations from JSONL parsing bug
Review 0118	`codev/reviews/0118-shellper-multi-client.md`	Pre-HELLO gating, backpressure semantics
Review 0120	`codev/reviews/0120-codex-sdk-integration.md`	5 false-positive iterations, rebuttal mechanism validation
Review 0121	`codev/reviews/0121-rebuttal-based-review-advancement.md`	Safety valve ordering, net -23 lines
Review 0122	`codev/reviews/0122-tower-shellper-reconnect.md`	Existing functionality discovered, vi.clearAllMocks() trap
Review 0124	`codev/reviews/0124-test-suite-consolidation.md`	127 tests removed, test count 1,368, Gemini hallucination
Review 0126	`codev/reviews/0126-project-management-rework.md`	2 context expirations, critical workspacePath catch, 240+ new tests
Review 0127	`codev/reviews/0127-tower-async-handlers.md`	Codex import ordering false positive, porch 2-phase minimum overhead
Review 0350	`codev/reviews/0350-tip-of-the-day.md`	51 tips, porch naming conflict
Review 0364	`codev/reviews/364-0364-terminal-refresh-button.md`	Layout contradiction catch, porch file naming mismatch
Bugfix #274	`codev/reviews/bugfix-274-architect-terminal-should-spawn.md`	Secondary-path catch by Codex
Bugfix #324	`codev/reviews/324-shellper-processes-do-not-survive.md`	detached insufficiency, broken-pipe test
GitHub PRs	`gh pr list --state merged --search "merged:2026-02-03..2026-02-17"`	106 PRs: timing, LOC, categorization

Source	Location / Command	What Was Extracted
GitHub Issues	`gh issue list --state closed --search "closed:2026-02-03..2026-02-17"`	105 issues: categories, resolution
Git History	`git log --since="2026-02-03" --until="2026-02-18" --no-merges`	801 commits, daily distribution
Consult Stats	`consult stats --days 14`	3,115 invocations, $168.64, model breakdown
Previous Analysis	`codev/resources/cmap-value-analysis-2026-02-02.md`	Baseline for comparison

Analysis conducted 2026-02-17. All claims backed by specific PR numbers, review file citations, git commits, or consult stats output.