TECHNICAL WHITE PAPER / FIELD NOTES 02

Building the product. Learning the practice.

The architecture and operating decisions behind Deadwax: three clients, a shared backend, source-backed pressing research, and an evolving workflow for building with AI agents.

Jason · Founder & product lead September 7, 2026 Implementation snapshot + research agenda

---

00 / SCOPE & PURPOSE

A small product with real operating conditions.

Deadwax helps vinyl collectors browse their Discogs collection and wantlist, identify records, compare pressings, and explore the stories around them. It has a web app, native iOS and Android clients, and a shared AWS backend. Android v1 is public; the newest native Collectors Corner update is still in release preparation.

I use the project to learn how AI-assisted development holds up across a complete product lifecycle. That includes the first implementation, but also the review, the release, the bug report, the infrastructure bill, and the decision to remove a feature that people do not find useful.

This paper describes the current design and several decisions that changed it. It separates shipped product capabilities from engineering tools, and both from the broader evaluation lab we are planning. It is a case study of one small live product. The examples are evidence of what happened here, not a benchmark against a conventional engineering team.

> The human contribution is substantial: product priorities, domain expertise, visual design, community work, release judgment, and scrutiny of the agents' output. Assigning an agent a role does not make it a staffed department.

---

01 / GUIDING PRINCIPLES

The harness begins with what matters to us.

Our mission is to help vinyl enthusiasts discover the best version of their favorite music. We put it in the repository, alongside the culture, product vision, design system, and decision log. An agent working on a pressing panel needs more than a description of the component. It needs to know why that panel exists, what collectors trust us to do, and what we have already decided.

The culture document calls for respect, kindness, ownership, grit, humility, and pride in the work. Those values shape the task: challenge an unsupported conclusion without dismissing the person who raised it; disclose uncertainty; finish the verification; preserve the collector's time and trust. The more concrete we make those expectations, the more useful they become in execution.

1. Humans own judgment. AI owns execution within scope.

Jason sets direction and makes the final product and release decisions. Hope supplies the visual identity and participates in brand review. Chris supplies pressing expertise and challenges domain claims. Caden brings the community voice. David has helped us improve the working method. Agents research, propose, implement, test, and review against that context.

This is a responsibility model, not a claim that models never exercise judgment. They do. The important distinction is accountability: an agent's confident output does not approve a brand change, create permission for a new account mutation, or authorize a store release.

2. Write the repeatable operation as a script.

If a task has stable inputs, a repeatable sequence, and a checkable result, we prefer a script. Gathering review context, preparing a branch, running checks, packaging evidence, and following a deployment should not depend on a model remembering the correct sequence of shell commands.

Instructions and skills handle the judgment around that operation: what the reviewer should question, whether evidence supports a claim, or when a change crosses a human decision boundary. The script gives the agent a reliable operation to invoke and an exit status or artifact to inspect. It also gives the next model the same operation.

3. Give the agent the smallest complete context.

"Small" must not mean missing the reason for the work. We use a lean task packet, then load the exact decisions, files, tests, and constraints the task requires. The packet should say what outcome matters, what is in scope, what must remain true, and what evidence will count as completion.

Loading an entire archive can bury the current decision under yesterday's instructions. Loading only the code can omit the product rule that makes the change wrong. Context engineering is choosing the useful middle and maintaining pointers to the durable record.

4. Let the acceptance criteria decide what to measure.

A script can check a schema, permission rule, deterministic score, or expected UI state. A model judge can assess whether an extracted opinion is faithful to its source. A person can decide whether the experience makes sense while holding a record. We choose the check from the failure we need to detect.

GitHub Actions runs the automated checks. Playwright exercises browser behavior against explicit assertions. The evaluator's rubric or test assertion defines what passes. An eloquent explanation from the authoring agent does not replace that evidence.

5. Put permission and cost boundaries around the work.

A task packet names allowed actions. Code and workflow controls enforce the boundaries we can enforce mechanically. Sensitive writes, new dependencies, schema or infrastructure changes, legal and brand decisions, and store releases have escalation or approval requirements. Expensive automation needs a batch size, a deadline, a quota, or a way to stop it.

6. Run a weekly postmortem with the agents, then make the lesson durable.

Every week we review what failed, what caused it, what evidence revealed it, and how the system should improve. The follow-through must live in something durable: a regression test, script, eval case, clearer task boundary, or recorded decision. That way, the next session inherits the improvement instead of repeating the failure.

The working hypothesis is that good context and a well-designed harness make a wider range of models useful. A more capable model gets the constraints it needs to spend its effort well; a smaller or faster model gets a bounded task and a clear test. Our next evaluation work will measure those tradeoffs rather than assume that model names settle them.

---

02 / AGENTS & HUMAN GATES

Give a role a contract, and a person a decision.

The role definitions describe responsibilities, inputs, expected artifacts, and collaboration. A Director coordinates work in Claude Code. The Codex Orchestrator coordinates Codex-owned tasks. A role can be invoked for a bounded task without remaining active all day, and the model behind a responsibility can change.

Role	What it contributes	What it must hand off
Director	Planning, task packets, dependencies, merge order and retrospectives.	A clear task, its decision context, and the next owner.

Role	What it contributes	What it must hand off
Product Manager	Requirements, priorities, stories, acceptance criteria and instrumentation.	A collector outcome that design and engineering can verify.
Designer / Researcher	Research, wireframes, interaction rules, accessibility and visual implementation.	Loading, error and empty states, mobile behavior, and design decisions ready for human review.
Marketing Manager	Positioning, community strategy, launch copy and go-to-market preparation.	Material a human can review and publish in the appropriate channel.
Legal / Compliance Advisor	Research on naming, privacy, licenses and API terms.	Sources, uncertainty and a decision brief. This is an AI advisory role, not licensed counsel.
Architect	Architecture, service contracts, infrastructure and independent review.	Concrete findings, their severity, and a review decision attached to the change.
Backend Developer	Authentication, API behavior, data pipelines and reliability.	A bounded implementation and evidence against the service contract.
Frontend Developer	Browser workflows, responsive layout and interaction details.	A usable implementation tied to the design and acceptance criteria.
Tester / QA	Regression strategy, browser checks, CI and release evidence.	Reproducers, machine-readable results, and failures that still need attention.
Swift iOS Developer	SwiftUI, Apple conventions, lifecycle and native parity.	Native implementation and local build/device evidence, separate from store permission.
Android Developer	Kotlin, Compose, Android conventions and native parity.	Native implementation, tests and Android-specific verification.

These eleven shared role contracts grew out of the launch work. The Codex launch configurations are a different set: they include an Orchestrator, omit an Android wrapper in the audited checkout, and explicitly disable the Director on Codex. A configuration file is evidence of an available role, not evidence that it runs continuously.

Historically, Claude carried the business roles and Codex carried much of engineering. The operating model later moved primary development to Claude and concentrated Codex on review and specialist work. Model settings and scheduled-job overrides add another layer, which is why a diagram that permanently assigns every job to one provider becomes misleading.

Design has a human source.

Hope drew the visual identity. The designer role turns that work and the design system into constraints developers can use: typography, colors, spacing, hit areas, behavior at phone widths, and the states a user encounters when something fails. Its contract requires Jason and Hope to review logo or palette changes before they are finalized.

Chris contributes more than a final approval. He helped choose the initial PI albums, challenged recommendation language and citation style, and later helped distinguish comparative evidence from synthetic scores. His knowledge changes what the model is being asked to produce and what the evaluator should consider correct.

Caden's handoff is different from an engineer's. Launch material needs to be usable outside GitHub. A team member should receive an artifact that fits their work, whether that is a design direction, an Instagram brief, a signed build, or a pull request.

The places we stop for human judgment

- **Product direction and ambiguity:** Jason decides the intended behavior, scope, and priorities.
- **Brand identity:** Jason and Hope decide changes to the visual identity and palette.
- **Domain authority:** pressing claims and disputed recommendations need source evidence and appropriate collector scrutiny, including Chris's expertise.
- **Sensitive changes:** auth, security, privacy, schemas, new dependencies, infrastructure, legal and material cost changes route for the required decision or review.
- **External commitments:** store actions, public messages, and spending require the appropriate human authority. A task to build a feature does not authorize those commitments.
- **Feedback resolution:** functional evidence on the released build determines whether a reporter can be told the issue is fixed.

Some boundaries are implemented in code, hooks, or scripts. Others remain explicit workflow requirements. We should name which is which; writing a rule in a role file does not turn it into a sandbox.

---

03 / REPOSITORY & TASK CONTEXT

The repository is part of the interface.

Code is only one of the artifacts an agent needs. The repository also holds the purpose of the product, the current work, the decisions already made, the procedures it can call, and the evidence it must leave behind.

This selected directory tree shows the boundaries in the audited checkout. It is not a complete listing.

```
discogs-app/
|-- AGENTS.md                       # Codex operating contract
|-- CLAUDE.md                       # Claude operating contract
|-- .claude/
|   |-- hooks/                      # workflow guards
|   `-- skills/                     # judgment-heavy procedures
|-- .codex/
|   |-- agents/                     # role launch configurations
|   `-- skills/
|-- .github/workflows/
|   |-- ci.yml
|   |-- deploy.yml
|   |-- docs-protection.yml
|   |-- auto-fix-path-guard.yml
|   |-- coverage-full.yml
|   `-- gemini-youtube-analyzer.yml
|-- agents/
|   |-- TODAY.md                    # current task context
|   |-- COMMS-TODAY.md              # recent decisions/routing
|   |-- REVIEW-LOG.md               # durable review evidence
|   |-- roles/
|   |-- decisions/DECISION-LOG.md
|   `-- execution/                  # bounded feature packets
|-- scripts/
|   |-- worktree-create.sh
|   |-- pre-pr-cleanup.sh
|   |-- pr-create.sh
|   |-- pr-review.sh
|   |-- codex-review.mjs
|   |-- pi-eval.mjs
|   |-- token-audit.sh
|   |-- codex/ship-loop.sh
|   |-- bots/                      # scheduled task wrappers
|   |-- tests/
|   |   |-- run-all.sh
|   |   |-- run-all.test.sh
|   |   `-- manage-flaky.sh
|   `-- vitals/ingest-test-results.mjs
|-- tests/
|   |-- flaky.json
|   `-- artifacts/runs/             # logs and reporter output
|-- apps/
|   |-- web/
|   |   |-- src/
|   |   |-- e2e/
|   |   `-- playwright.config.ts
|   |-- api/src/
|   |   |-- handlers/
|   |   |-- services/
|   |   |-- vitals/
|   |   `-- pi/evals/
|   |       |-- run-output-eval.ts
|   |       |-- run-judge-eval.ts
|   |       `-- golden/
|   |-- ios/Deadwax/Deadwax/
|   `-- android/
|-- packages/
|   |-- shared/
|   `-- config/
`-- docs/
    |-- company/                    # mission, culture, team
    |-- product/                    # vision, stories, backlog
```

```
    |-- design/                    # brand and interaction rules
    `-- engineering/               # architecture, tests, evals
```

Operating contracts point to context. Role files assign responsibilities. Skills describe procedures requiring judgment. Scripts package repeatable operations. Tests make behavior observable, and artifacts preserve what a particular run actually did. The decision log records why a constraint exists.

Keeping these separate lets a task load the small portion it needs. A scanner fix can read its native interaction rules and source files without loading every marketing plan. A marketing task still needs the mission, the audience, and the claims the product can support.

A reusable task packet

This template distills the practice into a handoff another team could adapt. It is a suggested format, not a required schema already enforced across every Deadwax task.

```
Task: One concrete change or failure.
Intent: The user outcome and why it matters.
Current symptom: State + action + observed behavior.
Success: State + action + expected behavior.
Ownership: Exact files or contracts; other workers' boundaries.
Read first: Current packet, relevant decisions, code and tests.
Constraints: Allowed writes, privacy, cost and platform rules.
Dependencies: Required upstream work and merge order.
Verification: Reproducer, checks and review evidence.
Completion: Release evidence and functional confirmation.
Handoff: Commit, artifacts, uncertainty and next owner.
```

For example: when a collector chooses Artist and Ascending, the request must carry both parameters together and the visible list must update. That is more useful than “fix sorting,” and it gives a test a specific request and state to inspect.

Keep history available without making it the startup prompt.

Our lean runtime files point to durable decisions and execution records. We open history by topic when it matters. The token-audit script reports words, lines, bytes, and prompt-load hotspots. It is a useful maintenance signal, not a model tokenizer or a currently enforced token-budget ceiling.

We learned this boundary after shared coordination documents entered the wrong code workflow and overwrote later work. Protected paths, cleanup scripts, and diff checks make ownership more concrete. Worktrees help isolate code edits; they do not automatically solve shared-document ownership or constitute a security boundary.

FROM DAVID / A PROMPT TO BORROW

Give your AI workflow a checkup.

A question from David helped us look at what the agents were reading, where tokens were going, and which repeated steps belonged in scripts.

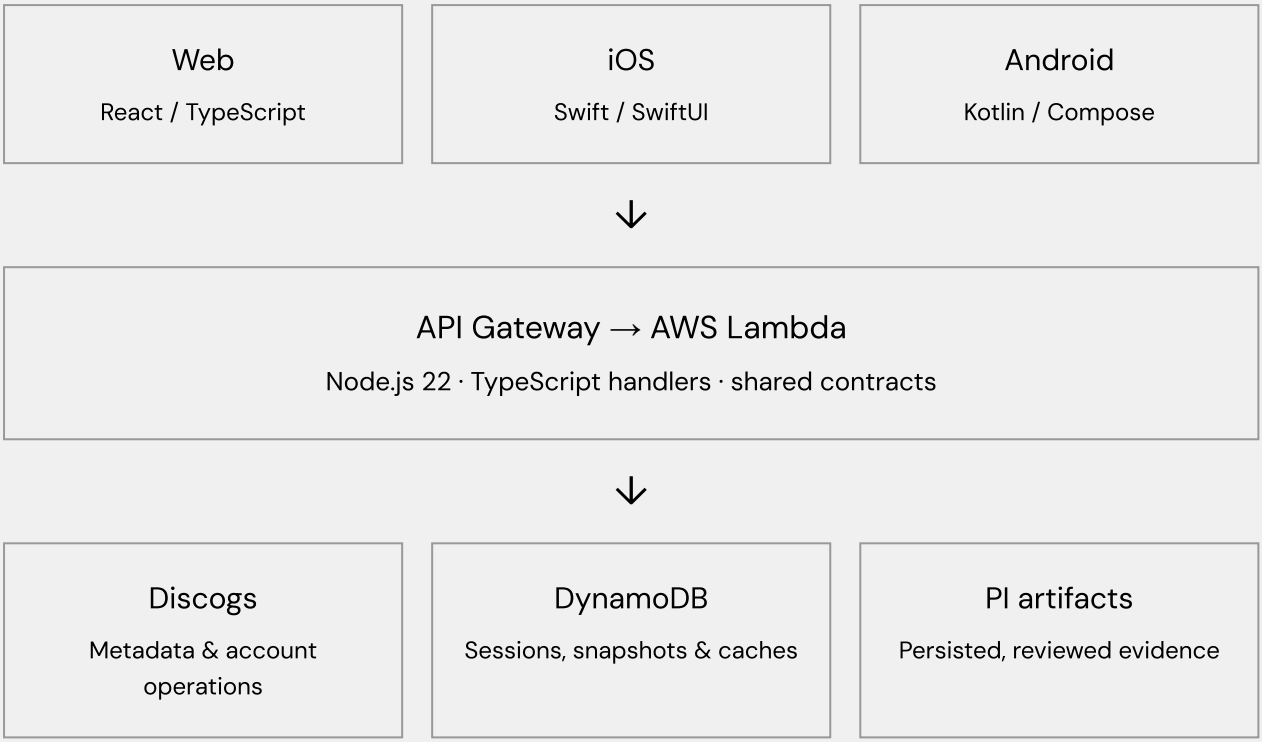
```
Analyze my Claude Code and Codex environment, including CLAUDE.md and AGENTS.md, for token usage, workflow optimizations, and tools vs. skills/script usage.
```

04 / PRODUCT ARCHITECTURE

Three clients.
One set of service contracts.

The clients share an API and product rules. They do not share every screen. The web app uses React and TypeScript; iOS uses SwiftUI; Android uses Kotlin and Jetpack Compose. Native navigation, camera behavior, keyboard focus, accessibility, and application lifecycle each need work on the platform where they run.

FIGURE 1 / THE SERVING PATH



Web assets are delivered through CloudFront from S3. This is a serving diagram; scheduled data preparation and request-time cover recognition are separate paths.

Layer	Current implementation	Why it matters
Web	React 19, TypeScript, Vite, React Router; static delivery through S3 and CloudFront.	Public content and browser workflows can ship independently of a store review.
Native	SwiftUI on iOS; Kotlin and Jetpack Compose on Android.	Shared data still requires platform-specific interaction and release checks.
API	API Gateway HTTP events, explicit TypeScript routing, Node.js 22 Lambda handlers.	The handlers own authentication, input validation, upstream calls, and response contracts.
State	DynamoDB for server-side sessions, collection snapshots, caches, and operational records.	A large collection can be served from a usable snapshot while synchronization continues.
Pressing evidence	Persisted read-model artifacts, curated mappings, and extracted source claims.	Opening a PI panel does not require generating a fresh model answer.
Configuration	AWS Systems Manager Parameter Store and server-side configuration.	Credentials belong to the service boundary, outside the client bundle.

Follow a collection request

A collector signs in through Discogs OAuth. The web client receives an opaque session identifier in a secure cookie; the backend retains the session state and Discogs credentials. A collection request resolves that session, reads the available snapshot, and uses the synchronization machinery to fetch and publish updates when needed.

Large collections made snapshot integrity and response delivery first-class concerns. The service has asynchronous synchronization and continuation paths; it should not replace a usable snapshot with a partially built one. The experience also needs an honest distinction between “you can browse this data” and “this data is fully current.”

Implementation anchors: `lambda.ts`, `handlers/index.ts`, `services/session-store.ts`, `services/collection-sync.ts`, and `pi/read-model.ts`. PostgreSQL adapters exist in the repository; the default PI read path described here serves persisted artifacts.

---

05 / EVIDENCE TO PRESSING

The hard part is which record a claim belongs to.

An album title is a poor identifier for a physical record. Artwork, mastering credits, catalog numbers, and release years can overlap across editions. A source may praise a particular mastering cut without identifying the pressing plant. Matching that opinion to one exact Discogs release requires more evidence than matching the album name.

Pressing Intelligence combines release facts, curated research, and claims extracted from sources such as forum discussions and video reviews. These inputs have different authority. A catalog fact describes an edition. A listener’s opinion describes an experience. An Album Story gives historical context. None should silently become proof of the others.

1. **Establish identity.** Work from the album and the candidate releases, preserving the edition details that distinguish one copy from another.
2. **Keep the source attached.** Normalized claims retain source references, release associations, strength, opinion status, and paraphrased evidence.
3. **Resolve only what the evidence supports.** A general statement about a mastering cut should not manufacture certainty about a specific plant or issue.
4. **Rank and explain.** Combine eligible evidence with the scoring rules, present the relevant sources, and retain disagreement or weaker confidence where appropriate.
5. **Evaluate the served result.** Check the actual output against versioned cases so a correct intermediate claim does not hide a wrong final recommendation.

How the pipeline runs now

The September 2026 evidence-quality postmortem found that every Pressing Intelligence guard asked whether anything bad was present and none asked whether the good evidence was still there, so a series of individually defensible changes removed coverage without a test failing. The redesign puts one channel allowlist in front of all four intake paths, records coverage floors beside the purity ceilings, groups served evidence one card per source document, and evaluates the shape of what is served on every pull request.

```
PRESSING INTELLIGENCE, END TO END          DEC-134 · September 2026

SOURCES            INTAKE — four paths, one shared gate
+--------------+   +-----------------------------------+
| YouTube      |──▶| Gemini analyzer      (weekly)     |
| (allowlisted |   +-----------------------------------+
|  channels)   |──▶| PI backfill bot      (hourly)     |
+--------------+   +-----------------------------------+
| Steve        |──▶| Feedback auto-fix bot (hourly)    |
| Hoffman      |   +-----------------------------------+
| Forums       |──▶| Hand curation        (pull request)|
+--------------+   +-----------------------------------+
| Discogs      |──▶ identity, credits, matrix / runout
+--------------+                    |
                                    v
   discovery-channels.json is the ONE allowlist all four read:
   `discovery` channels are polled; only `curated` publishers
   may carry a comparative claim.
                                    |
                                    v
THE REPO IS THE DATASTORE — no database
   persisted-artifacts.ts · curated-seeds.json
   gemini-claims.json · quality-baseline.json · corrections
                                    |
                                    v
GATES — every ceiling now has a floor
   Ratchet     floors may only RISE:  albums with video,
                 distinct documents, sources, picks,
                 explanations, timestamps
               ceilings may only FALL: forum-only share,
                 raw excerpt debt
               a deleted citation needs a replacement
                 or a logged waiver
               a bot change crossing a tier line waits
                 for a human
   Invariants  badge-shaped labels, filler ban, allowlist
   Shape eval  ⩾1 comparative document per recommendation,
                 ⩽3 documents per source type, ⩽65-word
                 excerpts, no usernames, one document ⇒ no tier
   Output eval which pressing wins (golden set)
                                    |
                        merge ⇒ deploy (one Lambda)
                                    v
SERVE — per request
   relevance gate on the raw text
     → rank on net-positive comparative evidence
     → group by DOCUMENT (one thread or video = one card,
        anchors nested)  → cap 3 per source type
     → 65-word excerpts, usernames stripped
     → consensus + tier (one document = INSUFFICIENT, no tier)
                                    |
                                    v
        Web, iOS and Android render the same source cards
```

A concrete identity failure

Collectors Corner can feature a specific edition of an album. A collector may already have another edition on their wantlist. Redirecting the featured card to the wanted edition prevents one kind of duplicate, but it can also make a rare featured pressing open an unrelated copy.

The September work changed that interaction: the card opens the featured pressing, makes the other wanted edition explicit, and asks for confirmation before adding another. We also corrected edition destinations and cover mappings. The lesson crosses the API and the interface: preserve the identity the collector chose, then explain the relationship to the record they already wanted.

What a recommendation does not establish

A top-ranked pressing is a recommendation from the evidence we have. It is not a guarantee about the condition of a seller's copy, a claim that all listeners agree, or proof that we have reviewed every pressing. Coverage varies. Conflicting sources remain relevant, and facts-only records should not acquire a strong recommendation merely because their metadata is complete.

Implementation anchors: `pi/claim-normalization.ts`, `pi/scoring.ts`, `handlers/pressing-intel.ts`, and the curated release mappings. The exact-release handling includes protections against replacing a forum-backed release identity with a broader overlay.

06 / WHERE MODELS RUN

Different jobs need different paths through AI.

"AI-powered" hides several distinct systems. Their latency, cost, and failure modes differ enough that they need separate controls.

Work	When it runs	How we contain it
Video evidence extraction	Batch preparation with Gemini, producing structured claims and versioned artifacts.	Review the output, preserve provenance, run regression checks, and serve persisted results.
Cover recognition	During a collector's scan. Claude vision is primary; Gemini is a fallback in the current implementation.	Authentication, daily quotas, deadlines, metered extra attempts, and a bound on downstream Discogs calls.
Barcode / label / deadwax recognition	Native capture and device tooling, including Apple Vision and Google ML Kit.	Let the collector inspect and correct recognition before committing to a search or edition.
Development and review	Task-driven Claude Code and Codex sessions, with bounded automation for maintenance.	Task contracts, scoped context, isolated work, tests, independent review, and release checks.
Claim-quality judging	On-demand evaluation with a separate Claude judge.	Explicit rubrics, known bad claims, repeated trials, and an "uncertain" outcome.

Recognizing a sleeve is useful for finding an album. A shared sleeve cannot prove which pressing is inside it. The interface has to preserve that distinction instead of turning a successful image match into false edition certainty.

Keeping PI generation out of the reading path also makes the product easier to operate. A page view reads prepared evidence. It does not invite a new round of model speculation, latency, and billing every time a collector opens the panel.

The label-and-voice workflow came from a human workaround.

Early runout OCR experiments struggled with glare, curved text, shallow etchings, and a collector moving the record to find the light. The useful alternative was already in Jason's behavior: scan the label for context, then read the markings into the phone. The native flow records the whole utterance, transcribes it, normalizes code-like speech, and lets the collector correct the result before search.

Chris's surprise at automatic microphone activation led to an explicit opt-in step. That decision is recorded in the implementation. A tool that can record audio still needs an interaction that gives the collector control. Native label/speech handling and remote cover recognition have different data paths and must not share a vague "everything stays on device" description.

The current iOS cover endpoint uses Claude Haiku as primary and Gemini as fallback after field comparisons. It generates album-search clues and corroborates them against Discogs. Native barcode and label workflows also serve Android; the newer cover-recognition surface is not being represented here as an Android launch feature.

---

07 / DEVELOPMENT WORKFLOW

The workflow grew up with the product.

I began by giving agents specialized roles. That was a useful way to divide work across unfamiliar areas. The more durable work was deciding what each task had to prove and how another person or agent could challenge the result.

At launch, we used a daily planning rhythm. After launch, bugs arrived continuously and larger features arrived in bursts. The daily paperwork started to drift away from the real work. We moved to continuous operations for fixes and bounded feature projects with their own scope, acceptance criteria, and closeout evidence.

Claude became the primary developer and orchestrator, with Codex used for independent review and selected specialist or rescue work. As review costs became harder to justify, we made the route sensitive to risk and capped repeated review rounds. The current router gives unknown or sensitive paths the deeper review route. Exceptions and substitute reviews are recorded.

A task's path through the system

1. **Capture the symptom and the intended outcome.** A useful report gives us something we can reproduce. Product ambiguity needs a decision before implementation.
2. **Assign bounded ownership.** A task names the files or contract it owns, dependencies, and the evidence needed at completion. Parallel work only helps when those boundaries hold.
3. **Load relevant context.** A lean runtime packet points to durable decisions. Code, tests, and constraints are loaded for the work at hand. Loading every document is not a substitute for choosing the right context.
4. **Implement and test locally.** Separate working copies reduce interference. Maintained scripts handle deterministic preparation, checks, and release mechanics.
5. **Review according to risk.** Another reviewer challenges the change. Backend, authentication, infrastructure, schema, and cost-sensitive surfaces receive particular attention.
6. **Release through the appropriate channel.** Web deployment and native store distribution are different events. A merged native change is not yet on a collector's phone.
7. **Confirm the behavior.** A feedback issue stays pending until the original symptom is shown to be resolved on the released build.

▶ Context, instructions, and scripts: the practical distinction

These are controls with limits. Worktrees isolate edits; they are not a security sandbox. Protected-file checks reduce accidental changes; they do not make an agent incapable of making a mistake. A second model can challenge the first and still share its blind spot. That is why we need several forms of evidence.

08 / CI, PLAYWRIGHT & FLAKES

The gate needs tests, too.

Our script-first principle applies to the machinery around a change. The scripts that gather evidence, classify a failure, or permit a release can be wrong. They need tests for their normal paths and for the exceptions they allow.

Operation	Script	Evidence or boundary
Prepare isolated work	`worktree-create.sh`	Creates a branch/worktree; ownership must still be assigned.
Prepare a code PR	`pre-pr-cleanup.sh`, `pr-create.sh`	Rebase, remove protected-file changes, invoke review, push and open the PR.

Operation	Script	Evidence or boundary
Retrieve and route review	`pr-review.sh`, `codex-review.mjs`	Collect context, pin the base, route the diff, and preserve the verdict.
Run regression work	`tests/run-all.sh`	A run ID, script tests, workspace tests, browser checks, logs and reporter files.
Handle known flakes	`tests/manage-flaky.sh`	Known signatures, a command retry, structured classification, and retained attempts.
Evaluate PI output	`pi-eval.mjs`	Versioned cases, deterministic output scoring or a model judge, Markdown/JSON reports.
Complete web release	`codex/ship-loop.sh`	The merged commit, its CI/deployment, and endpoint checks. User-symptom verification follows.

What GitHub Actions runs

Applicable code pull requests run lint, type checks, targeted security checks, and the maintained test bundle. That bundle includes tests of our scripts, application tests, and Playwright. Path filters add an unsigned iOS simulator build or Android debug build, lint, and selected native unit tests when those surfaces change.

CI also publishes changed-line coverage and test artifacts. Coverage percentages are currently advisory; a low percentage is not itself a failing threshold. Execution failures still matter. Documentation-only paths and ordinary main pushes do not all run the same heavy suite. A separate weekly workflow supplies full coverage, and explicit main-commit markers can request heavier validation.

Playwright supplies explicit assertions about browser workflows. We use controlled application data and a mock API in the test environment so a collector journey can be exercised repeatably. Auth smoke, pressing panels, feedback, and UI regressions have dedicated specifications. The browser and test server can still have timing failures; a scripted test is not a promise that its environment is deterministic.

A flaky browser check can teach the wrong lesson.

One recurring failure came from observing the wrong network request. A test inspected the last request in a list, but a background request could arrive later. Increasing a timeout did not fix the observation. We changed the assertion to follow the relevant request and tested it against the old failure mode.

Other failures involved Vite import contention and connection resets. The repair needed a reproducer and a diagnosis, not a broad decision to ignore browser failures. A noisy gate loses value if everyone learns that red usually means “try again.”

Vitals and quarantine: an explicit exception.

The test-health system ingests machine-readable results, identifies unstable tests, and keeps quarantine separate from a clean pass. The detector uses a 14-day window, at least 20 attempts, a failure rate of at least 5%, and evidence such as a retry pass or mixed results for the same commit. Quarantine carries an owner and repair deadline. It is temporary debt, not permission to delete the test.

Known quarantined tests still run. The flake manager can return a distinct protocol result when a persistent failure matches quarantine. The outer runner must then verify every reported failure, including the saved first attempt. One recognized flake must not excuse an unrelated failure beside it, even if that unrelated failure disappears on retry.

We test that exception with mixed failures, missing reporter files, a real failure that vanishes on retry, and truly quarantined-only failures. A runner crash with a stale report must not look like passing tests. That is harness engineering in a very practical form: test the mechanism that decides whether a test failure may be tolerated.

Retries exist at several layers: Playwright's per-test CI retry, the known-flake command retry, and a browser-install recovery path in the outer runner. They can compound. The important property is that attempts and classifications remain visible. We do not claim that the entire pipeline retries exactly once.

Code review adds a different check.

The reviewer can notice that an assertion proves the wrong behavior, that a cache changes security semantics, or that a rare path increases cost. Our routing script uses an allowlist of routine surfaces; an unknown or sensitive path sends the whole diff to the deeper reviewer. That rule emerged after attempts to enumerate only risky paths missed cases.

The routing, base selection, invocation, and verdict recording are scripted. The substantive review is model judgment. Review is a required practice with documented fallback and override policy; it is not an unbypassable mathematical proof. Keeping the reviewer different from the author broadens the challenge, while tests and human decisions supply other evidence.

Source anchors: `.github/workflows/ci.yml`, `apps/web/playwright.config.ts`, `scripts/tests/run-all.sh`, `scripts/tests/manage-flaky.sh`, `apps/api/src/vitals/`, and their regression tests. This describes configured behavior, not a fresh CI or branch-protection audit.

---

09 / SCHEDULED WORK

Turn signals from use into bounded work.

The two hourly jobs run through installed macOS launchd tasks, the maintained replacement for the older cron setup. A dedicated checkout isolates this work from an interactive development session. The jobs have process locks, empty-work checks, failure reporting, and disable controls.

Loop	Schedule	Evidence status
Feedback triage and fixes	Hourly at local minute :00	Installed and loaded locally at the September 7 audit.
Recent-opened-album PI	Hourly at local minute :30	Installed and loaded locally at the audit.
Admin snapshots	Daily, 09:40 UTC	GitHub workflow configuration; remote execution state was not rechecked for this paper.
AWS cost-history append	Daily, 11:40 UTC	GitHub configuration with a shared daily fetch claim.
Discogs profile cache	Monday, 09:17 UTC	GitHub configuration for bounded cache refresh.
Gemini video claims	Sunday, 15:30 UTC	GitHub configuration with a scheduled-run disable variable.
Full coverage	Monday, 08:00 UTC	GitHub configuration for the fuller coverage report.

Feedback: a small queue with a truthful finish line.

The feedback wrapper checks for actionable reports, new replies, and unfinished task packets before starting a model session. Its operating contract allows up to five fix tasks in a run, ordered by priority, effort, and age. That is a task limit, not a promise that five issues will be solved.

Permitted bug work follows the code PR path: a scoped change, focused checks, independent review, and the release loop. Sensitive or ambiguous work escalates. Store actions remain separately authorized. If the released behavior has not been confirmed, the feedback issue stays Fix Pending.

Album demand: research what people are opening.

The PI wrapper queries a bounded 180-minute window of `record_explored` events and prepares at most 80 candidate albums. It aggregates views and distinct-user counts, retains missing or unknown PI opportunities, and cross-checks uncertain native flags against local coverage.

The agent then researches pressing-specific evidence. Useful material can be a particular Steve Hoffman thread, a genuine comparison video, or verified Discogs facts. Evidence must attach to the right release, preserve disagreement, and respect existing human-verified facts. No useful evidence is a valid reason to make no change.

The additive data loop has a deliberately narrower path than a code fix. Its wrapper permits three data/audit paths, rejects changes outside them and rewrites of existing artifacts, then validates seeds, builds the API, and runs focused PI regression tests. This documented exception can commit additive data directly to main; a dedicated PI deployment workflow checks the data diff and verifies the runtime. It is not an individually Architect-reviewed code PR for each album.

The 80-album limit bounds candidate preparation. It is not a demonstrated hard ceiling on every outbound research call or model token. Query bounds, write scope, model cost, and agent behavior are separate controls and should be measured separately.

Gemini: a separate source-processing pipeline.

The weekly video workflow discovers recent material and analyzes up to 12 videos by default with Gemini. Changed artifacts go through regression checks and a pull request. Its PI output-quality eval is currently report-only. The score is visible for review, but the scheduled configuration does not block publication merely because it falls below the quality threshold.

The installed system is the source of truth for what runs.

The September 7 inspection found both local jobs configured to invoke Codex with Sol in their dedicated checkout. Main-repository defaults and some recent notes described Claude-based bots. We preserved that distinction here rather than turn an intended migration into an operating claim. No job configuration was changed for this paper.

This is one of the reasons I treat the harness as software: its installed configuration, failure behavior, and history need the same attention as its source. A scheduler entry proves a configured opportunity to run, not an uninterrupted history of successful outcomes.

---

10 / NORTH STAR & TELEMETRY

A North Star, and the questions around it.

Our business North Star is **Total Pressing Intelligence Panels Viewed**, cumulative over time. A qualified view means usable pressing data rendered. It represents the moment the product offers decision support, rather than merely receiving traffic.

We try to grow it through acquisition and deeper use, while improving the supply of credible pressing evidence. Search and scans help reach an album; collections and wantlists create recurring contexts; stories and corners invite exploration; source research makes more of those visits useful. The metric remains a proxy for value, not proof that every view caused a good purchase.

Guardrail	Definition	Target
PI activation	Distinct collectors with PI views divided by active collectors.	At least 40%.
PI availability	Usable PI views relative to panel attempts, including failed loads in the denominator.	At least 80%. This is not a catalog-wide coverage percentage.

Guardrail	Definition	Target
Collection success	Collection successes relative to success, failure and incomplete-index outcomes across clients.	At least 95%. The current card is narrower than the broader strategy for all browse/search paths.
PI load time	P90 of measured panel loading, retaining source/platform samples.	Under two seconds on cached/normal return paths.

These are defined targets. This paper does not present a newly collected performance snapshot or assert that every target is currently met.

The diagnostic layer in Admin

Area	What we instrument and report
Adoption and return use	DAU, WAU, MAU, stickiness ratios, new and cumulative users, activity trends, platform splits, and D1/D7/D14 retention with eligibility and coverage limits.
Collection reliability	Success, failure and incomplete indexing, affected-user counts, load/index time distributions, failure categories, and rollout comparisons.
Search and scanning	Searches and album-click rate; barcode, deadwax and combined label/deadwax starts and hit classifications. A start is not counted as a successful identification.
Pressing decisions	PI views and unique users, album opens with or without known PI, YouTube/SHF source clicks, wantlist actions, collection adds, and marketplace handoffs.
Collection tools	Dice tap/shake rolls, filter opens/applies, sort changes, theme changes and distributions, with platform and daily trends.
Discovery and supply	Most expanded albums and clicked tiles; curated-album growth; recommendation versus facts-only tiers; YouTube, forum and Discogs citation coverage.
Collectors Corner	Corner opens, album taps, story and note opens, resource clicks, pins/unpins/hides, sorting, and Discover tile customization.
Native behavior	iOS/Android active use, sessions, app opens, PI use, feedback, themes, top screens/tabs, version mix, and reliability events.
Service health	API/frontend errors, loading latency, Lambda timing and throttles, API invocations, rate-limit pressure, and authentication funnel fields.
Economics	Modeled per-user, cohort and platform cost; GitHub estimates; actual AWS daily billed history, month-to-date/forecast views, and modeled user versus background cost.

Area	What we instrument and report
Feedback and moderation	Submitted feedback, ratings and trends, issue outcomes, automatic fixes, escalations, automation health fields, and reported comment state.
Report trust	Snapshot age, refresh status, sample sizes, missing-history warnings, and distinctions between active, dormant, known, and anonymous activity.

Some diagnostics are present in the service payload without a dedicated headline card. The private user directory also contains support information, so “pseudonymous event telemetry” is more accurate than claiming the whole Admin system is anonymous. Public material should describe aggregates rather than reproduce collector identities, raw searches, or individual activity.

Measurement contracts are implementation work.

The event handler accepts a fixed set of properties, validates its pseudonymous identity field, and rejects credential-shaped or username-shaped input. That keeps arbitrary payloads from becoming a convenient route for sensitive data into product telemetry.

Legacy and current PI event names need careful handling. The current rollup takes the larger daily count of the two names to avoid simply adding them together. That is coarser than the desired interaction-level deduplication by user, session, and record. The distinction matters when interpreting the total.

Durable weekly North Star summaries now survive beyond raw-log expiry. Some early history had already been lost. Reports distinguish missing, partial, and measured periods instead of inventing a complete launch-to-today curve. The North Star’s cumulative history is separate from the normal 30-day window for log-derived guardrails.

Observation has an operating cost.

Admin page loads normally read cached DynamoDB snapshots. Daily scheduled rebuilds provide freshness; broad age-triggered page-view rebuilds default off. During an active rebuild, the UI polls the cheap snapshot read on a bounded schedule. Log lookbacks, row counts, query concurrency, and retries have limits.

Actual AWS cost history uses a shared daily fetch claim, so the append job and the snapshot refresh cannot both spend on the same daily Cost Explorer request. Modeled cost, a forecast, a spending cap, and an actual bill remain different quantities. These controls let us learn from use without making every dashboard visit an expensive investigation.

Source anchors: `docs/engineering/METRIC-STRATEGY.md`, `handlers/admin-metrics.ts`, `handlers/events.ts`, `AdminPage.tsx`, and the daily snapshot/cost workflows. No live log scan or private-user export was run for this paper.

11 / LESSONS FROM LIVE USE

The failures changed what we check.

1. A successful handler can still fail the request.

A collection of roughly 5,900 records exposed a response-size failure. The handler logged success, but the Lambda runtime could not deliver the full response. The client continued showing stale data. Small paged checks missed the failure in the request the collector actually used.

The recorded fix reduced a 6.3 MB response to about 1.24 MB with gzip and strengthened snapshot publication and synchronization behavior. Those are incident measurements, not a standing performance guarantee. The lasting change was to verify end-to-end delivery at realistic collection sizes.

2. A retention setting can erase the evidence for a metric.

A time-to-live policy removed old activity records that retention charts depended on. A cumulative identity index still knew the accounts existed, while the activity table no longer held the history. Fixing retention and preserving summaries helped going forward. It did not recreate data that had already expired.

We now distinguish retained evidence from missing history instead of filling gaps with plausible-looking numbers. Storage lifecycle is part of metric design, and observability has its own cost and retention requirements.

3. Model choice is a product decision with a bill attached.

Field tests of cover recognition exposed confident misidentification and availability problems with the initial primary provider. The comparison favored Claude for the cases we tested, so the primary and fallback order changed. The more useful model also cost more per scan. We lowered the global budget at the same time.

That is evidence for this task and these images. It does not establish a universal model ranking. It does show why model choice, retry behavior, accuracy, and spend limits should be evaluated together.

4. A working feature can make the product harder to use.

Batch scanning kept the camera open for the next record. Feedback showed that the flow hid pressing results and ownership information collectors wanted to inspect. We rolled it back. The useful success criterion was whether someone could understand the record in their hand, not how many camera captures the interface could accept.

Incident sources: collection-sync architecture notes; the August analytics-retention decision; the cover-scan provider decision; and iOS version history ↗. Detailed source references are retained with this edition's review material.

12 / SECURITY & COST

The lab has real users, credentials, and bills.

The opportunity to experiment does not remove responsibility for the product. We use explicit service boundaries and spend limits so routine use does not become unbounded background work.

Boundary	Current approach	Limit to keep in view
Account access	Discogs OAuth credentials remain server-side. Web sessions use opaque identifiers in HttpOnly, Secure, SameSite=Lax cookies.	Cookie settings are one control; authorization and sensitive-route review still matter.
Discogs writes	The generic proxy is read-only. Explicit handlers allow wantlist add/remove and collection add to folder 1, including condition grades on the added instance.	New mutations require an explicit policy decision; “read-only app” is no longer an accurate shorthand.
Cover scans	Per-user and global daily quotas, an extraction deadline, metered extra generation attempts, and at most four downstream Discogs calls in the handler’s budget.	Limits bound exposure; retries, providers, and prices still require monitoring.
PI reads	Serve persisted evidence and prepared read models.	Freshness and coverage need their own checks; cached does not mean correct.
Admin metrics	Cached snapshots and bounded scheduled work. Automatic age-based page-view refresh defaults off.	A dashboard visit should not trigger a broad live log scan. Missing history must be labeled.
Native releases	Build and local device verification are separate from store upload and public rollout. Store writes need explicit human authorization.	Build success is not distribution, and distribution is not proof that a reported symptom is resolved.

Serverless infrastructure reduces the capacity we have to manage. It does not eliminate storage, logs, scheduled jobs, external API costs, or CI minutes. We treat those costs as design inputs. Backfills and repair work should start with a dry run or a bounded batch; recurring work needs a way to stop it.

13 / EVALS: BUILT & PLANNED

Useful evidence. Still an unfinished system.

A normal test can show that a request is authorized, a value is calculated correctly, or a view renders. Model-assisted features add other questions: did the extracted claim match the source, did we connect it to the right edition, and did the final ranking preserve the intended evidence?

What exists today

Evaluation	What it measures	Current status
PI output regression	Versioned reference cases, precision at ranks 1 and 3, coverage, and expected placement in served output.	Implemented. Eight output cases in the audited dataset. Weekly analyzer workflow is report-only; strict execution is available separately.
Claim-quality judge	A separate Claude model grades source support, correctness, and hallucination against transcript excerpts and explicit rubrics.	Implemented, on demand. Nine reference cases include deliberately incorrect claims. Repeated trials and an uncertain verdict expose some judge instability.
Broader AI Evaluation Lab	Model, reasoning effort, prompt, context, tool, and workflow configurations on the same tasks.	Current first priority, at baseline and design stage. The benchmark platform and later routing experiments are not complete.

The answer key has to earn trust, too.

The first output labels were partly seeded from the extraction pipeline's own declared winners. That makes some results a check of extraction-to-ranking consistency, rather than an independent check of the source. Human re-verification and independently read source material are needed to strengthen the reference set.

A source can also declare one winning cut while the curated ranking places another edition first. The evaluator preserves those distinctions instead of treating any disagreement as a simple wrong answer. We should never relabel a reference case merely to make the score improve.

Using a different model as a judge helps challenge the extraction model. It does not make the verdict independent truth. Known-bad claims, repeated trials, written reasoning, and explicit uncertainty are useful checks on the judge itself.

> A passing result applies to the cases and rubric tested. We are not publishing a catalog-wide accuracy claim, and we are not claiming that every PI refresh is already blocked by an eval gate.

The next harness: a controlled comparison.

The proposed lab begins with instrumentation and baselines. Each run should identify the task, source version, prompt, model, reasoning setting, relevant context, tools, and workflow. Results need to retain quality judgments and failure categories alongside latency, token use, and cost. Without that record, a cheaper run could simply have done less of the work.

1. **Establish a baseline.** Close instrumentation gaps and run a fixed set of representative tasks. Record failure modes and the known limits of the labels.
2. **Compare configurations.** Change models, effort, prompt style, or workflow against the same work. Repeat enough runs to see unstable results.
3. **Calibrate a regression gate.** Decide which failures should block a change and which require review. Close the current gap between reporting a score and acting on it.
4. **Test handoffs.** Compare full replay, summaries, and structured state when work moves between models or environments. Measure which decisions and constraints survive.
5. **Use the evidence for routing.** Only after there is reliable history should measured results inform which configuration receives which task.

▶ A concrete example: one feature, two kinds of proof

The lab is engineering infrastructure, not a new collector chatbot. Experiments should run on demand with explicit spend boundaries. The current path-based review router is a rule we can inspect; the planned empirical router is a hypothesis we still have to test.

Implementation anchors: `scripts/pi-eval.mjs`, `pi/evals/run-output-eval.ts`, `pi/evals/run-judge-eval.ts`, and versioned golden cases. Research scope: the September AI Evaluation Lab charter. No fresh production eval run is represented by the counts above.

---

14 / OPERATING THE BUSINESS

Engineering is part of a larger feedback loop.

The project has also become a way to learn work I would not encounter in an isolated coding exercise. Preparing a store release and introducing it to a community both require judgment beyond the code.

We tested paid advertising on Facebook and Instagram in August. Ad review influenced which creative we could use. Later launch planning separates iOS and Android destinations and creative so we can read results by platform. That is a measurement plan, not a demonstrated return on advertising.

The community launch has its own constraints. The September Google Play post on the Steve Hoffman Music Forums leads with the collector's problem, thanks beta testers, and invites source-grounded corrections. The product story has to make sense to people who have no interest in our agent workflow.

These activities affect the engineering agenda. A campaign needs a working destination and a way to understand activation. Trustworthy business metrics need retained data. A store release needs accurate screenshots, metadata, and a real build. The learning comes from taking responsibility for those connections, not just producing more artifacts.

15 / A PRACTICE TO BORROW

What we are committing to learn next.

If you want to build this way

Start with a useful product problem and one bounded agent task. Write the mission, customer context, non-negotiable constraints, and an observable completion test. Make the repeated commands callable. Keep the diff small, have another reviewer challenge it, and preserve the evidence of what ran. Add another role or scheduled loop only when its ownership and limits are clear.

Then improve the workflow from its failures. Fix a noisy test before it teaches people to ignore the gate. Test the quarantine exception. Tie release evidence to the correct commit. Verify the user's symptom. Measure the model-dependent parts against stable cases before introducing more autonomous routing.

The current priority order begins with the AI Evaluation Lab, then carefully timed requests for user feedback. Collectors Corner is live on web, with the latest native release being prepared. Collection value and insight work follows at a lower priority, with historical and external-market data dependent on source and use decisions.

Further out, we want to understand whether agent handoffs can preserve the work that matters and whether model routing can be supported by measured evidence. Stereo equipment planning is another area to explore, but privacy, costs, sources, and the product scope still need decisions.

I do not expect the current agent setup to be the final one. Models, tools, prices, and our own needs will change. What I want to keep is a record of what worked, a way to notice when it stops working, and the habit of checking the result where a collector actually uses it.

See the current product roadmap ↗

16 / NOTES & REFERENCES

How to read the claims.

This edition was checked against the implementation, release records, decision log, and current backlog on September 7, 2026. "Implemented" describes available code or tooling; "public" describes a collector-accessible release; "planned" describes work without a completed implementation. The distinction matters most for native distribution and the evaluation lab.

Deadwax on the App Store ↗

Public iOS release history, including the May launch and the scanner rollback.

Deadwax on Google Play ↗

Public Android availability. The latest signed source build may be newer than the store version.

The September Android launch announcement ↗

How the product was introduced to the collector community.

Discogs API documentation ↗

Deadwax integrates with Discogs. It is an independent product, not an endorsed Discogs service.

Implementation and operating records

API handlers, session and collection-sync services, the PI read model and evaluators, release evidence, and the September backlog. File identifiers appear in the relevant sections; the underlying repository is private.

Historical incident measurements are attributed to the incident. Reference-case counts describe the audited dataset. No new performance, accuracy, acquisition-cost, or revenue benchmark is claimed in this paper.