

How I work with AI agents

Damon Young · AI Operator and Senior BI / SQL Developer

damonyoungwork@gmail.com · 562-972-3127 · Remote · [linkedin.com/in/damon-young-bi](https://www.linkedin.com/in/damon-young-bi)

This document is the long version of my resume. It describes the system I built to do my work with AI coding agents: how it's organized, how it keeps the work correct and safe, and what it has produced. System counts were checked against the system itself on October 23, 2026. Client delivery figures come from my project records.

IN SHORT

I spent 15 years as a senior BI and SQL Server developer. Most of that job was making executives trust numbers. In 2026 I rebuilt how I work around AI coding agents. I don't use one chatbot. I run a small team of agents from two vendors: Anthropic's Claude and OpenAI's Codex models. One agent orchestrates each project and delegates to cheaper specialists. Independent agents review each other's work. Every agent works under the same written rules, safety hooks and shared knowledge base, all kept in git.

I've reinvented my work before. I started in sales and finance as a stockbroker and then a financial advisor. I moved into IT as an instructor at a technical school, where students voted me the best teacher, and I taught myself SQL Server development to get my first consulting role. Learning to run AI agents well was the same move again: teach myself the new tool deeply, then use it on real work.

I use this setup for paid SQL and report work, for internal tools and for an iOS game that's now with outside testers. My role is architect, director, reviewer and decision-maker. The agents do most of the typing. I decide what gets built, set the standard it has to meet and check that it meets it.

WHY A BI BACKGROUND MATTERS HERE

AI agents are fast, fluent and sometimes confidently wrong. That's exactly the problem BI developers deal with every day: a number that looks right isn't the same as a number someone verified. My AI system is built on habits from reporting work:

- **Reconcile to the source.** Agent output gets checked against the data, not against how plausible it sounds.
- **Find the real question.** A good brief to an agent starts the same way a good report does, by finding out what the person actually needs.
- **Make it repeatable.** When something goes wrong, the fix becomes a rule, a test or a check, so it can't happen silently again.

1. THE OPERATING MODEL

One orchestrator per project. A Claude Opus agent plans with me, writes the briefs, delegates the work and checks the results. It never hands orchestration to another vendor's model.

Work routed by cost. Coding and code review go to OpenAI Codex models, which saves the more expensive Claude capacity for orchestration. Within Codex:

- one model handles coding and reviews by default
- a stronger one gives high-level second opinions on design and risk

- a light one does bulk mechanical work such as sorting and categorizing

Effort policy. Every model runs at medium reasoning effort by default. High effort costs about twice as much, so it needs a stated reason. Each model has a hard ceiling. A written fallback rule covers a vendor running out of capacity.

Written briefs. Child agents start with none of the conversation's context. Every brief names the files, the goal and what "done" means, and it carries the writing standard with it.

Several machines. The same rules run on a Mac, a Windows laptop and a Linux VM. The Mac orchestrator can read and message agent threads on the Windows laptop through a secured bridge. I used that bridge to have one machine's agent review another machine's release process.

2. QUALITY: REVIEW GATES

Not every change deserves the same scrutiny. A written review skill sets five tiers, and the highest one that applies wins:

Tier	Applies to	Review
Client	Anything a client sees or relies on	A strong model that did not write the work
Release	Anything that reaches users or testers	Two independent reviewers from different vendors. Every confirmed bug is fixed, with no round limit
Ships	Merged or deployed code, scheduled jobs, shared config	One reviewer, with more effort for risky changes
Second opinion	A design or approach before building	A strong reviewer focused on approach and risk
Routine	Small, reversible internal edits	Reread the diff and run the tests

The release tier also has rules for long reviews. From the third round on, reviewers check only the fixes. Each finding's pattern gets swept across the whole codebase, and bug reproductions become committed tests. A written rule decides when an unlikely edge case can be listed as a known limit instead of fixed. A bug that could lose a user's data never qualifies.

Findings from two independent reviews. On one release for external users, the Codex reviewer reported 19 findings (16 rated serious) and the Claude reviewer reported 26 (3 rated serious). They caught largely different kinds of bug, which is why release work gets both.

Other skills support the same goal:

- **blast-radius** traces what a change could break before it's finished.
- **roast** has five independent critics and a judge pressure-test a plan or business idea.
- **show-me-your-work** keeps an append-only decision log for long unattended runs.

3. SAFETY AND GUARDRAILS

A guard hook on the Mac and the Windows laptop. Before an agent runs any tool, a shared pre-tool-use hook checks the action. It blocks reads of credential files. For destructive commands in work folders and against remote machines, Claude Code agents must ask for approval. Codex can't prompt through the hook,

so its version denies those commands outright. The same applies to a command that would overwrite an existing file in a work folder: Claude agents must ask, and the Codex version blocks it. I added that check after benchmarking four small AI models as command classifiers on 100 commands. All four rated > file, which wipes the file, as read-only. So I added an explicit overwrite check with regression tests instead of relying on a model. The hook has 139 automated tests. Services on the Linux VM are isolated separately (section 5).

Client data stays off-limits. Agents may not search for or store a client's customer data. If they ever encounter it, they must stop and ask me.

Data provenance, learned the hard way. An agent once drafted a client report that presented values copied from a spreadsheet as if they came from the live database. I caught it before delivery and added checks to catch it in the future:

- Every report handoff now includes a table of data sources.
- An automated scanner flags hardcoded values in SSRS report files for review.
- An independent review by a model that didn't write the report is required before anything reaches a client.

Approval for anything hard to undo. Deletions, persistent services and new logins need my explicit approval.

4. KNOWLEDGE AND MEMORY

Agents forget everything between sessions unless the knowledge lives somewhere they can read. I built one shared knowledge base that serves every project. It's 288 Markdown pages in git, mirrored to two remotes with one push.

Before building it, I compared Obsidian, vector stores, graph stores, agent-memory products (Mem0, Letta, Basic Memory) and the "LLM wiki" approach. I chose plain Markdown in git, for these reasons:

- **Indexes.** Each area has an index page.
- **Search.** A command-line tool runs full-text search (SQLite FTS5), with semantic search optional.
- **Proposals.** Unverified findings go in a proposals folder until someone checks them.
- **Health check.** A check command finds broken links and stale pages.
- **Plain files.** Every agent can read and edit Markdown, with no special tooling.

I made one deliberate choice: project scopes are labels, not walls. A lesson learned on one project gets found and reused on the next. Memory is layered on top of the knowledge base:

- global rules
- per-project rules
- per-agent memory for preferences
- skills for handing a session over and preparing to compact context

Keeping the always-loaded files small matters. One project's running notes file grew to 112 KB in seven weeks, which cost about 28,000 tokens at the start of every session. I moved its history, word for word, into searchable knowledge base pages and kept only current state in the file.

5. INFRASTRUCTURE

- **Shared configuration.** One repository holds the global rules, nine skills, agent definitions and hooks. Symlinks install it into Claude Code and Codex on Mac and Windows. On Windows, the installer falls back to junctions or copies when symlinks need admin rights.
- **A hardened Linux VM for scheduled agent jobs.** Each service gets its own Linux account, a sandbox and firewall rules. Services are reachable only over a private Tailscale network. I ran a security audit and migration that retired one self-hosted agent and moved its jobs to a dedicated account.
- **Debugging the tools themselves.** A beta release of my agent GUI broke proxied Claude sessions with authentication errors. I traced the cause, built a small relay service as a workaround and filed the upstream bug report. When the maintainer responded, my agent re-tested on the newer build, confirmed the problem no longer reproduces and replied with the evidence.

6. COST AND CAPACITY

AI subscriptions have weekly and five-hour usage limits, and running out mid-project is expensive. So I treat capacity as something to engineer:

- **Routing.** Cheaper models take the bulk work.
- **Monitoring.** A single local service polls usage. It feeds a phone dashboard that shows remaining weekly and five-hour capacity for each vendor, colored green, orange and red.
- **A menu bar app.** A native macOS app shows the same figures. It's about 300 lines of Swift, built in one evening by directing agents, and it replaced a third-party app.
- **A reserve threshold.** The last 20% of weekly capacity is held back for my desktop and phone apps.

7. UNATTENDED AI PIPELINES

- **Daily AI news briefing.** It fetches sources, has a model rank and summarize them under editorial rules I tune, and publishes a private page. A weekly watcher reports changes in a public dataset of how leading practitioners configure their agents. That watcher went through two independent review rounds, which caught two real bugs. The pipeline has 55 passing tests.
- **Other scheduled pipelines** handle data collection and AI triage, plus a channel-digest service with model fallback.
- **Operations.** Daily sign-in checks, health alerts to my phone, and maintenance-window updaters with rollback.

8. WHAT THE SYSTEM HAS PRODUCED

Client SQL and report work. For municipal government clients on Tyler MUNIS and EnerGov, through a consulting firm:

- An 8-hour Crystal report rebuilt in SSRS to run in 12 seconds.
- Eight conversions and fixes, each quoted at 12 hours, delivered in 2 to 4 hours each and production-ready.
- Payroll allocation audits and data extracts under the provenance rules above.

An iOS game with outside testers. A logic-puzzle game in Swift and SwiftUI, with 991 commits so far. Builds 12 and 13 went to about ten TestFlight testers, with 18 levels in six languages. It isn't in the App Store

yet. It follows a staged process: idea, spec, core package, rough playable version, feel, content curve, shell, then release. Along the way: numbered decision records, a hint engine backed by a puzzle solver, localization, and Metal shader effects with measured frame timings. Tester builds go through the two-reviewer release tier. Build 14 is in its seventh review round, with bugs found in every round so far. The automated suite has 573 tests, with no failures.

Internal tools.

- A browser extension that summarizes YouTube videos into sections with clickable timestamps and an accuracy check. Any Claude or Codex model can join the conversation.
- The macOS menu bar app and phone dashboard described above.

Proposal research and a discovery assessment. For a prospective small-business client with strict data-residency requirements, I researched compliant data-processing options for Claude, and found that new Outlook can copy IMAP mail to Microsoft's cloud. I also wrote an unpaid discovery assessment of a nonprofit's systems, which its board praised.

9. MEASURE FIRST, THEN FIX: RECENT EXAMPLES

These are from the past week of work on the game and on the agent system itself.

- **Slow tests.** I suspected the test suite was slowing us down. An agent measured it before changing anything. Full-suite runtime had risen about 9%, but the larger delay came from repeated full runs and 64 UI tests added by mistake, which cost about 49 minutes. New rules: one full run per batch of changes, UI tests only when screens change, and one reused build folder.
- **Lean code.** A cleanup audit removed about 1,430 lines and made the app 5.6 MB smaller. A real save file from an earlier tester build loaded intact after the cleanup.
- **A check that caught bad data.** New puzzle data is rebuilt and compared against what's already committed before it's loaded. This week the check stopped an import because the order didn't match. An agent found the bug, fixed it and added a test so it can't recur unnoticed.

10. A REPORT CONVERSION, STEP BY STEP

How a typical Crystal-to-SSRS conversion runs now:

1. **Understand.** I read the original report and the request. Then I talk to the user about what they actually need, which often isn't the request as first stated.
2. **Specify.** I write the brief: business rules, parameters, expected totals and the tuning standard. The standard means pushing parameters upstream, staging core data in indexed temp tables and avoiding row-by-row logic.
3. **Build.** Agents write the SQL and the report definition. Verified schema facts and rules from earlier work come from the knowledge base.
4. **Verify.** I benchmark the runtime and reconcile totals to the source and to the old report. The scanner checks for hardcoded values. An independent reviewer checks anything the client will see.
5. **Finish.** I polish the layout by hand: headers, footers, parameters shown, page X of Y and branding. Experienced review still catches presentation and edge-case problems that agents miss.
6. **Record.** New verified facts go into the knowledge base, so the next report starts ahead.

11. WHAT I CAN DO FOR A TEAM

The same system can be set up for a team that's starting out with Claude Code or Codex, or one whose agent use has outgrown informal habits:

- **Assess.** Learn how the team works today, where agents already help and where they create risk.
- **Set up.** Shared rules in version control, model routing and effort policy, review tiers, guard hooks, a knowledge base the agents can use, and usage monitoring.
- **Coach.** Remote sessions where the team learns to write briefs, read agent output critically and turn mistakes into rules.
- **Modernize BI with agents.** Report migrations such as Crystal to SSRS or Tableau to Power BI, legacy report rewrites and data-quality audits, delivered faster with the provenance controls clients need.

WORKING PRINCIPLES

Principle	In practice
Rules live in files, not in chat	Every correction becomes a line in a rules file, a skill or a hook, committed to git
Agents check each other	The model that wrote the work never gives the final review on client work. Release work gets two vendors
Fix every real bug	No artificial round limit. Data loss is never acceptable as a known limit
Proportional rigor	Five review tiers. Small reversible edits skip formal review. Client and release work never do
Spend tokens where they matter	Cheap models for bulk work, medium effort by default, live usage monitoring
Own your tools	When a tool doesn't fit, agents build a small replacement in an evening
Security by default	Separate accounts per service, private-network access, credential-blocking hooks, approval for deletions

WHAT THIS IS, AND WHAT IT ISN'T

I operate, integrate and govern AI agents. I don't train models, and I don't have a machine-learning research background. Most of this system was built for my own work, and the agents did much of the hands-on coding. That is the AI operator job: deciding what to build, setting the standard, directing the work and verifying the result.

TOOLS AND TECHNOLOGIES

- **Agents:** Claude Code, Claude Opus and Sonnet, OpenAI Codex CLI, T3 Code, MCP servers, agent skills, pre-tool-use hooks, subagents
- **Built with agents in:** Python, Swift / SwiftUI / AppKit, Metal, PowerShell, Bash, JavaScript, SQLite FTS5
- **Operations:** Linux (systemd, sandboxing, nftables), Caddy, Tailscale, launchd, git and GitHub
- **Data:** SQL Server, T-SQL, SSRS, Crystal Reports, Power BI, Tableau, SSIS