

WHITE PAPER • AI-DRIVEN DELIVERY

Outcome First

Replacing a Commercial EHR in 11 Weeks with an AI-Driven Process

How a two-person team, directing AI agents and measured against the client's own go/no-go criteria, took an AI-built prototype rated 3/10 for go-live readiness to a production electronic health record system.



Executive summary

11 weeks

AI-built prototype to production

3/10 → live

Go-live readiness at the start

16 of 20

Spec modules delivered

686

Merged pull requests, every one reviewed

A healthcare-technology client needed to build its own electronic health record system (EHR) fast enough to retire a licensed commercial EHR before the contract for it renewed. Blue Pumpkin began with that desired business outcome and worked backward from there. The client's written go/no-go criteria defined "done", and every week of work was measured against those criteria, not against lines of code or features shipped.

In 11 weeks (mid-July to late September 2026), a two-man Blue Pumpkin team transformed an AI-built prototype, rated **3 out of 10** for go-live readiness, to a hardened, delivered system. The client now runs it in production on its own infrastructure. A conventional software development plan for the same scope had estimated 18 to 24 months and three development squads.

- **Outcome:** the system is live on the client's infrastructure ahead of its late-October decision on retiring the incumbent EHR.
- **Scope:** 16 of the specification's 20 core modules shipped, plus roles and delegation.
- **Safety:** every blocker on the client's go/no-go list is closed and held closed by a test or a build gate. That includes all 28 findings from the client's own engineering audit.
- **Throughput:** 686 merged pull requests and 6,964 automated tests. 88% of commits were written with an AI coding agent, and a person reviewed every change.

This paper follows the working-backward model the team adopted. It opens with the business outcome and the baseline, then covers the process, why it differs from conventional development, the benefits of the process that was used, how success was measured, and the iterations it took. It closes with lessons learned. The client is not named due to confidentiality requirements.

1. The business outcome.

The client's desired business outcome was not necessarily to "build an EHR." Its desired outcome was to **stop paying for a licensed commercial EHR and run its care programs on a platform it owned**. This transition had to happen in a timeframe that would allow the client to give the

incumbent vendor timely cancellation notice before the contract for the licensed EHR renewed.

The deadline. The incumbent EHR contract required cancellation notice by early November. The client set an operational target of the end of October to fully transition to the new EHR, and a decision gate a week before that to determine if such a transition was proving to be feasible. It was clear which date could move: the September go-live was an internal target, but the contract cancellation date was not.

The decision bar. The client agreed to write, before go-live, a quantitative bar for when the new system had "performed well enough" to cancel the old one, including: minimum days live, incident count, provider adoption, and uptime. The late October decision gate call would be a decision based around agreed criteria, not a debate.

The launch path. The transition plan included producing an MVP by mid-September, then a soft launch with one partner program, expansion to a second, and around October a larger specialty program moved onto the new EHR.

What the product had to deliver. The client's specification set four aims:

- reduce administrative burden;
- remove the "toggle tax" of switching between systems;
- speed up the revenue cycle by creating the claim the moment a visit note is signed;
- stay fully HIPAA compliant.

Where AI had to earn its place. AI was expected to pay off twice:

- **In the product:** An integrated AI voice-to-text physician note drafting functionality cut documentation time for providers while incorporating review and confirmation workflows to enable providers to sign off on every AI-generated note draft for compliance purposes.
- **In delivery:** the client's own plan described this as its first agentically generated, spec-driven product. AI-assisted development had to deliver a production-grade system on a timeline conventional software development methodologies could not meet.

The constraint. The client's plan made July a security and stabilization month, not a feature month, and launch was gated on security, QA, clinical safety, and operational readiness. Speed only counted if the system was safe to run.



2. The baseline: a spec, a prototype, and a readiness score.

Development of the EHR began in late April 2026 as a prototype produced with an AI app-builder platform. In its first week it already had a practice home page, a three-panel patient chart, billing screens, and a command-bar interface. By June it was running on AWS with ICD-10 search, FHIR-backed allergies and medications, and a passing test suite.

The scope for the project came from a master specification, version 1.0 at the project kick-off in late April, followed by version 1.1 in mid-May. The spec defined three subsystems (clinical EHR, practice management, and billing), seven user roles, and 20 core modules, from self-scheduling and intake to e-prescribing, quality measures, e-fax, and a zero-menu command bar. Billing was to be triggered only after a provider signs the visit note. Version 1.1 added a care-coordinator role, provider-by-provider delegated permissions, pre-charting, and AI physician note tools. The spec named .NET, C# and SQL Server as the preferred stack. The Blue Pumpkin team converted this spec to TypeScript and PostgreSQL instead, which suited AI-assisted development and the hosted FHIR store.

The prototype looked close to done. Before building further, the parties measured its performance ability versus the full project requirements. In early July, a go-live readiness review was performed along six parallel parameters: security, HIPAA, infrastructure, data and integrations, operations, and functional completeness.

The review came back with a go-live readiness score of **3/10** for the EHR prototype, with six hard blockers and 15 high-priority findings:

- no production environment and no HTTPS;
- authorization enforced only in the UI, so cross-patient access was possible;
- a live database that was unmanaged, unencrypted and publicly reachable;
- no PHI access audit log;
- fabricated data on live screens, including patient search and the whole patient portal.

At that point, the baseline plan for completing the EHR project by the required deadline was formed up. There was a clear specification, 97,600 lines of prototype code whose demo path worked, a readiness score of 3/10, with an estimate of 18 to 24 months to build the new EHR system using conventional software development methods.

The proposed stack to complete the build was: React 19, Tailwind and shadcn/ui on the front end; Express with tRPC on the back end; PostgreSQL through Drizzle ORM; a hosted FHIR server for clinical data; AWS ECS, RDS, and ALB, managed with Terraform; CI on GitHub Actions.

The heat, for the Blue Pumpkin AI development team, was on.

3. The process: outcome gates outside, an AI build loop inside.

The process had two layers. The client's gated plan defined what had to be true before launch. Inside those gates, Blue Pumpkin's AI-assisted loop produced each change, with evidence that it met the bar.

The outer layer: a gated plan with named sign-offs.

The client's engineering lead wrote an MVP plan running from early July to mid-September, in seven phases:

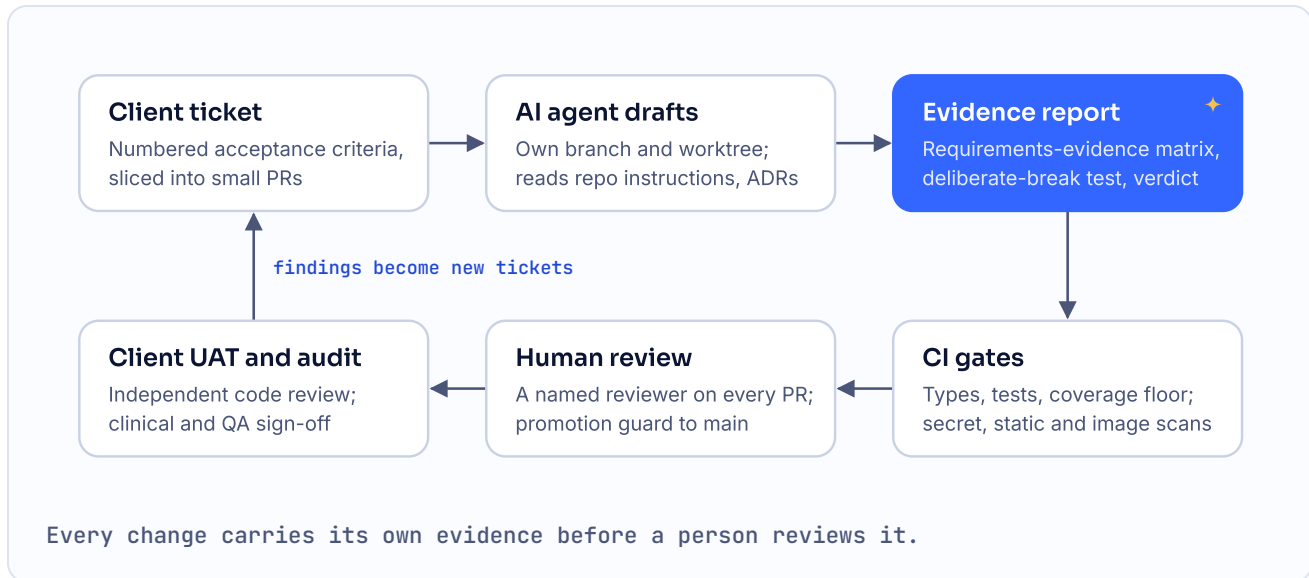
1. environment setup;
2. security gate;
3. physician UAT;
4. scope freeze;
5. engineering and QA review;
6. compliance and operations readiness;
7. final UAT and a go/no-go decision.

In mid-July, the plan was re-baselined around the contract deadline, so the September go-live became a step toward the October decision rather than the finish line.

- **Named sign-offs per discipline.** Product, engineering, QA, DevOps and security, operations, and clinical (the CMO) each owned a gate. Our lead owned overall MVP readiness.
- **Written go/no-go criteria.** These included no anonymous access to patient data, PHI audit logging, HTTPS everywhere, and completed reviews.
- **Scope discipline in one sentence:** "New feature requests do not automatically enter MVP scope." A gap-analysis dashboard from the product owner closed the scope conversation.
- **Acceptance by audit, not by milestone.** The client ran AI-assisted code reviews and end-to-end test passes in July, then a formal 28-finding engineering audit in late August. We answered with evidence reports (see the next section).

The inner layer: AI writes, evidence decides.

An AI agent drafted most of the code. A human decided what shipped, and evidence decided what the human accepted. Each change went through the same path, shown below.



1. Start from a candid, parallel audit. The July readiness review ran as six independent, read-only reviews: security, HIPAA, infrastructure, data and integrations, operations, and functional completeness. Each finding got a severity and an issue. The document was kept as an audit trail, and live tracking moved to one consolidated go-live list.

2. Work one ticket at a time, in an isolated worktree. Each issue got its own branch and git worktree, so several agents could work in parallel without colliding. Large tickets were sliced by acceptance criterion. One scheduling ticket shipped as seven PRs, each titled with the criteria it satisfied (for example "slice 5, AC 5–9").

3. Prove it before anyone reviews it. Every material change came with a pre-review evidence report. It has four parts:

- a requirements-to-evidence matrix;
- the real verification commands and their output;
- a **mutation challenge**: deliberately break the fix and show the new test fails;
- a READY or NOT READY verdict.

Reports also correct the ticket where it was stale: "the issue cites line 641; on the current branch it is line 2179 and already fixed, but nothing pinned it."

4. Let CI hold the line. Twelve GitHub Actions workflows run on every PR and promotion:

- type-checking;
- the full test suite with enforced coverage thresholds;
- secret scanning (gitleaks) and static analysis (Semgrep);
- journey tests and a read-only smoke suite against the deployed environment;
- a promotion guard that blocks unreviewed code from reaching the main branch.

5. Human review, then client audit. A person reviewed every PR. The client's engineers ran independent AI-assisted code reviews, end-to-end test passes, and a formal engineering audit. Findings came back as issues and went through the same loop. The late-August audit produced 28 findings, remediated in a single tracked PR.

6. Write down what the next agent needs. Each repository carries a project instructions file for AI agents, architecture decision records, and runbooks. When a document was found stale, the fix was a commit: one PR corrected "six stale claims" across the instructions, code comments, and the to-do list. Agents are only as good as the context they are given, and stale context produces confidently wrong code.

Guardrails specific to clinical software:

- **One permissions source of truth.** A shared matrix of 8 roles × 38 capabilities is read by both server guards and the UI. Inline role checks are forbidden.
- **Every PHI read is audited.** The audit log also records denied access attempts, per HIPAA §164.312(b).
- **No PHI to an AI model without a BAA.** Note drafting sends PHI only to an LLM endpoint covered by a BAA.
- **"Coming soon" must be honest.** Placeholder screens were either wired to real data or visibly disabled, never left showing mock data.

4. How this differs from conventional software development.

Conventional delivery is organized around output: staff a team, write the code, review the code, test near the end. Adding AI to that model generally just makes the standard steps faster. Industry surveys show the gap this leaves. In one McKinsey survey cited by AWS, 80% of respondents said AI made them more productive, but only 37% saw a positive impact on earnings. To flip the script on making AI usage for the EHR project, the process was changed, itself, by starting with a clear understanding of the desired outcome.

	Conventional plan for this scope	What this project did
Estimate	18–24 months, three squads, an MVP in 6–9 months	Production-ready MVP scope in 11 weeks, with two Blue Pumpkin engineers directing AI agents, plus the client's reviewers.
Starting point	Requirements document, then build from scratch	Specifications developed, then a clickable prototype available within days.
Definition of done	Features complete	The client's go/no-go criteria and ticket acceptance criteria met.
Who writes the code	Engineers write; peers read the diff	AI agents draft; people set direction and check evidence.
Unit of review	The code change	An evidence report: requirements matrix, real test output, and a deliberate-break test.
When quality is checked	A QA phase near the end	Security first, in July; build gates on every change from then on.
Scope change	A change-request cycle	Tickets sliced by acceptance criterion and shipped in days.
Documentation	Written afterward, if at all	Maintained with each change, because it is the agents' working context.
Technology stack	The spec named .NET, C# and SQL Server	TypeScript and PostgreSQL, chosen to suit AI-assisted development.

The common thread is where human attention goes. People stopped spending their hours typing and reading code. Instead they spent them deciding what "done" means, checking evidence, and making the calls only the client could make.

5. The benefits of focusing on the outcome.

Each benefit below is measured against the business outcome, not against activity.

- **Time to outcome.** The system reached production in 11 weeks, against a conventional MVP estimate of 6–9 months. The client goes into its late-October gating decision with a running system and real usage data, not a half-built one.
- **Low cost of change.** Small, sliced changes (686 merged pull requests) kept each fix cheap. All 28 findings from the client's August engineering audit were remediated within a week, and UAT

defects were typically fixed within days.

- **Quality at speed.** The system has 6,964 automated tests, 0.73 lines of test per line of product code, and twelve build gates. Every security blocker found stays closed because a test fails if it reopens.
- **Ownership.** The client installed the code, infrastructure definitions, runbooks and tests on its own infrastructure and moved to production itself. Owning the platform was the point of the project, so this is part of the outcome, not a side effect.
- **AI in the product.** An ambient scribe, AI note drafting through a HIPAA-covered model, and CMS-HCC risk scoring target the documentation and revenue-cycle aims in the client's specification.

What the 11 weeks produced

The application grew from **97,600 to 406,000 lines of TypeScript** in 11 weeks, a factor of 4.2, measured at the integration branch in late September 2026:

Area	What it holds	Files	Lines
Server	API routers, billing engine, integrations, migrations, co-located tests	724	199,733
Client	React pages, components, modals	550	159,073
Test suites	Contract, journey and end-to-end (Playwright) tests	115	24,751
Shared	Permissions matrix, types, date and validation logic	86	14,355
Schema	Database schema and relations	2	6,937
Scripts	Seeding and operations tooling	6	1,147
Total		1,483	405,996

Two splits matter more than the total:

- **Tests versus product.** 829 test files hold 170,400 lines and 6,964 test cases. Production code is 233,200 lines, so there are 0.73 lines of test for every line of product code. CI enforces coverage thresholds on every PR.
- **Written versus kept.** The 11 weeks added 727,000 lines and removed 48,000. Much of the difference is churn: code written, reviewed, and replaced before it shipped.

Around the application sit 1,511 lines of Terraform, 12 CI/CD workflows, 110 Markdown documents (44,500 lines, including architecture decision records, runbooks, and per-change evidence reports), and 131 npm dependencies. The product surface is 153 pages and 128 API router modules.

The largest single files show where complexity lives: the patient chart (9,300 lines), the startup migration (6,800), and settings (6,700). They are also the obvious candidates for refactoring.

6. How success was measured.

Success was measured by the client's own criteria, written before the build began. The primary test was the plan's go/no-go list: the system could not launch unless every item was true. Blue Pumpkin was accountable for eight of the ten; the other two were the client's operational sign-offs.

The client's go/no-go criterion	Evidence	Status
Server-side role authorization is enforced	Capability check on every PHI procedure; tests fail if any guard is removed.	Met
No anonymous PHI access is possible	Authentication required on every PHI route, plus per-patient access scoping.	Met
HTTPS and secure cookies are active	HTTPS-only load balancer; httpOnly session cookies with idle and absolute timeouts.	Met
PHI audit logging is active	Every PHI read and write logged, including denied attempts.	Met
Demo data is removed from production paths	All live screens wired to real data; fake seed metrics removed.	Met
Production secrets are managed securely	Secrets manager plus short-lived OIDC credentials; the app refuses to start without its secrets; secret scanning blocks the build.	Met
Backups and rollback are tested	Automatic rollback on failed deploys; rollback and database runbooks; deletion-protected database.	Met
Engineering and QA reviews are complete	Client engineering audit (28 of 28 findings remediated), independent code reviews, end-to-end test passes.	Met
Administrative and clinical sign-offs are complete	Given by the client when it moved the system to production.	Client-owned
Support process is ready	The client's operations team, using the delivered incident runbooks.	Client-owned

Three secondary measures tracked progress along the way:

- **Readiness baseline.** The 3/10 review's blockers, each closed and pinned by a test (section 7).

- **Acceptance criteria.** Each client ticket carried numbered acceptance criteria, and PRs were titled with the criteria they satisfied.
- **Specification coverage.** Measured module by module against the client's spec, below.

The **measure still ahead** is the one the whole project was built for. At the late-October gate the client will decide whether to retire the incumbent EHR, as determined against the client's pre-determined criteria for days live, number of incidents, provider adoption, and uptime.

Spec versus delivered

16 of the spec's 20 core modules have been delivered, plus roles and delegation. Three additional modules have been delivered in part, and one is deferred until a vendor integration is in place. "Delivered" means the module's core workflow is built, tested, and in production use. Each module also lists refinements planned for later releases.

Spec module	Status	What shipped	Next enhancements
4.1 Patient registration	Delivered	USCDI demographics, multi-policy insurance with card upload, duplicate detection at registration, chart merge, patient tags.	Administrative caption; admin-managed tag list
4.2 Scheduling	Delivered	Day and week calendar, tracking board, cancel, reinstate and reschedule, no-show policy, SMS and email reminders, auto-generated video links, tabbed charts.	Scheduled batch eligibility; copay at check-in
4.3 Self-scheduling and check-in	Delivered	Public booking wizard filtered by state and program, confirmations with video link, automatic intake forms, a coordinator queue for web bookings.	Booking from inside the portal
4.4 Clinical profile	Delivered	FHIR-backed allergies (including drug classes), ICD-10 problems with a billable-code check, RxNorm medications, vitals, CVX vaccinations, history, care team, confidential notes.	Care plans and PHQ-9 trending
4.5 Visit notes	Delivered	Three-panel SOAP editor, pre-charting imports, versioned templates, AI ambient scribe, dictation, vitals with flags, sign-to-claim, coding workbench, amendments.	In-note ordering and care-gap alerts
4.6 Telehealth	Delivered	Zoom or RingCentral per practice, auto-created meetings with a waiting room, start and join from the schedule and the portal, note reminders after each visit.	Video embedded inside the chart

Spec module	Status	What shipped	Next enhancements
4.7 E-prescribing	Delivered	Certified eRx through an embedded, Surescripts-connected vendor: pharmacy search, prescribing, renewals, change requests, fill notifications, EPCS setup, chart sync.	Switch on with production credentials
4.8 Lab and imaging	Partial	Lab results: PDF upload with a deterministic parser and patient-identity check, review and sign-off, trend view.	Electronic ordering through a lab network
4.9 Billing and RCM	Delivered	Live clearinghouse (837P claims, 835 remittance, 277 status), denials and retries, secondary billing, payer rules, real-time eligibility, fee schedules, analytics.	Patient statements and collections
4.10 Patient portal	Delivered	Appointments, reschedule requests, secure messaging, signed notes, medications, photo upload, intake, education; English, Spanish and French.	Lab results; online card payments
4.11 Intake and forms	Delivered	Form builder with 10 field types and e-signature, mobile token-link forms, auto-send by SMS and email, multi-form packets, versioned consent, completion analytics.	Conditional logic
4.12 Automation engine	Partial	Built-in automations: configurable reminders, intake auto-send, note-due reminders.	Admin-defined rules builder
4.13 Reporting	Delivered	Operational dashboard, billing analytics with denial rate, patient-status, communication-log, referral and audit-log reports, CSV export.	A/R aging; PDF export
4.14 Quality and value-based care	Partial	Quality dashboard, care-gap tasks, CMS-HCC V28 risk scoring validated against the official CMS model.	Measures calculated automatically from chart data
4.15 E-fax	Delivered	Inbound and outbound fax, an inbox with unread badge, assign to staff or file to chart, send from the chart, history.	Priority flag
4.16 AI tools	Delivered	Ambient scribe (visit to draft SOAP note, vitals and ICD-10 candidates, clinician confirms), consent reminder, audited audio playback, AI note drafting through a BAA-covered LLM.	AI letter drafting and translation
4.17 Referrals	Delivered	Compose from the chart to a recipient directory, up to 10 attachments, PDF packet preview, send by fax, referral tracking per chart.	Direct messaging; NPI search

Spec module	Status	What shipped	Next enhancements
4.18 Medication fill history	Deferred	Fill notifications for prescriptions sent through eRx.	Full 6-month "Download Fills" history
4.19 Practice settings	Delivered	About 20 settings areas, from providers, templates and calendars to insurance, CPT, clearinghouse and messaging templates; fee schedule, eRx, role and delegation admin.	Workspaces; letterhead
4.20 Command bar	Delivered	⌘K command bar with fuzzy search over patients, ICD-10 and all 13 spec intents, modal overlays, recents, role filtering.	Natural-language commands
Roles and delegation (§3)	Delivered	8 roles × 39 capabilities, editable custom roles per practice, provider-by-provider delegation with dual-identity audit.	Delegation for more document types
Multi-tenancy (§2, §9)	Deferred	Single-practice deployment by design (architecture decision record), with per-patient access control.	Tenant isolation before a second practice

Built beyond the spec:

- **AI and coding:** CMS-HCC risk scoring matched exactly to the official CMS model, and a coding workbench with provider queries.
- **Data exchange:** staged reconciliation of clinical data from inbound faxes, plus third-party record import and FHIR bundle export.
- **Scheduling:** provider calendar sync, and scheduling filtered by program and state licensure.
- **Integrations:** a fax-vendor registry with a test that fails if any vendor is marked implemented when it isn't.

Where it stands

The blockers inside the application are closed: HTTPS, server-side authorization, audit logging, encryption, real data, session security, monitoring and rollback. Each one has a test or CI gate that keeps it closed.

The last launch step is also done. The client took the codebase, installed it on its own infrastructure, and moved it to production. The hardened configuration traveled with the code: the Terraform, CI gates, runbooks and tests described here.

The arc is typical of AI-built prototypes. The demo path works, but the controls a clinical system needs are not visible on screen. Finding those gaps, closing them, and keeping them closed was the first job of the engagement.

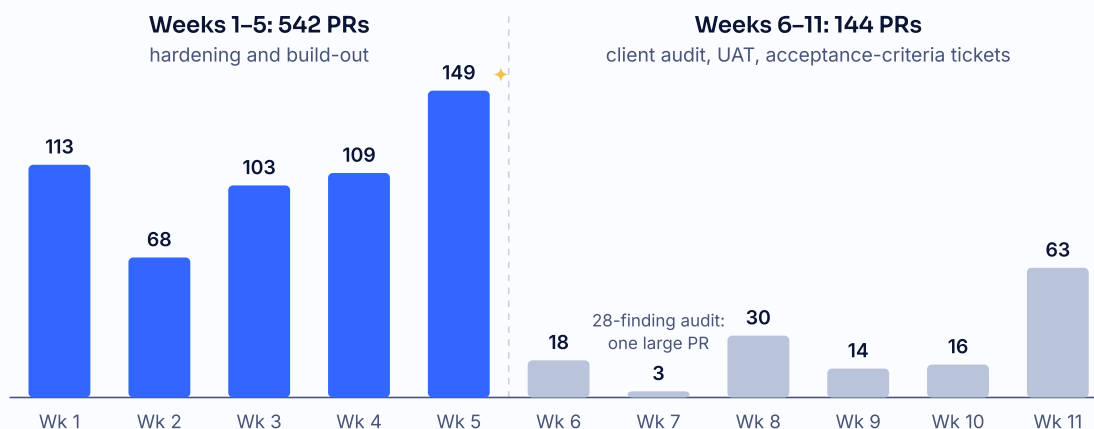
7. The iterations it took

Meeting the measures took five iterations. Each one ended when an independent check passed, not when a feature list ran out.

- 1. Security and stabilization (early to mid-July).** The goal was to close the readiness review's blockers. By mid-July, five of the six hard blockers and all 15 high-priority findings were closed. The sixth blocker, a separate production environment, was the client's to stand up.
- 2. Client review and course correction (mid-July to early August).** Independent code reviews, the client's end-to-end test passes, and a second security pass each produced findings, which went through the same fix-and-verify loop. In mid-July, the plan was re-baselined around the contract deadline.
- 3. Build-out to scope freeze (late July to mid-August).** Feature completion against the specification peaked at 149 merged PRs in a single week.
- 4. Engineering audit and UAT (late August).** All 28 findings from the client's formal audit were remediated, along with the UAT defects.
- 5. Acceptance-criteria backlog (September).** Client tickets were worked one acceptance criterion at a time until the client moved the system to production.

The chart shows the shape: a burst of output while the foundations went in, then fewer, larger, gated changes as the client's reviews took over.

Most PRs merged in the first five weeks; later work was audit- and ticket-driven



Pull requests merged per week on the development repository, mid-July to late September 2026.

Week	Dates (2026)	What shipped
11	Late Sep	Clinical charting depth: structured medication directions (route, form, frequency) that build the sig automatically; allergies to a drug class; problem-list reordering by drag and by keyboard; chart-summary quick-add; technical jargon removed from clinician screens.
8-10	Early to mid-Sep	Client backlog, one acceptance criterion at a time: Quick Book scheduling (7 PRs), duplicate-chart detection and merge, "Log a Call", a public-booking queue coordinators can claim, versioned intake forms, and coordinator-drafted notes that clinicians pick up.
7	Late Aug	Client engineering audit: all 28 findings remediated. UAT defects fixed, including \$0 claims to the payer, a silent bill-pay no-op, temporary passwords that never expired, and the MRN exposed in the URL. Container and dependency CVEs patched.
6	Mid- to late Aug	Code moved into the client's own GitHub organization, and Google SSO restored as a second sign-in route.
5	Mid-Aug	Peak week, 149 PRs: a date and timezone correctness sweep (new hand-rolled date formatters are now banned by a test), report fixes, and the confidential-notes guard extended to chart PDF export.
4	Early Aug	AI note drafting routed through a BAA-covered LLM endpoint; remaining FHIR gaps opened as tracked issues; C-CDA descoped after written client confirmation; live duplicate-contact check.

Week	Dates (2026)	What shipped
3	Late Jul to early Aug	Security hardening: an SSRF-safe inbound MMS pipeline for patient photos, images validated and re-encoded, NPI Luhn validation at the write boundary, and a provably read-only production smoke suite.
2	Late Jul	Client end-to-end review findings fixed: the patient portal shows the patient's real, self-scoped data; denied access attempts are logged; forced password reset is enforced at the API; telehealth visit lifecycle added.
1	Mid-Jul	Five of six go-live blockers closed by mid-July (HTTPS, server-side authorization, an encrypted managed database, a PHI audit log, and no fabricated data); static cloud keys replaced with OIDC federation; an editable, per-practice permissions engine with custom roles.

What the baseline found, and how each item was fixed

Found in early July	Remedy	Closed
No HTTPS; PHI and credentials in cleartext	TLS certificate and an HTTPS-only load balancer that redirects HTTP; database connections now verify TLS certificates.	Early Jul
Live database unmanaged, unencrypted and publicly reachable	Brought under Terraform; encrypted, private, and deletion-protected. A drifted secret that could have repointed the app on the next deploy was removed.	Early Jul
No PHI access audit log	Audit middleware records every PHI read and write, including denied attempts (HIPAA §164.312(b)).	Early Jul
Authorization enforced only in the UI; cross-patient access possible	A capability check on every PHI procedure, and per-patient scoping through one shared access check. Tests fail if any guard is removed.	Mid-Jul
Fabricated data on live screens	Patient search, billing, ICD-10 search and fax wired to real data; fake quality metrics removed from seed data. The portal was moved onto the patient's own records the following week.	Mid-Jul
Weak login and sessions	Fail-closed rate limiting; 15-minute idle and 8-hour absolute timeouts; revocable sessions; password hash no longer sent to the browser; the app refuses to start without its secrets.	Mid-Jul
Long-lived cloud access keys	Replaced with short-lived OIDC federation for CI/CD.	Mid-Jul
No alarms, runbooks or rollback	Error-rate alarms, a deployment circuit breaker that rolls back bad releases automatically, graceful shutdown, and four incident runbooks.	Mid-Jul
Silent failures in billing, reminders and messaging	Claims blocked from signing without a fee, with failed claims tracked for retry; reminders actually send; a Postgres insert bug in messaging and tasks fixed.	Mid-Jul

Found in early July	Remedy	Closed
Container ran as root, base image unpinned	Non-root user, digest-pinned base image, and a health check.	Mid-Jul

Hardening continued after the baseline

Closing the first list was the start. Each later review went through the same find, fix and verify loop:

- **Mid-July, independent review cross-check.** Two external AI code reviews were checked finding by finding against the code. The one critical issue they surfaced, messaging and tasks failing on PostgreSQL, was fixed the same day.
- **Late July, client end-to-end review.** The patient portal was confirmed to show only the patient's own data. Denied-access logging was added, and forced password resets were moved into the API.
- **Early August, second security pass.** The pass produced 13 findings, including safe fetching of inbound photos (SSRF protection), atomic portal account setup, and paginated FHIR exports. Twelve were closed outright; the thirteenth, required status checks, was built.
- **Early August, scanners made blocking.** Secret scanning (gitleaks), static analysis (Semgrep), and a container image scan (Trivy) now fail the build. Rebuilding the production image removed 34 fixable image CVEs. A read-only security smoke suite runs against the live environment on every promotion.
- **Late August, the client's formal engineering audit.** All 28 findings were remediated, along with the defects found in UAT. Container and dependency CVEs were patched.
- **September onward, evidence on every change.** Each PR ships with the evidence report and CI gates described below, so hardening happens by default rather than as a separate phase.

8. Lessons learned

1. **Start from the outcome and the date that cannot move.** Here, the contract deadline set the plan, and the go-live date was a means to it. Framing it that way let the client re-baseline in July without losing sight of the goal.
2. **Get "done" in writing before the build.** A go/no-go list, acceptance criteria on every ticket, and a quantitative bar for the final decision turned reviews into checks rather than debates.
3. **Audit first, in parallel, and write it down.** Six narrow reviews found more than one broad one would have. A written baseline turned "is it ready?" into a list that could be burned down in a

week.

4. **Make evidence the unit of review.** AI agents produce code faster than people can read it. Requirements-to-evidence matrices and deliberate-break tests let reviewers check the claim instead of re-reading the code, and they caught tickets that were already stale.
5. **Put guardrails in code, not in memory.** A single permissions matrix, a test that bans ad-hoc date formatting, a promotion guard, and secret scanning hold the line on every one of hundreds of PRs, whoever or whatever wrote them.
6. **Keep agent context current.** Instructions files, architecture decision records, and runbooks are inputs to the agent. Stale documentation produced confidently wrong changes more than once. Correcting it is real work and belongs in the normal flow of changes.
7. **Slice client tickets by acceptance criterion.** Small PRs titled with the criteria they satisfy made client UAT fast and made regressions easy to trace.
8. **Let the paper follow the work.** A client-written plan with named gates governed the build well. When scope pivots, though, write the SOW or change order to match, so both parties' obligations stay clear.

What comes next: the system is in production on the client's own infrastructure. Next is switching on e-prescribing, which is built and awaiting production credentials. Electronic lab ordering, the automation engine, and multi-practice tenancy are the largest deferred items.

Appendix: how the engagement was contracted

The contract framework was conventional. What worked in practice was not. The paper said scope, hours, and change orders. The work was actually governed by phase gates, named sign-offs, and repeated independent review.

The contract stack

- 1. Master Services Agreement (September 2025).** Services exist only through SOWs signed by both parties. Each SOW must state specs, milestones, rates, timeframe, fees, and deliverables. Billing is monthly in arrears. Acceptance means work meets the specs "to the client's reasonable satisfaction", and non-conforming work is redone at no charge. All work product belongs to the client.
- 2. Phase 1 SOW (April 2026).** It covered a patient portal with SMS one-time-code login and AI-assisted intake, a care-coordinator app, an EHR integration layer, and email and alert services. Scope came straight from the client's PRD.
 - **Estimating:** each feature was T-shirt sized, from XS (4 hours) to XL (32 hours). That gave 1,000 engineering hours, plus 10% each for buffer, project management, and QA, plus 25 hours of design: 1,325 hours in total.
 - **Guardrails:** explicit exclusions, success metrics, 10 open questions each with an owner, a BAA before any PHI access, and a rule that no out-of-scope work starts without a signed change order.
- 3. The EHR execution plan (July 2026).** When the client's priority moved to a full EHR, the client's engineering lead wrote an MVP execution plan rather than a new SOW. It ran from early July to mid-September in seven phases: environment setup, security gate, physician UAT, scope freeze, engineering and QA review, compliance and operations readiness, and final UAT with a go/no-go decision.

What we would paper differently. The pivot from portal to EHR was large enough to deserve its own SOW or change order, and the SOW should state rates and totals as the MSA requires. The execution plan governed the work well. But an execution plan is not a contract, and both sides are better off when the paper matches the work.



Blue Pumpkin

Outcome-first, AI-driven software delivery. Start with the result your business needs; we work backward from there.

`<mybluepumpkin.com />`