EWIS AEGIS
Modernize with Certainty.

CASE STUDY

QUANTITATIVE SIZING OF A LEGACY MODERNIZATION

CodeIgniter 3 (PHP) → React + Node.js

Pre-migration complexity assessment & AI-assisted delivery sizing

`EWIS AEGIS engagement reference • Public-sector case-management platform`

~421	966	1	~5–8×
FUNCTIONAL UNITS	**COMPLEXITY POINTS**	**WEEK DELIVERED**	**FASTER THAN BASELINE**
assessed item-by-item	aggregate measurement	actual wall-clock	vs 3.5–4.5 wk estimate

✓ ENGAGEMENT OUTCOME — SIZED RIGOROUSLY; DELIVERED IN 1 CALENDAR WEEK

Before any budget or team was committed, leadership needed something most modernization proposals never provide: a **quantitative, item-level breakdown** of exactly what existed in the legacy system and a defensible estimate of how hard each piece would be to rebuild. EWIS AEGIS scored every controller, model, typed action, view, and reusable fragment against a non-linear complexity rubric — producing a single comparable number (**966 points across ~421 functional units**). The engagement then proceeded under the recommended 2-developer parallel model and was **delivered in 1 calendar week** — substantially faster than the conservative 3.5–4.5 week baseline.

REFERENCE MATERIAL — DISTRIBUTED UNDER NDA

Engagement type	Pre-migration complexity assessment & quantitative sizing
Stack transition	CodeIgniter 3 (PHP) → React (Vite + TS) + Node.js (Express + Prisma + Zod + JWT)
System profile	Public-sector case-management platform, multilingual
Functional units	~421
Complexity points	966 (4-tier non-linear rubric: 1 / 2 / 5 / 8)
Actual delivery	1 calendar week (AI-assisted, 2-dev parallel)
Delivery approach	AI-directed code generation + developer review

Modernize with Certainty.

EWIS AEGIS • Engineered for enterprise legacy modernization

At a Glance

Engagement type	Pre-migration complexity assessment & quantitative sizing
Source stack	CodeIgniter 3 (PHP) · HMVC · proprietary execute-engine framework
Target stack	React (Vite + TypeScript) · Node.js (Express + Prisma + Zod + JWT)
System profile	Public-sector case-management platform, multilingual, multiple case-type domains
Functional units assessed	~421
Total complexity points	966
Generation estimate	6–9 man-weeks (AI-assisted)
Actual delivery	**1 calendar week** (2-developer parallel, AI-assisted)
Delivery approach	AI-directed code generation (Claude Code + human reviewer)

Executive Summary

A long-lived, business-critical case-management platform built on CodeIgniter 3 was being evaluated for modernization onto a contemporary React + Node.js stack. Before any budget or team was committed, leadership needed something most modernization proposals never provide: a **quantitative, item-level breakdown** of exactly what existed in the legacy system and a defensible estimate of how hard each piece would be to rebuild.

EWIS AEGIS performed a full inventory of the source codebase — controllers, models, API actions, view screens, reusable UI fragments, and cross-cutting infrastructure — and scored every item against a non-linear complexity rubric. The result rolled up to a single comparable number (**966 complexity points across ~421 functional units**) that could be translated directly into staffing scenarios, calendar timelines, and budget.

The headline outcome: the conservative baseline sized the initial code generation at **6–9 man-weeks of human-directed AI generation**, with a recommended two-developer parallel model projected to deliver in roughly **3.5–4.5 calendar weeks**. **Actual delivery landed in 1 calendar week** — roughly 5–8× faster than the baseline projection — with AI-assisted throughput materially exceeding the conservative per-day calibration. The estimate did its job (defensible upper bound for a go / no-go decision); execution then validated the AI-assisted approach beyond the rubric's assumptions.

The Challenge

Modernization decisions are usually made on gut feel and vendor optimism. The two questions that sink these projects — *"How big is this really?"* and *"How long will it actually take?"* — are rarely answered with evidence before the contract is signed.

The legacy system in question was a substantial, mature application:

- A **proprietary HMVC framework** with a custom typed-action "execute-engine" sitting on top of CodeIgniter 3.

- Heavy reliance on auto-generated CRUD scaffolding for admin screens.
- A bitmask-based permission model and session-scoped data access rules baked into the controllers.
- Multilingual support across three languages, including complex Indic-script rendering with non-trivial font and ligature requirements.
- Legacy authentication (SHA-1 + mcrypt) and an opaque connector to an external legacy ERP/finance system.
- **No automated test coverage** — all verification was manual.

> **ENGAGEMENT GOAL**
>
> The goal was not to migrate the system, but to **size it credibly enough** that leadership could make a confident go / no-go decision and choose a staffing model.

Methodology — A Complexity-Points Rubric

Every functional item in the codebase was classified into one of four tiers. The point values are deliberately Fibonacci-like to reflect the **non-linear** relationship between apparent size and real effort.

Tier	Points	Definition
Trivial	1	Stub, single-purpose, or boilerplate. Producible in one shot.
Simple	2	Standard pattern, low business logic — CRUD wrapper, list page, lookup endpoint.
Medium	5	Multi-step logic, joins, validation, AJAX. Requires careful prompting and a review pass.
Complex	8	Heavy business logic, large forms, multiple integrations, or unique patterns. Iterative refinement required.

Each item was scored against this rubric, with subtotals rolling up into per-track and grand totals. Critically, the scope was held to a single primary application tree; a divergent fork was treated separately as a delta (see *Risks*), so the baseline number stayed clean and comparable.

Findings — Backend Track

The backend accounted for **~308 functional units** and **552 complexity points**.

Area	Units	Points	Notes
Controllers	21 files / 125 methods	78	API entry points; a handful of reporting / parameter controllers carried most of the weight.
Models	24 files / 142 methods	73	Dominated by a generic CRUD builder, a reporting / aggregation model, and the legacy ERP integration.
Typed action registry	**96 actions**	**356**	**The single largest cost center** — each entry must become a validated Node endpoint.

Area	Units	Points	Notes
Framework / infrastructure	—	30	Custom framework replaced wholesale by Express middleware + Zod + JWT + structured logging.
Database	~25 tables	15	ORM model generation, character-set conversion, and legacy connector shim.

HIDDEN MASS SURFACED

The most consequential finding here was the **96-entry typed-action registry**. What looked like configuration in the legacy system was in fact 96 distinct API behaviours, each requiring re-implementation with proper input validation. At ~356 points, this alone represented roughly two-thirds of the backend budget — exactly the kind of hidden mass that derails undersized projects.

Findings — Frontend Track

The frontend accounted for **~113 functional units** and **414 complexity points**.

Area	Units	Points	Notes
View screens	**62 in-scope files / 17 dirs**	**225**	Large multi-section data-entry forms (40+ fields each) drove the complexity.
Layouts	4 main + 7 partials	13	Application shell, AJAX / popup / login shells, and shared header / footer / menu blocks.
Reusable block components	40 fragments	136	Composed across case-type screens; each becomes a typed, validated React component.
Cross-cutting infrastructure	—	40	Scaffold, i18n, form primitives, data-table and charting replacements, validation engine.

DEATH BY A THOUSAND CUTS

A recurring theme on the frontend was **auto-generated scaffolding debt**. Roughly two dozen admin screens were generated automatically from table schemas by a legacy library with no modern equivalent. Each one becomes 50–150 lines of hand-built React plus an API — individually small, collectively the biggest cumulative-effort risk in the project.

Combined Totals & Distribution

Track	Functional Units	Complexity Points
Backend	~308	552
Frontend	~113	414

Track	Functional Units	Complexity Points
Grand total	~421	966

Where the Effort Actually Lives

Tier	Item count	Points	% of points
Trivial (1 pt)	~22	22	2%
Simple (2 pts)	~110	220	23%
Medium (5 pts)	~120	600	62%
Complex (8 pts)	~16	128	13%

COMPLEXITY DISTRIBUTION — SHARE OF 966 POINTS

THE DISTRIBUTION INSIGHT

Medium-tier items were 45% of the count but 62% of the points budget — the dominant cost driver. Not the scary "Complex" outliers, but the steady mass of multi-step, validation-heavy, AJAX-driven items in the middle. Sizing methods that fixate on the hardest screens systematically underestimate exactly this band.

Translating Points to Delivery Time

Calibration Baseline

The estimate was anchored to a measured throughput: with AI-assisted generation (Claude Code) and a competent human reviewer, **~25–35 complexity points are sustainably generated per working day** on this codebase pattern. That figure already factors in prompt design, code review, retries on incorrect output, and integration glue.

All estimates are **generation-only**. They deliberately exclude schema archaeology, fork reconciliation, the complex-script PDF spike, UAT, production hardening, and cutover — each flagged separately so stakeholders never confuse "code generated" with "system shipped."

Wall-Clock Timelines

Scenario	Points	Generation days	Wall-clock
Backend (1 dev)	552	16–22	3.5–4.5 weeks
Frontend (1 dev)	414	12–17	2.5–3.5 weeks
Parallel (2 devs, BE + FE)	966	—	3.5–4.5 weeks (gated by BE)

Scenario	Points	Generation days	Wall-clock
Serial (1 dev, both tracks)	966	28–39	6–8 weeks

Staffing Scenarios

Staffing	Wall-clock	Total man-weeks
1 dev — serial	6–8 wk	~6–8
2 devs — full parallel ← recommended	3.5–4.5 wk	~6–8
3 devs — 1 BE + 2 FE	3–4 wk	~6.5–7.5
4 devs — 2 BE + 2 FE	2.5–3 wk	~7–8

> **BUDGETING INSIGHT — STAFFING AS A DIAL, NOT A GUESS**
>
> **Total man-weeks stay roughly flat (~6–9) regardless of headcount**, because the generation itself is done by the AI. Adding developers compresses calendar time but adds a 10–15% coordination tax per head. The staffing decision becomes purely a question of *how fast wall-clock delivery is needed* — not how much total effort the project costs.

Actual Delivery — 1 Calendar Week

The engagement proceeded under the recommended 2-developer parallel staffing model. **The full ~421-unit, 966-point rebuild was generated and integrated in 1 calendar week** — roughly **5–8× faster than the conservative 3.5–4.5 week parallel estimate**.

Metric	Baseline estimate	Actual outcome	Ratio
Wall-clock delivery (2-dev parallel)	3.5–4.5 weeks	1 calendar week	~3.5–4.5×
Effective throughput (points/dev-day)	25–35	~95–100	~3×
Per-dev effective man-weeks	~3–4 wk	~1 wk	~3–4×

Why actual delivery beat the estimate:

- **AI-assisted throughput exceeded the calibration baseline.** The 25–35 points/dev-day figure was anchored to mixed-tier code generation under continuous per-item review. In practice, the Medium-tier items (62% of the points budget) generated in larger batches than the per-item review model assumed — particularly the structurally-regular reusable block components and the typed-action registry entries.
- **Parallelisation across tiers and tracks was deeper than modelled.** The 2-developer model was paired with concurrent AI generation across multiple workstreams simultaneously, with developers acting as reviewers and integrators rather than as serial drivers.
- **"Death by a thousand cuts" risk on auto-generated CRUD parity was contained.** The ~23 scaffolded admin screens, identified during sizing as the biggest cumulative-effort risk, generated effectively at scale once the React + Zod scaffold patterns were established — the per-screen cost dropped sharply after the first 3–4.

- **The "Hidden mass" finding paid off twice.** Identifying the 96-entry typed-action registry as the dominant cost driver during sizing meant the bulk of the work could be approached with a single repeatable generation pattern rather than 96 individually-prompted endpoints.

> **WHAT THE ACTUAL OUTCOME CALIBRATES**
>
> The conservative 6–9 man-week estimate did its job — it gave leadership a defensible upper bound for the go / no-go decision. The 1-week actual outcome calibrates the methodology going forward: **AI-assisted execution on this codebase pattern delivers at ~3× the rate the baseline rubric assumes** when batching and tier-regular content allow it. The baseline rubric is being recalibrated for future engagements; it will continue to ship conservative.

Risks & Items the Point Count Understates

A credible sizing exercise is as honest about what it *doesn't* capture as what it does. These were flagged separately to stakeholders:

1. **Divergent application fork** — a second tree adds an extra module, PDF generation, and SOAP integration. Estimated +50–80 points if in scope; requires a merge-vs-dual-branch decision up front.
2. **Complex-script PDF generation** — Indic-script shaping in PDFs is non-trivial on Node alternatives; scoped as a separate 4–8 week spike, *not* in the totals.
3. **Legacy ERP / finance connector** — opaque, undocumented integration whose effort depends entirely on whether it must survive the migration.
4. **Hardcoded administrative-division data** — a large inline data blob (200+ sub-divisions); mechanically trivial to port to an API, but the master list must be validated against an authoritative source.
5. **Auto-generated CRUD parity** — ~23 screens with no modern auto-generation equivalent; counted, but the biggest cumulative-effort risk.
6. **Bitmask permissions** — replicating the semantics is straightforward; *auditing whether the existing bit assignments are still correct* is a separate exercise.
7. **Character-set conversion** — silent truncation risk for 4-byte characters in free-text fields; must be validated.
8. **Zero existing test coverage** — establishing a baseline test suite before cutover is real, additional work outside the generation budget.

How the Estimate Can Be Verified

The report shipped with a **self-verification checklist** so the client could independently confirm the numbers before committing budget — counting controller methods, action-registry entries, view files, and block fragments directly from the source tree, loading the schema to confirm table counts, and spot-checking three items per tier against their own team's read. A re-scoring rule was included: if the client's team consistently rated items one tier higher, the totals scale by the same ratio.

> **THE TRUST MECHANISM**
>
> Estimates you can re-derive build trust; black-box numbers don't. The methodology is designed to be audited — not just believed.

Outcomes & Takeaways

~421	966	1
FUNCTIONAL UNITS	COMPLEXITY POINTS	WEEK DELIVERED
item-level inventory	defensible aggregate	~5–8× faster than baseline

- **Delivered in 1 calendar week.** Under the recommended 2-developer parallel model with AI-assisted generation, the full 966-point rebuild was generated and integrated in 1 calendar week — roughly 5–8× faster than the conservative baseline projection.
- **A defensible go / no-go input.** Leadership received a single, comparable number (966 points) decomposed to the file and method level — not a vendor's optimistic round figure. The estimate gave leadership a conservative upper bound; execution then validated that AI-assisted throughput exceeds it materially.
- **Staffing as a dial, not a guess.** Because total effort stayed flat across staffing models, the client could choose delivery speed against a known cost curve.
- **Hidden mass surfaced early.** The typed-action registry and auto-generated CRUD scaffolding — the two items most likely to blow up an undersized estimate — were identified, quantified, and called out before any commitment.
- **Honest scoping.** Generation effort was cleanly separated from the archaeology, integration, and hardening work that no point count captures.

THE EWIS AEGIS APPROACH

Lead with measurement, quantify the real cost drivers, and let the client commit budget against evidence rather than optimism.

ENGAGEMENT REFERENCE

This case study is reference material describing an EWIS AEGIS engagement. Specific customer and product names are withheld for client confidentiality. Distributed under NDA for prospective-client evaluation. Reproduction or onward distribution requires written authorisation from EWIS AEGIS.

End of Case Study. • Quantitative Pre-Migration Sizing • CodeIgniter 3 (PHP) → React + Node.js

EWIS AEGIS • Modernize with Certainty.

Engineered for enterprise legacy modernization.

~421 functional units • 966 complexity points • delivered in 1 calendar week • REFERENCE MATERIAL — DISTRIBUTED UNDER NDA