SPRING POINT SOLUTIONS · TECH BLOG · ISSUE 03

How We Actually Build Software Now

Inside the engineering process behind the modernization, and the delivery data that proves it's working.

Date: September 2026 **Topics:** Engineering · AI · Process · Architecture **Read time:** ~7 min

IN THIS ISSUE

01 · THE BIG PICTURE

"AI-Centric" Is a Claim Everyone Is Making Right Now. Here Is What We Mean by It.

Every software vendor has an AI story this year. Most of them are the same story: a chatbot bolted onto an existing product, announced in a press release, with no detail on how it was built or what happens when it's wrong. That is not what this issue is about.

In January 2026, we made a deliberate decision that most of our peers have not: AI writes 100% of our code. Not a chatbot suggesting a line here and there while an engineer still types the rest. Everything that ships from our team starts as AI output. We didn't rebrand what we were already doing. We changed it, and we have been running that new process for about a year.

That claim only means anything next to the one that has to sit right beside it: a documented, human-reviewed process governs every line before it reaches your system. This issue is about that process, not a product. The software you rely on gets built faster and more reliably when the process behind it is disciplined, not when a vendor simply puts AI in the description.

THE DISTINCTION THAT MATTERS

AI-Centric Does Not Mean AI Decides What to Build

Plenty of ERP vendors will tell you their engineers use AI. Ask them what percentage of their shipped code an AI model actually wrote, and watch the answer get vague fast. Ours doesn't: all of it does, and that is the reckless-sounding part of this story until you look at what it runs through before it reaches you. AI-centric does not mean AI decides what to build. It means the typing got faster while the judgment stayed put.

> *"We don't do vibe coding. We write the spec, AI writes the code, and a human still checks the work before it ever reaches you."*
>
> *Spring Point Solutions Engineering Team*

The rest of this issue backs up that sentence: what the process actually looks like, and what it produced when we measured it.

02 · OWNERS & OPERATORS

What This Means for Your Shop

Plain English, no process jargon

You don't need to know how a truck's engine is timed to trust that it starts every morning. You don't need to know how we build software to benefit from how we build it either. But you've been told for years that we run on old technology, so it's fair to ask what's actually changed underneath the hood, and whether "AI wrote it" should make you more comfortable or less.

Here is the honest answer: AI writes the first draft of the code. It does not decide what ships.

Someone still defines exactly what a feature needs to do. Before it goes anywhere near your data, automated tests run against it and an engineer reviews the result against that definition. AI changed who does the typing, not who is accountable for the result.

WHY YOU SHOULD CARE

Faster, Without Lowering the Bar

Because it's why things are moving faster. Building software the traditional way (one person hand-writing every line) is slow, and slowness is the single biggest complaint we hear about how quickly requests get addressed. The process described in this issue is how we're closing that gap without lowering our own bar for what's allowed to reach production.

A simple report and a complex one were never the same job, with or without AI, so we're not going to claim every future request now takes a day. What we can tell you honestly is that the same category of work is measurably faster than it used to be, and Section 03 shows the actual numbers behind that claim.

WHAT HAS NOT CHANGED

Your Data. Your Access Controls. Human Sign-Off on Everything.

Your data is handled the same way it always has been. This issue is about how we write code, not about what our engineers or our AI tools can see in your live system. Nothing here changes the access model.

Every line of AI output goes through the same testing and review standard that governed releases before AI was part of how we write code. A human reviews it before anything ships.

THE ENGINE ITSELF

A New Engine, Not a Faster Old One

Here's the part worth saying plainly, because it's the part some of you have wondered about for years without asking directly: is the truck getting a faster old engine, or an actual new one?

A new one. Everything we build under this process is written in modern, containerized C#, running in the cloud on AWS, on the same kind of scalable architecture you'd expect from a company founded today rather than one adapting a 25-year-old codebase.

We still have work to finish. Issue 01 was upfront that some MotorBase screens run on our older stack during the transition, and we're not going to pretend otherwise. But new development does not happen on MS Access, and it does not happen on the legacy technology that earned us the reputation some of you have carried for years. That reputation was fair once. It describes less and less of what we actually ship every month, and everything we build from here forward makes it describe even less.

03 · TECHNICAL OPERATIONS

How a Feature Actually Gets Built

The pipeline: specs, scaffolding, validation loops, architecture, and the evidence

WHY UNSTRUCTURED AI CODING DOES NOT WORK FOR AN ERP

We Learned This the Hard Way

Handing an AI model a vague instruction and letting it write production code against a 25-year-old business system is a bad idea, for a predictable reason. The model has no ground truth for

decades of embedded business logic that exists only in old code and in a few people's heads. Without a specification to check itself against, the model will confidently produce output that looks correct and isn't. There's a name for that error mode: a hallucination.

We hit this directly in year one. Some of our earliest AI-assisted development (before we had the process described below in place) required heavy manual correction. One of our most complex financial reports took several months to complete because we were still learning how to give the model enough grounding to get it right. That experience is why the process below exists.

THE PIPELINE

Five Steps, Used the Same Way for Every Feature

1

SME documents the ground truth

A subject matter expert documents how it actually works today: the tables, the calculations, the edge cases, and a sample of the expected output. This is what the rest of the pipeline checks itself against.

2

AI converts that into a machine-readable specification

We use YAML: a structured, explicit format that leaves far less room for ambiguity than prose. A recent specification for one of our more complex reports ran over 1,500 lines. That level of detail is the point, not a byproduct.

3

Scaffolding, not a blank page

New code is generated from templates that already match our application architecture (the same layers, patterns, and conventions as everything else in the codebase). AI isn't inventing structure here. It's filling in a structure our engineers already established and reviewed.

4

A generation-and-validation loop, not a single shot

The model writes the code, then a build runs, then automated tests run, then the output is checked against the actual database. If any step fails, the model revises and the loop runs again, automatically, before a human is even looking at it.

5

A human reviews every result

> Architecture, correctness, test coverage, and for the SME who requested it, whether the output actually matches what they asked for. Nothing merges without that sign-off.

Speed came from cutting out the manual, repetitive part of the job: steps 2 through 4 running in a loop instead of a person typing every line by hand. Steps 1 and 5 stayed exactly where they were.

ARCHITECTURE

High-Level Stack

New services are built on a standard, layered architecture (Clean Architecture, with CQRS to separate reads from writes) that keeps business logic, data access, and presentation cleanly separated. That structure is what makes the scaffolding step possible: AI-generated code has a well-defined place to go, and it's straightforward to verify that it stayed there.

We run automated architecture-conformance analysis (NDepend) as part of our review process, and we maintain thorough automated test coverage (unit and integration) on everything that ships. The stack: C#, containerized, deployed to Kubernetes on AWS. Not MS Access. Not the ASP.NET MVC pattern some older interfaces still run on during the transition. The pipeline described in this issue generates into the same modern stack Issue 01 described.

THE EVIDENCE

What Changed When We Measured It

The following is a sample from one specific area (financial reports) covering roughly the first half of 2026, before we officially put this full process in place. Five examples, chosen because they show the trend clearly.

REPORT	RELATIVE COMPLEXITY	TIME TO COMPLETE
Profit & Loss	Very high — new architecture, hierarchical accounts, on-the-fly calculations	**~5 months**
Balance Sheet	Very high — similar complexity to P&L	**21 days**

REPORT	RELATIVE COMPLEXITY	TIME TO COMPLETE
Trial Balance	Medium-high — related to Balance Sheet	**8 days**
AR Aging	Medium	**5 days**
AP Aging	Medium	**1–2 days**

A few honest caveats before drawing the wrong conclusion. Complexity varies widely — this is not a benchmark for any given report, and upcoming work will sometimes take longer; that's expected, not a regression. It's also a sample from one area of a larger modernization, not a scoreboard of everything completed.

The trend is the point. The Profit & Loss report was the first one we attempted with this level of rigor, and it showed. Every report after it got faster, because the specification and templates matured as the underlying AI models improved substantially over the same year. Both factors mattered.

WHAT COMES NEXT

The Same Process Powers Everything We're Building

This same pipeline (specification first, AI-generated code, automated validation, human review) is what powers the WebMB modernization work described in Issue 01 and the Beacon assistant introduced in Issue 02. It is also the reason we can commit to that work continuing at pace rather than stalling the way a traditional, hand-coded rebuild of a 25-year-old system would.

We use an internal platform called **Spring Point Fabric** to handle the build and deployment side of this pipeline once code is written. That's a deep enough topic to deserve its own issue rather than a paragraph here.

Being AI-centric was never the point. What it produced is: every line of code you now run started as AI output, checked by a documented process before a human let it anywhere near your data. That combination is why the work in this issue moved as fast as it did. That's not a faster version

of the shop you've known for 25 years. It's a different engine, doing the same job better, and the data in this issue is how we're holding ourselves to that claim, issue by issue.

Questions about how our process works? We're happy to walk you through it.

Email us at support@springpt.com · Call 888-382-7835

Spring Point Solutions · Technology Team · September 2026