C&R Software
Credit Risk Management

White Paper

Why build it yourself sounds better than it works

It's a prototype trap.

You've seen the LinkedIn posts. Someone with little to no coding background spins up a working prototype in two hours using Cursor or Claude. They demo it at a stand up. Everyone nods. Someone says, "why are we paying six figures for a vendor again?"

Fair question. Wrong context.

The problem is that debt collection, regarding actual, at scale, enterprise debt collection, isn't the same as building a calculator or a Slack bot. It looks like it should be. It feels like it should be. But then you hit a Tuesday in March and discover why your prototype was actually a land mine.

The four-hour trap

Let's be specific. A smart analyst on your team with no coding background just built a dialer optimization tool in four hours. It segments accounts by propensity, prioritizes by expected recovery, and routes calls to the right agent pool. It works. In your test run, it's clearly better than your current rotation logic.

So far, so good.

Now scale it. Your portfolio is 2.3 million accounts. The tool was built on a clean data dump from last week. But your real system has:

- Accounts that move between portfolios three times a day
- Disputes that flip status in real time
- Partner agencies that refresh data on different schedules
- Accounts flagged for hardship that your prototype doesn't even know exist yet
- Regulatory holds, bankruptcy flags, and state specific rules that look invisible until you miss one

Your prototype sat on clean data. Your production system is a living, breathing mess of real world chaos.

The analyst who built it in four hours? Now they own it. Forever. Until it breaks. Which it will, probably during your busiest collection period.

The compliance surprise

Here's the thing nobody mentions in the LinkedIn posts: collections has rules. Lots of them. FDCPA, TCPA, FCRA, UDAAP, state debt collection laws, banking regulations, and increasingly, AI governance frameworks that even regulators are still figuring out.

Your prototype probably doesn't think about any of this. It was built to solve a business problem, not a legal one.

An actual case: A lender built an internal AI system to optimize call timing. Technically elegant. Great at finding windows when customers were most likely to answer. Then their compliance team discovered it was systematically calling certain ZIP codes more frequently than others. In a pattern that happened to correlate with race.

They never intended to discriminate. The model didn't "know" about race. But the algorithm was optimizing for a variable that acted as a proxy, and now they're in a situation where they need to prove the discrimination was coincidental. That's not a fun audit conversation.

A purpose built collections platform has thousands of hours baked into bias detection, disparate impact testing, contact rule enforcement, and model governance. Your prototype has none of this. Not because your team is careless, more because they didn't know they needed it.

The data plumbing problem

Debt collection AI only works if it has clean, unified data. Core system data. Servicing data. Dialer data. Payment history. Bureau data. Dispute data. Customer channel preferences. Consent records. Hardship flags. Complaint history. Your prototype? It probably ran on a CSV export.

Real integration means:

- Nightly batch processes that handle time zones, data refresh conflicts, and the inevitable situations where two systems disagree about the truth
- Circuit breakers that stop the AI from using stale data
- Audit trails that show exactly which data version drove which decision
- Real time syncs with payment systems so customers don't get contacted after they just paid
- Failover logic when one data source goes down

None of this is optional. All of it's tedious. And all of it takes months to get right.

A vendor platform has already solved these problems for thousands of portfolios. Your team is solving them for the first time, on their own time, while also trying to run collections.

Your prototype was built to solve a business problem, not a legal one.

The audit disaster you didn't plan for

Let's say your prototype works great for six months. Your cure rates are up. Everyone's happy. Then the examiners come in.

They ask: "Show me the model. How was it developed? What testing did you do? What's your bias assessment? Who approves changes? How do you know it's not violating TCPA contact rules? Where's your documentation?"

And your analyst pulls up a folder of Jupyter notebooks and a Slack thread where someone said, "Yeah, we tweaked the scoring logic last month."

That's the moment you realize your prototype wasn't just a tool. It's technical debt you didn't know you were accumulating.

A platform vendor has compliance documentation, model cards, approval workflows, and version control built in. Not because they're better at software engineering, but because regulators require it.

Your team now gets to retrofit all of that, retroactively, into a system they never designed with governance in mind.

The real cost of DIY

So what does this actually cost?

The four hour prototype? That's real. The next 2,000 hours of integration, testing, hardening, compliance documentation, and bug-fixing? That's the invisible part.

We've seen this pattern at dozens of enterprises:

Month 1–2: Prototype works, team is excited, ROI looks amazing.

Month 3–4: Data integration issues surface. Unexpected edge cases. The analyst is now spending half their time on production support instead of strategy.

Month 5–6: Compliance team asks questions. Someone realizes the model doesn't handle bankruptcy stays correctly. Another person discovers it's not logging decisions in a way that passes audit.

Month 7–12: Team spends most of their time maintaining the system instead of improving it. ROI disappears. They start looking at vendors again, but now they're defensive about admitting the internal build didn't work out.

Year 2: The system is still running, but nobody wants to own it. It's become a drag on the organization. Updating it means regression testing the entire chain. Adding features means months of planning.

The analyst who built it could have done ten other valuable things for the business. Instead, they're the de facto collections technology team.

When DIY actually makes sense

Let's be fair: there are cases where building internally works.

If you're solving a hyper specific problem that no vendor addresses, maybe a unique portfolio strategy or an edge case in your specific geography, and you can afford to have someone own it long-term. Go ahead.

But if you're solving a problem that vendors have already solved for thousands of other companies? You're not innovating. You're duplicating effort and assuming you'll do it better than someone who's had ten years and millions of dollars to get it right.

The real opportunity with AI in collections isn't building your own tools from scratch. It's using a modern platform that lets you:

- Configure strategies without writing code
- Plug in your own models and test in a governed environment
- Experiment rapidly without building compliance risk
- Scale what works without maintaining infrastructure
- Free up your team to focus on strategy instead of software engineering

The bottom line

The person who's built a working prototype in four hours is genuinely talented. And this is exactly why you should have them working on harder problems rather than keeping a homegrown system alive.

The platform approach isn't about locking you in or preventing innovation. It's about letting your team innovate on top of a foundation that's already solved the boring, mandatory, regulation heavy problems.

Build to learn what you need. Buy to deliver it at scale.

That's the real strategy. And it's already happening at the enterprises that are winning in collections right now.

Learn more at **inquiries@crsoftware.com**