# Buy, Build,
# or Partner?

*Why the Build vs Buy Debate Is the Wrong Question —*
*and What Financial Advice Firms Should Be Asking Instead*

---

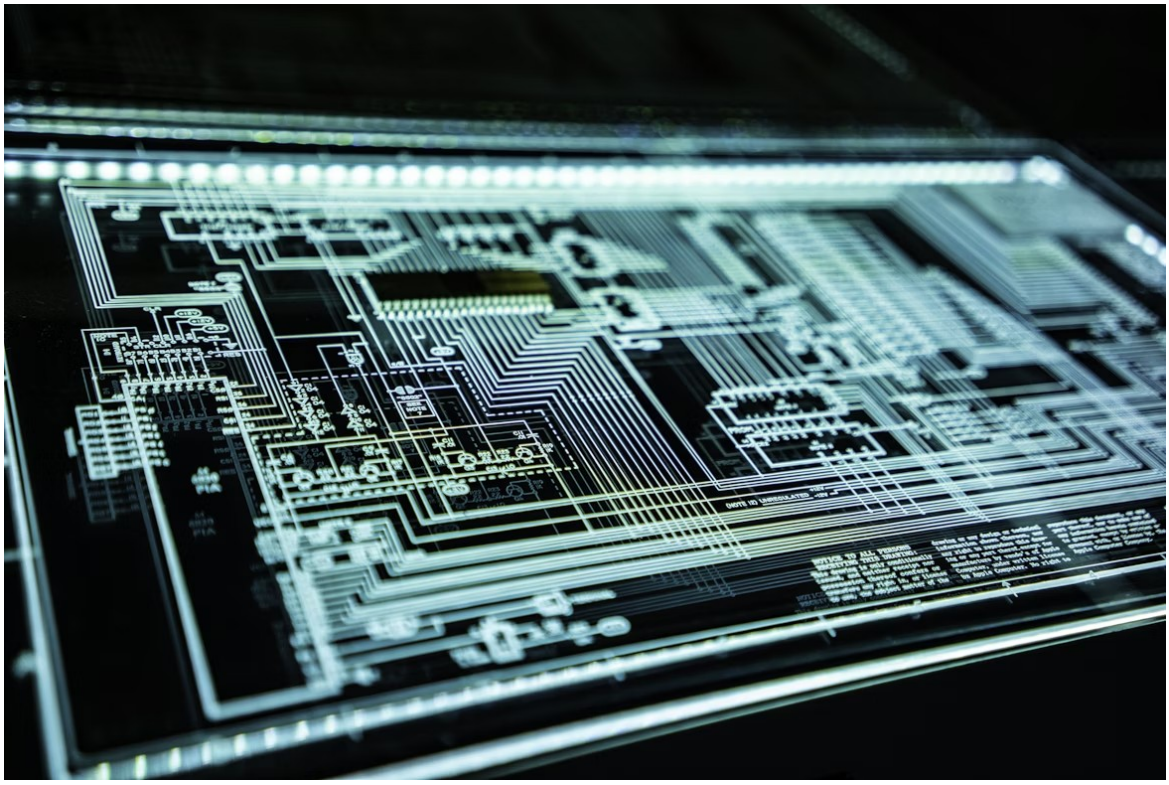
**Peter Ridlington FPFS**
CEO & Founder, Ningi
March 2026

# Contents



*This essay draws on research from across industries — healthcare, law, accounting, enterprise software, and professional services — to examine the three paths firms take with technology, and to make the case for a fourth option that most have not considered.*

# Introduction: A Familiar Conversation



We hear this conversation regularly. A firm has spent five, ten, sometimes fifteen years building its own technology. Usually on Microsoft Power Platform, sometimes on Salesforce, occasionally on something more bespoke. They have invested serious money. They have paraplanners-turned-citizen-developers, a relationship with a consultancy, maybe a small internal team. And at some point, someone in the room asks the question: should we just keep building on this?

It is an honest question, and it deserves an honest answer. Not a sales pitch, but a genuine examination of the options, because the decision about whether to build, buy, or find another path entirely is one of the most consequential a firm can make. It shapes not just what technology you use, but how your business operates, what it costs, how quickly it can move, and ultimately, what experience your clients receive.

This is not a niche problem confined to financial advice. The build-versus-buy question is one of the most studied decisions in business strategy. Geoffrey Moore, in his influential work on technology adoption, drew a distinction between core and context: core is what differentiates your business and creates competitive advantage; context is everything else. His argument, now widely accepted in business strategy, is that companies should invest their best resources in core activities and outsource context to specialists. The question is not whether you can build something, but whether building it is the highest and best use of your time, money, and attention.

*Do what you do best and outsource the rest.*

Peter Drucker put it more bluntly. The Standish Group's CHAOS reports, which have tracked software project outcomes for three decades, consistently find that around 70% of custom software projects fail to deliver on time, on budget, or on specification. McKinsey's research on large IT projects tells a similar story: 45% go over budget, 7% over time, and 56% deliver less value than expected. These are not amateur efforts. These are well-funded projects with dedicated teams.

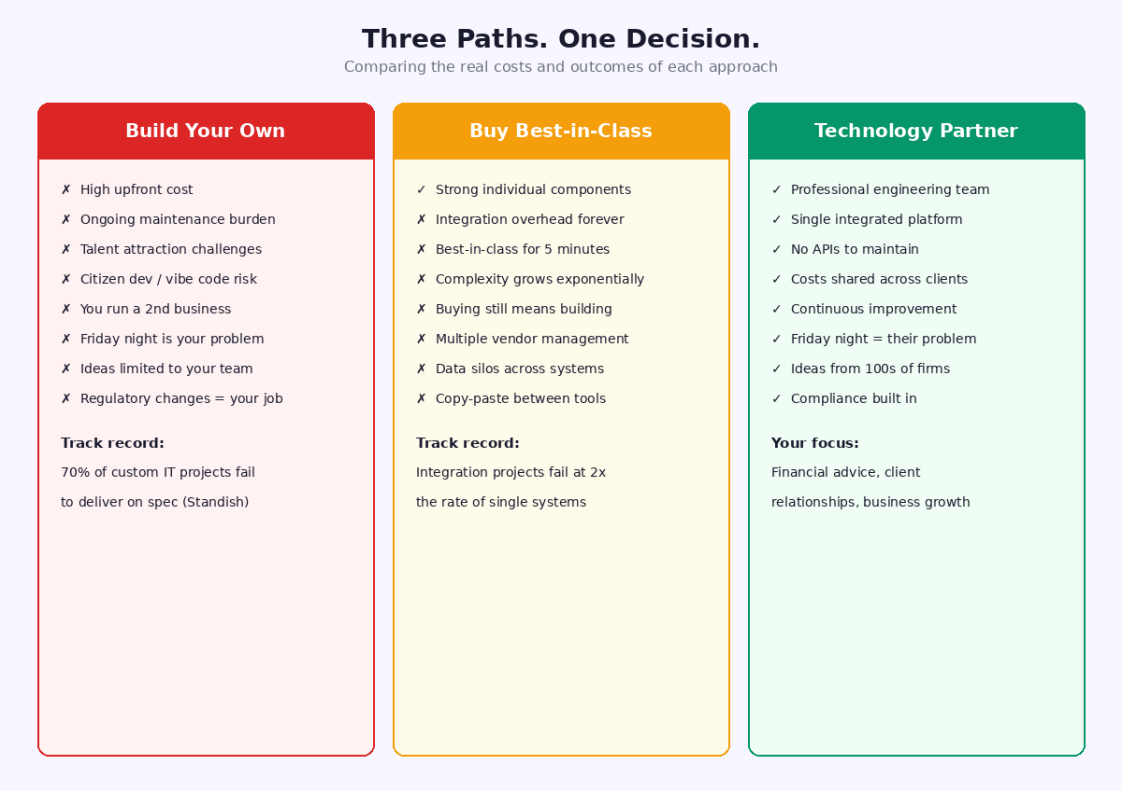



*Figure 1: The three approaches to technology — and their real-world track records*

This essay is an attempt to lay out the examination clearly, drawing on evidence from across industries that have faced the same choice. It is written from the perspective of someone who has spent years building technology for financial advice firms, and who has watched many firms navigate this decision in different directions. Some of those directions worked. Many did not. The patterns are remarkably consistent, and they are not unique to our industry.

# Part One: The Self-Build Path

## How It Usually Starts

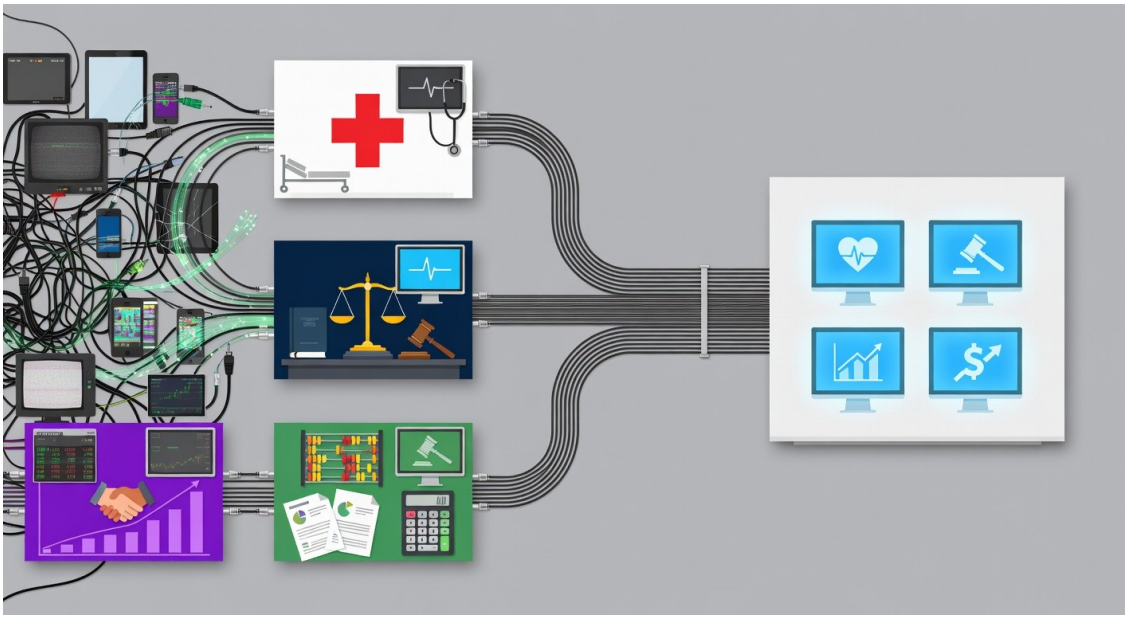
The appeal of building your own technology is entirely understandable. You know your business better than anyone. You know the workflows, the pain points, the little inefficiencies that drive your team mad. Off-the-shelf software never quite fits. It has features you do not need and lacks the ones you do. So you start building.

In the early days, it feels like a revelation. A paraplanner who is good with computers builds a workflow in Power Automate. Someone creates a client dashboard in Power Apps. A SharePoint site gets configured into something resembling a portal. Each piece solves a real problem, and because the person who built it understands the business, it solves it in exactly the right way.

*It always starts with a clever spreadsheet*

This is the honeymoon period. It typically lasts one to two years. And it is not unique to financial advice. The pattern has played out across every professional services industry you can name.

## The Pattern Across Industries



Healthcare provides perhaps the most expensive cautionary tale. For decades, hospitals and health systems attempted to build their own electronic health record systems rather than adopt commercial

platforms. The logic was identical to the logic used by advice firms today: we understand our workflows better than any vendor, our needs are unique, and off-the-shelf products do not fit. The US Veterans Health Administration spent over twenty-five billion dollars building and maintaining its custom VistA electronic health record system across multiple decades, enduring repeated cost overruns, delayed deployments, and a system that ultimately could not keep pace with modern healthcare requirements. The programme was eventually restructured around commercial platforms. The UK's National Programme for IT, the NHS's attempt to build a unified health records system, was abandoned in 2011 after spending an estimated ten billion pounds across nine years. A National Audit Office review concluded the programme had been too ambitious, too centralised, and had failed to account for the sustained complexity of building and maintaining production-grade software at scale.


*Healthcare learned this lesson at a cost of billions*

The legal sector followed a similar arc. Through the 1990s and 2000s, large law firms invested heavily in custom case management, document assembly, and knowledge management systems. The reasoning was familiar: our practice is unique, our workflows are specific, no vendor understands litigation the way we do. By the 2010s, most had migrated to commercial platforms. The cost of maintenance, the difficulty of retaining specialised developers who would rather work at technology

companies, and the relentless pace of regulatory and compliance changes made custom systems untenable. Firms like Clifford Chance, DLA Piper, and Allen & Overy, with budgets that dwarf anything in financial advice, could not sustain internal technology development at the level required. If they could not make it work, it is worth asking what makes an advice firm with fifty people think it can.

*Law firms with budgets that dwarf financial advice could not make self-build work*

Accounting firms discovered the same thing with tax preparation and practice management software. The regulatory environment, with its annual tax law changes, compliance updates, and filing requirement modifications, created what amounts to a perpetual maintenance burden. Mid-market CPA firms that built custom billing or time-tracking systems through the 2000s and 2010s found themselves spending thirty to forty percent of their development effort simply keeping up with regulatory changes, leaving almost no capacity for innovation. Most eventually migrated to commercial platforms from Thomson Reuters, Intuit, or BlackLine, often at significant migration cost.

## What Happens Next

The financial advice version of this story follows the same trajectory. The problems begin when you try to scale. That clever workflow the paraplanner built needs to handle edge cases. The dashboard needs to talk to your back-office system. The portal needs to be secure, accessible, and actually pleasant to use. The person who built it gets pulled into maintaining it, then extending it, then fixing it when it breaks in ways nobody anticipated.

Before long, you have a pseudo-development team. People whose job was supposed to be financial planning are now spending significant portions of their time maintaining technology. The budget for external consultants grows. Licence costs for the underlying platform accumulate. And the technology itself, because it was built by domain experts rather than professional developers, starts to show its limitations: inconsistent design, poor error handling, no automated testing, no release control, no quality assurance process.

*There is no silver bullet that eliminates the essential difficulty of software engineering.*

This is not a criticism of the people involved. They are often talented and deeply committed. But building production-grade software is a discipline, and it is not the same discipline as financial planning. Frederick Brooks made this point in The Mythical Man-Month in 1975, and it has only become more true as software systems have grown more complex. The difficulty is not in the typing. It is in the thinking.

## The Citizen Developer Fallacy

The technology industry has spent the last decade promoting the idea of the citizen developer: the domain expert who, armed with low-code or no-code tools, can build the solutions their business needs without involving professional developers. Microsoft, in particular, has invested heavily in this narrative through Power Platform. Salesforce has done the same with its ecosystem. The promise is compelling: your people know your business, give them the tools and they will build what they need.

The reality is more nuanced than the marketing suggests. Low-code platforms make specific things easier, but they make everything else significantly harder. They restrict the full power of modern web development. They impose architectural constraints that become painful at scale. Finding developers who specialise in these platforms is expensive and difficult, because the talent pool is smaller and the skills less transferable. And the licence costs, which initially seem reasonable, compound into significant sunk costs as your usage grows. Microsoft 365 licensing for a Power Platform-heavy organisation can easily run to hundreds of pounds per user per month once you factor in premium connectors, AI Builder credits, and the Dataverse storage that any serious application requires.

Gartner has repeatedly warned about the shadow IT risks created by citizen development programmes, noting that without proper governance, organisations end up with dozens or hundreds of ungoverned applications, many of which become business-critical without anyone planning for them to be. The analyst firm found that by 2025, the majority of low-code applications built by citizen developers would need to be rebuilt by professional developers within three years of deployment, as they hit scaling, security, or maintainability limits.

More fundamentally, citizen development lacks the disciplines that make professional software reliable: rigorous design, quality assurance, release control, version management, security auditing, performance testing. It carries the same risk as the clever person who built the spreadsheet your entire business now depends on, except that person left two years ago and nobody fully understands how it works.

Beyond a certain point, if you want to build anything of substance, you need a team. You need to manage that team. You need processes, standards, and oversight. At which point, the citizen developers may as well be professional developers, because you are running a development operation whether you intended to or not. And the question becomes: how good is the development team you can assemble compared to the team at a dedicated technology company? The honest answer, almost always, is not as good.

## The Runaway Budget

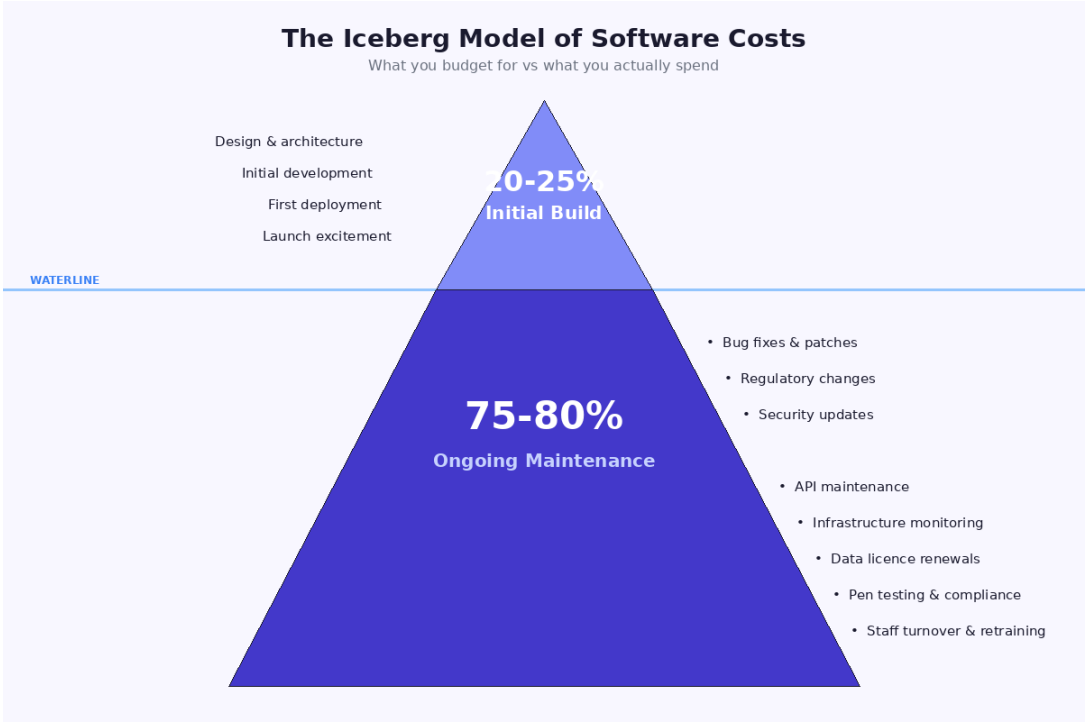



*Figure 2: The iceberg model of software costs — initial build is just the tip*

The financial trajectory of self-build projects follows a depressingly predictable curve. Early enthusiasm and modest investment give way to escalating costs as the scope grows, maintenance demands increase, and the gap between what has been built and what the business actually needs becomes apparent.

This is consistent with what the software industry calls the iceberg model of software costs. Research consistently shows that the initial development of a software system represents only about twenty to twenty-five percent of its total lifetime cost. The remaining seventy-five to eighty percent is maintenance: fixing bugs, adapting to changing requirements, updating integrations, managing security patches, and keeping pace with the platforms and dependencies the system relies on. A firm that budgets for the build but not the ongoing maintenance is seeing only the tip of the iceberg.

And here is the part that rarely gets discussed honestly: even if you are prepared to make that investment, you still need to deal with everything that sits beneath the application layer. Data licences. Penetration testing. ISO 27001 compliance. Origo integration for data feeds. Morningstar for fund data. Regulatory change management. Security patching.

Infrastructure monitoring. These are not optional extras. They are the cost of doing business in regulated financial services technology, and they are substantial. When a technology vendor quotes you a subscription fee, these costs are already baked in. When you build your own, every single one of them is your problem.

## The Friday Night Problem

There is a practical test that cuts through much of the theoretical debate: what happens when it breaks at 7pm on a Friday?

*You build it, you run it.*

When you rely on your own technology, built by your own team, the answer is that someone on your team has to fix it. If that team is a paraplanner who is good with computers, they may not be available. If it is a junior developer, they may not have the experience to diagnose the problem quickly. If it is an external consultant, you are paying emergency rates and waiting in a queue.

When a tax regime changes, when a regulatory requirement shifts, when a data feed provider updates their API specification, the burden falls entirely on you. There is no product team monitoring these changes. There is no engineering team rolling out updates. There is just your team, doing their best with the resources available. Amazon's Werner Vogels has a useful phrase for this: "You build it, you run it." In the context of a technology company with dedicated operations teams, that is a healthy principle of accountability. In the context of an advice firm where the builder is also the paraplanner, the compliance officer, and the person trying to get home for dinner, it is a recipe for burnout and failure.

# Part Two: The Best-in-Class Illusion

## The Dog’s Dinner

The alternative to building your own technology has traditionally been to buy the best available solution for each component of your business process. The best CRM. The best cashflow modeller. The best client portal. The best compliance tool. The best research platform. Pick the winner in each category, plug them together, and you should end up with a best-in-class technology stack.

In theory, this is elegant. In practice, it produces what I once described as a dog’s dinner, and the industry needs to stop eating it.

*Individually polished. Collectively chaotic.*

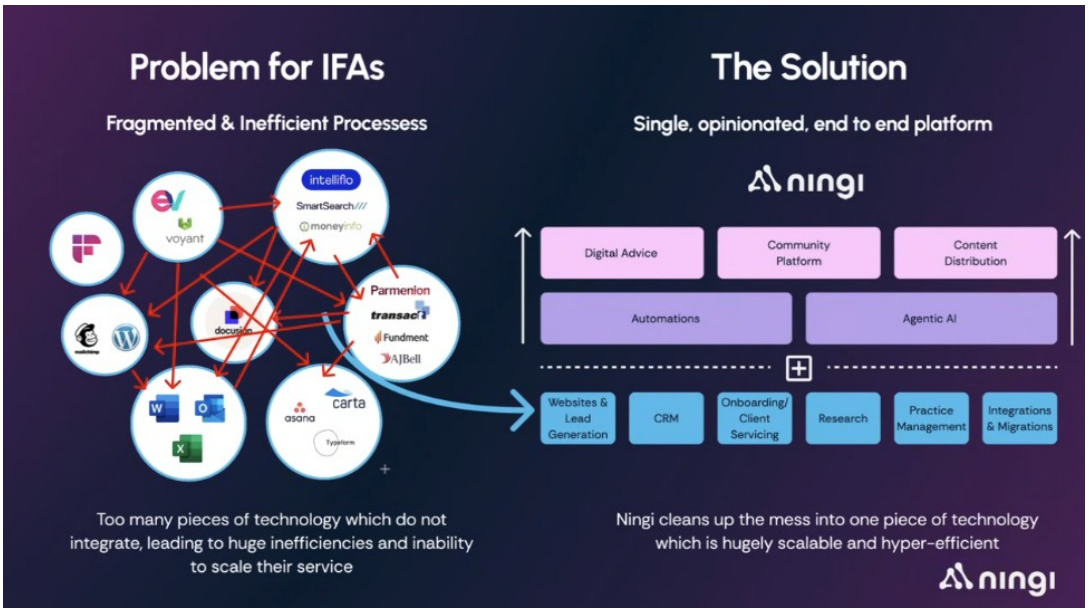



*Figure 3: The fragmented tech stack vs a unified platform*

The problem is not with any individual component. That client portal might genuinely be five percent better than the alternatives. That CRM might have a couple of features the others lack. But when you step back and look at the whole picture, you have a bag of bits that needs plugging in, configuring, integrating, and managing. Not just to get it started and working, but continuously, indefinitely, if you want it to keep working.

The enterprise software world learned this lesson expensively through the best-of-breed versus integrated suite debates of the 1990s and 2000s. Companies that assembled best-of-breed ERP components from different vendors, rather than adopting an integrated suite from SAP or Oracle, consistently found that the integration costs consumed any advantage the individual components provided. A landmark study by the MIT Sloan Center for Information Systems Research found that companies with more integrated technology architectures generated significantly higher revenue per employee and profit margins than those with fragmented best-of-breed environments. The integration overhead was not a one-time cost but a permanent tax on the organisation's agility and efficiency.

## Best in Class for Five Minutes

There is another problem with the best-in-class approach, which is that best-in-class is a temporary designation. The technology market moves quickly. The client portal that was the clear winner six months ago may have been overtaken by a competitor. The CRM that had those extra features may have stagnated while others innovated.

Do you keep hot-swapping components in and out? Do you continuously survey the market for the latest thing? Do you keep buying and changing and absorbing all the disruption that comes with it? The answer, if you are honest, is that you cannot. The switching costs are too high, the integration work too painful, the organisational disruption too great. So you end up locked into a collection of tools that were each best-in-class at the moment you selected them and have been drifting from that position ever since.

*The audiophile fallacy: individually perfect components that sound terrible together*

There is an instructive parallel in the world of high-fidelity audio. Audiophiles who assemble systems from the “best” individual components — the finest turntable, the finest preamp, the finest speakers, each from a different manufacturer — often find that the system as a whole sounds worse than a well-designed integrated system from a single manufacturer. The components were each optimised in isolation but never designed to work together. Impedance mismatches, cable interactions, and resonance issues mean the sum is less than its parts. The technical term for this is system-level optimisation, and it applies to business technology stacks as directly as it applies to audio equipment.

*An orchestra where every musician plays from a different sheet*

## When Buy Still Means Build

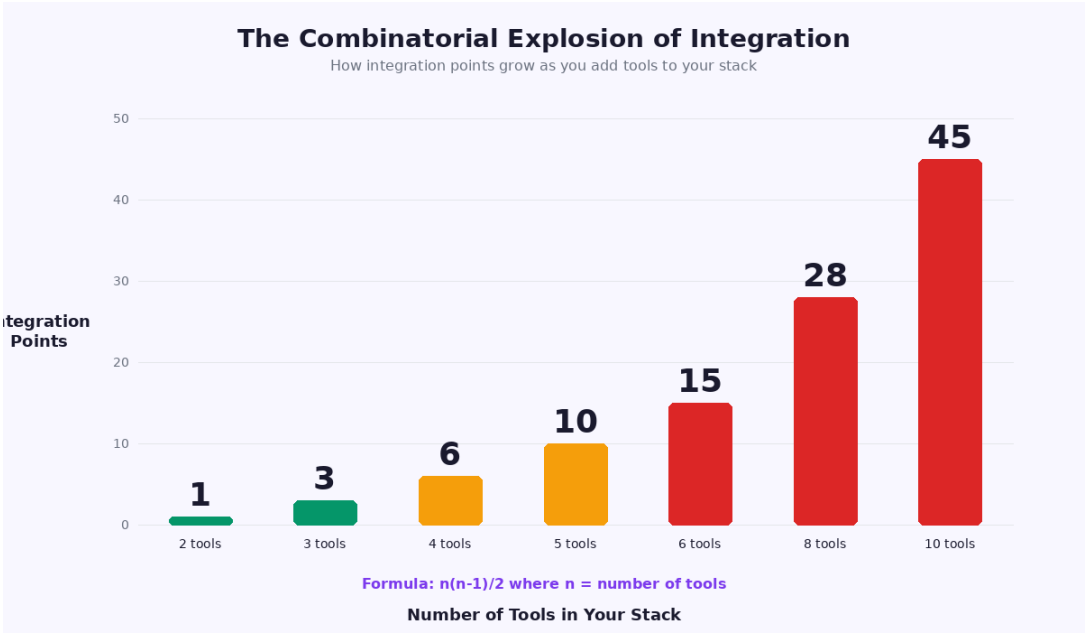



*Figure 4: Integration points grow exponentially — 10 tools means 45 potential failure points*

Here is the insight that is often missed: buying best-in-class and then integrating it is not really buying at all. It is building. You are building the integration layer. You are building the workflows that connect the systems. You are building the data pipelines that keep everything in sync. You are maintaining all of it, indefinitely.

The APIs are great in principle, but someone has to maintain them. Every time a vendor updates their platform, your integrations are at risk. Every time you add a new tool to the stack, the complexity does not increase linearly. It increases combinatorially. A system with three components has three possible integration points. A system with ten components has forty-five. Each connection point is a potential failure point, and you are responsible for all of them.

*When your integration layer starts to resemble your lunch*

A whole cottage industry has grown up around this mess. Consultancies that help advisers find the latest technology, evaluate which widget is best right now, configure workflows, connect APIs, and manage the resulting complexity. The existence of this industry is itself evidence of the problem. The question these businesses answer is essentially: how good is each piece of technology at cleaning up its own mess? That is not a question we should need to be asking.

## The System Is Bigger Than the Software

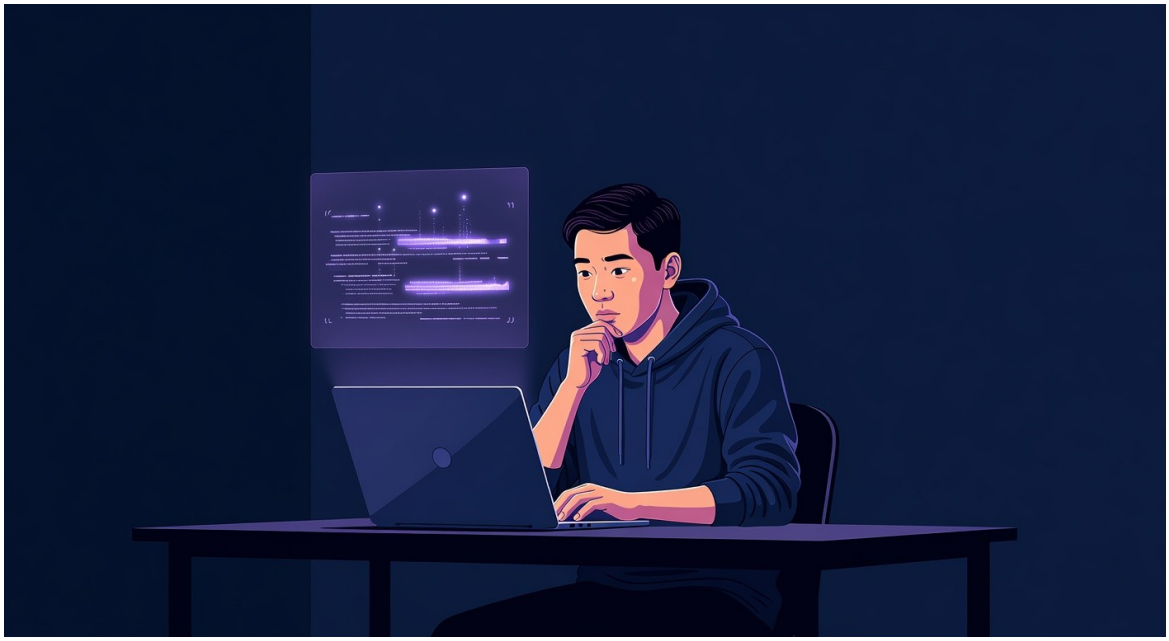
> *"The performance of a system is not the sum of the performance of its parts. It is the product of their interactions."*
>
> — W. Edwards Deming

When we talk about the best system for running an advice business, we should not be talking about individual pieces of technology. We should be looking at the system as a whole: the combination of technology, people, processes, and tools used to deliver a service. Optimising each vertical component while ignoring the horizontal connections between them is like tuning each instrument in an orchestra independently without ever rehearsing together. The individual components might each be excellent. The performance will still be a mess.

I do not believe the financial advice industry is in a position where everything is so complex and so finely tuned that firms need to hunt for

one percent improvements by chasing the pinnacle of technology in each vertical. Much of the industry still runs significant processes on spreadsheets. The opportunity is not in finding marginally better components. It is in getting the basics right across the entire process, end to end, in a way that actually works.

# Part Three: The Vibe Coding Mirage



## A New Version of an Old Mistake

The emergence of powerful AI coding assistants has introduced a new variation on the self-build temptation. The reasoning goes something like this: AI has democratised software development. Anyone can build an app now. We will hire a couple of graduates, give them Claude or Cursor licences, and build what we need in-house at a fraction of the cost.

This is the citizen developer argument repackaged for the AI age. And it carries all the same flaws, plus some new ones.

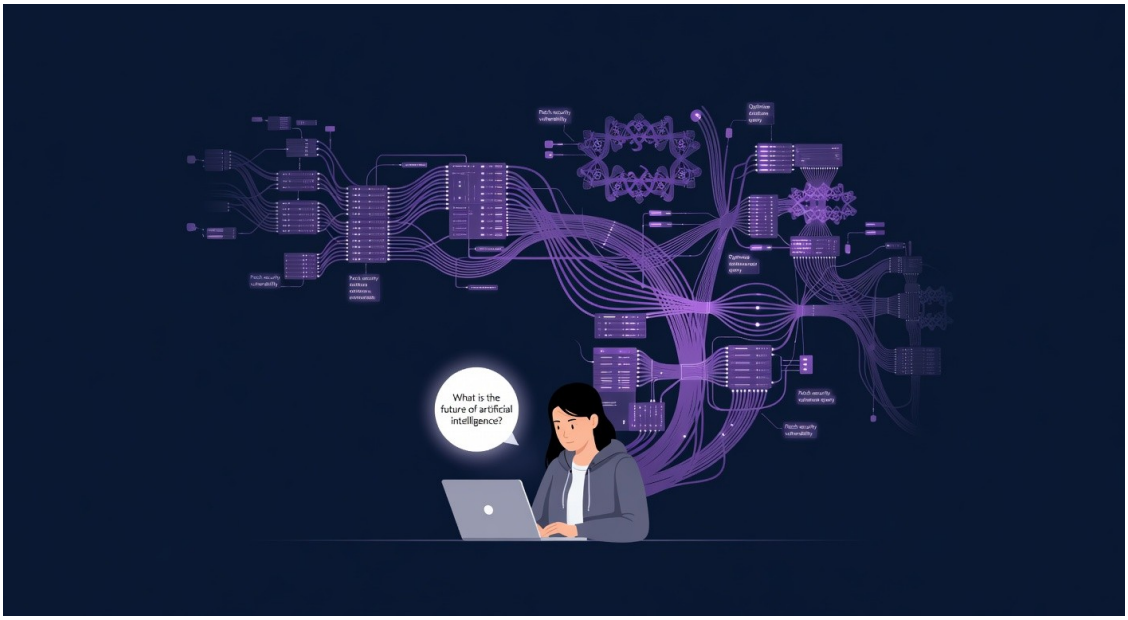
AI coding tools are genuinely impressive. They can accelerate development, reduce boilerplate, and help less experienced developers produce working code more quickly. But they do not solve the fundamental challenges of software engineering: architecture, design, testing, deployment, maintenance, security, scalability, and the thousand other considerations that separate a prototype from a production system.

*The prototype looks deceptively simple. Everything after it is not.*

*The AI helps with the typing. It does not help with the thinking.*

A graduate with an AI coding assistant can build something that works. Building something that works reliably, securely, at scale, over time, through regulatory changes, API updates, and evolving business requirements is a different proposition entirely.

## The Faustian Bargain



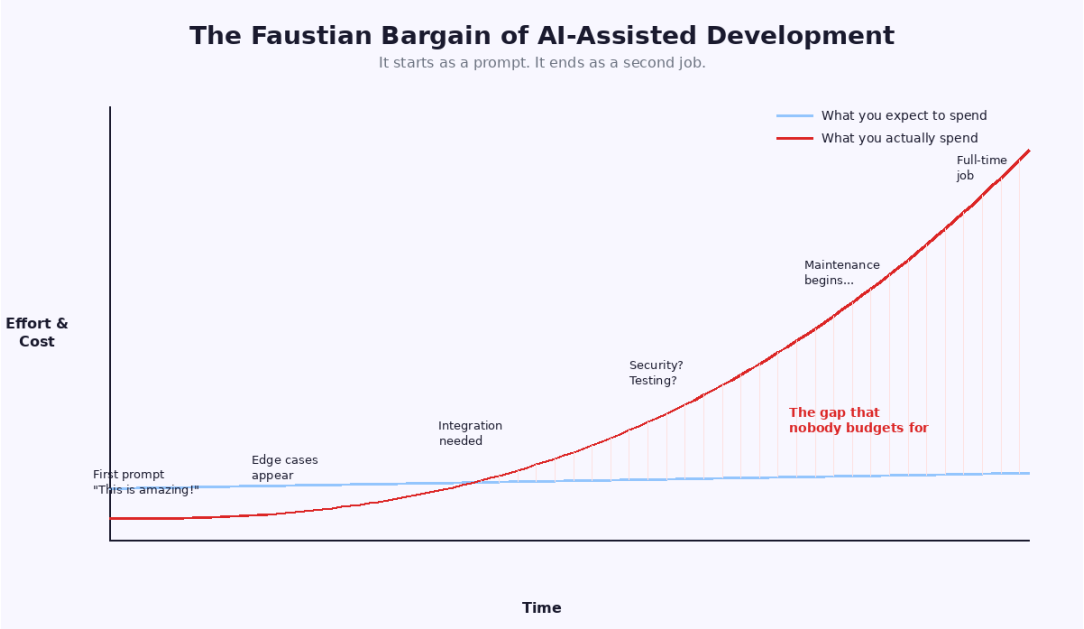



*Figure 5: What you expect to spend vs what you actually spend — the gap that nobody budgets for*

There is a concept gaining traction among experienced software developers that describes the AI coding experience with uncomfortable precision: the Faustian bargain. It starts as a prompt. A simple idea, expressible in a few sentences. The AI produces something that looks right, quickly. You feel powerful. You feel fast. But then you need to adjust something. The adjustment requires more context, more prompting, more waiting. Each fix introduces new edge cases. Each edge case requires another round of prompting. The cycle lengthens. The complexity compounds.

Before long, you have multiple things running, multiple ideas half-built, and a growing sense that while you are producing more code than you ever have, you are not actually accomplishing what you set out to accomplish. As one developer described it: “The work is multiplying, the task juggling is getting out of control. The productivity is through the roof, yet nothing really happens.” Ten thousand lines of code in a day is easy now. But that does not mean those ten thousand lines were any good.

The deeper insight is one that experienced engineers have always known but that AI tools make dangerously easy to forget: the programming was never the hard part. The hard part was solving the right problem, in the right way, with the right architecture, and maintaining that solution over time as the world changes around it. AI dramatically reduces the cost of

producing code. It does not reduce the cost of understanding what code to produce, or of living with the consequences of getting that decision wrong.

> *"Atoms are cheap, but process is pricey. The raw materials of a product are rarely the expensive part. What costs money is the knowledge of how to combine them, the processes to ensure quality, and the institutional learning that accumulates over years."*

*Writing code is the atoms. Software engineering is the process. AI has made the atoms essentially free. The process is as expensive as it has ever been.*

## The Maths Does Not Work

The economics of in-house AI-assisted development are less attractive than they initially appear. Two graduates with AI coding licences cost twenty thousand pounds or more annually before you account for management overhead, tooling, infrastructure, and the opportunity cost of the time required to direct, review, and course-correct their work. Add API token costs for the AI tools themselves, which can run to hundreds or thousands per month for heavy usage, and the total climbs further.

That is not actually cheap. And it buys you developers who, with the greatest respect, are learning on the job. They will reinvent solutions that already exist. They will make architectural decisions they do not yet have the experience to evaluate. They will build things that work in development and fail in production.

Do the same maths on your own time, if you are a firm principal who is spending evenings and weekends directing the technical work. What is your hourly rate? What else could you be doing with that time? If you bill at three hundred pounds an hour and spend ten hours a week directing development, that is a hundred and fifty thousand pounds a year of opportunity cost that never shows up on a spreadsheet. The hidden costs of in-house development are almost always underestimated, because they are distributed across the organisation in ways that are difficult to see until you add them up.

## The Talent Problem

If you decide to build a proper development team rather than relying on graduates and AI tools, you face a different challenge: talent.

The best developers in the market are not, by and large, looking for jobs at financial advice firms. They want to work at technology companies, where they can build products at scale, work with modern tools and practices, and develop their careers alongside other experienced engineers. Stack Overflow's annual developer surveys consistently show that the factors developers value most in a role are the technologies they work with, the quality of their colleagues, and the impact of what they build. An advice firm with a two-person development team running Power Platform cannot compete on any of those dimensions with a technology company building a product used by thousands.

There is a blunt way of putting this: if your developers were capable of building a world-class technology platform, they would probably be doing it for themselves or for a technology company that could offer them the environment, the compensation, and the career trajectory they want. What you get, more often, is competent people who want a safe, stable role. That is not a criticism of them. But it does mean you are likely building with a B-team compared to the A-team at a technology vendor that lives and dies by the quality of its software.

And you still have to manage them. You need to understand what good looks like in software development, how to evaluate architectural decisions, how to set technical direction. If you are not a technologist yourself, you are managing a discipline you do not fully understand, which is a recipe for expensive mistakes. When you buy software, the vendor is making decisions informed by feedback from hundreds of customers and built on years of domain expertise. When you build in-house, every decision has to come from you, and you have to get them right with a fraction of the information.

# Part Four: The Uncomfortable Truth

## You Are Starting a Second Business

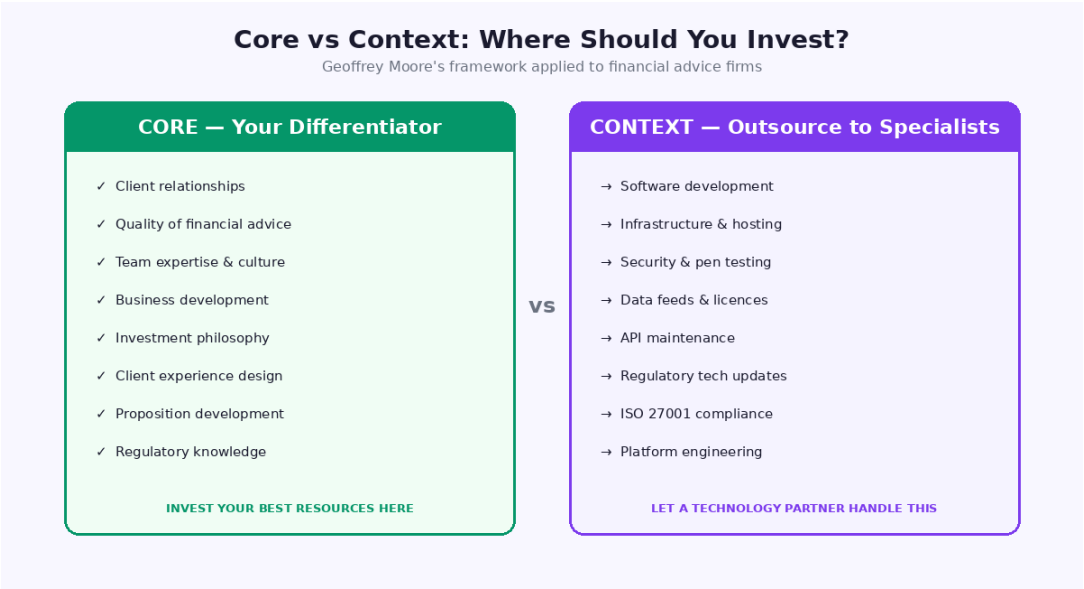



*Figure 6: Geoffrey Moore's framework — invest your best resources in what differentiates you*

Here is the thing that firms considering self-build rarely confront directly: building and maintaining technology at any meaningful scale is not a side project. It is a business.

It requires sustained investment. It requires specialised talent. It requires processes, governance, and management attention. It requires you to keep pace with a market that moves faster than almost any other. It is, in effect, a second business running alongside your advice firm.

This is not inherently wrong. Some firms may decide that owning their technology is a strategic imperative and be willing to commit the resources required to do it properly. If so, they should do it properly: hire professional developers, invest in real infrastructure, build genuine engineering capability, and compete with the best technology companies in the market. That is a legitimate business decision.

But most firms do not want to run a technology company. They want to run an advice business. The technology is a means to an end, not the end itself. And for those firms, trying to build their own technology is like someone who describes buy-to-let property as a passive income stream. It is not passive. It is a job. It requires constant work, attention, and money.

*Restaurants do not build their own point-of-sale systems. They focus on the food.*

Restaurants do not build their own point-of-sale systems. They do not build their own booking platforms. They do not build their own delivery logistics. They buy Toast, they use Resy, they integrate with Deliveroo. And nobody thinks less of them for it, because everyone understands that the restaurant’s competitive advantage is in the food, the service, and the

experience, not in the software that processes the bill. The same logic applies to every professional services firm, including yours.

---

## The Competitive Delta Has Not Changed

The argument for self-build in the AI age often rests on the idea that AI has levelled the playing field. If anyone can build software now, the thinking goes, then the advantage that technology companies had over their customers has been eroded.

This is wrong, and it is wrong for a simple reason: technology companies have access to the same AI tools you do. Every advantage that AI coding assistants give you, they also give to the professional development teams at software vendors. The delta has not changed. They still have the engineering expertise, the architectural experience, the testing infrastructure, the deployment pipelines, the security practices, and the scale advantages. AI has made both sides faster. It has not made them equal.

*AI has made both sides faster. It has not made them equal.*

If anything, AI has widened the gap. A professional engineering team augmented by AI can move dramatically faster than it could before, because they have the experience to direct the AI effectively, to evaluate its output critically, and to integrate its contributions into a coherent, maintainable system. An amateur team augmented by AI can produce working prototypes faster, but still lacks the experience to turn those prototypes into reliable, secure, maintainable production systems. The gap between prototype and production has not changed. If anything, it has grown, because the ease of creating prototypes encourages people to underestimate the difficulty of everything that comes after.

## What Is Your Actual Business?

*This is what you should be spending your time on*

The question every firm should ask itself before committing to self-build is the one Geoffrey Moore would ask: what is your core and what is your context?

If the answer to “what is our core?” is financial advice, then your competitive advantage lies in the quality of your advice, the depth of your client relationships, and the expertise of your team. Technology is context. It is the infrastructure that enables your core advantage, not the advantage itself. Spending your time, money, and attention building technology means spending less of those resources on the thing that actually differentiates you.

If the answer is technology, then build technology. Do it properly. Hire the best people. Invest at scale. Compete in the market. Productionise your code and sell it. That is a legitimate path, and some firms have taken it successfully. But be clear-eyed about what you are doing: you are starting a technology company. Staff it like one, fund it like one, and hold it to the same standards as one.

But you cannot be half-in. You cannot get your cousin Dave who is good with computers and compete with dedicated technology companies. You cannot hire two graduates and expect them to build what a team of

experienced engineers at a specialist firm has spent years developing. The middle ground between buying software and building a technology company is expensive, exhausting, and almost always produces an inferior result. It is the uncanny valley of business strategy: not quite a technology company, not quite focused on your core business, and therefore not excellent at either.

# Part Five: The Third Option

## What Has Been Missing

The reason the build-versus-buy debate keeps recurring in our industry is that neither option has historically been satisfactory.

Building your own technology is expensive, distracting, and produces inferior results unless you are prepared to invest at a scale that most advice firms cannot justify. Buying best-in-class components and integrating them produces a fragmented, complex, maintenance-heavy technology environment that is best-in-class for five minutes and a headache forever. Buying a single platform from a traditional vendor gives you a product that was designed for the average customer, not for you, and that evolves according to the vendor's priorities, not yours.

What has been missing is a third option: a technology partner that has all the code, all the engineering capability, all the infrastructure, but that also wants to work with you. Not just ship a product and walk away. Actually partner with you. Understand your business. Build with you, not just for you.

## Why People Do Not Think of This

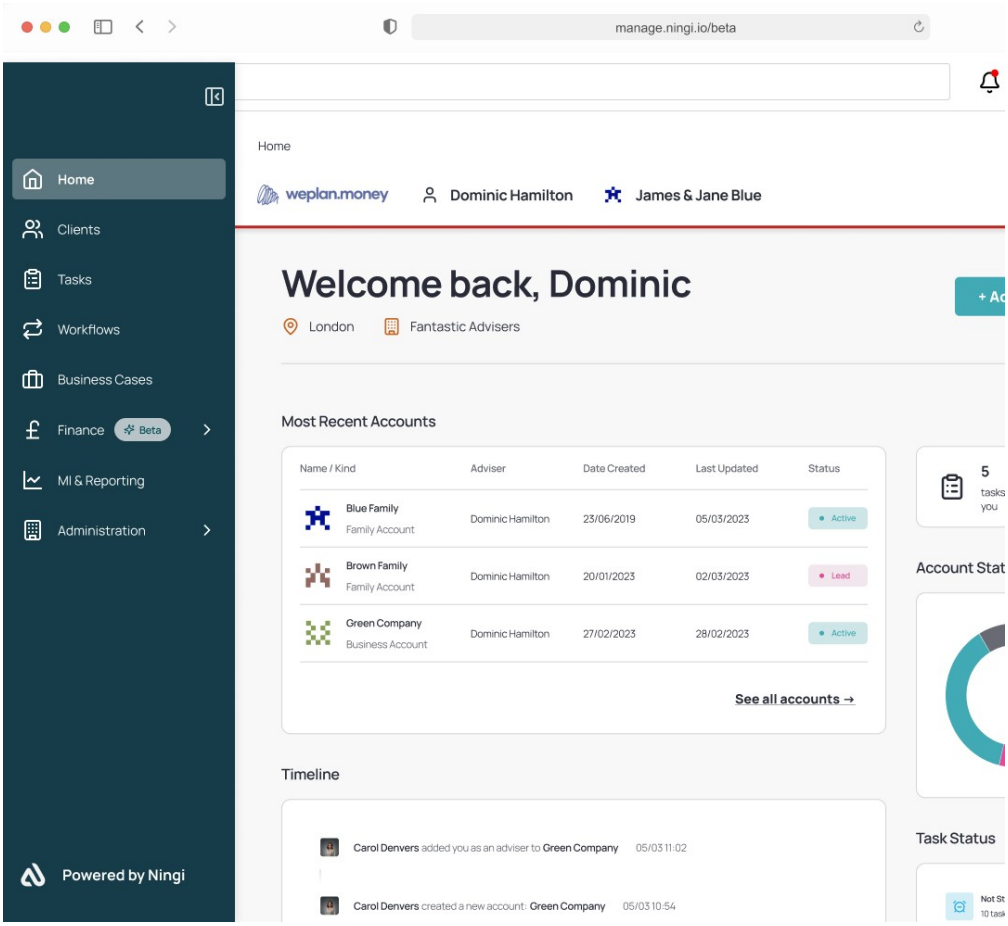
The reason most firms in our industry do not consider this option is simple: it has not existed. There has not been a technology company in the UK financial advice market that operates this way. The market has been structured around two models: vendors who sell products, and consultancies who help you build your own.

Daniel Kahneman, the Nobel laureate whose work on cognitive biases transformed economics, described a phenomenon he called WYSIATI: what you see is all there is. We make decisions based on the options we are aware of, not the options that exist. When the only models you have seen are “buy a product” and “build your own,” those feel like the only choices. But they are not. They are just the ones the market has historically offered.

If you designed the ideal relationship between an advice firm and its technology provider from first principles, you would not design either of the existing models. You would design something that combines the engineering excellence of a specialist technology company with the business understanding and responsiveness of an in-house team. You would want a partner who takes ownership of the complexity rather than pushing tools back to you to figure out yourself.

## What Partnership Looks Like


*Strategy sessions at Ningi — where product direction meets real adviser needs*

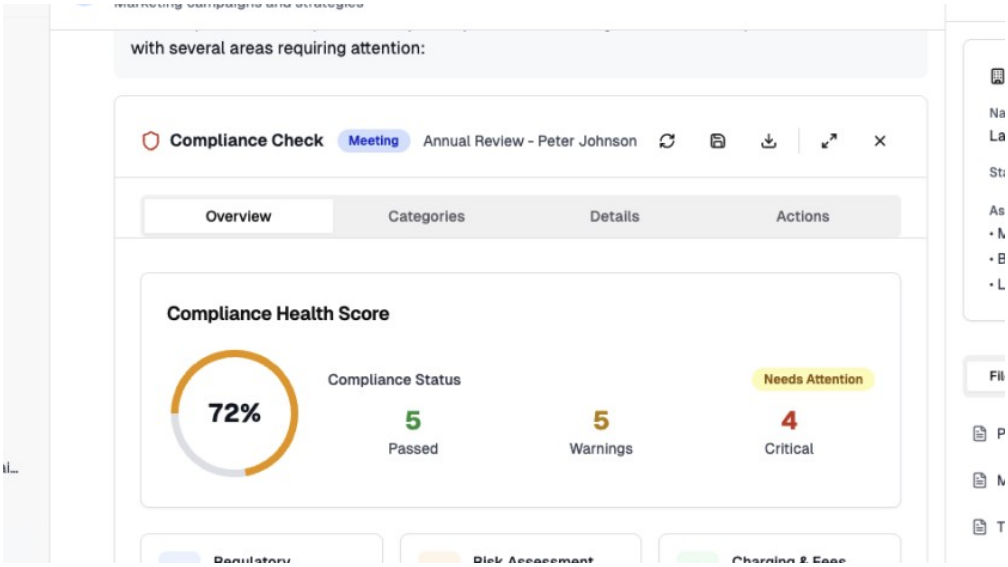



*The Ningi platform — a single, unified environment spanning the full advice process*

A genuine technology partnership means a single platform that spans the full advice process, built and maintained by professional engineers, but shaped by the real needs and feedback of the firms that use it. It means your technology evolves continuously, informed by insights from across the client base, not just your own firm's requirements. It means when tax rules change, the update is rolled out before you need to ask. When AI capabilities advance, they are evaluated, tested, and integrated into your environment by people whose entire job is to do that work.

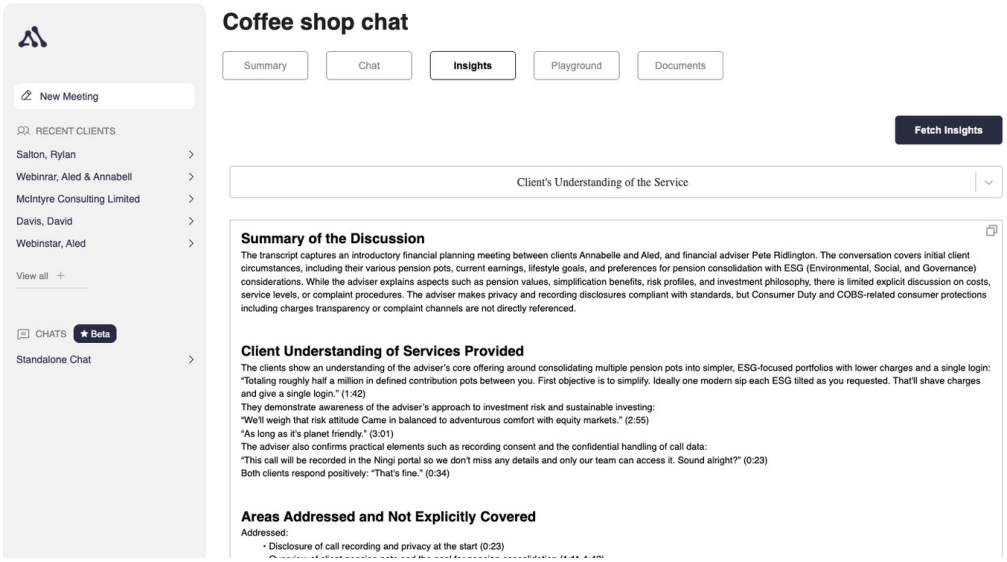
It means you do not need to maintain APIs, because there are no APIs to maintain. Everything lives in one system. It means you do not need to worry about penetration testing, ISO 27001, data feeds, or infrastructure, because that is the partner's responsibility. It means when something breaks at 7pm on a Friday, there is an engineering team on it, not a paraplanner trying to debug a Power Automate flow.

*The engineering team behind the platform*

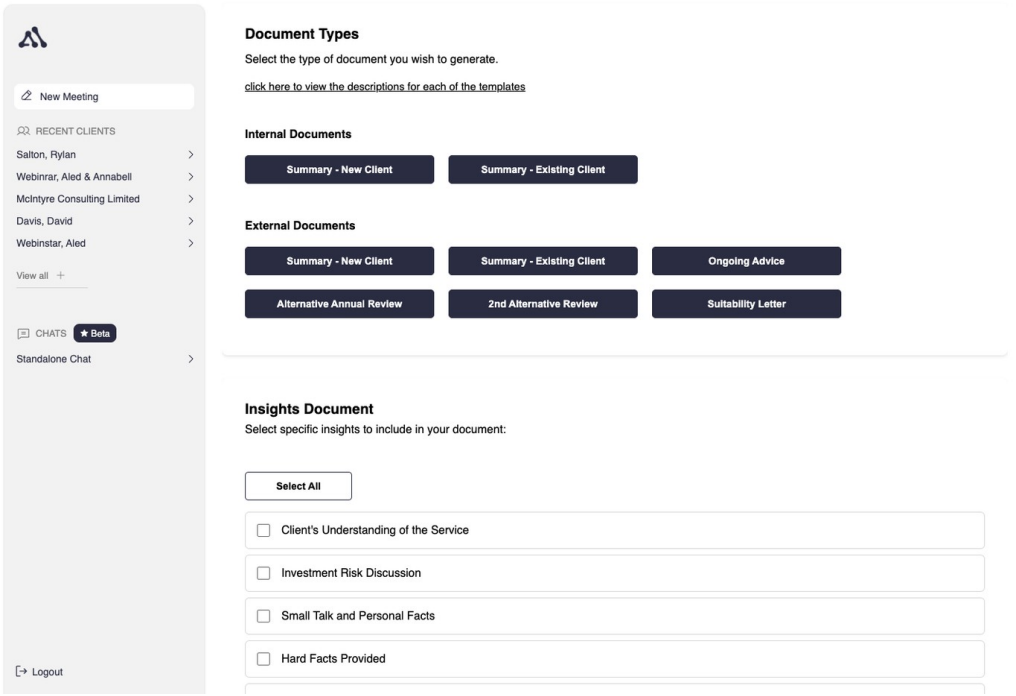



*AI-powered compliance checking — built in, not bolted on*

And it means the ideas that shape the product come from hundreds of firms, not just one. When you build in-house, every idea has to come from you. Every innovation has to be conceived, designed, and executed by your team. When you work with a technology partner, you benefit from the collective experience and insight of every firm they work with. Your competitors' best ideas become your features. Eric Raymond, in his influential essay on open-source software development, articulated this as Linus's Law: "Given enough eyeballs, all bugs are shallow." The same principle applies to product development: given enough customers, all needs are visible. A technology partner sees more of the market than any individual firm can, and that perspective compounds into a better product for everyone.




*AI meeting insights — intelligence that compounds across your entire client book*




*Automated document generation — from meeting to suitability letter in minutes, not days*

## The Cost Efficiency Advantage

There is a hard economic reality underpinning all of this. A technology company that builds one platform and serves many firms can invest in that platform at a level that no individual firm can match. The cost of a professional engineering team, the infrastructure, the compliance, the data licences, the security practices, all of it is spread across the entire client base. Each firm gets the benefit of investment that would be impossible to justify on its own.

This is not a marginal advantage. It is structural. It is the same reason you do not generate your own electricity or build your own telephone network. Economists call it economies of scale, and it is one of the most powerful forces in business. Some things are better done by specialists, at scale, with the costs shared across everyone who benefits.

The firms that try to replicate this capability internally will always be at a cost disadvantage, because they are bearing the full burden of development, maintenance, compliance, and innovation on their own revenue base. And they will always be at a capability disadvantage, because they cannot attract the same calibre of talent or invest at the same scale as a dedicated technology company. These are not problems

that can be solved by working harder or spending more. They are structural features of the market that favour specialisation.


*What calm, unified technology actually looks like*

# Conclusion: An Honest Assessment

The build-versus-buy debate in financial advice technology is not new. But the stakes have never been higher.

AI is accelerating the pace of technological change. The firms that have the right technology infrastructure will compound their advantages as AI capabilities improve. Those that are stuck maintaining fragile, self-built systems, or wrestling with fragmented best-in-class stacks, or burning through tokens and burning out graduates trying to vibe-code their way to a platform, will fall further behind with each passing quarter.

The evidence from across industries is consistent and unambiguous. Healthcare systems that tried to build their own technology spent billions and failed. Law firms with budgets that dwarf anything in financial advice abandoned custom development in favour of commercial platforms. Accounting firms that built their own tools discovered that regulatory maintenance alone consumed their development capacity. The pattern is the same everywhere: building technology is a full-time job, and if it is not your full-time job, you will do it badly.

*Technology should be the thing that enables your business, not the thing that consumes it.*

The honest assessment is this: if you are a financial advice firm, you should be spending your time, money, and attention on financial advice. On your clients, your team, your proposition, your growth. Technology should be the thing that enables all of that, not the thing that consumes it.

Building your own technology makes sense if you are genuinely prepared to build a technology company. Most firms are not, and should not pretend otherwise. Buying best-in-class components makes sense if you are prepared to accept the permanent overhead of integration, maintenance, and vendor management, and if you believe those marginal gains in individual components outweigh the systemic cost of fragmentation. Most firms underestimate that cost significantly.

The third option — a genuine technology partnership — has been missing from this industry. That does not mean it is not the right answer. It means the right answer has not been available until now.

*Simplicity and calm — what your technology experience should feel like*

> *"In the journey of building Ningi, we have come to value simplicity and calm over hype and the mindless feature race. We believe better is always to be favoured over newer. We take ownership of complexity rather than pushing tools to advisers for them to do it themselves."*

The question is not build or buy. The question is: do you want to run a technology company, or do you want a technology partner who runs it for you, with you, and who is as invested in your success as you are?

We think the answer is obvious. But then, we would.

# Let’s Talk

*Ready to explore what a genuine technology partnership looks like?*

**Pete Ridlington**
CEO & Founder
pete@ningi.co.uk

**Jym Brown**
COO & Co-Founder
jym@ningi.co.uk

**Alex**
Lead Developer
alex@ningi.co.uk

**ningi.co.uk**