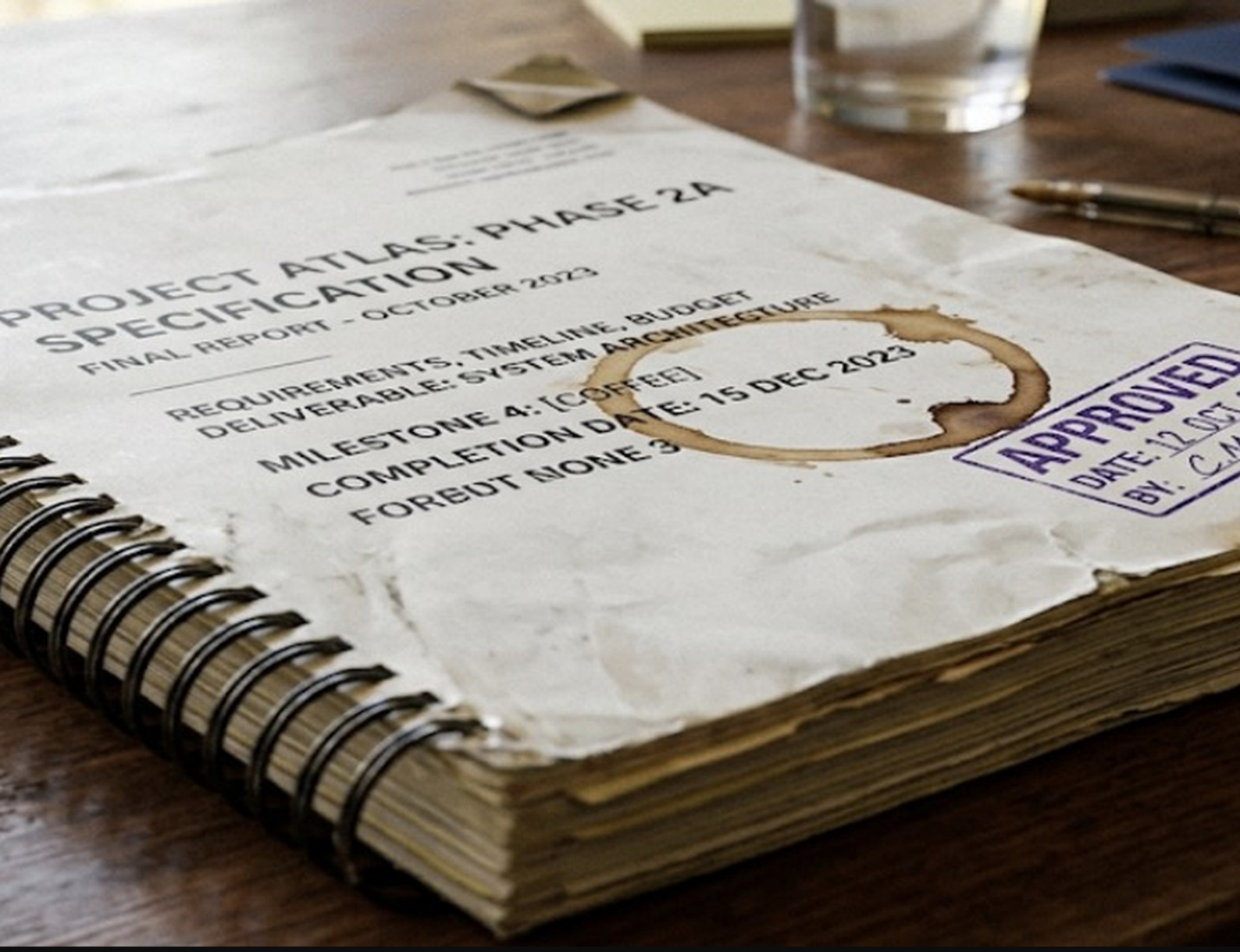

INVESTPUPPY UNVARNISHED

InvestPuppy

Why financial software implementations fail

*For anyone else who's been in **that** room.*

Serious about outcomes. Not serious about ourselves.

About InvestPuppy

InvestPuppy builds Vektor, a systematic portfolio management platform for wealth management professionals.

What this series is

Every experienced practitioner thinks it. Almost nobody publishes it: *there has to be a better way.*

We thought it long enough that we went and built one.

Before that, we spent decades in the rooms.

The same project was failing for the third time. The project plan had been signed off before anyone had understood the first requirement.

The relationship was too warm for anyone to say so.

It isn't incompetence.

It's the incentive structure doing its job perfectly.

Nobody planned it this way. Nobody needed to.

We got frustrated enough to write it down. Then frustrated enough to build something. These papers are the writing-it-down part.

The something we built is a separate conversation.

> The same failures repeat across every financial software implementation in this industry. The same conversations happen. The same silences happen. The same blame is distributed in the same directions. Nobody writes it down because nobody benefits from writing it down.
>
> This series does.

1. The Pattern

The pattern is recognisable to anyone who has been through one of these implementations. Not recognisable in retrospect — recognisable in real time, from week three, while it is still happening. The problem is not that the pattern is invisible. The problem is that by the time it is visible, the incentives that produced it are already in place.

A financial software implementation begins with optimism on both sides. The vendor has sold a capable platform. The client has bought a solution to a real problem. The project plan looks comprehensive. The timelines look achievable. The kick-off meeting produces the right kind of energy.

By week six, something is wrong. Not dramatically wrong — not a visible crisis — but wrong in the specific, quiet way that financial software implementations go wrong. A deadline has slipped. An integration point has turned out to be more complex than anticipated. A workflow assumption that seemed reasonable at signing has turned out not to reflect how the practice actually operates. The people who should be making decisions are not making them.

By month three, the project is in a different shape from the one that was sold and bought. The question of whose fault this is will eventually need an answer. The answer will be complicated.

This is not a story about a single implementation. It is the same story, told with different names on the door, across decades of financial services technology adoption.

2. The Failure Modes

Five patterns appear in every failed implementation. They rarely appear alone.

Nobody owns it.

At signing, the vendor believes the client owns the implementation. The client believes the vendor owns it. Both are correct. The accountability gap is not a misunderstanding — it is a structural feature of how these projects are sold and bought. The next paper in this series examines this in full.

The plan was built before anyone understood the workflow.

The implementation plan was written before week one. Before a single conversation about how the practice actually operates. Before the specific quirks of the client's approval process were understood. Before the edge cases in the data integration were mapped. The plan reflects what the vendor knows how to deliver. It does not reflect what the client needs to receive.

Integration was the afterthought.

Every implementation has integration touchpoints — inbound data, outbound order files, connectivity to existing systems. In every failed implementation, these touchpoints were identified late, scoped inadequately, and tested under pressure at go-live. Integration is not a peripheral concern. It is the implementation. Treating it as a marginal consideration guarantees a marginal result.

Go-live was declared before confidence was earned.

The go-live date was in the contract. It was on the project plan. The pressure to hit it was real from both sides — the vendor needed to close the project, the client needed to demonstrate progress internally. The confidence required to actually go live — the practitioner's settled sense that the system is behaving correctly — was not present. The date was hit. The confidence was not.

The vendor relationship prevented honest conversation.

The sales relationship that brought the project into being is the same relationship that makes it difficult to say, six months in, that something is wrong. The warmth that was an asset in the sales process becomes a liability in the delivery process. The honest conversation that would have identified the problem in week three is the same conversation the relationship cannot afford.

3. Why They Keep Happening

These failures are not caused by incompetence. They are caused by incentive structures that make them the rational outcome.

The vendor's incentives are aligned toward signing, not toward implementation success. The sales team is measured on contracts, not on outcomes. The delivery team inherits the commitments made in the sales process and is measured on hitting the timeline, not on whether the client's practice actually changed.

The client's incentives are aligned toward purchasing, not toward adoption. The person who bought the platform is measured on having bought a credible solution, not on whether their team uses it. The people who will use it are not always the people who decided to buy it.

> The implementation fails in the space between these two incentive structures. Not because anyone planned for it to fail. Because neither side's incentives required it to succeed.

This matters because the fix is not found in the people. Better project managers, more experienced consultants, stronger governance — these mitigate the symptoms. They do not address the cause. The cause is structural. It will reproduce itself with every new project until the structure changes.

4. What This Series Is

This series is not a consulting framework. It is not a methodology. It is not a set of best practices derived from industry research. It is field notes — observations from inside the implementations, from both sides of the table, across enough projects to know that the patterns are structural, not accidental.

Each paper in this series takes one failure mode and examines it directly. Not to assign blame. Not to propose a universal fix. To name the mechanism accurately — so that the people sitting in those rooms can recognise it when it is happening, not in retrospect, but in real time.

This series locates the cause.

The Unvarnished Series

Thirteen field notes on why financial software implementations fail.
Published under our own name. All freely available, no registration required.

Why Implementations Fail (UNV-01) (you are here)
The same implementation failures repeat across the industry, unrecorded, because no party benefits from writing them down. This paper names the pattern openly, once, rather than let it repeat quietly again.

Nobody Owns It (UNV-02)
Ask a vendor who owns an implementation and they say the client. Ask the client and they say the vendor -- both are right, and that shared, undefined ownership is the actual failure mode, not any single decision either side made.

Complexity as Cover (UNV-03)
A consultant's real product is often the document justifying the advice, not the advice itself -- built to survive the project's failure, not prevent it. Complexity becomes the cover story for that gap.

Plan Last (UNV-04)
Project plans are written before anyone understands the workflow they claim to schedule, in columns spanning weeks one through thirty-six. Planning first and discovering later gets the sequence backwards.

The Change Request Trap (UNV-05)
A change request is a formal admission that the original specification was wrong, and also a commercial mechanism for charging separately for what should have been included from the start. Neither party is surprised when it happens.

Invisible Stakeholders (UNV-06)
Every implementation has a second set of stakeholders who never attended a workshop or reviewed a specification, yet are the ones who have to live with what was built. Their absence from the room doesn't make them absent from the consequences.

The Hammer, the Tool, and the Methodology (UNV-07)
A tool gets picked up, used for the job it suits, and put down. A methodology requires certification, governance, and a steering committee -- and the industry keeps reaching for the second when it only ever needed the first.

Life After Live (UNV-08)
Go-live is the final milestone on the Gantt chart, but it's the beginning of the harder problem: an organisation built around a legacy system now has to actually live inside the new one. The project plan ends; the real work doesn't.

The U in UAT (UNV-09)
UAT sign-off documents carry the right signatures -- from consultants testing scripts written by other consultants, not from anyone who actually uses the system day to day. The U in UAT is the part everyone skips.

Nine Women (UNV-10)
Fred Brooks named it in 1975 and the same boardroom conversation still happens fifty years later: a project is late, so the proposal is more people. The logic feels intuitive and has always been wrong.

The AI Revolution — Automating Bad Practice at Scale (UNV-11)
AI doesn't fix a broken process -- it executes it faster, at greater scale, with fewer chances for the human hesitation that might have caught the problem. The real question isn't whether an AI implementation will work, but what it will end up working very hard to get wrong.

Model Bank — the perfect solution for banks that don't exist (UNV-12)
A 'model bank' configuration is a composite of institutions that don't exist, built from someone else's defaults before your institution was ever part of the conversation. Every subsequent customisation is just correcting an assumption that was never really about you.

The parameter that broke the camel's back (UNV-13)
A platform demo answers every question with yes, right up until one specific parameter turns out to be the one nobody actually tested. This paper is the account of exactly where that line sits, and what breaks on the other side of it.

Our Better Way

This is what we built instead.

VEKTOR BY INVESTPUPPY

"So, what did you do about it?" — a fair question after thirteen papers on what goes wrong. InvestPuppy builds Vektor, a systematic portfolio management platform for wealth management professionals — the direct answer, series by series.

These four papers are the how to the thirteen whys above — not a sales pitch, but a technical account of the specific choices Vektor made where the industry's default pattern would have produced the same failures just described.

– Technical Debt (BW-01)
– Vektor House Implementation (BW-02)
– The Configurability Gap (BW-03)
– AI as Instrument (BW-04)

More at investpuppy.com