INVESTPUPPY UNVARNISHED

InvestPuppy

Nobody owns it

*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.

Ask a vendor who owns the implementation.

They will say the client.

Ask the client.

They will say the vendor.

Both are right. That is the problem.

1. How It Looks From The Inside

At signing, the ownership question does not feel like a question. The vendor has a delivery methodology. The client has a project sponsor. The contract specifies deliverables and timelines. There is a project plan. There are weekly status meetings. There is, on paper, a clear structure.

By week six, the ownership question has become the only question that matters. The decision that needed to be made last week has not been made. The sign-off that would unlock the next phase of configuration has not arrived. The escalation path that was supposed to resolve the integration question leads to someone who leads to someone else who is waiting for clarity from the vendor.

Nobody planned for this. Nobody needed to. The ownership vacuum assembles itself from the normal operation of both parties' incentives, without a single bad decision being made by anyone.

2. The Blame Architecture

The blame architecture assembles itself before the project begins. It is constructed from two perfectly rational decisions made independently by two parties who would both say they want the project to succeed.

The vendor does not assign a named individual with authority over the implementation outcome. The vendor assigns a project manager with authority over the vendor's deliverables. These are not the same thing.

The client does not assign a named individual with authority over the implementation outcome. The client assigns a project sponsor with authority to sign off on milestones. These are not the same thing.

The gap between these two arrangements is where implementations fail.

> *Ask a vendor who owns the implementation. They will say the client. Ask the client. They will say the vendor.*
>
> *Both are right. That is the problem.*
>
> *Vendors train their sales teams to build relationships, not accountability structures. Clients promote the people who don't own things that fail. Nobody planned this. Nobody needed to.*
>
> *When things go wrong, the vacuum takes the blame. Not a planning failure. Not a communication failure. A rational response to a rational incentive — from both sides, without a word spoken.*
>
> ***The project belongs to no one.***

3. The Regulatory Dimension

In a regulated financial services context, unclear implementation ownership is not merely an operational inconvenience. It is a compliance exposure.

The audit trail for a financial software implementation — who approved what, when, and on what basis — is not a document that can be reconstructed after the fact. It requires clear ownership at every stage. When the question of who approved the go-live decision, who signed off on the integration testing, and who validated the output of the first live allocation cannot be answered clearly, the audit trail does not simply have gaps. It has the specific kind of gaps that compliance officers and regulators are trained to find.

Unclear implementation ownership, in a regulated environment, is a risk that is rarely priced into the project at signing.

4. What Ownership Actually Looks Like

Ownership is not a title on an org chart. It is the answer to a specific question: who has the authority and the accountability to make this implementation succeed, and who will bear the consequences if it does not?

Three things need to be true before configuration begins.

One. A named individual on the client side has the authority to make workflow decisions without referral upward. Not to report to someone who will make those decisions. To make them.

Two. A named individual on the vendor side has the authority to commit the vendor's resources to resolving issues that arise during implementation. Not to escalate to someone who has that authority. To exercise it directly.

Three. Both individuals have agreed, explicitly, on what success looks like — not in commercial terms, but in operational terms.

Own it. Before the configuration begins. Before the relationship makes it uncomfortable to ask.

Before week six.

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)
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) (you are here)
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