INVESTPUPPY UNVARNISHED

InvestPuppy

# The parameter that broke the camel's back

*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.*

About InvestPuppy

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

## 1. The Demo

The platform does everything. You watch it screen a universe, optimise a portfolio, allocate capital, generate an order file. You ask whether it supports your market, your fee structure, your client-specific risk parameters. The answer is yes. You ask whether it handles multiple clients with different configurations. Yes. You ask whether it connects to your custodian. Yes.

The demo is a single-client, single-market, single-configuration environment. It was built to answer yes. It was not built to survive client 3.

This is not deception. The platform does do those things — in the sense that the capability exists somewhere in the system, or in the vendor's roadmap, or in the professional services team's backlog. The demo shows the answer as if it were already built. The client does not have the vocabulary to ask whether what they are watching is architecture or aspiration.

Nobody explains the difference. Nobody needs to. The demo closes contracts.

## 2. What Configurable Means

Every serious financial software platform is configurable. This claim appears in every brochure, every sales deck, every RFP response. It is not false. It is incomplete.

The question the claim does not answer: configurable by whom?

There are two kinds of configurable. The first kind lives in data. A parameter table, an admin portal, an API endpoint. An operations manager adds a new market, adjusts a fee threshold, sets a client-specific risk limit. No

developer is involved. No ticket is raised. No deployment is required. The change is live in minutes.

The second kind lives in code. A developer opens a file, adds a conditional, writes a test, raises a pull request, waits for review, waits for deployment. The operations manager raises a ticket. The ticket enters a queue. The queue has a backlog. Weeks pass.

Both are called configurable. They are not the same thing. The brochure does not specify which one you are buying.

## 3. The Overheard Remark

Some years ago, a senior manager at one of the platforms named in this document was overheard in a conversation about how to bring new consultants up to speed.

> "Read the system parameters document — there are about 400. Six are fundamental to how the system works. Around 20 are important in configuring for a specific client. The other 400 are there to confuse junior consultants."
>
> — A senior manager at one of the platforms named in this document

This was not institutional policy. It was not a training manual. It was said quietly, in passing, by someone who had been inside the system long enough to know. Said because the room seemed safe. Not intended to be heard.

What it describes is not a documentation problem. It is a dependency model. Four hundred parameters create a surface area that cannot be navigated without institutional knowledge. That knowledge lives with the vendor's senior consultants. It does not transfer to the client. It does not need to, from the vendor's perspective. The client's inability to navigate the surface area is what makes the senior consultant billable.

The complexity is not incidental. It is load-bearing.

## 4. The Number That Grew

That remark was overheard approximately twenty years ago.

***The same product today has over 700 parameters.***

The moat did not shrink. It grew. Each additional parameter represents a client requirement that could not be handled by the existing configuration — a gap patched with a new entry rather than a structural solution. Each one a scar from a previous implementation. Each one also a new source of confusion for the next junior consultant, and a new reason to call the senior one.

Seven hundred parameters is not depth. It is sediment.

The platform was configurable. By the vendor. On the vendor's timeline. At the vendor's day rate.

## 5. The Boundary That Was Never Drawn

Every institutional implementation contains weeks — sometimes months — of what participants diplomatically describe as lively discussion about where the boundary sits between configuration and customisation.

Configuration is included in the contract. Customisation is additional.

There is no industry definition of the boundary between them. There has never been one. Ask any two vendors to define it and you will receive two different answers. Ask the same vendor at the start of a project and at the end of a change request conversation, and you may receive two different answers from the same person.

The lively discussion is not a misunderstanding. It is a negotiation with no external reference point. The vendor holds the reference. The vendor wrote the contract. The vendor's senior consultant is the one who knows which six parameters actually matter.

**The absence is structural.**

A clear industry definition of the configuration/customisation boundary would benefit clients. It would allow independent assessment of whether a requested change falls within the contracted scope. It would limit the vendor's ability to reclassify included work as billable additional work.

The definition has not emerged. The party with the greatest commercial interest in its absence has also had the greatest influence over the industry conversations in which it might have been produced.

Nobody planned for the definition not to exist. Nobody needed to.

## 6. What Client 3 Looks Like

Client 1 onboards. There are accommodations — a few parameters adjusted, a couple of manual workarounds, some things that will be tidied up in a future release. The vendor is close. The relationship is warm. The platform works.

Client 2 requires a different fee structure. A different market. A different rebalancing rule. Each difference is accommodated with a code change, a conditional, a configuration entry that didn't exist before. The workarounds from client 1 are still there. The platform still works.

By client 3, the codebase carries three clients' worth of if-statements, special cases, and accumulated debt. The vendor is no longer close. The gaps that were the vendor's problem are now the client's problem. The manual accommodations from client 1 are now operational dependencies. The feature the client thought was included is now a customisation.

The demo never had a client 3. The demo was a single-client environment. Client 3 was never in the room when the contract was signed.

**The quiet conversation.**

At some point — client 3, client 4, sometimes earlier — someone inside the vendor's engineering team is having a quiet conversation about whether to refactor the codebase or rebuild it. This conversation is the real cost of building for the demo. Not the workarounds. Not the change requests. Not the lively discussions about configuration versus customisation.

The rewrite. It was predictable from the first architectural decision. The demo never revealed it because the demo was not built to survive it.

> At 700 parameters, you are not a client.
>
> You are a hostage.
>
> We built the parameter layer before we had a second client. Because we have seen what client 3 looks like.

# 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)**
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) (you are here)**
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*