INVESTPUPPY UNVARNISHED

InvestPuppy

Model Bank — the perfect solution for banks that don't exist

*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 model bank is a composite of institutions that do not exist. Its defaults were not derived from your institution. They were derived from someone else's institution, filtered through the vendor's understanding of it, and configured before your institution was in the conversation.
>
> The configuration is not done. It has been deferred.

1. The Promise

The pitch is seductive. Instead of a blank-canvas implementation — months of discovery, configuration workshops, parallel testing — you take delivery of a system that has already been configured. Best practice embedded. Industry-standard workflows pre-built. The hard decisions already made. A bank, fully formed, waiting for your name on the door.

The model bank. Faster. Cheaper. Lower risk. The vendor has already implemented this platform for other institutions. The common patterns have been extracted. The configuration is done. You are not building from scratch — you are inheriting from experience.

This pitch has closed a very large number of contracts.

It has also produced a very large number of failed implementations. The connection between these two facts is not accidental.

2. What a Model Bank Actually Is

A model bank is a set of defaults. It is the configuration the vendor chose when they had to choose something — because a system cannot be demonstrated without being configured, and a configured system looks more credible than a blank one.

The defaults were not derived from analysis of the institution buying the platform. They were derived from the institutions the vendor has previously implemented — or, more precisely, from the vendor's understanding of

those institutions, filtered through the decisions made by whoever configured the demonstration environment.

These defaults represent something. They represent a plausible financial institution of the type the vendor has worked with before. They do not represent your institution. They have never been in the same room as your institution.

The gap between defaults and requirements.

The gap between a model bank's defaults and a specific institution's requirements is not a gap of detail. It is a gap of kind. A default chart of accounts reflects someone's accounting decisions. A real institution's chart of accounts reflects that institution's regulatory environment, ownership structure, product set, reporting requirements, and accounting history. These are not the same thing in different quantities. They are different things.

Every workflow in the model bank was designed for an institution that does not exist. The model bank is a composite — a weighted average of prior implementations, smoothed into something that offends nobody and fits nobody precisely.

> *It is institutional furniture from a showroom.*
>
> *It looks correct.*
>
> ***It is not yours.***

Some requirements can be approximated. Regulatory obligations cannot.

A model bank can encode a plausible approach to how decisions are made. It cannot encode the specific legal framework under which those decisions must be made — because that framework is not an average. It is a specific set of obligations, in a specific jurisdiction, under a specific licence. The model bank was not configured for your licence. It was configured for someone else's.

The model bank has two regulatory failure modes, not one. The first is the gap: regulatory obligations specific to the institution are absent because the model bank was built elsewhere. The second is contamination: regulatory configurations from the model bank's home jurisdiction are present and active in a context where they do not apply. The institution discovers the gap at go-live. The institution may not discover the contamination at all.

An institution takes delivery of a model bank configured for European commercial banking. The transaction monitoring system is pre-configured with thresholds derived from EBA guidelines. The institution is licensed in Singapore. MAS Notice 626 specifies different thresholds, different customer risk categories, and different escalation requirements.

The model bank is not just missing the MAS configuration. It is running the wrong configuration, producing monitoring outputs that do not satisfy MAS requirements, and generating no alerts about this discrepancy — because the system is configured, it is running, and it looks correct.

There is no average compliance.

The regulation changed. The model bank didn't.

The gap and the contamination are present at go-live. The third failure mode arrives later — silently, without announcement, and without anyone necessarily noticing.

Regulations are amended continuously. The model bank's configuration is not. From the moment of go-live, the gap between the configuration and the current regulatory environment begins to widen. The system continues running. It produces outputs. It generates reports. It files returns. And it is wrong — not because it was never right, but because it was right once and the regulatory environment moved.

The ownership vacuum compounds this. The vendor assumes the client manages their own regulatory compliance. The client assumes the vendor's 'regulatory-compliant system' stays compliant. A regulation is amended. The amendment is published. The compliance team reads it. They raise a request with IT. The request enters a queue. The queue has a backlog. Months pass. The system continues running the pre-amendment configuration. No bad decision was made. The process produced non-compliance through normal operation.

A model bank is wrong in three dimensions. It was configured for a different institution. It was configured for a different regulatory environment. And it was configured at a different point in time — before the most recent regulatory update, and before the one after that.

The gap widens silently, from the moment of go-live, without anyone watching.

3. The Configuration That Was Never Done

The model bank does not eliminate the configuration requirement. It defers it, disguises it, and — in the worst

cases — assumes it has been completed when it has not.

The implementation begins. The vendor demonstrates the platform. It works. Workflows run. Reports generate. The system processes transactions. The client team, who have never seen a financial platform implemented from scratch, observes a functioning system and draws the natural conclusion: it is configured. The hard work has been done.

It has not been done. The system is demonstrating its defaults. The defaults will not survive contact with production data, real client accounts, actual regulatory requirements, or the specific operational structure of the institution implementing it.

The discovery that the configuration has not been done arrives in one of two ways. The first is in UAT, when testers attempt to process real transactions and encounter systematic failure. The second is at go-live, when real users encounter systematic failure at production volume. Neither is the right time to discover that the configuration requirement was not understood to exist.

The best practice that wasn't.

The model bank carries a specific claim: that its configuration represents best practice. This claim does substantial work in the sales process. It positions the vendor's defaults not as arbitrary starting points but as validated decisions — the distilled wisdom of multiple implementations.

The claim is difficult to evaluate and rarely evaluated. Who determined what best practice is? For what regulatory environment? In what market? For an institution of what size, what product mix, what client profile? The best practice embedded in a model bank built for a mid-size European commercial bank is not best practice for a private bank in Singapore. It is not even best practice for a different mid-size European commercial bank with a different product mix.

Best practice is context-dependent. A model bank eliminates the context.

4. Why the Myth Persists

The model bank myth persists because it serves the interests of everyone involved in the procurement conversation — except the institution that will eventually have to operate the system.

The vendor benefits because the model bank shortens the sales cycle. A demonstration is faster than a blank-canvas discovery process. It produces a tangible artefact. It closes contracts.

The implementation team benefits because the model bank provides a starting point that looks like progress. Configuration from a baseline is easier to defend than configuration from scratch. The decisions embedded in the baseline are someone else's decisions. If the baseline turns out to be wrong, the starting point is responsible.

The client's procurement team benefits because the model bank provides a reduction in apparent risk. A pre-configured system looks less risky than an unconfigured one. The procurement team is measured on the purchase decision, not on the operational outcome. A compelling demonstration that closes quickly and within budget is a success by procurement metrics — regardless of what happens eighteen months later.

The incentive structure, again.

Nobody in the procurement conversation is incentivised to say: this model bank does not reflect how our institution operates, and before we proceed we should understand exactly what that means for the implementation.

That conversation is slow. It is uncomfortable. It requires the client to admit they do not know their own processes well enough to evaluate the gap. It requires the vendor to acknowledge that the demonstration environment is not the same as a configured production system. It requires the implementation team to commit to a discovery phase that delays the configuration work they are being paid to do.

Nobody planned for this conversation not to happen. Nobody needed to.

The vendor moves on. The client does not.

When the implementation closes — on time, within budget, with the model bank delivered and signed off — the vendor moves to the next engagement. The implementation team closes the project. The gap, the contamination, and the widening obsolescence remain. They remain with the institution, which will operate this system for the next decade, under the regulatory framework the model bank was not configured for, discovering the configuration gaps one by one as they surface in production.

The people who benefit from the myth are not the people who live with the consequence. This asymmetry is not incidental. It is structural.

5. The Institutions That Don't Exist

Every model bank is a solution for an institution that does not exist.

Not because the vendor invented it — but because the averaging process that produces a model bank, by mathematical necessity, produces something that matches no specific institution exactly. The model bank is optimised for the centre of the distribution. Real institutions live in the distribution, not at its centre. Every institution has specific characteristics — specific regulatory obligations, specific product structures, specific client relationships, specific operational constraints — that deviate from the average. These deviations are not edge cases. They are the institution.

A model bank built for a European commercial lender comes pre-configured for syndicated loan processing. A Singapore private bank does not process syndicated loans. The workflow exists in the system, fully configured, for a product the institution does not offer. The configuration is not neutral — it consumes system resources, appears in user interfaces, generates compliance documentation for transactions that do not occur, and requires active management to suppress.

The institution did not ask for it. The model bank assumed it.

This is why every model bank implementation eventually requires a discovery phase. Not because the discovery phase was planned — in most cases it was not. But because the gap between the model bank's defaults and the institution's actual requirements becomes impossible to ignore at some point in the implementation. The question is only when the gap is discovered, and how much has been built on top of the wrong foundation before it is.

6. What Thoughtful Implementation Looks Like

The alternative is not the absence of a starting point. It is a different question about what the starting point is for.

A financial system implementation begins with a question the model bank sales process is specifically structured to avoid: how does this institution actually operate?

Not how it is supposed to operate. Not how its process documentation says it operates. How it actually operates — the workflows that run in production, the decisions that are made by feel rather than by rule, the exceptions that have become standard practice, the regulatory requirements specific to this institution's licence, structure, and market.

This question takes time to answer. It requires conversations with the people who run the processes, not the people who designed them. It produces a specification that the vendor's demonstration environment cannot match — because it describes a specific institution, not a composite one. The configuration that results will not look like the model bank. It will look like the institution. That is the point.

The model bank as a starting point.

There is a legitimate use of a model bank: as a starting point for discovery, not as a substitute for it. A pre-configured demonstration environment can accelerate the identification of gaps — if the client team is using it to find where the defaults diverge from their requirements, rather than assuming the defaults are correct.

This requires an explicit conversation at the start of the engagement: the model bank represents a plausible institution, not your institution. Every default is a question, not an answer. The implementation work is the work of answering those questions correctly — for this institution, in this market, under this regulatory framework.

This conversation rarely happens. When it does, the implementation has a fighting chance.

7. The Test

Before accepting delivery of a model bank — or any pre-configured system — three questions. Not as a framework. As a minimum.

Which defaults in this system reflect decisions specific to my institution?

If the answer is 'all of them', the model bank has not been configured. It is a demonstration environment. The implementation work has not started.

Which workflows were designed for an institution with my specific regulatory obligations, product mix, and client profile?

If the answer requires more than a sentence, the gap between the model bank and the institution has not been assessed. Stop. Assess it. The implementation can wait. The gap cannot be allowed to close at go-live.

If this system's defaults are wrong for my institution and we discover that at production volume, what is the remediation cost?

If the answer is uncomfortable, it should have been the first question in the procurement conversation — not the last question in the implementation.

The model bank is a composite of institutions that do not exist.

Your institution does exist.

The configuration is not done.

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) (you are here)
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