
INVESTPUPPY UNVARNISHED

InvestPuppy

# The U in UAT — kind of important...

*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 UAT sign-off document has the right signatures on it. The signatories were not users of the system. They were consultants working from test scripts written by other consultants, testing a configuration reviewed by a project team that had not spoken to the people who would use it in production. The document says PASS. The system goes live. Three weeks later, the people who use it every day explain, at length, why it does not.***

## 1. What the U Is For

User Acceptance Testing. Three words. Each one is load-bearing.

*Testing* means that something is being evaluated against a standard. *Acceptance* means that the evaluation results in a judgment — this works, or it does not. *User* means that the person making that judgment is the person who will use the system in production, on real transactions, under real conditions, with real consequences for getting it wrong.

Remove the user from User Acceptance Testing and you have acceptance testing. Which is a different thing. A system can pass acceptance testing and fail UAT. This happens. It is, in fact, one of the most common outcomes in financial software implementation.

## 2. How the Outsourcing Happens

The decision to outsource UAT is rarely announced as a decision. It arrives as a resource allocation, a timeline pressure, or a project management convenience.

The users are busy. The system is going live in six weeks. The project team has UAT test scripts. The consulting firm has testers. The testers are available. The users are not. The project manager makes a practical decision: the consultants will run the UAT.

> The users are busy because the organisation has not protected their time for UAT. The organisation has not protected their time for UAT because UAT has been treated as a project team activity rather than an operational activity. The resourcing constraint that makes consultant-led UAT feel necessary is a planning failure, not a fixed condition.

## 3. What the Consultant's UAT Actually Tests

A consultant running UAT with a test script is not testing whether the system works. They are testing whether the system produces the outputs described in the test script, given the inputs described in the test script, in the sequence described in the test script.

This is a narrower question. It excludes, by design, everything that is not in the test script. It excludes the transaction that occurs twice a month. It excludes the workflow exception that the operations team handles manually. It excludes the report that the risk team runs every Friday.

> *The test script was written from the specification. The specification was written from the workshops. The workshops were attended by the stakeholders who were available.*
>
> *The users who were not available did not attend the workshops, did not shape the specification, and did not write the test scripts. A consultant running those test scripts is not testing whether the system works for the users. They are testing whether the system is consistent with a document that the users had limited input into.*
>
> ***The U was removed long before the UAT phase began.***

## 4. What Real UAT Looks Like

Real UAT is operationally inconvenient. It requires the users to stop doing their jobs for long enough to test the system that will change their jobs. It requires test scripts that were written with user input, not only from the specification. And it requires someone with the authority to say, without commercial consequence, that the system is not ready.

Three things that make UAT real rather than documentary:

**One.** Users — the actual people who will use the system in production — run the tests. Not a representative, not a delegate, not a consultant with a test script.

**Two.** The test scripts include the transactions that are not in the specification. The exceptions, the edge cases, the processes the users know exist and the project team does not.

**Three.** The sign-off authority rests with the users, not with the project team. A UAT phase in which the project manager can override a user's rejection of a test case is not a UAT phase. It is a documentation exercise.

**User Acceptance Testing.**

**The word 'user' is not decorative.**

# 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) (you are here)**
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*