NEXT313

DISCOVERY DELIVERABLE · ILLUSTRATIVE EXAMPLE

Sample Discovery *Findings.*

An illustrative example of how a Next313 Discovery uncovers the real, structural problem in a business — before a single line of code is written.

Representative sample. Names, figures, and identities are anonymized — this is not a live client file.

THE DISCOVERY QUESTION

A working problem, brought *before* a build.

A small internal team came to Next313 with friction in how their work actually moved — not a feature list, and not a decision to build anything yet. Discovery started where it always does: by listening to how the work happens.

We listened first

Interviews and process walks with the people doing the work daily — where the day slows down, where the same task is redone, where information goes to hide.

We looked for the pattern

Not the surface complaint, but the structural reason it keeps recurring. A stated problem is a symptom; Discovery is the work of naming the cause.

> *"The request was a better place to put files. The finding was that two different kinds of work had been quietly sharing one shape — and neither fit."*

THE PROBLEM WORTH SOLVING

Three teams, *one* shared gap.

The same structural friction showed up across three very different working contexts. Each had improvised a different workaround; none had solved the underlying problem.

A production team

Recurring client relationships — recordings, deliverables, approvals — where the same friction repeats every cycle, and each new cycle starts by rebuilding context that already existed.

A creative team

Brand-level assets that are always true, managed alongside project-specific work that is not — flattened into the same folder structure until neither is easy to find.

An independent operator

A body of work that collapses into one undifferentiated drive folder or chat thread, with nothing findable by type, by client, or by project.

THE SHARED STRUCTURAL GAP

There was no persistent, “always true” layer kept separate from the recurring, “this cycle’s work” layer — and no shared space that worked for both sides of a client relationship without one side bending around the other.

THE STRUCTURAL INSIGHT · THE CENTERPIECE FINDING

Every recurring relationship has *two layers* of work.

This was the finding, not a feature. One layer persists across the whole relationship; the other resets every cycle. The tools teams reach for — general storage, flexible workspaces, task trackers — do not enforce that distinction, so the two layers keep collapsing into one.

Persistent layer

ALWAYS TRUE ACROSS THE RELATIONSHIP

- Identity, brand, and standing agreements
- Reference material reused every cycle
- The context a new collaborator needs on day one

Recurring layer

THIS CYCLE'S WORK, THEN THE NEXT

- Deliverables and approvals for one cycle
- Working files that close out when it ships
- Status that should not clutter the next cycle

ACCESS IS A PROPERTY OF **BOTH** LAYERS

A temporary collaborator needs scoped access that spans both layers at once. Most tools force a bad trade — oversharing everything, or over-managing permissions by hand. Naming the two layers is what makes scoped access possible.

VALIDATING AGAINST THE MARKET

The market already tried — and *missed* the seam.

A good insight should explain why existing tools fall short. Each category below solves part of the problem; none enforces the persistent-versus-recurring distinction that the two-layer finding names.

CATEGORY OF TOOL	WHAT IT GETS RIGHT	WHAT IT MISSES
General cloud storage	✓ Familiar; anyone can find a file by folder	✕ No idea what is persistent versus per-cycle; access is all-or-nothing
Digital-asset and brand managers	✓ Strong on the persistent, always-true layer	✕ Weak on recurring, per-cycle production work
Flexible workspace tools	✓ Adaptable to almost any structure	✕ Adaptable to the wrong one too; the distinction is left to discipline
Agency client portals	✓ Built for two-sided client relationships	— Portal-first, not work-first; the two layers still blur inside

The gap is not a missing feature in any one tool. It is that none of these categories treats the two layers as structurally different — which is exactly the seam the Discovery surfaced.

DE-RISKING THE BUILD

Prove it inside a *committed* environment first.

Discovery does not end at an insight — it ends at a plan that lowers the risk of building the wrong thing. The recommendation was to pilot inside an already-committed environment before any external launch.

Requirements from real users

The first users are already committed, so the specification comes from how people actually work — not from assumptions written down in a planning document.

Tested on real work

The pilot runs against live production work, not synthetic test cases, so the two-layer model is validated where it has to hold up.

Adoption risk, removed early

Because the first users have a reason to use it, low adoption cannot quietly mask a design that does not fit.

GENERALIZED ACROSS TWO DOMAINS

The same insight was pressure-tested in two different pilots — one production-focused, one campaign-focused — to confirm the two-layer model held across domains without a redesign. An insight that only fits one team is not structural.

PATH FROM DISCOVERY TO LAUNCH

Three phases, each *earned* by the last.

Discovery hands over a staged path, not a launch date. Each phase has to prove itself before the next begins, so risk is retired in order.

01

Internal validation

Prove the core against real workflows inside a committed environment. Requirements and refinements come from live use, over a span of weeks rather than a fixed calendar.

02

Controlled external launch

Open to a small, warm cohort who already understand the problem. Learn where the model bends for teams outside the original context — before scale makes changes expensive.

03

Broader market entry

Once the core is proven and adoption is real, widen access. The sequence means every later decision is grounded in usage, not assumption.

> PRICING PHILOSOPHY, IN BRIEF
>
> Price to the unit that scales with the value delivered, and make capacity — not access to features — the thing that differentiates one plan from the next. Discovery names the pricing shape; it does not lock a number before the value is proven.

WHAT THIS MEANS FOR YOU

This is what Discovery looks like *before* a single line of code.

A Next313 Discovery finds the structural pattern underneath a stated problem, tests that pattern against how the market already tries to solve it, and stages the build so real usage — not assumption — drives every decision that follows.

- **Find the structural pattern**
- **Test it against the market**
- **Stage the build around real use**

Let us find the structural insight in your business.

Every engagement starts with a paid Discovery. Tell us how your work actually moves, and we will tell you whether Discovery is the right next step.

launch@next313.com

next313.com