Neuphlo

USE CASE · QA & TEST CONSULTANCIES

Test design as a discipline, not a spreadsheet.

How QA teams and test consultancies run structured test design in Neuphlo — equivalence partitioning, boundary values, decision tables, state transitions and pairwise — with execution history and an agent that reviews coverage.

6 design techniques | Runs & pass rates | QA Agent

01 — THE PROBLEM

§ Where test design goes to die

Real test design — partitions, boundaries, decision logic — mostly lives in Excel. It's invisible to the team, disconnected from the work it verifies, and rots the day the requirements change.

Test management tools track execution but treat design as a text field. Spreadsheets hold the techniques but nothing else. And the development work the tests exist for lives in a third system entirely. For a consultancy, that fragmentation is billable hours lost to glue work — and for any QA team, it's why coverage erodes quietly.

Neuphlo makes testing a **first-class module** in the same workspace as the tasks, conversations and documentation: designs built with real techniques, cases generated from them, runs recorded against them, and an agent that reviews the gaps.

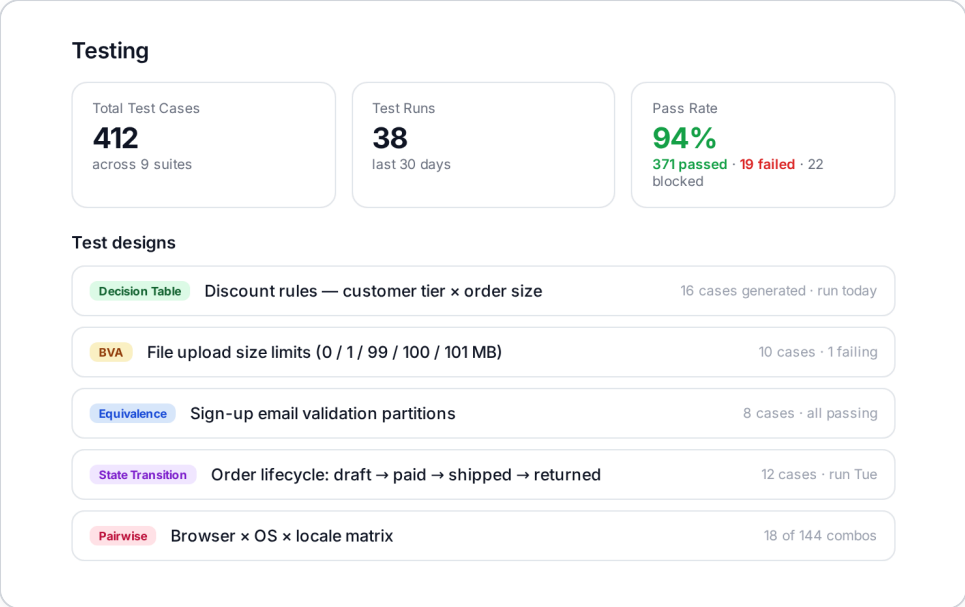
6	1	history
technique editors: EP · BVA · DT · ST · pairwise · use case	click from failing test to bug task	per-case result timeline across runs

In this paper: the Testing module's dashboard, designs and suites; a decision table that generates its own cases; and the QA Agent that pressure-tests coverage. Screenshots are taken directly from the product.

02 — THE MODULE

§ Designs, cases, suites, runs

The Testing module is organised the way testers actually think: designs hold the technique work, cases are generated from designs, suites group them, runs record execution.



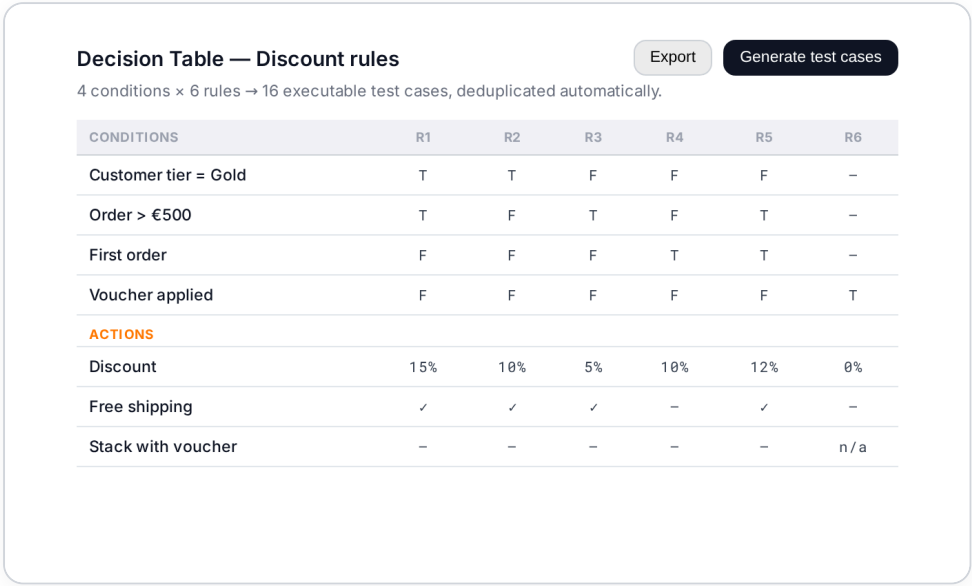
The testing dashboard. Case counts, run volume and pass rate up top; below, each design carries its technique — decision table, BVA, equivalence, state transition, pairwise.

Every case keeps a **result history** across runs, so flaky areas show themselves as patterns rather than anecdotes. Pass, fail and blocked counts roll up to the dashboard where a test manager — or a client — can read the state of quality at a glance.

03 — TECHNIQUES

§ The techniques, executable

Each technique has a dedicated editor that does the combinatorics for you — the decision table below becomes sixteen executable cases with one click.



Decision table editor. Conditions and actions as structured data; case generation and export built in. BVA, equivalence partitioning, state transition, pairwise and use-case editors follow the same pattern.

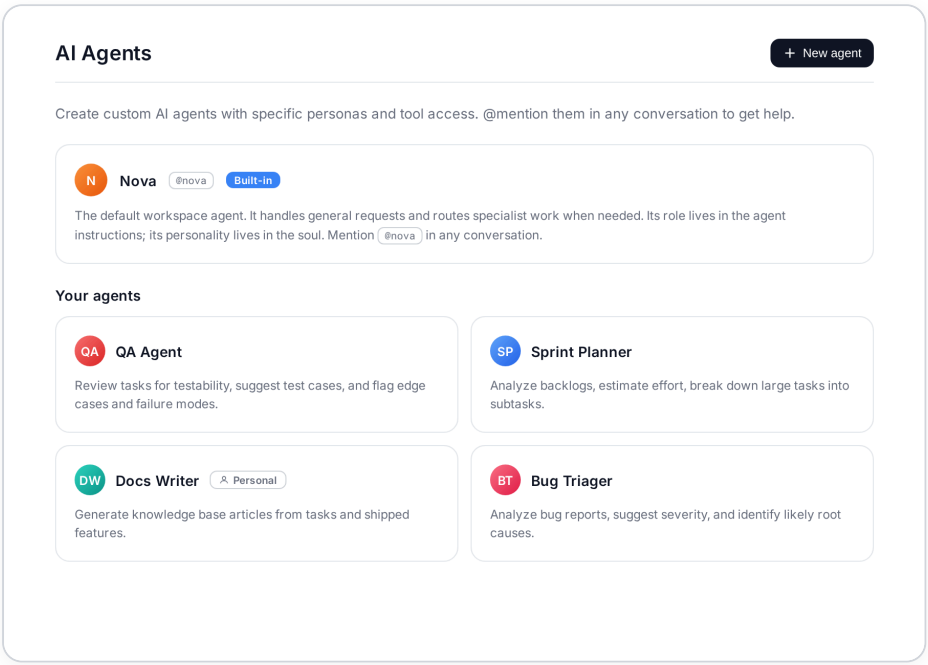
Because the design is data rather than prose, it stays alive: change a condition and regenerate, instead of reverse-engineering last quarter's spreadsheet. Pairwise narrows a 144-combination matrix to a covering 18; state transition editors turn lifecycle rules into paths. For consultancies, exports give clients the deliverable they expect — while the living version stays in the workspace.

04 — THE AGENT

§ A QA specialist on every task

The QA Agent reviews work items the way a senior tester would — before anyone writes a line of test code.

Mentioned on a task, it analyses the description and acceptance criteria for testability, generates concrete cases across happy path, boundaries, error states and security, flags ambiguity, and checks related work for existing test patterns via semantic search. Its instructions demand specificity: not "test the login", but credentials, expiry, injection.



Specialists on demand. QA Agent and Bug Triager ship as templates — tool-scoped, tunable, and @mentionable on any task in the workspace.

Paired with the **Bug Triager** — severity with justification, duplicate detection, escalation — the routine analytical work around testing is handled, and the testers keep the exploratory and risk-based work that actually needs them.

05 — CONNECTED QUALITY

§ Tests that know what they verify

Because testing lives beside tasks and knowledge, the traceability that consultancies normally build by hand is just how the workspace works.

- **Failing case → bug task** in one step, with the run context attached; the fix closes the loop on the same board.
- **Designs link to requirements** — the spec page, the task and the test design reference each other, so coverage questions have answers.
- **Documentation by agents** — Docs Writer turns recurring findings into knowledge articles; Release Notes Writer reports what shipped and what was verified.
- **Client-ready reporting** — pass-rate trends and run history come from the dashboard, not from a Friday-afternoon spreadsheet merge.

For consultancies specifically

NEED	HOW IT LANDS
Separate clients	Separate workspaces — isolated data, members and agents per engagement
Client confidentiality	BYOM per workspace; sensitive engagements pinned to local models or self-host
Deliverables	Design exports plus a published knowledge site for test documentation
Repeatable method	Agent templates and workflow templates carry your methodology between engagements

06 — TRUST & GETTING STARTED

§ Rigour, with receipts

- **Auditable agent work.** Every QA Agent suggestion is a visible run with steps and reasoning — reviewable like a junior's work, at a fraction of the turnaround.
- **Guardrails.** Agents suggest and comment by default; what they may change is bounded by tool scope and confidence thresholds.
- **Your data, your model.** Client systems under NDA never need to touch a third-party model — local Ollama on self-host keeps the whole loop in-house.

§ Getting started

Enable the **Testing** app (with Tasks and Knowledge), pick the Bug Tracking workflow template, and add the QA Agent. Start with one real feature: design a decision table for its rules, generate the cases, run them — and let the agent review what you missed.

> **In short:** Neuphlo treats test design as engineering — structured techniques, generated cases, recorded runs, agent-reviewed coverage — in the same workspace as the work under test.

Neuphlo

Learn more → **neuphlo.com**