PERINIMBLE · WHITE PAPER · GRID DATA FAMILY

Testing as a Release Gate for Utility Software

A release should ship because the tests passed, and everyone should be able to see why they did.

PRODUCT	Grid Data Testing Platform
AUDIENCE	QA leads and testers, release and DevOps engineers, IT managers

Every release touches the bill

Utility software carries real consequences. A change to a meter data interface can affect validated reads, billing determinants, settlement files and the numbers regulators see. A regression that slips through does not stay in a test environment. It shows up in customer bills, in exception queues, and in the weeks of cleanup that follow.

Yet testing is often the least automated part of the release. Test cases live in spreadsheets. Regression runs happen by hand, late in the cycle, when time is shortest. The automated tests that do exist were written by a few engineers, and only they can maintain them. The pipeline builds and deploys the software but does not ask whether it still works. So releases go out on judgment and hope, and everyone braces for the first bill run afterward.

Why testing utility software is hard

Utility systems expose many APIs, and the behavior that matters rarely sits in one of them. A realistic test is a business flow: create a service point, install a meter, load a day of reads, request validated data, and check the result. Each step depends on values produced by the step before it, such as an identifier created at the start and needed at the end. Testing endpoints one at a time misses exactly the failures that hurt.

Test suites drift. When a case is copied into several suites, a fix to one copy leaves the others stale, and a suite that once meant something quietly stops meaning it.

Results often arrive as a bare pass or fail. When a run fails, someone has to reproduce it to understand it. When a run passes, nobody can say exactly what it proved.

Much of the knowledge about how the business should behave sits with functional testers and analysts who do not write code. If every test must be programmed, their knowledge reaches the test suite slowly, or not at all.

Principles of a good approach

Build tests from the API's own contract. Testers should build steps in the browser from what the API already offers, rather than typing requests by hand or writing code.

Test business flows, not just endpoints. A test case should run several steps in order, with later steps building on what earlier steps created, and a flow built once should be reusable as the setup for another.

Reuse without duplication. A test case should be shared by every suite that uses it, so a fix made once applies everywhere.

Make the pipeline ask. The CI/CD pipeline should be able to run a suite against the environment it is testing and get back a clear pass or fail. A failing suite should stop the release. That is what turns testing from an activity into a gate.

Show results as they happen. Teams should be able to watch cases pass and fail while a run executes, not wait for a report at the end.

Keep reports that explain. Every run should keep its step-level execution flow, request logs, generated identifiers and assertion failures. A failed run should tell a developer what went wrong without a second attempt to reproduce it. Dashboards should show totals and trends over time.

Let AI draft, and let testers refine. Describe a test in a sentence, and AI drafts the executable steps. The tester reviews the draft, adjusts assertions, and saves it. The tester stays in charge of what the test proves.

Work for any API. A utility's release touches its own services and its vendors' services. The same builder, suites and reports should work for any API a team registers, not only one vendor's products.

What it looks like in practice

> **ILLUSTRATIVE EXAMPLE (HYPOTHETICAL)**
>
> A utility is preparing a release of the interface that sends validated reads to its CIS. The QA lead builds a test case in the browser from the interface's published contract. Step one creates a test service point. Step two loads a day of interval reads against it. Step three requests the validated data and asserts on the status and the totals. Step four requests the billing determinants and checks that they match. Each later step works with that same service point.
>
> For a new scenario, a functional tester types a sentence describing a meter exchange partway through the billing period. The AI assistant drafts the steps. The tester corrects one assertion and adds another, then saves the case to the same suite. Because cases are shared across suites, a later fix to the service point setup applies everywhere that setup is used.
>
> The release pipeline runs the suite against the release candidate environment. The team watches the run live. The meter exchange case fails. The report shows the step that failed, the request that was sent, the response that came back, and the assertion that did not hold: a field had been renamed in the release candidate. The pipeline stops. The developer fixes the change the same day, the suite passes, and the release goes out.
>
> After deployment, the same suite runs again as a smoke test against the freshly deployed environment. It passes, and the report is kept with the release record.
>
> Nobody found this regression in a bill run. The gate found it, and the report explained it.

Questions to ask any testing platform vendor

- Can testers build multi-step API tests in the browser from the API's own contract, without writing code?
- Can later steps build on earlier ones, and can one flow serve as setup for another?
- When we fix a test case, does every suite that uses it pick up the fix?

- Can our pipeline run a suite against any environment and get pass or fail back?
- Can we watch a run live, and does every run keep step-level logs and assertion failures?
- If AI drafts tests, can testers review and edit every step before it is saved?
- Does it work for any API we register, or only the vendor's own products?

Releases that prove themselves

The Grid Data Testing Platform lets QA teams build API tests in the browser, chain whole business flows, curate reusable suites, watch runs live, and gate every release pipeline on the result. It comes with ready-made test modules for the Grid Data Management Platform and Grid Data Simulator, and it works just as well for any other API a team registers.

Learn more about the Grid Data Testing Platform, or talk to our team about turning your regression suite into a release gate.

> **ABOUT PERINIMBLE**
>
> PeriNimble builds mission-critical grid data software for energy and utility companies. Founded in 2022 and headquartered in Bloomington, Minnesota, we support more than 15 million meters and over 1 billion interval reads per day in production.
> **Built for systems that can't afford failure. · perinimble.com · perinimble.com/contact**