

Testing the Plan: DR Exercises and the After-Action Report

Mon, Aug 24, 10:00 AM EDT · <https://scottslab.io/posts/dr-testing-and-after-action-reviews>



TL;DR

Five test types climb in rigour and risk, from read-through to full interruption. But the honest problem is that almost every DR test is announced, scheduled in business hours, with the right people present and everything else working. A real disaster is none of those. The failures that actually bite are circular dependencies, DNS TTLs and replication lag, none of which a tabletop will surface. And an after-action item without an owner and a date is a wish.

The uncomfortable truth about disaster recovery plans is that most have never been run, which makes them documentation rather than capability. Testing is how you find out whether the plan survives contact with reality.

The five levels, and what each actually proves

A **read-through** is the lightest: key personnel read the plan and confirm they know their responsibilities. It catches "wait, that's my job?" and not much else. But that's worth catching.

A **walkthrough**, or tabletop, has the team step through the plan together against a scenario a facilitator presents. Nobody touches a system. It surfaces assumptions and gaps a solo read never

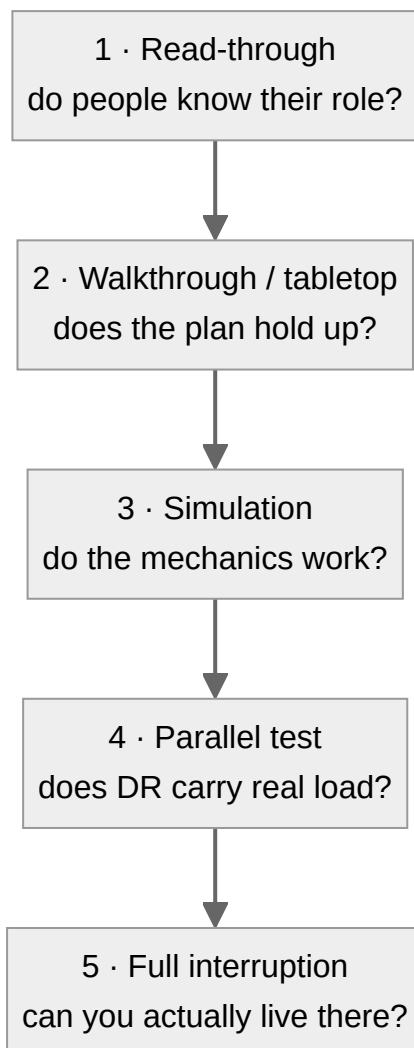
will, and it is by far the best value per hour spent.

A **simulation** actually activates components (restoring from backup, for instance) without impacting live systems. This is the first level that tests *mechanics* rather than *documents*, and it is where most plans first break.

A **parallel test** stands up the full DR environment alongside primary and runs it for real, without switching production over. High confidence, low blast radius.

A **full interruption** test shuts the primary down and runs operations entirely from DR. The most disruptive and the most honest, which is exactly why it's rare and planned carefully.

NIST's contingency planning guidance (SP 800-34) frames these as tabletop, functional and full-scale exercises, and expects at least annual testing for high-impact systems. Annual is a floor, not a target. A plan tested once a year is a plan that is eleven months stale on average.



The five levels, and what each actually proves

The problem with how everyone tests

Here is the thing worth saying out loud. Almost every DR test is **announced in advance, scheduled during business hours, run with the right people available, and performed while everything else is working.**

A real disaster is none of those. It happens at 2am, the person who knows the failover is on a plane, and the thing that broke has taken three other systems with it.

So the failures that actually bite in production are the ones a comfortable test is structurally incapable of finding:

Circular dependencies. The DR plan lives in the wiki that runs on the cluster you're recovering. The credentials are in a password manager behind SSO that depends on the directory that's down. The runbook says "contact the on-call rota" and the rota tool is hosted in the failed region. This is the single most common way a technically-correct DR plan fails on the day. *Print the critical path and store it somewhere that cannot fail with you.*

DNS TTLs. You failed over in eight minutes and customers were down for an hour, because a record had a 3600-second TTL and resolvers everywhere cheerfully kept serving the old address. Nobody discovers this in a tabletop.

Replication lag versus your stated RPO. You promised fifteen minutes. Measure the actual lag under peak write load, not at 3am on a quiet Sunday.

Capacity and licensing on the DR side. DR environments are routinely smaller than production and sized for the diagram rather than the load. Some licences are node-locked and won't start on different hardware.

Recovery order. Restoring the application before the database it depends on wastes the window, and dependency order is exactly what nobody documents.

The fix isn't more testing. It's *less comfortable* testing: run one unannounced, hand it to the second-string team, and disable one thing you assumed would be available.

The after-action report

Every activation ends with one, real disaster or test. That includes the runs that went well, not just the ones that went badly, because the smooth runs teach you what to keep.

A good report has a fixed shape: executive summary; background; the detailed facts (who, what, when, where, why, how); lessons learned; and next steps.

Two things determine whether it's worth writing.

It has to be blameless. The moment a report can end someone's career, people stop volunteering what actually happened, and you lose the only information that mattered. You want the engineer who typed the wrong command to say so within the hour.

Every action item needs an owner and a date. "We should improve backups" is not an action item, it's a mood. Track them where you track real work, and review them before the *next* test. The most reliable finding in this field is that the last report's lessons were never implemented, and the next incident is the same incident.

Communication is part of the test too. Notify internal staff so nobody mistakes the drill for a real outage or the reverse, external partners who depend on you, and regulators where required. A DR test that quietly triggers someone else's incident response has failed regardless of whether the failover worked.

A plan is a hypothesis. A test is the experiment. Run the cheap experiments often, the expensive ones deliberately, and write down what you learned every single time. The next disaster won't wait for you to remember what you meant to fix.