For: engineering leaders

Distributed Systems Testing Landscape

Where does your distributed-system testing problem live?

Igor Konnov, Thomas Pani demo@wunderspec.com

Executive summary. Distributed-systems outages often start as **subtle, high-impact behavioral failures**: one unlucky ordering, one stale observation, one retry at the wrong time. Adding more example tests is often not enough. The opportunity is to make expected behavior executable early, so teams can find design flaws before they become production incidents.

1. Is This Relevant?

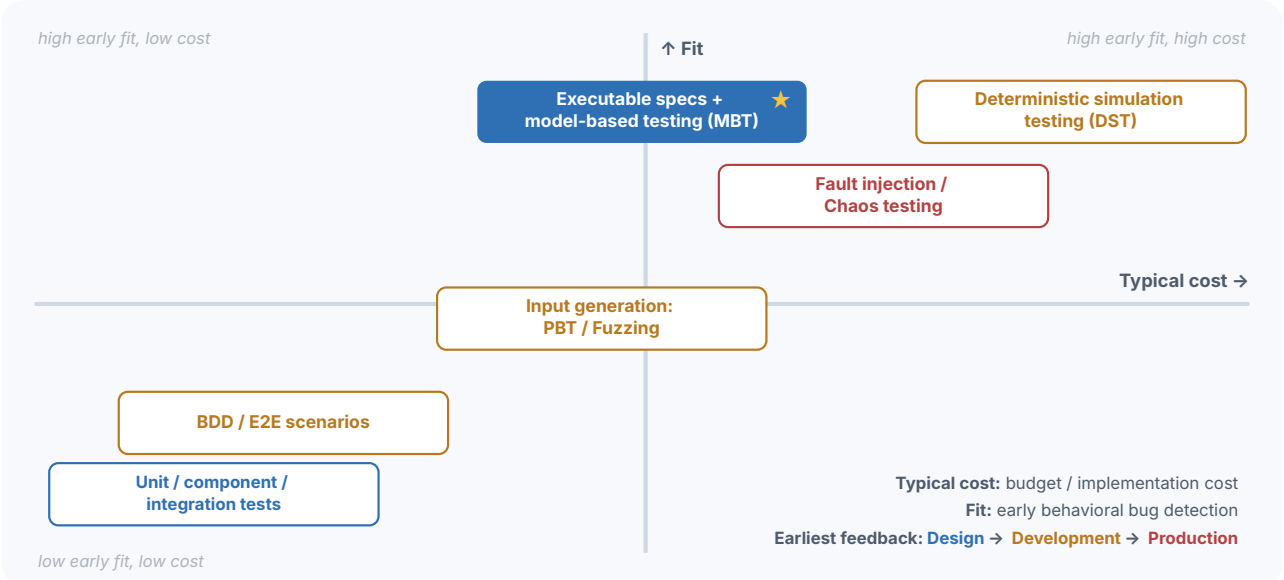
This is worth considering if your system:

- ☐ manages complex state or implements a state machine
- ☐ contains a control loop, scheduler, protocol, or reconciliation process
- ☐ must behave correctly under reordered events, delayed messages, retries, crashes, or partial failure
- ☐ has incidents that are hard to reproduce
- ☐ has important invariants that must always hold
- ☐ has no executable reference for correct behavior

Note: For simple, low-risk, or mostly CRUD systems, executable specs are usually not the first tool to try. If you are unsure, send us a short description and we will place it on the testing landscape.

2. Testing Landscape

Here is how we think about the distributed-system testing landscape: the chart below positions six common bug-finding approaches across two axes: **typical cost** and **fit for early behavioral bug detection**. Color shows the earliest SDLC stage where an approach provides feedback: ■ design, ■ development, or ■ production.



Read this as a decision aid: the star marks the approach this guide focuses on, not a universal winner for every system. Executable specs deliver value in design; MBT starts in development and continues via CI checks into production. The ultimate goal is **earlier feedback** *("shift left")*: find expensive distributed-systems bugs while design choices are still cheap to change.

Approach	Helps with	Watch out
Unit / component / integration tests	Known cases, regressions, TDD-style checks, and fixed technical interactions between components.	Passing tests prove only the paths you wrote down; rare interleavings and failures remain unexplored.
BDD / E2E scenarios	User-facing flows, acceptance criteria, and fixed cross-component workflows.	Scenario coverage is sparse; timing and failure interleavings remain mostly unexplored.
Input generation: property-based testing (PBT) / fuzzing	Unexpected inputs missed by example-based or table-style tests.	Needs strong oracles and good generators; otherwise it finds shallow input bugs while missing ordering, failure, and protocol behavior.
Executable specs + model-based testing (MBT) ★	Design clarity, protocol enforcement, and generated checks from executable specs.	Spec drift: executable spec and implementation may diverge. Use CI to mitigate.
Deterministic simulation testing (DST)	Real-system behavior under faults, timing constraints, and complex schedules.	High cost: lots of compute and significant integration work (packaging, instrumentation, ...).
Fault injection / Chaos testing	Resilience validation and failures visible only under faults or production pressure.	Often learns late; by then, users may already be affected.

Common distributed-system testing approaches and their trade-offs.

3. When Executable Specs Help

Executable specs help when the hard part is not generating more inputs, but **defining and exploring correct behavior**.

An executable specification can:

- ✓ explore behavior before production
- ✓ generate happy-path and adversarial scenarios for model-based tests
- ✓ act as a test oracle against the real system
- ✓ make invariants reviewable by humans and tools

The executable spec is not just documentation.
It is an executable reference that can generate tests and check behavior.

4. Want a Short Assessment?

Send us five bullets and we will tell you where your system fits on the testing landscape.
Rough notes are enough:

1. What system or subsystem are you thinking about?
2. What important behavior must never break?
3. What makes those behaviors hard to test?
   Concurrency, retries, time, partial failure, ordering, scale, state, or external services?
4. What are you worried your current tests miss or find too late?
5. Who would care if this failed in production?

> **We will reply with a short assessment:** where your system falls on the landscape, which approaches are most likely to help, which are poor fits, and whether executable specs + MBT are worth considering. You can forward the assessment internally to decide whether a small pilot is worth discussing.

→ wunderspec.com/diagnosis

About us

Igor Konnov, PhD, and Dr. Thomas Pani work on executable specifications, model checking, model-based testing, and fuzzing for distributed systems. Their work spans TLA+, Apalache, Quint, Lean, protocol verification, conformance testing, and practical test generation against real systems.