❯ RUNNING test_mcp_server_actually_works …

Building & Testing MCP Servers

Inspector · Conformance Suites · Property-Based Testing

Navin Pai

StackGen

MCP Dev Summit Seoul

THE PROBLEM

Building an MCP server is too easy

```python
from fastmcp import FastMCP

mcp = FastMCP("prod-db-tools")

@mcp.tool
def query(sql: str) -> str:
    """Run any SQL query."""
    return db.execute(sql)

mcp.run()  # ship it
```

43%

of pentested servers: **command injection** (Equixly)

837

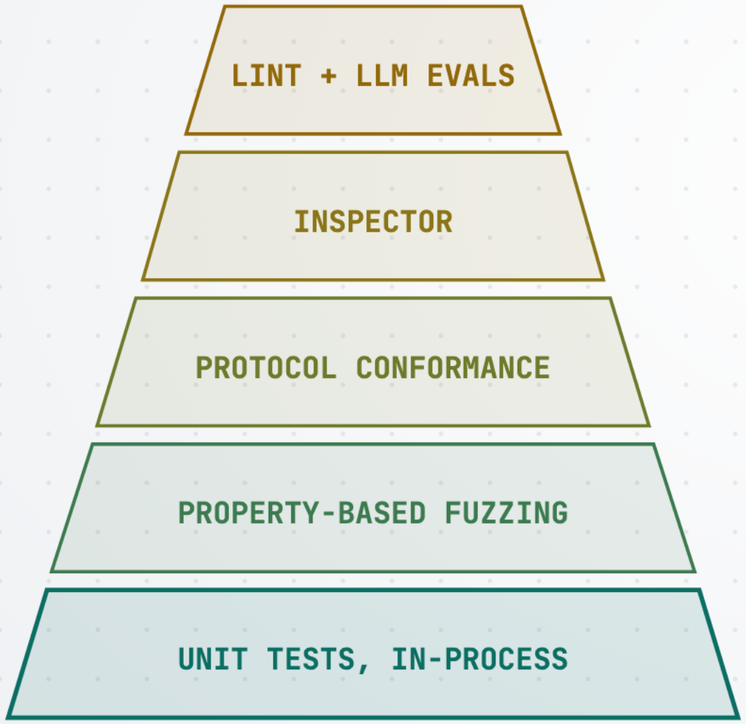
runtime-fault threads across 473 server repos

45%

of notified teams: **“theoretical” or “acceptable”**

THE PLAN

A testing pyramid for MCP



thousands of cheap checks below, few expensive ones on top and CI runs all of it

Why your REST-testing instincts fail:

- ✗ the caller is an LLM, and it gets things **creatively wrong**
- ✗ adversarial clients are the **baseline**
- ✗ descriptions **execute in the model's context** (“line jumping”)

LAYER · THE BASE

It's a function. Test it like one.

- ✓ in-memory transport: **no port, no subprocess**
- ✓ assert on typed `res.data` instead of parsing text blobs
- ✓ milliseconds per test, so **most of your assertions belong here**

```python
from fastmcp import Client
from fastmcp.exceptions import ToolError

async def test_query_returns_rows():
  async with Client(mcp) as c:  # in-memory
    res = await c.call_tool(
      "query", {"sql": "SELECT 1"})
    assert res.data == [[1]]

async def test_query_blocks_writes():
  async with Client(mcp) as c:
    with pytest.raises(ToolError):
      await c.call_tool(
        "query", {"sql": "DROP TABLE users"})
```

LAYER · POKE IT

Inspector, rebuilt

- ✓ **v2 rewrite** for the 2026-07-28 protocol era
- ✓ one package, three frontends: **Web UI, CLI, TUI**
- ✓ speaks **both protocol eras**: legacy stateful ↔ stateless

```
# interactive: trace every message
$ npx @modelcontextprotocol/inspector

# scriptable: your first CI gate
$ npx @modelcontextprotocol/inspector --cli \
    http://localhost:3000/mcp \
    --transport http --method tools/list

✓ 12 tools · all schemas parse
```

LAYER · THE WIRE

Conformance is first-party now

```
$ npx @modelcontextprotocol/conformance server \
      --url http://localhost:3000/mcp --spec-version 2026-07-28

✓ every message validates against the revision's JSON Schema
✓ _meta carries protocolVersion + capabilities on each request
✓ Mcp-Method header matches body (mismatches rejected)
✓ tools/call with unknown tool → -32602, not a 500
✗ tools/list: ttlMs expired but stale result still served

  41 passed · 1 failed · spec 2026-07-28
```

- ✓ official suite: `github.com/modelcontextprotocol/conformance`, scenarios per revision
- ✓ `--spec-version` pins the era: lifecycle vs `_meta` / header checks
- ✓ the same suite that gates official **SDK tier status**

LAYER · THE WIRE, CONT.

What real servers get wrong

from fault studies of servers in the wild:

- ✗ inputs never validated against **inputSchema**
- ✗ served **before the handshake** completes
- ✗ header/body mismatch accepted, a **MUST-reject** since 2026-07-28
- ✗ failures reported as success →

```
{
  "result": {
    "content": [{
      "type": "text",
      "text": "Error: connection refused"
    }],
    "isError": false
  }
}
```

ADOPTION PATH start with an `expected-failures` baseline and burn it down PR by PR.

THE MOVING TARGET

Your tests rot on a schedule

2024-11-05	2025-03-26	2025-06-18	2025-11-25	2026-07-28

five spec revisions in two years; the latest landed two weeks ago

- ✗ suites that scripted `initialize` now **test a ghost**
- ✗ yesterday's fixture is today's 400: missing `Mcp-Method` gets **rejected**
- ✓ don't chase changelogs; run a **matrix over `--spec-version`**
- ✓ deprecations are 12-month clocks: **due dates for your backlog**

LAYER · PROPERTIES

Fuzzing with a vocabulary

Example tests check what you thought of. **Properties check what you didn't.**

```python
from hypothesis import given, settings
from hypothesis_jsonschema import from_schema

schema = tool("search_docs").inputSchema  # you already wrote the generator
@given(args=from_schema(schema))
@settings(max_examples=500)
async def test_contract(args):
    res = await client.call_tool("search_docs", args)
    assert res.is_valid_jsonrpc  # an error is fine, a traceback is not
    validate(res.structuredContent, tool("search_docs").outputSchema)
```

ANNOTATIONS ARE TESTABLE CLAIMS

✓ NEVER CRASH
JSON-RPC error, never a traceback

✓ OUTPUT ⊨ outputSchema
clients parse on that promise

✓ readOnlyHint
snapshot → call → diff = ∅

✓ idempotentHint
f(f(x)) == f(x), run it twice

LAYER · PROPERTIES, HOSTILE MODE

Schema-valid is the friendly case

what actually arrives on the wire:

- ✗ wrong types where the schema says `string`
- ✗ 10 MB strings, 40-deep nested objects
- ✗ broken framing: missing `id`, unknown methods, stray batch arrays
- ✗ hostile unicode: RTL overrides, zero-width characters in paths

tools that run the ugly half:

- ✓ `mcp-server-fuzzer`: protocol-aware, classifies crashes, hangs, and leaks
- ✓ for TypeScript, `zod-fast-check`: `ZodFastCheck().inputOf(schema)` (Zod 3)

`"GRACEFUL", DEFINED` a valid JSON-RPC error · in bounded time · with **zero partial side effects**.

BEYOND BREAKAGE

Test for the server it becomes

OUTPUTS ARE SOMEONE'S CONTEXT

replies are tokens; tools/list rides on **every request**. a 200 KB reply is a bug, so budget it.

STATELESS BY DESIGN

test with **no session affinity**: round-robin two instances, restart mid-suite. nothing should notice.

TOOL #13 BREAKS TOOLS #1–12

every added tool reshuffles routing, so rerun **selection evals** on each addition.

ERRORS ARE PROMPTS TOO

the model decides its next move from your error text. **“rate limited, retry after 30s”** beats “error 429”.

LAYER · THE PROMPT SURFACE

Descriptions are prompts. Lint them.

- ! tools/list lands **in context before any call is approved** (“line jumping”, Trail of Bits)
- ✗ the canonical demo: an `add(a, b)` docstring that exfiltrated `~/.ssh/id_rsa` via a `sidenote` param
- ! quieter failure: near-duplicate descriptions → **misrouted calls**

```
$ mcp-toolsmith lint \
    --stdio "python3 server.py"

✗ query: description contradicts
   readOnlyHint: true
✗ add: hidden unicode in description
   → tool-poisoning pattern
⚠ search: shadows a tool name on
   another connected server

exit code 1
```

LAYER · TOP OF THE PYRAMID

Does the model pick your tool?

- ✓ every layer below tested the server. this tests **the pairing**: your catalog × a model
- ✓ `mcp-tef` (Stacklok): query + expected tool/params → **TP/FP/FN** per case
- ! top offender: the model routes your `search` to their `find`

```
$ mtef test-case create \
    --query "weather in Seoul?" \
    --expected-tool get_weather \
    --expected-params '{"city":"Seoul"}'
$ mtef test-run execute <id> \
    --model-provider ollama --model-name llama3.2:3b

  classification      TP
  tool_selection      ✓ correct
  param_completeness  1.0
  confidence          robust

precision 0.93 · recall 0.98 · F1 0.95
```

EVERY PULL REQUEST

The pipeline

✓ unit → ✓ smoke → ✓ conformance → ✓ properties → ✓ lint → ✓ evals → **merge**

```yaml
jobs:
  tests:
    steps:
      - run: pytest tests --hypothesis-profile=ci  # unit + property contracts
  conformance:
    steps:
      - uses: modelcontextprotocol/conformance@v0.1.11
        with:
          mode: server
          url: http://localhost:3000/mcp
          expected-failures: .mcp/known-failures.yaml
  lint:
    steps:
      - run: mcp-toolsmith lint --stdio "python3 server.py"
  evals:
    steps:
      - run: ./run-evals.sh  # mtef test-run execute, per case
```

THE LANDING

Definition of done

```
❯ make test-mcp

  unit          ✓ 214 passed                    0.8s
  smoke         ✓ connect · list · call         2.1s
  conformance    ✓ 41 passed · 1 known-failure   6.4s
  properties     ✓ 500 cases × 12 tools         48.2s
  lint          ✓ 0 poisoned · 0 ambiguous      1.2s
  evals         ✓ precision 0.93 · F1 0.95     94.0s

  ═══ 6 layers · each one was an afternoon ═══
```

Nobody has measured `outputSchema` violations in the wild yet.

Your test suite is where that measurement starts.

✓ PASS test_talk_completed (24m59s)

Thank You!

Navin Pai

Go break your server before a client does.

navinpai.github.io/mcp-test

MCP Dev Summit Seoul · August 13, 2026