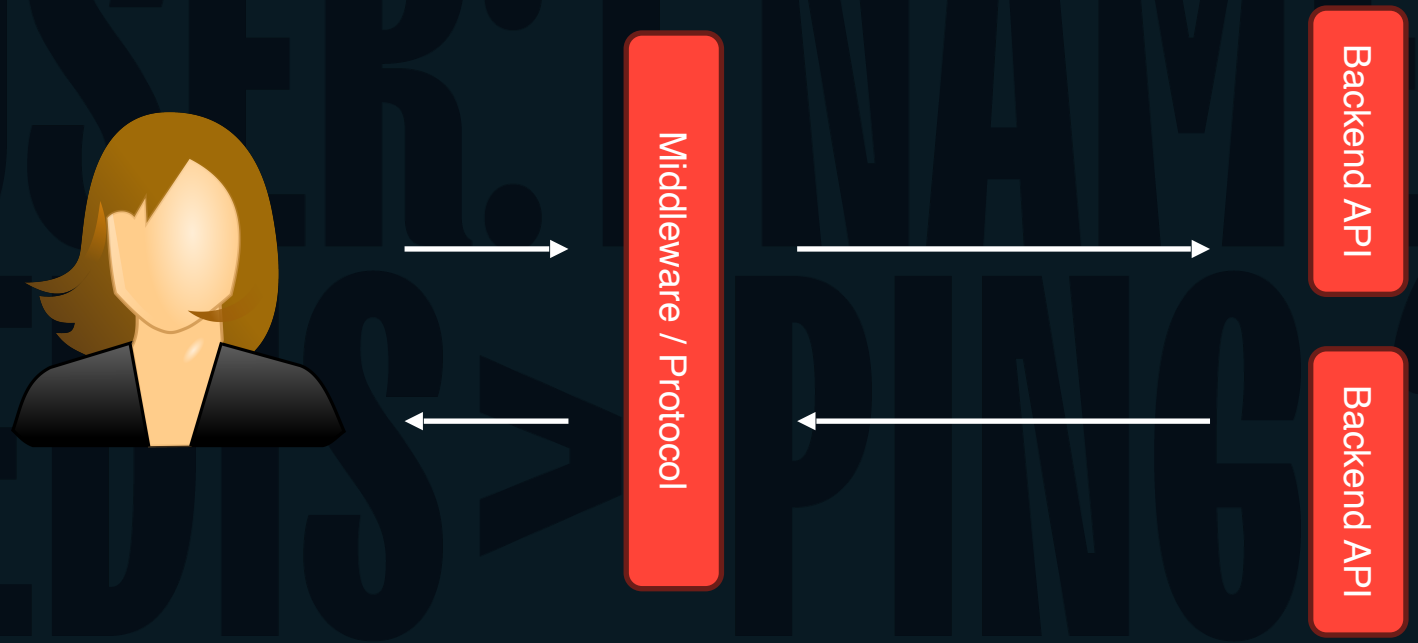
The Seven Deadly Sins with MCP

Ricardo Ferreira
Lead, Developer Relations @ Redis

Other talks have shown you how to build MCP systems. This one is about how to tell when you're building one badly.



“This ORB always fails.”
“This ESB is super slow.”
“Service Meshes are complex.”

The failure often starts in the capability design, but the blame lands on the most visible layer.

What is usually broken?

- Backends accepting generic commands
- Permissive deployment deleting master data
- Errors wrapped into high level exceptions
- Too many retry attempts breaking the backend
- Full payload of entities materialized on-heap
- Over engineered validation frameworks
- Functionality added just because it looks cool

“MCP always fails.”

“MCP is super slow.”

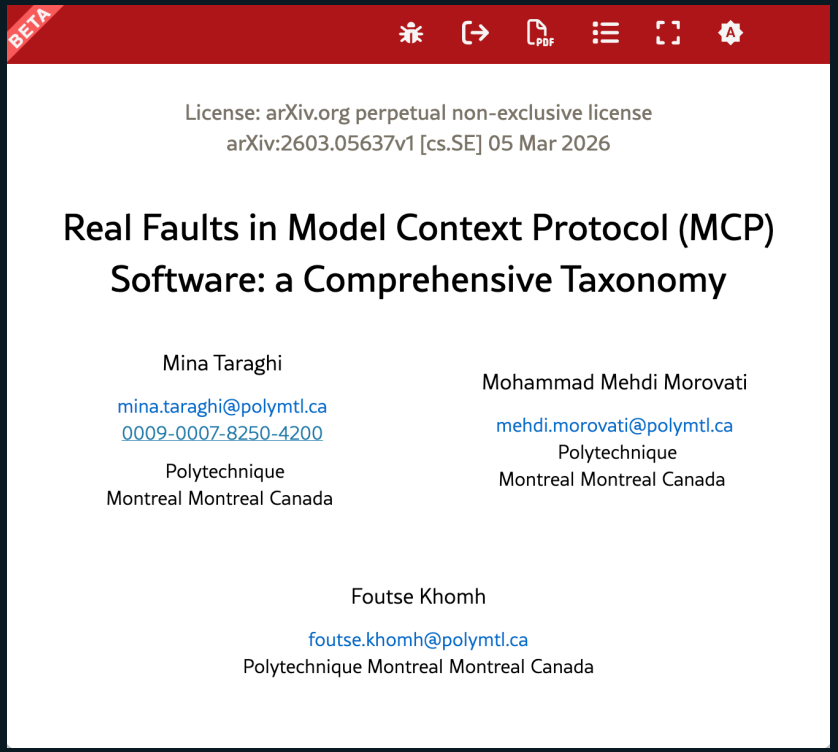
“MCP is complex.”

“MCP is dead.”

MCP is usually not the problem. MCP is the amplifier.

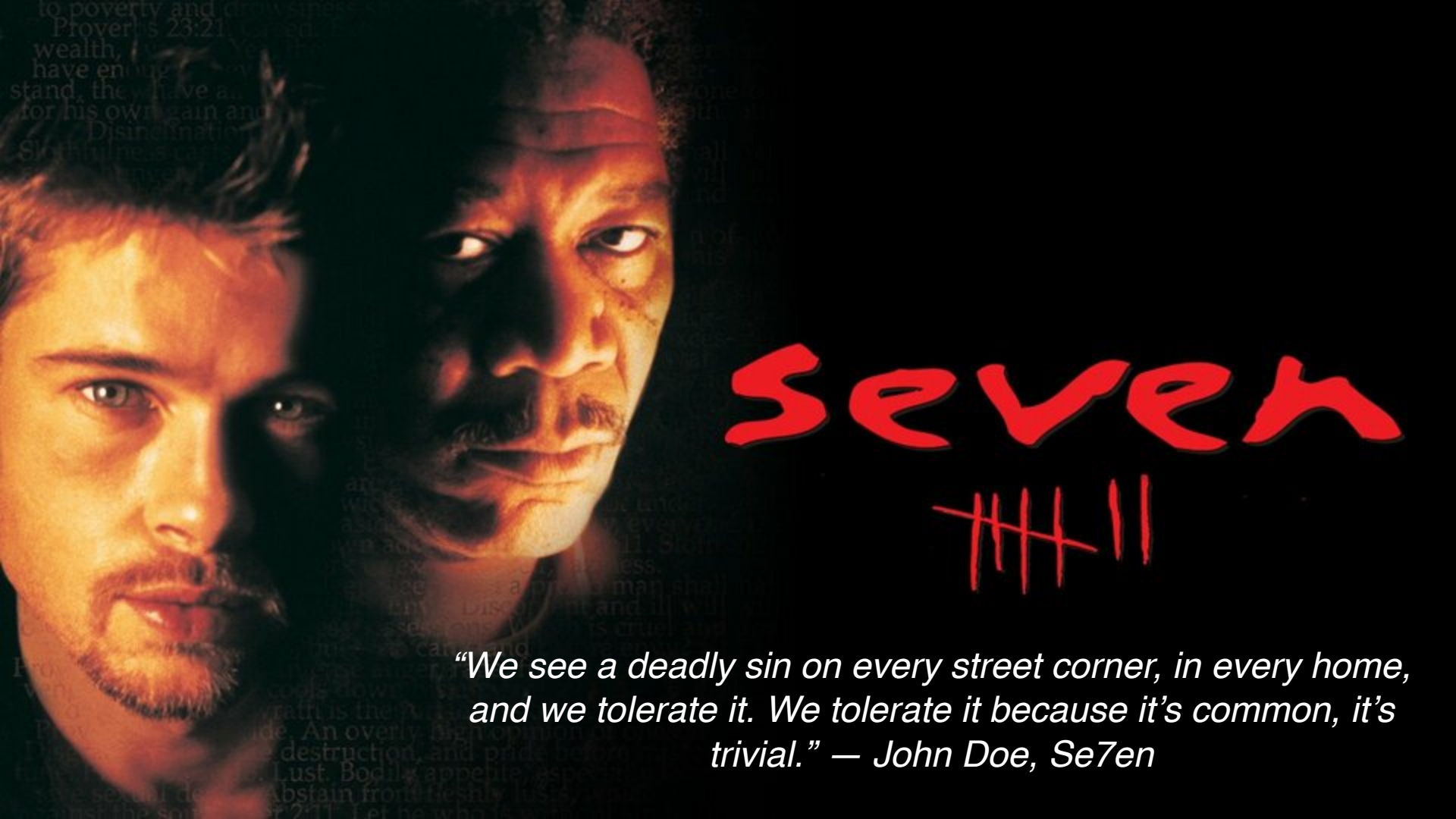
Real faults in MCP software: a comprehensive taxonomy

- Real-world faults in MCP servers
- 3,282 GitHub closed issues
- 407 MCP server-related issues
- No protocol-spec issues reported



Most MCP bugs are caused by engineering impulses

- The bug you see popping up in production is just a symptom
- The problem is the instinct that made the bug feel reasonable
- We act on them often enough, casually enough, and confidently enough



“We see a deadly sin on every street corner, in every home, and we tolerate it. We tolerate it because it’s common, it’s trivial.” — John Doe, Se7en

I like to think of these engineering impulses as deadly sins.

The seven deadly sins with MCP

- Security sins

  Decides blast radius

  (Lust and Greed)

- Operational sins

  Truthful system failures

  (Sloth and Wrath)

- Design sins

  Long-term carrying cost

  (Gluttony, Pride, and Envy)

Security sins: Lust
(Too much intimacy)

Security sins: Greed (Too much power)

Operational sins: Sloth (Too little care)

Operational sins: Wrath (Too much force)

Design sins: Gluttony
(Too much waste)

Design sins: Pride
(Too much ego)

Design sins: Envy
(Too much imitation)

The road to redemption is not better intentions. It's better defaults.

A simple refactoring order

- Lust: reduce the blast radius of the shell, database, and filesystem
- Greed: narrow scopes, credentials, and access boundaries
- Sloth: make failures intentionally specific and observable
- Wrath: stop runaway retries and reconnect storms; add idempotency
- Gluttony: shrink your payloads, limit data exposure
- Pride: simplify abstractions once the system is stable enough
- Envy: reduce the tool sprawl and fix naming ambiguity

The safest MCP systems are boring on purpose.

Redemption starts with boring defaults

- Prioritize narrow capabilities
- Least privilege by default
- Typed errors people can act on
- Retries with limits, backoff, and idempotency
- Small outputs with hard budgets
- Telemetry that shows when things start going weird

If it only lives in your head, it will fail

- Review capabilities before they ship
- Test the ugly paths, not just the happy path
- Measure payload size, retries, and failure codes
- Block bad defaults in CI
- Make ownership painfully explicit


When double-checking your work, Ricardo, remember that Senior Developer Mark always inquires about certain aspects. It would be wise to
Redis
Read the Full Blog Series
Let's connect on LinkedIn