



AGENCY TRUST LTD · GUEST LECTURE · OCTOBER 2026

AI Building AI

How a shipping company built its own AI coding agent —
and used it to ship enterprise software faster



qcli · qcli-control · qSMA

John · Agency Trust Ltd. — 40 minutes, questions welcome throughout

What we will cover

- | | | | |
|---|----------------------------------|---|--------|
| 1 | The setting | A shipping company that builds enterprise software — qSMA | 4 min |
| 2 | qcli | Why and how we built our own AI coding agent | 11 min |
| 3 | qcli-control | From one agent to a supervised fleet | 5 min |
| 4 | The multiplication effect | How each tool amplified the others — with numbers | 7 min |
| 5 | The systems behind it | Gates, memory, reflection, verification | 7 min |
| 6 | The honest ledger | What AI development costs — and what it pays | 2 min |
| 7 | Lessons | What transfers to your career — and close | 2 min |

Ship managers who run on their own software

- ▶ **Ship management, not software sales:** HSQE, technical, operational, commercial and crewing management for third-party vessel owners — plus the IT that runs it.
- ▶ **One platform:** qSMA covers 40+ business domains — crewing, chartering, voyages, bunkers, defects, ISM compliance, daily reports, supplies, repairs, accounting.
- ▶ **A legacy mountain:** the platform is a full port of a 20-year-old JSP/EJB system — every screen, formula and permission re-verified against the old behavior.
- ▶ **A small dev team with domain expertise:** the constraint was never ideas — it was typing speed. We chose to multiply the team we had.



**“We manage ships.
And we build the software
that manages the ships.”**

qSMA — the Ship Management Application

3,799

Java source files in the
backend alone

40+

business domains, from
crewing to accounting

254

@Toolbox endpoints the
built-in AI can call

3,421

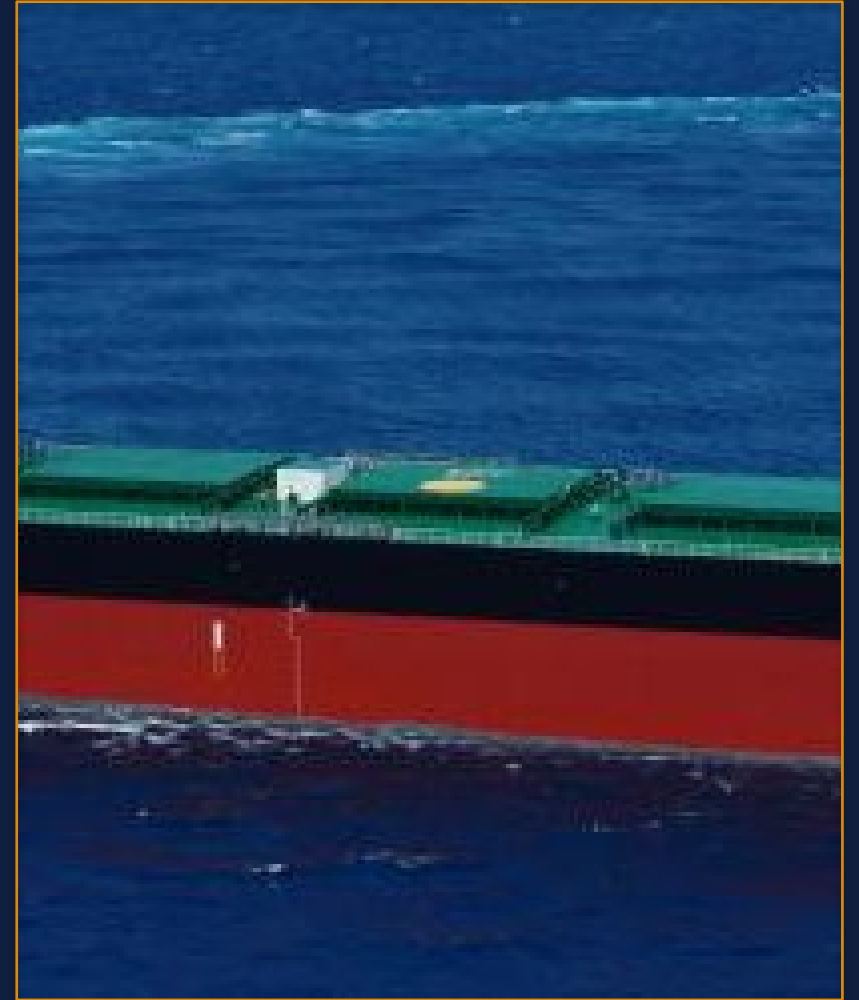
commits since April 2025

- **Stack:** Java 25 · Quarkus 3.39 · jOOQ · PostgreSQL + MSSQL · React 19 / TypeScript · Vite + Quinoa — one deployable JAR.
- **Serious engineering discipline:** PIT mutation testing as the final quality gate, generated API clients, parity audits against the legacy system per domain.
- **And it ships AI to its users:** an assistant with 6 modes (GeneralV2 default, Agentic, Constrained, Documents, Summaries, Tags) that reads and acts on live shipping data.
- **The question in mid-2026:** how do we keep this pace with the team we have — without shipping AI-built code we can't trust?

PART 1

Build the multiplier, not just the feature

In June 2026 we stopped treating the AI assistant as a product feature and started treating development leverage itself as the product to build.

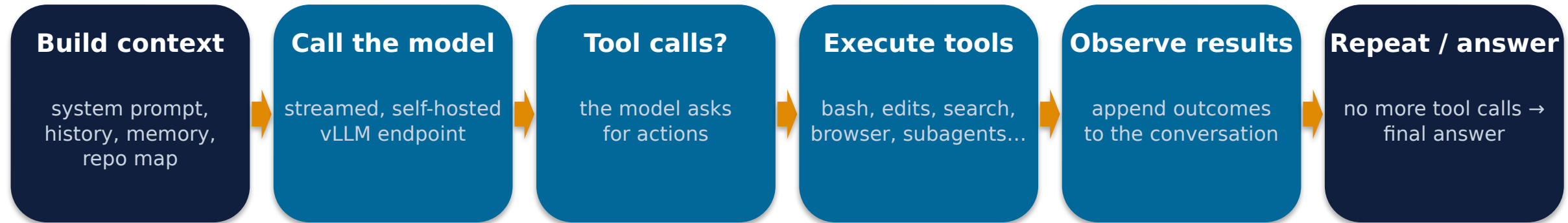


Why build an agent instead of buying one?

- ▶ **Data sovereignty:** commercial shipping data — chartering, crew, financials — cannot go to a third-party cloud API. We run a self-hosted vLLM GPU server; every agent assumes that, not a paid cloud key.
- ▶ **Control:** when the harness is yours, the loop, the tools, the context, the gates are all yours. Off-the-shelf agents are a black box at exactly the layer that matters.
- ▶ **Cost model:** self-hosted inference means marginal cost is GPU time, not per-seat subscriptions — so we can run fleets of agents, not one.
- ▶ **Dogfooding:** a coding agent that codes is the perfect first customer of itself. Every qcli improvement immediately pays back into qcli's own development.
- ▶ **The bet:** existing tools assumed cloud APIs and single users. Nothing fit a self-hosted, multi-agent, verification-first workflow. So we built qcli — Quarkus-native, like everything we ship.

The core: an agentic tool-calling loop

The model is not asked once — it works. Reason → act → observe → repeat, until it can answer.



↻ loop bounded by max-iterations and cancellable at any point (steering)

- ▶ **Termination is a contract:** the run ends when the model stops calling tools — or when the harness refuses to accept incomplete work (verification gate). The harness, not the model, decides when work is done.
- ▶ **60 built-in tools, zero registration:** tools are CDI beans auto-discovered at startup; adding one means writing one class. MCP adapters extend this to external servers.
- ▶ **Everything is inspectable:** every event — model calls, tool results, compactons — streams out as structured events; sessions persist to SQLite and can be resumed, branched, replayed.

The toolbox — what the agent can actually do

Code & files

- ▶ bash with timeouts & guards
- ▶ read / edit / patch families
- ▶ grep, glob, structural search
- ▶ repo map: PageRank over symbols
- ▶ call graph: callers / callees
- ▶ LSP: diagnostics, definitions, renames

Intelligence

- ▶ web search: multi-provider RRF fusion
- ▶ browser: scripted Playwright testing
- ▶ debug: breakpoints on the live JVM
- ▶ git history mining: bug-magnet files
- ▶ hybrid code search (BM25 + vectors)
- ▶ audit: architecture overviews

Scale & people

- ▶ subagents: parallel, forked context
- ▶ agent teams: message passing
- ▶ todo / goal: durable planning
- ▶ ask_user_question: real dialogs
- ▶ eval kernel: persistent REPL
- ▶ browser screenshots for UI work

A key design rule: every capability the agent gives a developer, it gives to itself. The browser tool tests qcli's own UI; the debug tool debugs qcli's own loop; the subagent tool parallelizes qcli's own work.

The value is context engineering, not the LLM call

Repo map

Tree-sitter parses 100+ languages; PageRank ranks symbols; the prompt gets a map of the 2 files that matter, not 1,798 files it can't read.

Memory pipeline

Every session's learnings persist to learnings.jsonl; recall is BM25 + cosine + RRF rerank + trust promotion + a knee-point cut. Weeks later, the agent remembers.

Compaction

Six strategies shrink long conversations without losing the thread — and every compaction logs a survival probe checking that key facts survived.

Verification gate

Policy-driven: surfaces, risks, gates. Compile and tests must pass before the run may finish; convergence codes are machine-readable, no free-text 'I did it'.

KV-cache discipline

The tool catalog is byte-stable across launches so the vLLM prefix cache hits; cache hit ratio is a first-class metric, shown live.

PART 2

One agent became a supervised fleet

If one agent is a force multiplier, a coordinated fleet with a control plane is an engineering department. qcli-control runs the fleet of agents from the browser — and from your phone.



A mission control for agent fleets

- ▶ **Launch and supervise:** agents run in tmux sessions started from the web UI; live output, thinking streams, todo boards and subagent trees render in the browser.
- ▶ **Steer, don't kill:** messages and slash commands reach a running agent mid-task over WebSocket (the qcli Control Protocol) — redirect work without losing context.
- ▶ **Human-in-the-loop from anywhere:** approval requests and questions from agents pop up in the dashboard and the mobile app; 'unattended auto-deny' lets overnight runs proceed down safe paths.
- ▶ **A real task board:** queued and dispatched work, retries, a verification ledger — and board-health sweeps that terminalize rows whose agent died.
- ▶ **Automations:** cron-scheduled prompts, CI-fix tasks that re-trigger failed GitHub runs, per-tool failure-pattern hooks — the fleet does the night shift.

1,620

commits to the hub itself
since July 11, 2026 — built by
the agent it controls

≈ 18 commits/day

6,119

automated tests in the hub:
2,955 backend + 3,164 frontend
— every agent-built change gated

SpotBugs 0 · Sonar 0

1 socket

QCP — one JSON-over-WebSocket
protocol drives everything:
launch, steer, approve, verify

Spec: [docs/reference/PROTOCOL.md](https://docs.reference/PROTOCOL.md)

Trust machinery: the fleet verifies itself

Verification ledger

A task decomposes its acceptance criteria into one on-disk assertion per clause. The hub — never the agent — evaluates each clause at run end. Task cards show 2✓ 1○ 1✗; the ledger survives restarts.

Hub-evaluated, not self-reported

Agents may declare PENDING assertions, but the hub alone promotes what it can check. Uncheckable rows stay gray forever. Self-reports never turn green.

Behavioral firewall

Declarative rules evaluate every tool call: deny aborts, flag injects an advisory note. Fail-closed, persisted, restart-surviving — a safety control that disarms on restart is worse than none.

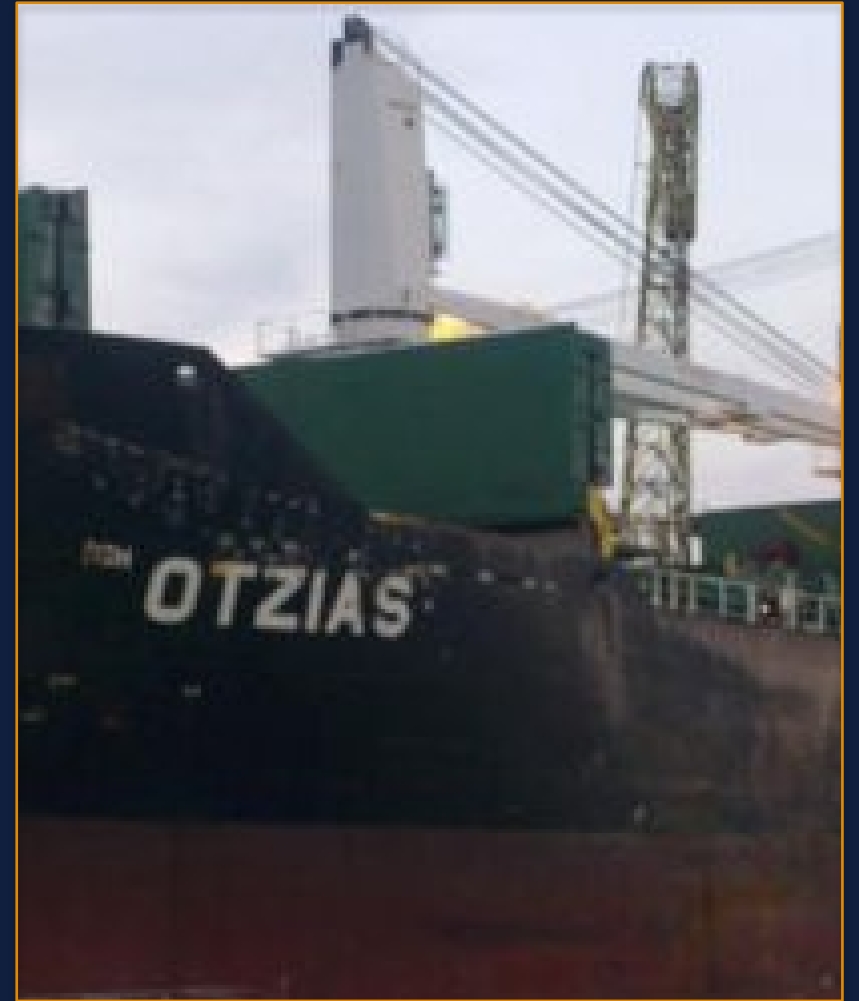
Tool-call policy + secrets

Path/regex/contains rules per agent or fleet-wide; profile-scoped secrets are injected via env files at launch and never transit any hub store.

PART 3

The multiplication effect

AI built the tools that build with AI: each system's output became the next system's input. This is where the cost of building your own tooling is repaid — with interest.



The flywheel



What the loop produced — measured

+26%

qSMA commit velocity after the agent fleet arrived (Jun 2026):
6.0 → 7.6 commits/day

higher pace, stricter gates

24/day

qcli's own commit rate:
2,552 commits in its first
106 days of existence

one developer + agents

155

commits on our best single day
(Oct 2, 2026) across the three
repos — from a small core team

human-reviewed, gated

17,027

test cases across the three
systems (7,561 + 6,119 + 3,347)
— AI-written, human-gated

0

SpotBugs findings and Sonar bugs
allowed on every merge —
enforced, not aspirational

BugInstance 0 policy

68%

code-coverage floor enforced as
a FAILED BUILD in the hub CI —
measured, then locked in

jacoco:check at 0.68

The honest asterisk: commit counts measure activity, not value. The reason we trust them here: quality got STRICTER over the same window — 17,027 test cases, mutation testing, coverage floors and zero-bug gates all arrived with the agents, not despite them.

qcli patterns became qSMA product features

Agentic loop → product

qcli's observe→reason→act loop is being ported into qSMA as a new AI mode: the same architecture, but the tools are SMA's 254 domain toolboxes instead of files and bash.

From 6 modes to a worker

SMA's assistant already has GeneralV2, Agentic (plan-then-execute), Constrained, Documents and Summaries modes. The ported loop adds a true worker that reasons between every call.

Scheduled department agents

Mail Analyst, Machinery/PMS Analyst, ISM Analyst — cron-driven agent runs that produce daily trended reports and feed each other serially. The night shift, shipped as software.

Infrastructure reuse

pgvector RAG, AI request-review logging, telemetry, self-check services — all pre-positioned by qSMA's earlier AI work, now gaining a real engine.

PART 4

The systems behind it — loops on loops

AI-generated velocity without control systems is a liability. Four loops keep the fleet honest: gates, memory, reflection, verification.



Gates: nothing lands because a model said so

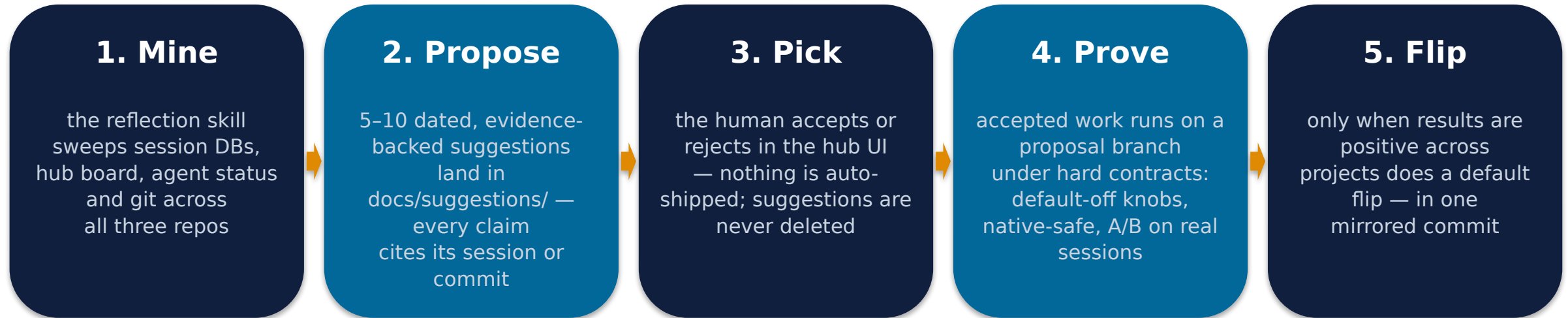


- ▶ **The agent runs this ladder, not a CI server:** gates execute inside the loop, before the run may complete. A failing gate is context the model must fix — not a red email a human reads tomorrow.
- ▶ **Mutation testing is the truth serum:** PIT deliberately breaks code and checks whether tests notice. It kills the classic AI failure mode — plausible tests that verify nothing.
- ▶ **Everything is pinned by tests that failed first:** the dispatch-cap fix was pinned by a test that failed 11-vs-2 before the fix; the coverage floor was proven red at 0.99-of-nothing and green at the measured 0.68.

Memory: the fleet that remembers

- ▶ **Write path:** agents persist durable learnings to per-project learnings.jsonl — build commands, project conventions, gotchas, root causes — white-box listed and editable with forget/maintain.
- ▶ **Read path is a pipeline:** scope + temporal filtering → BM25 + cosine retrieval → RRF rerank → trust promotion with recency tie-break → knee-point cut. You get the 6 learnings that matter, not 60.
- ▶ **Failure memory warns before repeating:** learnings marked FAILURE:: token-match bash commands at the tool seam — the warning rides the tool result itself, arriving before the mistake is made.
- ▶ **Knowledge pages:** the winning cluster's members are replaced by one canonical, human-editable page — memory that consolidates, not accumulates.
- ▶ **The payoff is compounding:** session N+1 starts where session N ended. A three-month-old fact about this codebase reached today's work automatically — no human re-explained it.

Reflection: the fleet audits itself every week



- ▶ **Running weekly since Sep 17, 2026:** each reflection mined thousands of tool rows across the fleet's sessions; 27 improvements shipped in the first three weeks — the fleet gate-lock, the release-gate script, stranded-worktree rescue, the model-endpoint probe.
- ▶ **Why it matters:** without it, an agent fleet optimizes for whatever the last prompt asked. With it, the system itself is under version control — the fleet's own behavior gets pull requests.

The honest ledger: what AI development costs — and pays

THE PRICE

- ▶ **1 in 4 commits is a repair.** qcli 24%, the hub 29% — a quarter of the fleet's output goes to fixing the system itself.
- ▶ **Testing effort = development effort.** 859 test files vs 858 production files in qcli. The safety net is as big as the product.
- ▶ **Static analysis is constant work.** 177 commits in the tool repos just clearing analyzer findings — the tools catch AI noise every single day.
- ▶ **AI bugs get subtle.** A theme palette computed HSL lightness mathematically, not perceptually — unreadable yellow on white, caught by a human eye, pinned by a 49-theme regression test.

THE RETURN

- ▶ **1.16M lines added** to the two tool repositories alone in under four months — every line human-gated.
- ▶ **32 reverts, total.** Across all three repos: bugs are fixed forward, not rolled back. History stays linear.
- ▶ **17,027 test cases** guard every merge — most AI-written, all human-reviewed, mutation-tested.
- ▶ **Bugs die before merge.** That is the point of the gates: the chaos happens in the sandbox, never in production.

The honest summary: AI writes a lot of code — and about a quarter of the fleet's energy goes into verifying and repairing it. The economics still win, but only because the testing machinery is automated too: automation must verify what automation writes.

What transfers to you

- ▶ **Buy speed with tools, not headcount.** The highest-leverage thing a small team can build is the thing that builds everything else. Feature work compounds linearly; tooling compounds exponentially.
- ▶ **Self-hosted is a strategy, not a compromise.** Owning the inference layer turned a per-seat cost into a fleet-sized budget — and kept shipping data in-house.
- ▶ **The harness decides when work is done.** Never the model. Verification gates, convergence codes, mutation testing — trust machinery beats trust.
- ▶ **Make the feedback loop the product.** Memory, weekly reflection, competitor sweeps (30+ agents tracked, features stolen shamelessly) — the system improves itself on a schedule.
- ▶ **Dogfood the loop, then port it.** The agent built its own control plane; the control plane ran the agent; the whole pattern is now becoming a product feature. Bootstrapping is a business plan.

Thank you

The stack: qSMA — ship management platform · qcli — self-hosted coding agent · qcli-control — agent-fleet hub

All of it: Quarkus + Java 25 + React, self-hosted vLLM, quality-gated (SpotBugs 0 · Sonar 0 · PIT · 68% floor)

Built by: one core team and its fleet of agents — agencytrust.com



Questions — and thank you for listening.

