AI TINKERERS — MEXICO CITY — SEP 1, 2026

I Gave Six AI Agents a Scrum Board and Measured What Happened

Richard Kasperowski · Alli Connect (Techstars '22)

PART 1 — WHAT I BUILT

A Scrum team of AI agents that ships production healthcare software

THE CONTEXT

Alli Connect — predictive mental healthcare for first responders. 30+ fire & police departments, HIPAA-grade. I'm the whole tech team, the CTO doing biz functions, the security officer, and the PO.

THE TEAM — A SCRUM TEAM

- **Scrum Master** — an AI agent that facilitates; it's the chat agent
- **Product Owner** — me, the human
- **Developers** — six AI agents, three pairs, pair programming, generally autonomous

They pull their own work off the sprint backlog and have full autonomy over how they do it. I don't assign. I don't dictate.



Watch the terminal

LAUNCHING A REAL BATCH, LIVE

PART 2 — HOW I BUILT IT

One versioned prompt, a hard commit gate, a measurement from git

- **Claude Code** runs every agent — Driver, Reviewer, and a read-only code-audit subagent
- **One Markdown prompt** (v20, with a changelog) is the entire orchestrator — not a framework I'd have to learn, wait on, or trust to maintain. I wrote the prompt; Claude figures out the rest.
- **Test tiers are the commit gate** — lint, tsc, Jest, Cypress, JUnit, Cucumber, Gradle, Playwright, Maestro — all green or no commit, checked after every review, not just before merge
- **MCPs and CLIs** connect the agents to almost every external service they need — a partial list below
- **Git worktrees**, used as needed, keep pairs from colliding on trunk-based `main`; **Qodana + CodeScene** hold a zero-findings quality bar

`Jira` `GitLab CI/CD` `DigitalOcean App Platform` `Datadog` `React Native/Expo` `Micronaut` `PostgreSQL` `Python + git log`

PART 3 — MEASURED AGAINST MY OWN BASELINE

Four eras, one baseline: me, solo, no AI

SOLO, NO AI

1×

– the unit
– 550 lines/mo
– churn 0.77

WITH OLGA — CHAT → IDE AI

4→6×

– pair/mob, AI arrives gradually
– churn 0.76 → 0.48

SOLO + AI IN THE IDE

8×

– single-threaded
– 4,360 lines/mo
– churn 0.28

SOLO + ORCHESTRATED TEAM

40×

– now
– 22,194 lines/mo
– churn 0.31

Source: monthly git-derived report, regenerated 2026-09-01. Product + test code only; lockfiles, generated files, config and docs excluded.

PART 4 — BUILDER TAKEAWAY

The safety isn't in the agents. It's in the gate.

- Agents never push — a bad agent decision costs a cycle, never a broken `main`
- The orchestrator is a prompt you version, not code you write — it started without even version control and grew, iteration by iteration, to v20 with a changelog
- Measure your own output from git, not from feel — and expect your first measurement to be wrong



Richard Kasperowski

Co-Founder & CTO, Alli Connect (Techstars '22)

awesomelab.org · kasperowski.com · agilesoftwarecourse.org

Looking for collaborators. Ask me for help — any time, I mean it.

Feedback, and stay in touch
menti.com/alp38g4x237e

`↳ how we measured` `↳ quality, not just volume` `↳ DORA standing` `↳ how the prompt evolved`

`↳ how the orchestration works`

Q&A — HOW WE MEASURED

Straight from `git log`

- Every changed file classified: **product + test code count**; lockfiles, generated files, config-data and docs don't
- Baseline is one pinned, defensible window — me, solo, no AI, 46 days — not an average or a guess
- Bulk imports (e.g. a forked library) excluded outright; per-commit insertions capped at 1,000

← back to Q&A

Volume with low rework is defensible. Volume alone isn't.

- **Qodana** — zero static-analysis findings across 75k lines, checked in the IDE daily and in CI
- **CodeScene** — code health and hotspot analysis
- **Zero production incidents** across the entire agent era — the record that matters more than any line count

CHURN IS NOT A QUALITY METRIC

Churn (deletions per insertion) is directionally ambiguous — deleting code is often a *quality* behavior. Call it rework, not quality, and lead with Qodana/CodeScene/incidents instead.

Qodana ↗ CodeScene ↗

← back to Q&A

Q&A — DORA STANDING

One DORA metric is measured. Three are still asserted.

METRIC	STATUS	BASIS
Deployment frequency	Asserted	Many times daily — trunk-based on `main`, deployed by GitLab CI
Lead time for changes	Measured*	PBI cycle time from Jira — start of work to live in prod
Change failure rate	Asserted	~0 — no measured production incidents
Time to restore service	Asserted	Not tracked

*Richard's own definition — start-of-work to live-in-prod. DORA's own lead-time definition (commit-to-deploy) would be trivial to prove too — every commit-to-live cycle runs under an hour — it's just not the number worth tracking here.

The other three could be measured today from a combination of GitLab CI, DigitalOcean App Platform logs, and Expo build logs. Not built yet.

← back to Q&A

Q&A — HOW THE PROMPT EVOLVED

v20 didn't start as version 1 of anything

- Started as ad hoc instructions I re-typed by hand — not even version-controlled at first
- I noticed **I** was the bottleneck to going faster — Theory of Constraints: find the constraint, remove it
- Every revision since has asked the same question: how could I — or we — be more effective tomorrow, or next week?
- Now v20, with a changelog, revised at the retrospective after each run

← back to Q&A

Input, fancy orchestration, output

INPUT
Sprint backlog
(Jira)

→

FANCY ORCHESTRATION
Pairs of AI agents, self-organizing

→

OUTPUT
Pushed code, live in prod

tmux + Tailscale — I can check in and steer the team from anywhere, not just from my desk

I push when I'm satisfied — I'm the PO, I decide when we're done.
Every push goes to prod, in under 10 minutes.

↳ what's inside "fancy orchestration"

← back to Q&A

Q&A — INSIDE "FANCY ORCHESTRATION"

One kickoff a day. Then it loops.

ONCE, TO START THE DAY

1. **Kick off** — I paste the orchestrator prompt into Claude Code
2. **Select** — developers decide which 3 PBIs to work on
3. **Set up** — the SM creates a design & status file
4. **Negotiate** — the team resolves dependencies together, using that file as the shared hub
5. **Pair up** — 3 pairs form, Driver/Reviewer roles assigned

REPEATS, UNTIL DONE

6. **Pair-program & test** — Driver/Reviewer cycle; write and run tests together, including automated UI tests on mobile simulators; swap roles each cycle
7. **Commit** — pairs commit; they never push
8. **Unblock** (as needed) — the SM asks the PO to clarify intent, or to clear an impediment that needs a human

↻ back to pair-program & test — until the PBIs are done or the day ends

← back to the overview