MICKEL TECH · FIELD GUIDE 04

Is your legacy codebase agent-ready?

20 checks that decide whether agent-led modernisation will hold up, and what to build first if it will not.

FOR

CTOs, VP Engineering, heads of platform

FROM

Gordon Mickel · mickel.tech

PRACTICE

Agentic SDLC · legacy migration and modernisation

USE

Answer honestly, per system, with someone who knows it

01 / HOW TO USE THIS

20 checks, one afternoon.

Agents read a legacy codebase faster than any team you can hire, and they can do the bulk of a migration. Whether the result holds up depends less on the model than on what surrounds it: what the agent can verify, what it may touch, and who signs off. Most of that lives in your codebase and your organisation rather than in the tool.

Work through the checks with someone who knows the system. Tick what is true today, not what is planned for the quarter. Each check carries a line on why it matters, so an unticked box tells you what to build rather than only what you lack.

Score per system. A single organisation is routinely ready in a young service and nowhere near it in the core that carries the revenue, and the gap between those two answers is usually the more useful finding.

The checks are deliberately technology-agnostic. COBOL, Delphi, VB6, an ageing Java or .NET monolith, a PHP application that grew for fifteen years: readiness looks much the same from the outside.

> *Agents do the heavy lifting on old code. What keeps that work correct is the engineering around them, and most of it is built before the first agent runs.*

Can the work be verified?

01

The safety net

- ☐ **A test suite exists that goes red when behaviour changes.**
  An agent cannot correct itself against a suite that stays green whatever it does. Verification is what turns generated code into reviewable code.

- ☐ **The suite runs end to end from one command, in under fifteen minutes.**
  Agents iterate. A feedback loop measured in hours is a loop nobody runs, and unverified work ships.

- ☐ **The paths that carry the revenue are covered, or you know exactly which are not.**
  A coverage percentage hides the gap. Naming the uncovered paths tells you where a person still has to look.

- ☐ **Someone can pin down current behaviour in a test without a week of setup.**
  Capturing what the system does today is the first phase of the work. If that step is expensive, everything after it is expensive.

02

A reproducible environment

- ☐ **A new engineer can build and run the system from a clean machine inside a day.**
  An agent gets exactly the environment you wrote down. Undocumented setup turns into silent failure.

- ☐ **The build is scripted rather than performed from memory.**
  Agents work through commands. A build that lives in the habits of one operator cannot be handed to an agent.

- ☐ **Dev, test and production are close enough that a passing test means something.**
  Parity is what lets you trust a green pipeline instead of hand-testing after every increment.

Can it be understood, and moved in pieces?

03

Knowledge outside the code

- ☐ **The rules the business depends on are written down somewhere other than the source.** Where they are not, reconstructing them is real work. Better to budget for it than to discover it halfway.
- ☐ **More than one person can explain why the odd parts are odd.** Old systems are full of deliberate exceptions. When the knowledge sits with one person, interviewing them becomes the critical path of the whole migration.
- ☐ **House conventions live in a file the tooling reads, such as AGENTS.md.** Agents follow written conventions well and guess badly. One page of rules removes a whole class of rework.

04

Room to move in slices

- ☐ **The system has boundaries you could move one at a time: domains, modules or services.** Clean boundaries let the work move one bounded context at a time, and a measured first context is what makes the estimate for the rest believable.
- ☐ **Old and new could run side by side on real traffic for a while.** Parallel running is the cheapest proof of parity there is. It turns a leap of faith into an observation.
- ☐ **A release can be rolled back without a maintenance window.** Reversibility is what keeps the steps small. When every step is one-way, steps get bigger and rarer.
- ☐ **Data can move in stages, or be read from both systems during the transition.** Data is the part of a cutover that is hardest to reverse and most expensive to get wrong.

Can you prove it, and who says stop?

05

Something to measure against

- [ ] **You know the numbers you have today, or could capture them within two weeks.**
  Cycle time, change failure rate and defect counts before the work starts. Without a baseline, "it feels faster" is the only claim left, and it does not survive a board meeting.
- [ ] **Production behaviour is observable: logs, errors, the transactions that matter.**
  Observability is how divergence surfaces while it is still cheap to fix.
- [ ] **There is an agreed definition of done for a migrated component.**
  Otherwise every increment gets re-argued, and the work runs at the speed of the argument.

06

People and limits

- [ ] **Review capacity exists for more change than usual.**
  Agents raise output. Where review is already the bottleneck, that output simply queues, and the gain disappears.
- [ ] **It is clear who signs off on a change to the critical core.**
  Control points need names on them. An unnamed control point is the one skipped under deadline.
- [ ] **Agents run against stated limits: what they may touch, what they must prove, which checks are mandatory.**
  Limits are what make autonomy safe, and they are what makes the run auditable afterwards.

What the count tells you.

16-20	Ready. Move one bounded context end to end, check it against the old system, and use what it cost to estimate the rest.	TICKED
10-15	Workable. Close the gaps in the safety net and in the room-to-move rows first; those two gate everything else on the list.	TICKED
under 10	The first phase of the work is building the ground the rest stands on. That is the normal starting point for a system of this age, and it is far cheaper to find here than mid-migration.	TICKED

What an expert adds

Everything above you can do yourselves, and if you do, the migration gets easier whoever ends up running it. That is a fine outcome for this document.

What I add is a fixed-shape engagement, sized in scoping. A typical shape is about a third of one team for about three months: we generate a documentation layer for the whole codebase, move one bounded context checked against the old system, and turn what that cost into an estimate for the rest. If it stalls, you keep far better documentation than you had.

The context crosses over by one of two routes. Strangler-fig rebuilds it feature by feature while old and new run side by side; clean-room rebuilds the whole context from its specs and switches it over in one step. Whether the data can move in stages usually decides which.

ENGAGE

A 30-minute discovery call is the cheapest way in.

Bring your answers. We go through where the gaps are, whether agent-led modernisation is the right shape for this system, and what a first slice would look like. If it is not a fit, I say so on the call.

BOOK
cal.com/gmickel/discovery30

EMAIL
gordon@mickel.tech

READ
mickel.tech/sdlc/legacy-modernisation

PRACTICE
Binningen, CH · DE / EN