

Brownfield AI — supplementary guide

Companion to the PMI Agile 2026 session *Your AI Writes Code Fast. Your Codebase Has Opinions* — Roman Lobus. From the forthcoming book *Brownfield AI: Shipping Safe Changes to Existing Systems with Coding Agents*.

This is the post-session take-home. The talk put the core model and the calibration framework on the slides; this page carries the deeper cuts there wasn't time for — the delegation ladder, the failure modes, the governance stack, and the nine-step playbook — plus the sources behind every figure on screen.

The model, in one paragraph (recap)

Two specs -> bounded delta -> evidence-first review. Every change to an existing system carries an **actual-behaviour spec** (only verified current behaviour in the slice you're touching — evidence, never intent) and an **intended delta** (only what you've authorised to change — intent, never evidence; a line cannot belong to both). From them you get a **bounded delta** — what changes, what stays (named, not implied), what's out of scope, and open questions *not to be resolved in code*. The change ships with a **proof packet** — what preserved behaviour was verified, what changed and what proves it, what downstream effects were checked, and what is *not* claimed — and review reads preserved-behaviour claims first, the delta second, the diff third. (*The five rigour variables, the three rigour levels, and the ownership map were on the slides; they're not repeated here.*)

The delegation ladder

How much autonomy a coding agent has earned on a given change:

1. **Discussion and clarification** — agent explores, human decides. For slices where current behaviour is still being established.
2. **Bounded edit** — defined scope and acceptance condition, human reviews. Where most routine existing-system work belongs.
3. **Plan-plus-edit loop** — agent plans, human approves the plan, agent executes in bounded increments.
4. **High-control workflow** — requirements, tasks, and validation linked and reviewable at every stage.

Gate for moving up a rung: can the agent evaluate its own output against the acceptance bar? That requires a precise spec. Autonomy is earned by spec precision, not granted by capability.

Spec maturity, three modes

- **Spec-first** — specification written before implementation starts.
- **Spec-anchored** — specification exists and is maintained alongside the work.
- **Spec-as-source** — implementation derives from the specification; the spec is the authoritative record.

In an existing system, a fourth dimension cuts across all three: the quality of the recovered baseline. Spec-first without a verified baseline is intent anchored to a guess.

Five failure modes to design against

1. **Scope widening** — the patch solves a broader problem than authorised, because the broader version is more coherent. A narrow delta removes the room to infer a redesign.
2. **Open questions resolved silently in code** — nobody decided the policy, so the agent decided it. The delta's open-questions section exists to catch these.
3. **Stale context between sessions** — a fresh read of the codebase quietly diverges from the recovered baseline.
4. **Governance theatre** — artefacts present, review gates passed, nobody engaged with what the artefacts said.

5. **Shallow characterisation** — edge cases listed in prose but not pinned by tests. The spec looks solid; the suite isn't.

The five-layer governance stack

1. **Context** — what the agent knows before it starts.
2. **Enforcement** — structural constraints (hooks, path restrictions, change-size limits), not skippable process steps.
3. **Specification** — the two specs and the bounded delta.
4. **Proof** — characterisation tests and the proof packet.
5. **Review** — evidence-first, verifying from the goal.

Governance concentrated only at layer 5 arrives too late: by the time a pull request exists, the agent has already resolved every ambiguity it met. (This is why telling an agent eleven times not to do something is a governance failure of *architecture*, not of emphasis — structure constrains, instructions only guide.)

The nine-step safe-change playbook

1. Classify the request (rigour level, delegation rung) before opening the codebase.
2. Establish local truth — what the system actually does in the slice you're touching.
3. Characterise current behaviour with tests, before implementation.
4. Define the behavioural delta — changes, stays, out of scope, open questions.
5. Map change points, test points, and pinch points.
6. Add conservative seams before editing.
7. Implement with additive patterns — new behaviour alongside old before replacing it.
8. Separate cleanup from behaviour change; never in the same commit.
9. Externalise newly learned context — write down what you recovered before closing the branch.

Steps 2 and 9 are the ones teams skip. Step 2 is what everything downstream stands on; step 9 is what makes the next change cheap.

Pick the first pilot

The right first workflow answers yes to all four: it runs in production today; it has real downstream consequences if it breaks; it changes often enough that the baseline investment pays back within a few months; it is not shared core infrastructure. Measure the pilot on one hard question — *how many changes were specification-compliant but behaviourally incorrect?* That number, not velocity, is what the model exists to reduce.

Sources cited in the session

DORA 2026, citing Stanford productivity research (greenfield vs existing-codebase gains); Faros AI, *The AI Productivity Paradox* (10,000-developer telemetry); Sonar developer survey, January 2026 (trust and verification gap); DORA *State of DevOps* 2024/2025 (AI adoption and delivery stability); METR randomised controlled trial (experienced developers, familiar codebases — predicted +24%, measured -19%; the slowdown is the cost of fixing plausible-but-wrong output, which evidence-first review front-loads instead of paying in production); Demirer, Musolff & Yang, NBER (lines-of-code and pull-request growth vs release rate); Wang, Pradel & Liu, ICSE 2026 (test-passing patches that are behaviourally wrong); Intercom (revert rates under evidence-first review); Pereira, Graylin & Brynjolfsson, Stanford (the review obligation under improving tooling); Michael Feathers, *Working Effectively with Legacy Code* (characterisation tests).