AI-NATIVE ENGINEERING

THE FIELD MANUAL

SHIP FASTER, SAFER, AND SMARTER WITH AI

1000

900

A PRACTITIONER'S GUIDE

Nearform_

CHARTING THE ROUTE TO AI-NATIVE ENGINEERING

THE COST OF INACTION

Enterprises everywhere are experimenting with AI, but most are stuck in pilots that fail to scale. Delivery is still too slow, fragile, and costly. Boards are demanding results. Competitors are moving ahead. The cost of standing still is losing ground.

THE MATURITY JOURNEY

In our work – and echoed by McKinsey's research – we see **four clear stages of AI maturity** in software development:

1 **AI-assisted coding:** Copilots help with autocomplete and snippets, but rarely shift delivery outcomes.

2 **AI-augmented workflows:** AI speeds up tasks like testing, documentation, and planning, but impact remains uneven.

3 **Spec-driven development:** Developers write specifications, agents generate code/tests. A real step-change, but still immature for most enterprises.

4 **Agentic delivery at scale:** Multi-agent teams coordinating across the full software development life cycle. The potential for faster, safer delivery at scale, but few are ready to reach this summit.

Most enterprises today are stuck between stages 1 and 2 – seeing activity, but little real ROI.

N_

G H I J K

LEADERS

WHO THIS MANUAL IS FOR

Senior technology leaders and their teams who feel the pressure to move faster, more safely, and at scale, but know that simply adding more tools won't get them there.

WHY THIS MANUAL EXISTS

This Field Manual is not a theory of the future. It's a set of plays you can run now to escape pilot fatigue and start seeing impact:

START WITH THE TOIL

Free engineers from repetitive work.

TINY TEAMS, BIG OUTCOMES

Lean hybrid teams of senior engineers &
AI can outship much larger squads.

TACKLE GNARLY PROBLEMS

Apply AI where the stakes are high and
progress can't be ignored.

Think of the **maturity model as the map** – and this manual as the field guide for taking the next step.

START WITH TOIL

The safest way to get value from AI is to point it at the boring work nobody wants to do.

WHERE AI HELPS

Code understanding and documentation:
Auto-generate explanations, diagrams, and docs for unfamiliar or legacy codebases.

Code generation and refactoring:
Produce boilerplate, translate old code to modern frameworks/languages, and refactor for best practices.

Testing & quality assurance:
Generate unit and integration tests, create regression tests from logs, and highlight risky changes.

Debugging and issue resolution:
Analyse logs, trace errors, suggest fixes, and surface likely root causes quickly.

Data and system modernisation:
Automate schema translation, migration pipelines, and wrap and expose legacy functionality via APIs.

WHY START HERE

Enterprise teams we've worked with see up to 50% uplift in productivity by starting with toil. It frees senior engineers to focus on architecture and design, accelerating delivery of customer-facing features and reducing technical risk while proving ROI early. Emerging approaches like spec-driven development show bigger future gains, but these tasks are low-risk, high-return entry points.

IMPACT

MAKE IT STICK

Baseline first: Capture today's coverage %, defect rates, or time-to-complete so you can measure real change.

Pick visible pain points: Target jobs that engineers already complain about - endless security alerts, dependency patching, version upgrades, and re-platforming tasks.

Use IDEs with MCP connectivity to ground AI in your own artefacts: Code repos, documentation, backlogs. Combine this with spec-driven practices (like BMAD) so generated scaffolding, boilerplate, and tests follow clear requirements, not guesswork.

Keep senior engineers in the loop: Always route AI-generated work through PR review, and back it with static analysis, security scans, and performance checks. Expect syntactically correct code, but not always optimised.

Measure real outcomes: Look for reductions in LoC, higher test coverage, a smaller backlog of Snyk alerts, faster PR merging, quicker burndown on tech debt tickets

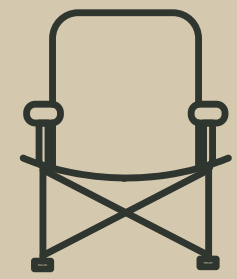
Lock in gains: Bake the improved workflow into your repo and delivery process, so you don't slide back into manual toil.

IN PRACTICE

Nearform: On a five-year-old M&A management platform, AI raised test coverage, refactored and removed dead code, and generated up-to-date docs. This boosts developer confidence, frees up hours each week, and lets them focus on higher-value product improvements.

Bain's 2025 research shows enterprises see early ROI by applying AI to unglamorous but time-consuming work - dependency patching, test automation, and framework upgrades. These tasks rarely excite developers, but automating them frees teams to focus on higher-value challenges

FREE TEAMS
FROM THE GRIND



04

N_

TINY TEAMS, BIG OUTCOMES

A couple of experienced engineers plus AI agents can out-ship much larger teams.

WHERE AI HELPS

Lean staffing: Staff with senior engineers only; let AI agents handle scaffolding, code generation, tests, documentation and setup.

Reuse design systems: Eliminate Sprint 0 by bringing Storybook and component libraries straight into build.

Automate environments: Use Terraform, CDK, or Pulumi to spin up dev/test/prod environments in hours, not weeks.

Build from specs: Apply spec-driven methods like BMAD so agents generate working code from specifications, while engineers refine architecture and domain logic.

ACCELERATION

WHY IT MATTERS

Bigger teams aren't always faster. They often bring more coordination and complexity. Lean teams of senior engineers, amplified by AI, avoid the overhead and deliver faster, with cleaner code and less rework. **An independent study** shows they achieve 20% more merged PRs and up to 40% higher test coverage.

LEADERS

1000

G H I J K

N_

MAKE IT STICK

Pair senior engineers with agents: Let agents wire up logins, scaffolding and test generation, while senior engineers focus on integration and architecture.

Prototype early: Deliver working software in week one - not just wireframes, so clients give feedback sooner and teams converge faster on what drives real user or business value.

Keep engineering discipline: Stick to peer reviews, code standards, and security checks. AI accelerates delivery, but only principles keep systems resilient.

Measure throughput, not headcount: Track features shipped, quality maintained, and cycle times cut.

IN PRACTICE

On greenfield projects, Nearform has deployed AI to handle scaffolding and environment setup, while engineers focused on core logic, eliminating sprint 0 and accelerating delivery.

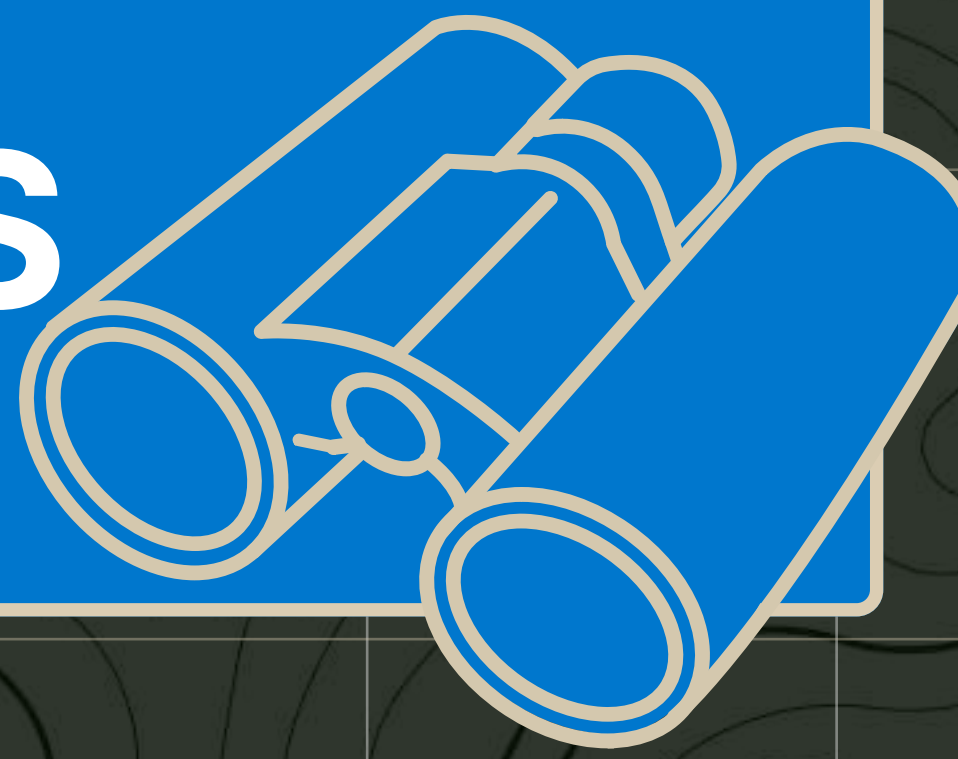
On a brownfield project, two Nearform engineers plus an Agentic Squad cracked a complex NLP problem. Using spec-driven development and senior oversight, the team found an approach up to 80% faster than their baseline, accelerating time-to-value for the client's core platform.

A GitHub study showed developers built an HTTP server 55.8% faster with Copilot, highlighting how AI can meaningfully speed up software development when applied to well-defined tasks.

GET THERE FASTER

TACKLE THE GNARLY PROBLEMS

Real ROI doesn't come from safe side pilots, it comes from solving the big, hairy challenges that force progress.



WHERE AI HELPS

Platform upgrades: E.g. Node 10 to Node 22 migrations or other long-term support jumps.

Dependency modernisation: Upgrading large frameworks or libraries without risking regressions.

Retrofitting tests: Generating unit and functional coverage across a legacy codebase before refactoring.

Codebase clean-ups: Stripping out anti-patterns, dead paths, or inconsistent implementations.

EXPERIMENT

WHY IT MATTERS

According to McKinsey and Gartner, most projects stall in **"pilot hell"** - lots of small experiments, little impact. Tackling gnarly problems forces AI to prove its value: reducing technical debt, unlocking stalled upgrades, and translating engineering efficiency into measurable business outcomes like faster releases, improved reliability, and lower maintenance costs.

MAKE IT STICK

Pick bounded but painful problems: Choose work that's too costly for engineers alone, but constrained enough to validate with tests.

Test first, migrate second: Retrofit coverage (80% or more is achievable) before running AI-led migrations, so you've got a safety net.

Automate refactors: Use spec-driven tools (BMAD, Kiro, Spec Kit) to handle repeatable changes across large repos.

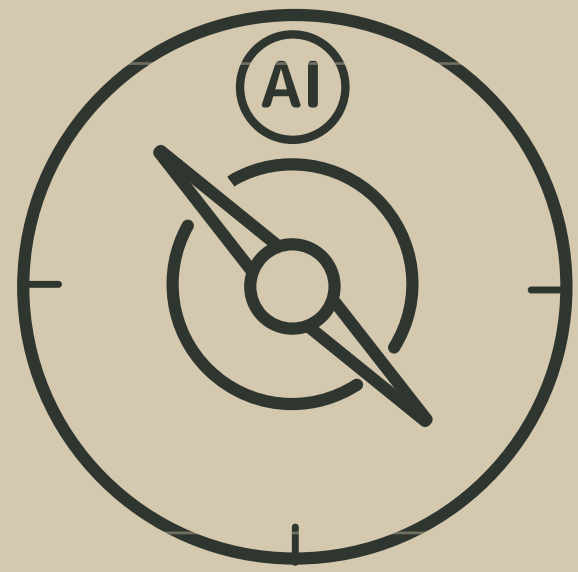
Keep humans on architecture: Engineers guide strategy, validate edge cases, and make the call on trade-offs.

Measure at system level: Look at stability, time saved, and maintenance risk reduced - not just lines of code touched.

IN PRACTICE

Nearform is helping a client modernise a legacy platform. AI parsed and mapped complex mainframe data, automated migration, and validated quality to ensure accuracy. It also accelerated web app development by generating boilerplate code, tests, and UI components - freeing engineers to focus on business logic and user experience.

Dropbox found engineers using AI shipped more pull requests while reducing change failure rates - showing teams can move faster without sacrificing stability, even on complex systems.



AI GUIDED ASSISTANTS

You've read the manual. Now put it to work. Our **Illuminate Sessions** are practical, fast-moving workshops built for senior technology leaders. We'll walk through real examples from the Field Manual, assess your current maturity, and **identify where AI can accelerate your delivery - safely, measurably, and fast.**

CHART YOUR COURSE

Find your first AI win.

Book an **Illuminate Session** today at
nearform.com/ai_native_engineering

Nearform_