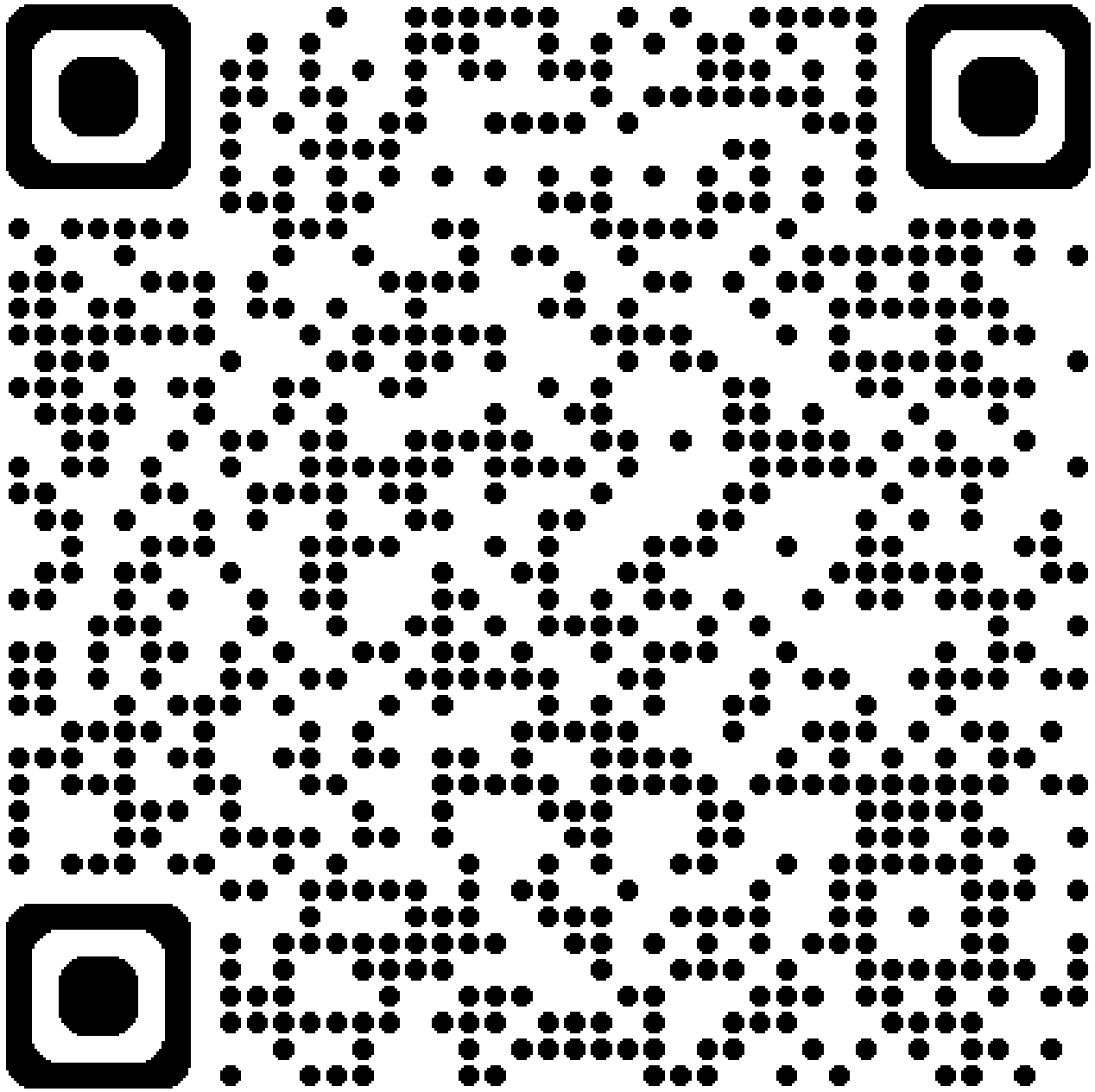
How to Get Started With Automated Testing In the Age of AI

Daniel Ward



Me

LEAN TECHNIQUES

MVP

- Software developer, consultant
- Microsoft .NET MVP
- Co-organizer of the San Antonio/Austin .NET User Group
- daninacan.com
- @danielwarddev
- daniel-ward-dev
- danielwarddev.bsky.social

SAN ANTONIO .NET

AUSTIN .NET

https://www.meetup.com/sadnug/

https://www.meetup.com/austin-net-user-group/

Outline

- Why should we test?
- How does AI change testing?
- How to use AI with testing

Why should we test?

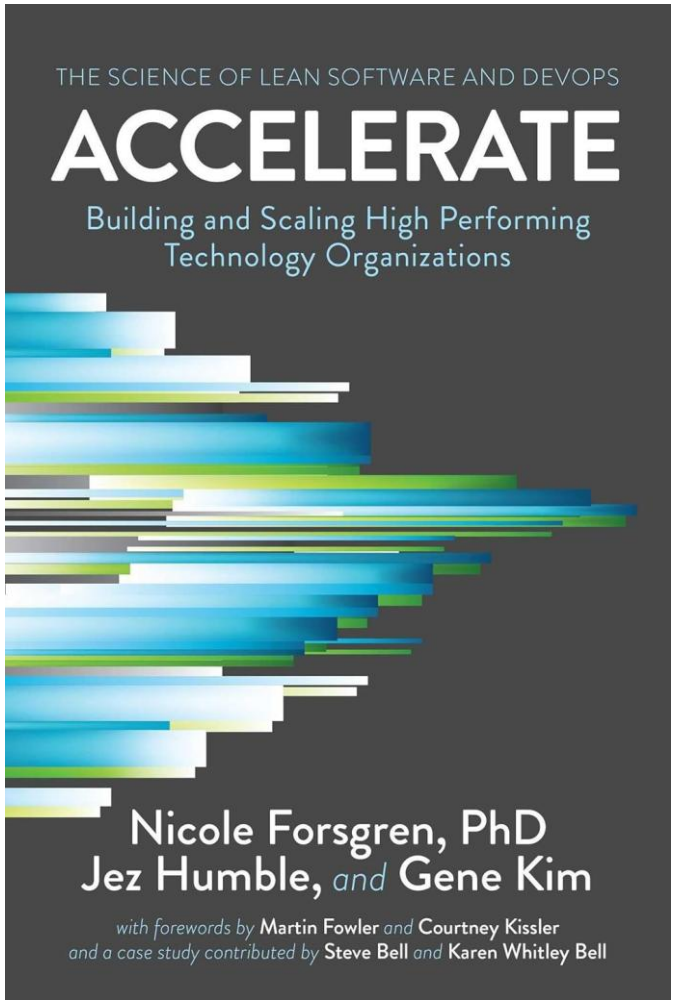
Why do we test?

- Prevent regressions
- Design by acting as the first consumer of the system
- Validate assumptions

DevOps Research and Assessment (DORA)

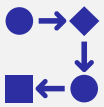
1. What practices predict high performance?
2. How can we foster those practices?

- 30k+ surveys over 5 years
  - Startup to enterprise
  - For profit, not-for-profit
  - Legacy + modern systems



The DORA metrics

Deployment frequency. How often you go to production

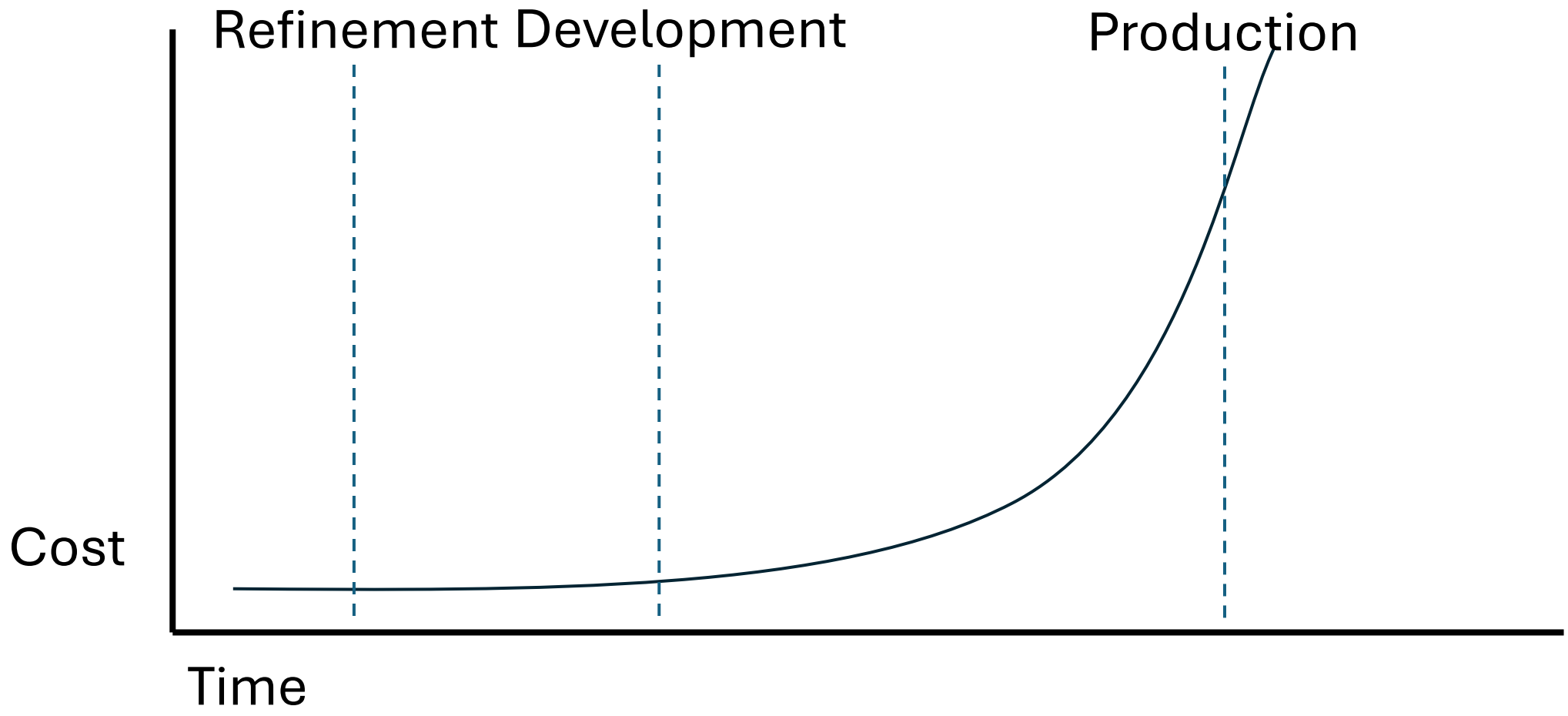


Lead time for change. The time it takes for a change to go to production

Change failure rate. % of deployments that cause a failure

Time to restore service. How long it takes to recover from a failure

Cost of a defect



Confident green

- If all your tests pass, do you feel confident to deploy to prod? Why/not?
- Are you missing test cases? Flaky tests?
- Inverse: if your tests fail, do you believe something is wrong?

How does AI change testing?

DORA and AI

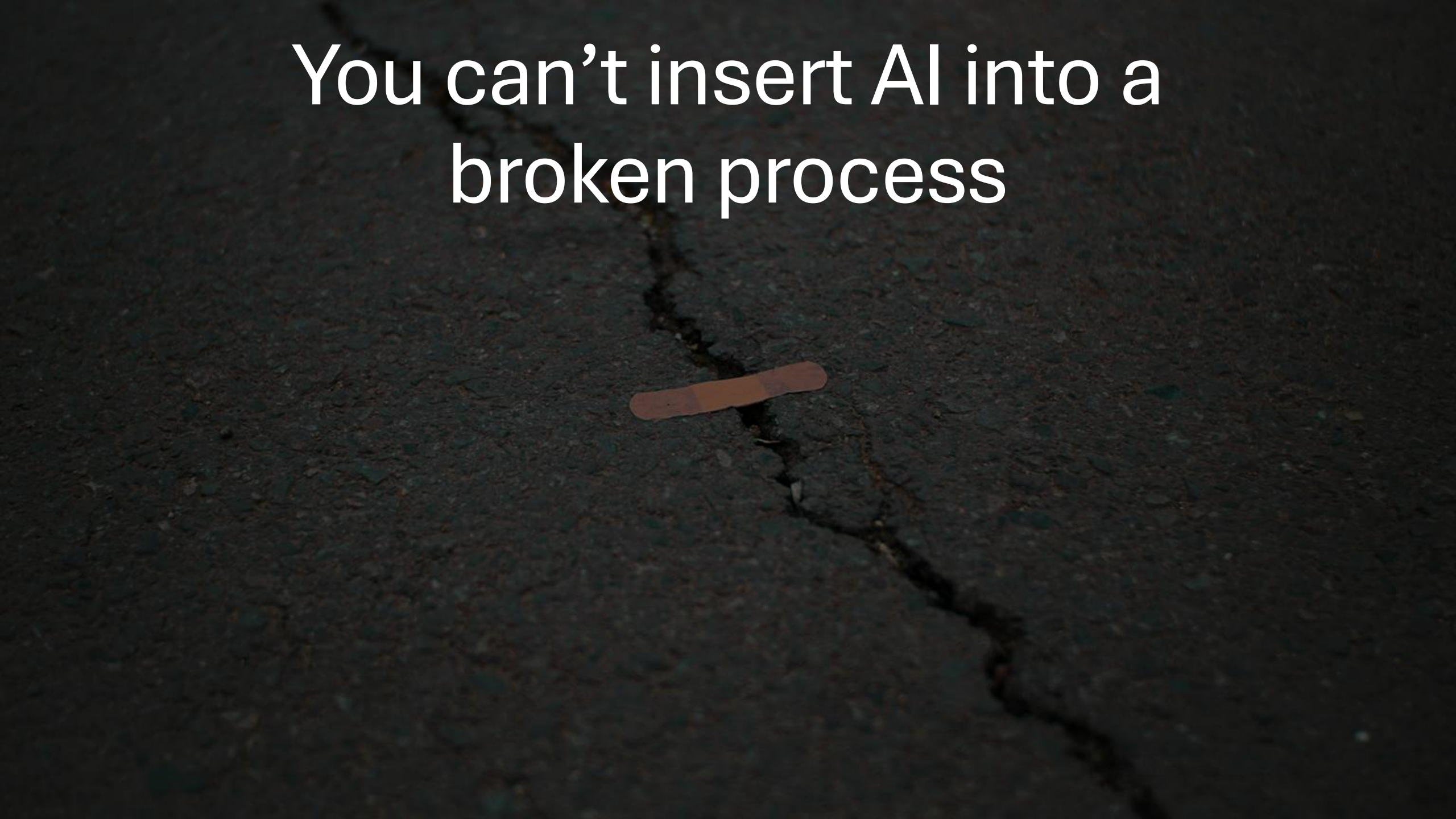
"This confirms our central theory - AI accelerates software development, but that acceleration can expose weaknesses downstream.

Without robust control systems, **like strong automated testing**, mature version control practices, and fast feedback loops, **an increase in change volume leads to instability**.

Teams working in loosely coupled architectures with fast feedback loops see gains, while those constrained by tightly coupled systems and slow processes see **little or no benefit**."

https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report, emphasis added

You can’t insert AI into a broken process



How does AI change testing?

Before AI

Make changes

Did it work?

Verify

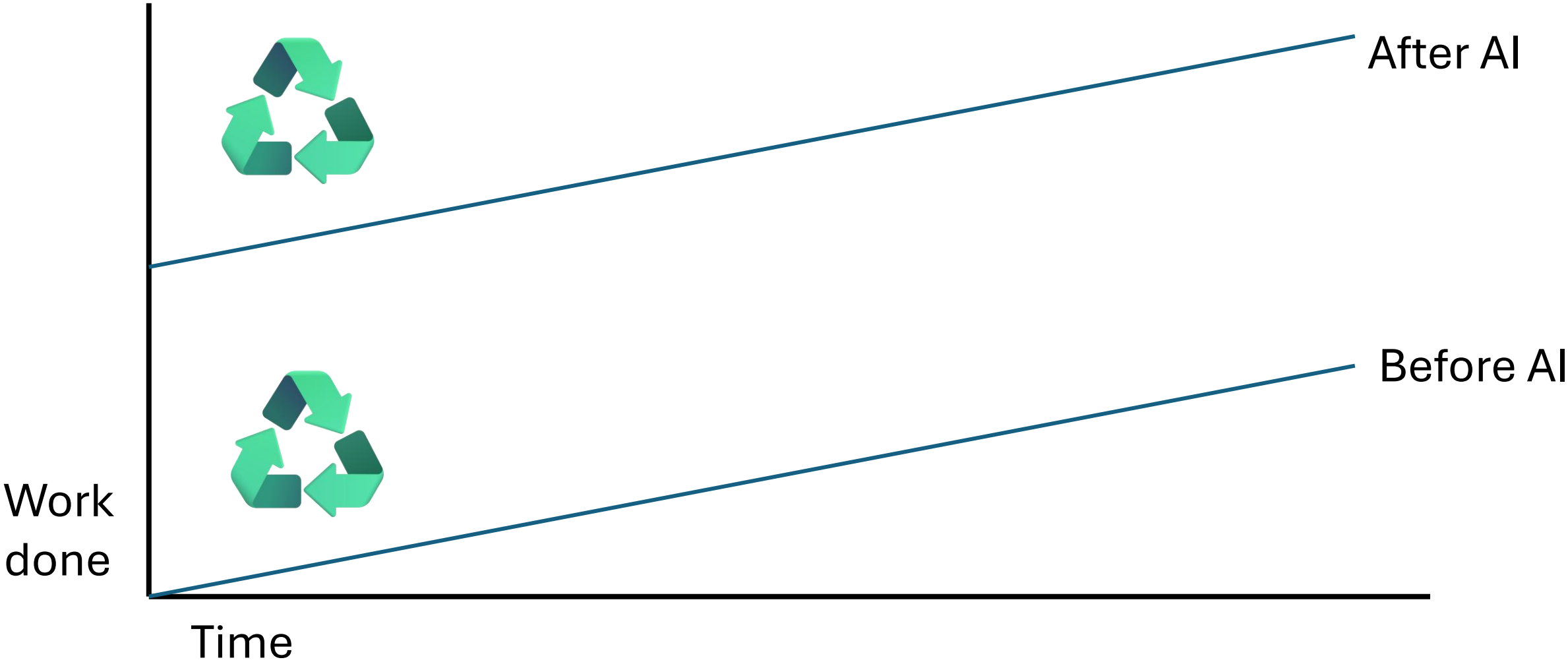
After AI

Make changes

Did it work?

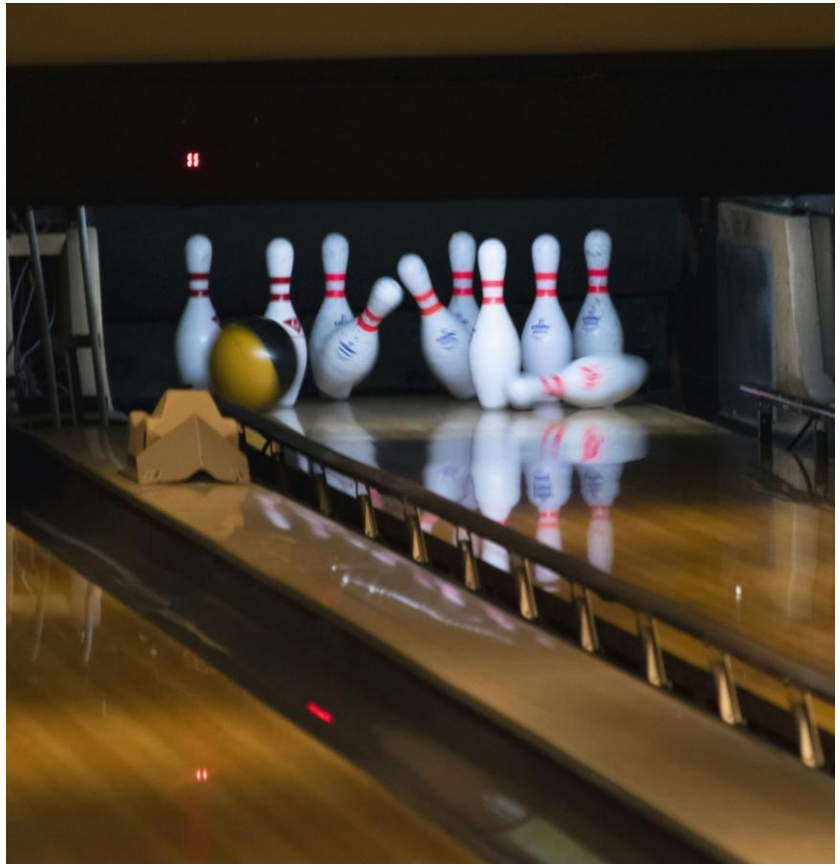
Verify

Before AI vs after AI



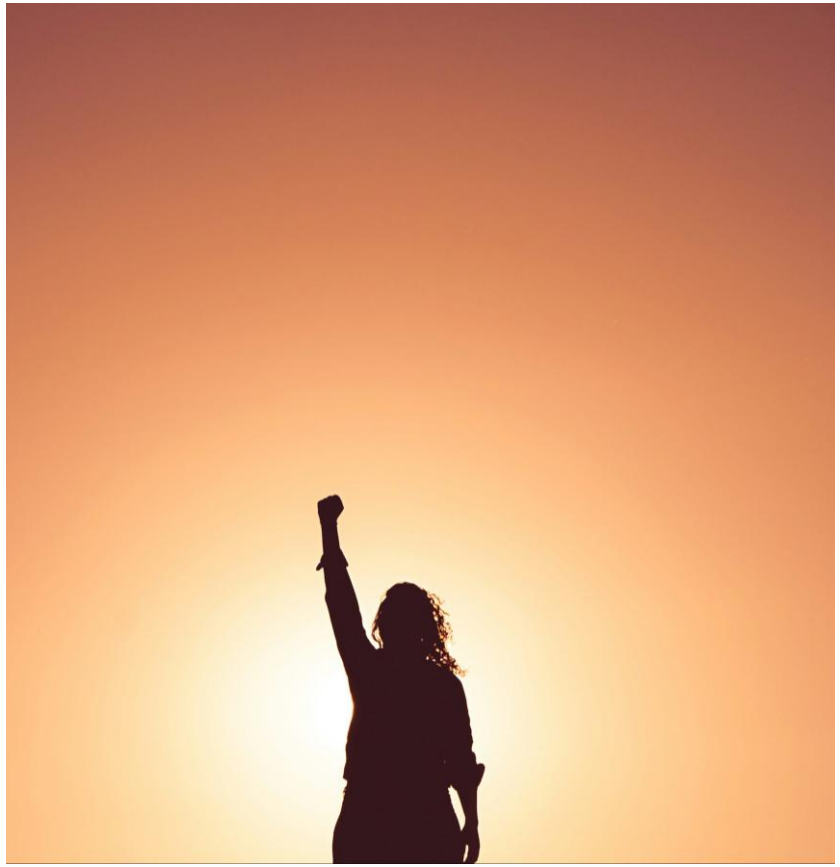
Tests as guardrails

- Test names are plain English requirements for AI
- Test output can be used as feedback for AI



Tests as confidence

- Gives you permission to go faster
- Lowers barrier to entry
- Easier to get confidence in a system
- Easier to get false confidence in a system



Tests as documentation

“Normal” docs

- Updated manually
- Ages quickly once it falls behind
- Provided to AI manually

Tests as docs

- Part of the codebase
- Impossible to fall behind
- Test names are English requirements that the AI can automatically pick up as context

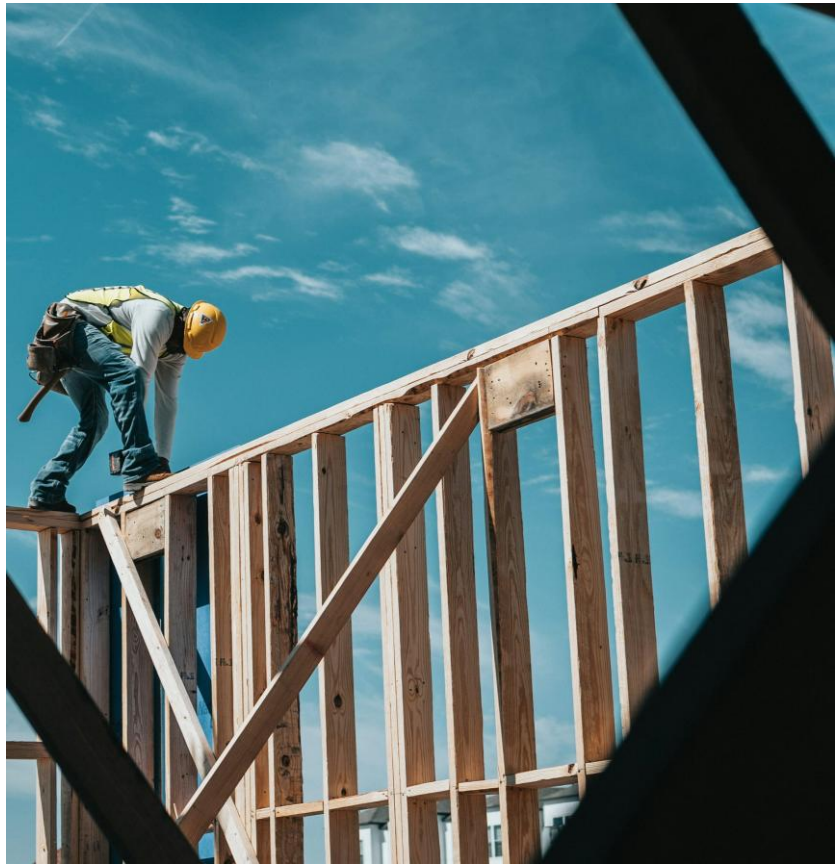
Risks with using AI

- AI makes too many/not enough tests
- Hardcoding passing value
- Removing prod code/tests
- Working in small batches reduces risk

How to use AI with testing

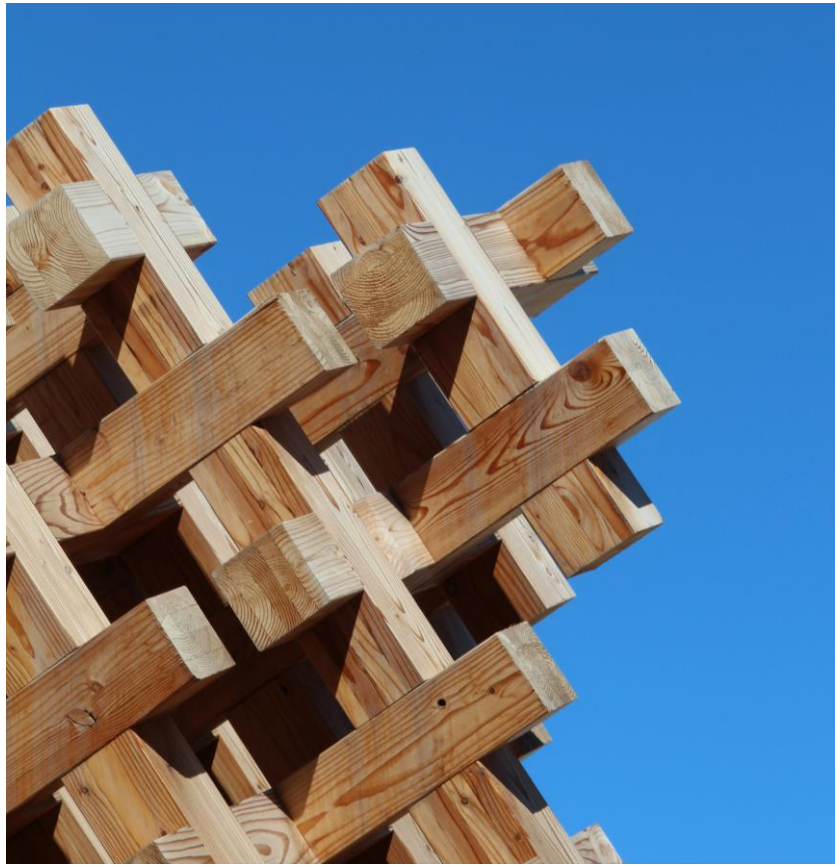
Build up your instructions over time

- agents.md, copilot-instructions.md, claude.md, etc
- Codify corrections
  - “Name tests like <example>, not <example>”
  - “Don’t test method calls in unit tests. Write an integration test instead”
  - “Whenever you write new code, you MUST write relevant unit and integration tests for it”



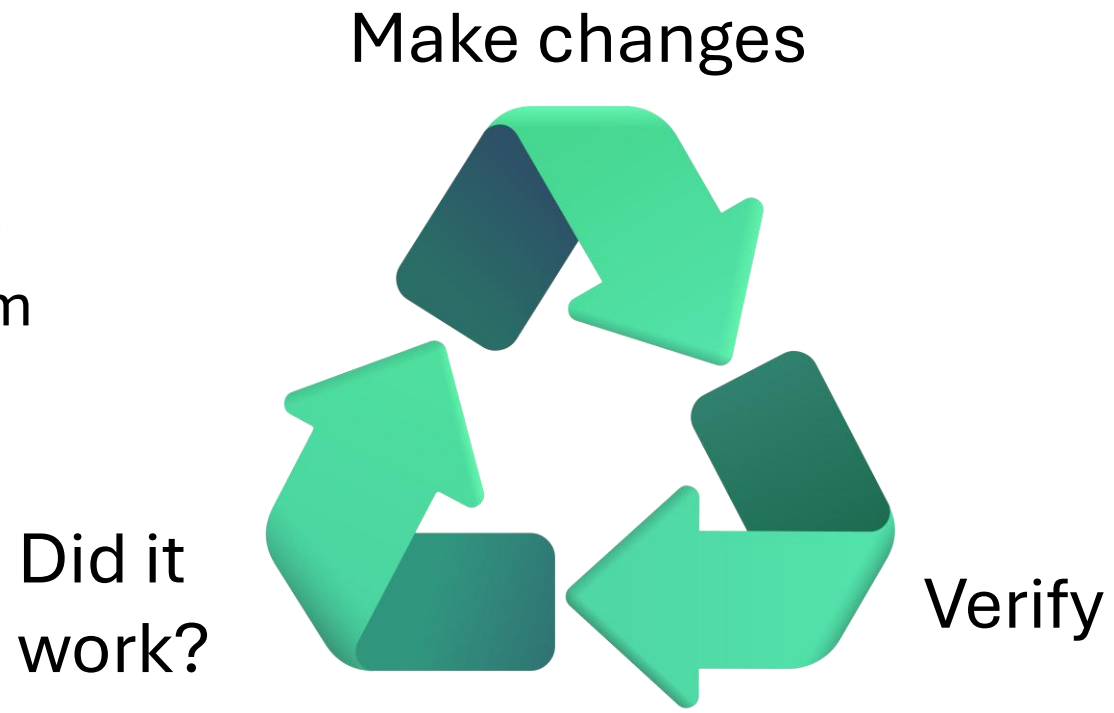
Test structure & naming

- Arrange, Act, Assert
- Clear variable/method/class names
  - result vs paymentIds
  - process() vs applyLateFee()
  - Processor vs ClaimsReimbursementProcessor
- Human-readable test names
  - Given, when, then
  - When_Header_Row_Is_Invalid_Then_Throws
  - “when the modal is reopened, then the form should be reset”



Feedback loop

- “Whenever you change code, you must run the linter, build, and tests. If they fail, continue making changes until they pass”
- AI automatically get test output as feedback
- verify: npm run build && npm run lint && npm run test && npm run start



Agent skills

- Instructions to AI that it uses automatically
- Don’t need too many of these

```
---
name: coding-standards
description: 'General code and UI architecture standards for MindMapAi. Use when changing
C#, Razor, services, or project structure. For detailed test workflow, use the
testing-standards skill.'
argument-hint: 'Code area or architectural change'
---

# Coding Standards
```

Common skills I use

coding-standards	testing-standards
• Keep business logic out of frontend components • Follow the Robustness Principle for types • Follow SRP	• Test outcomes, not method calls, • Move shared setup into a single spot • Name tests “when x happens, then y should happen”

Prefer pure functions

- Same input will always give the same output
- No side effects
- Less edge cases for the AI to mess up

```
function getDiscountedPriceImpure(price: number): number {
  const rate = parseFloat(process.env.DISCOUNT_RATE ?? "0");
  return price * (1 - rate);
}
```

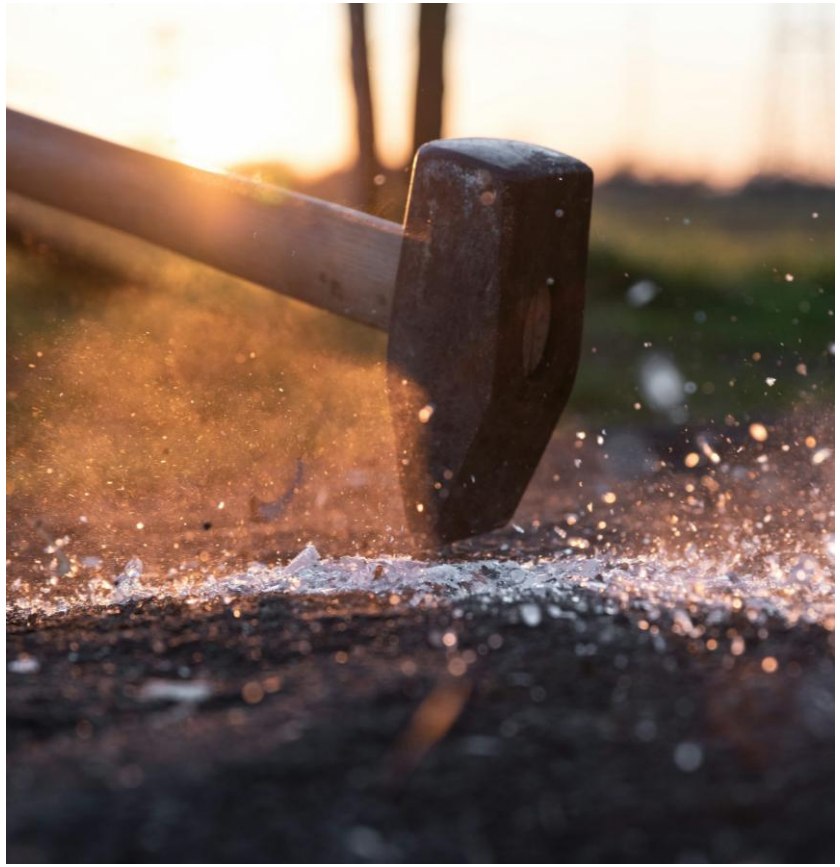
```
function getDiscountedPrice(price: number, discountRate: number): number {
  return price * (1 - discountRate);
}
```

Blackbox tests

- Characterization tests
- Great for legacy codebases

Try to break it

- “What edge cases am I missing for this function?”
- “What inputs could break this?”
- Mutation testing
  - Turn + into –
  - True into false
  - etc



High quality test data

- Easier to create fake data & data factories
- CSV, JSON, etc for integration tests
- ⚠️ Warning!
  - Do you need all the test data?
  - Are you testing the easiest way?

Parameterized tests

```
Run | Debug
it("adds 1 and 2", () => {
  expect(add(1, 2)).toBe(3);
});
```

- Easier to generate with AI
- More test confidence + more context/guardrails for AI
- Can be harder to read
- Make sure they actually add value

```
it.each([
  [1, 2, 3],
  [0, 0, 0],
  [-1, -2, -3],
  [-1, 5, 4],
])("add(%i, %i) === %i", (a, b, expected) => {
  expect(add(a, b)).toBe(expected);
});
```

5 point recap

1. Test automation is a big part of making quality software
2. AI helps us do **everything** faster
3. Work in small batches and iterate to lower risk
4. Review all AI output!
5. Getting started – have AI write a test and build up your instructions over time; blackbox tests for legacy code

Thank you!