💬 lets talk about the most popular non-strategy in software delivery...
💬 Just Try Harder

- heard it → said it

Problem: 💬 like when
- bugs **escape**, estimates **slip**, releases **break**
- CTO → incident → CTO things
- fire out → retro → just try harder

Response: 💬 so we end up with
- approvals → checklists → friction

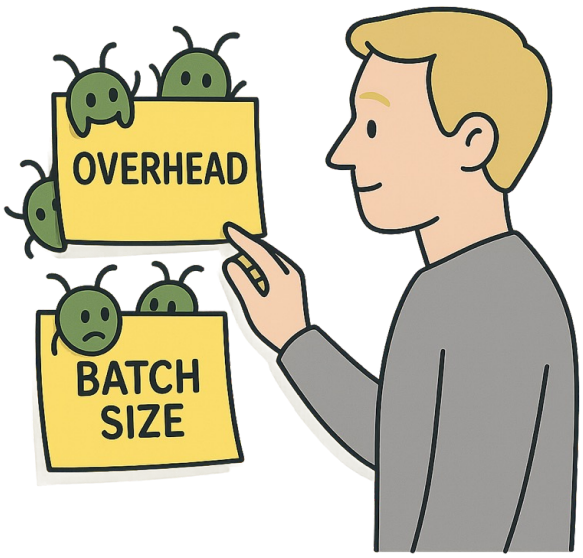
Real Problem: 💬 here's the thing
- not strategy → broken system → failure easy

Why Care: 💬 and why do we even care about bugs?
- features
- **break** flow, **shake** trust, **kill** momentum

💬 so let's stop blaming people, and start fixing the system

What the Research Says

→ From Boehm to DORA – creating systems where bugs can't hide



headforwards

💬 I wanted to take a step back, and look at why bugs slip through the net
💬 the research has said the same thing for decades

1981: 💬 Boehm found
- early catch → cheap fix

today: 💬 DORA found that
- **speed** → **stability** → hand in hand
- fast feedback → better outcomes
- Build **small**. Ship **often**. **Avoid** bugs. Fix **fast**.

- more approval → worse quality

Chain: 💬 here's how I like to think about it
- Overhead → Batch Size → Complexity → Hiding Place → Bugs

💬 so, that's the theory
💬 here's how it played out on our HealthTech platform

Fast Feedback

→ Get feedback in the time it takes to make a cup of tea

headforwards

💬 we had a test suite that took over 30 minutes to run

Problem: 💬 and what did the team do?
- avoided running → batch changes → push everything → cross fingers

Problem: 💬 and when the pipeline failed?
- quick and dirty → make it pass

Impact: 💬 the result?
- bugs slipped through
- context lost → stress levels high

Impact: 💬 we ended up debugging code from days ago
- ask what?

Takeaway:
- delayed feedback → bigger batches → complexity → risk

💬 so we needed a reset, we made a new rule
💬 “Get feedback in the time it takes to make a cup of tea”

Fast Feedback

→ Get feedback in the time it takes to make a cup of tea

headforwards

💬 we set out to make it real

Solution: 💬 we set some targets
- 1 min local → 5 min pipeline
- **local runs**: analyse slow → prioritise → turn off
- **pipeline**: optimise → no framework → no database → hunt anything

Result: 💬 and the result?
- hit targets

- **get** faster feedback → **push** smaller changes → **make** cleaner commits
- in flow → fix as created

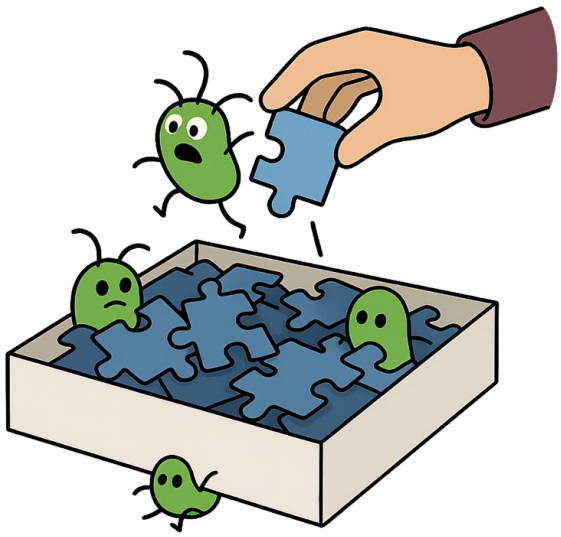
- 20 hours → sort of → free build and test
- once → multiple

💬 but, even with fast feedback...
💬 our platform made every change feel risky

Smaller, Safer Changes

→ You can't inspect a fourth leg onto a three-legged table

headforwards



💬 fast feedback had helped
💬 but it didn't solve the fear of touching our legacy systems

Problem:
- small change → big blast-radius
- one change → chaos in five

Problem: 💬 and it was a full-time job
- triage & firefight
- bugs
- customers & staff

Reaction: 💬 up until now
- catastrophe →
- checklist-item **appeared**
- process **introduced**

💬 it reminded me of something I heard at university

Smaller, Safer Changes

→ You can't inspect a fourth leg onto a three-legged table

headforwards

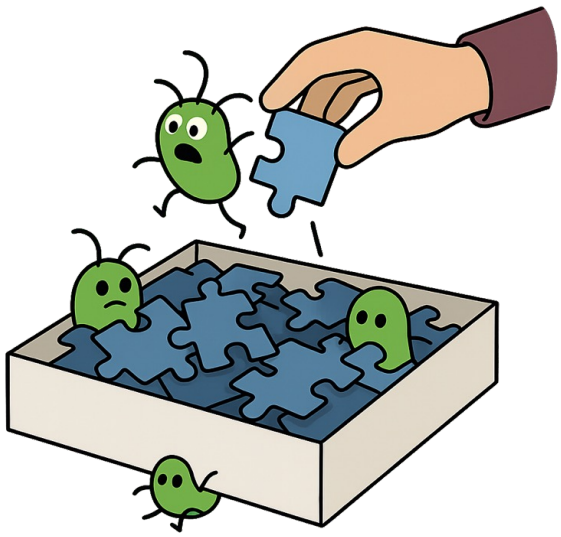
💬 you can't inspect a fourth leg onto a three-legged table

- inspectors
- factory
- white coats
- clipboards
- three legs
- should be four

💬 and there was nothing they could do

Smaller, Safer Changes

→ You can't inspect a fourth leg onto a three-legged table



headforwards

💬 so we took a different approach

Solution: 💬 we followed the strangler fig pattern
- replace old → chunk by chunk
- rebuild → independent
- good tests → fast pipeline → from day one

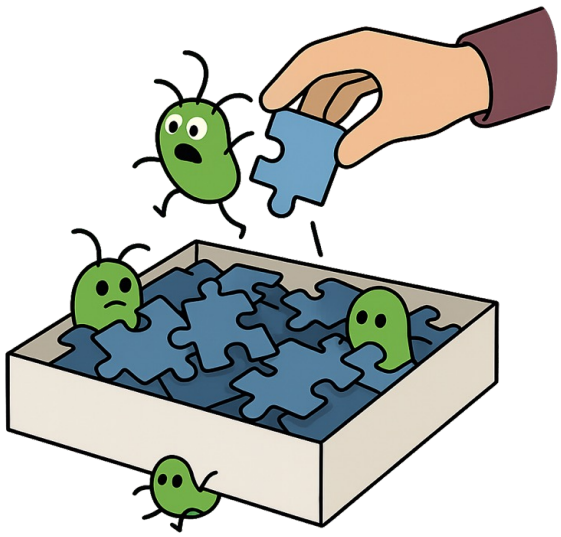
- ready → switch → remove
- Expand. Migrate. Contract.

Results:
- smaller scope → safer releases
- fewer approvals → faster flow
- release on demand

💬 that gave us confidence in the new
💬 but the legacy was still holding us back

Smaller, Safer Changes

→ You can't inspect a fourth leg onto a three-legged table



headforwards

💬 we were deploying the new services multiple times a day
💬 but the legacy? still took two weeks (if we were lucky)

Problem: 💬 every sprint finished with a scramble
- cherry-pick → extract → feature → last minute
- cross fingers → hope → ship

Solution: 💬 so we shifted our mindset
- towards trunk-based → always releasable
- automate pipeline → stop manual releases

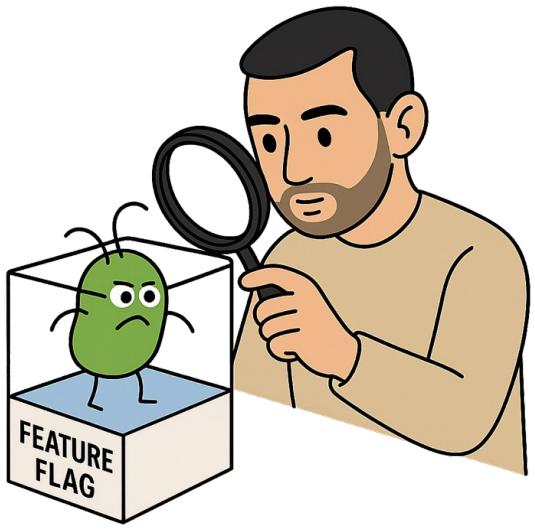
Result: 💬 and the result?
- business → trust → firefighting → reduce
- duty engineer → triage → fixes & features → fires
- 2 weeks (maybe) → 2 days (if needed)

Takeaway:
- more approval → know → worse quality

💬 now we could ship often, the next step?
💬 ship without fear

Controlled Delivery

→ Find bugs before your customers do



headforwards

💬 shipping fast was great, but we still needed to ship safely

Problem: 💬 because no matter how good your tests are
- production → final test
- users → data → load
- edge cases

Problem: 💬 and in production
- bugs → stand out → hide below

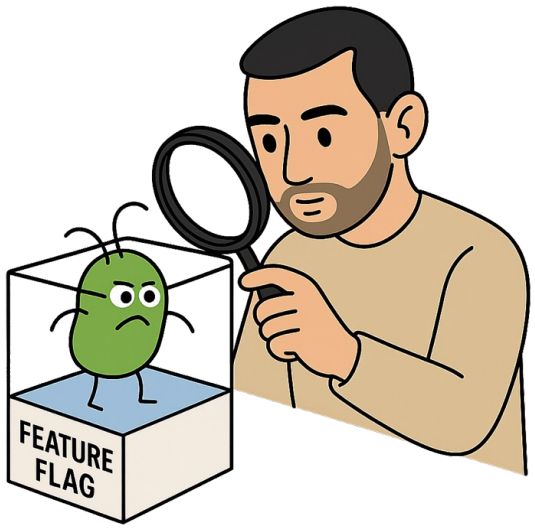
Problem: 💬 we were about to launch a new service
- send → sensitive → health
- **lab** results → **heart** measurements → appointment **reminders**
- mistakes → embarrassing → regulatory risk → real-world harm

Strategy: 💬 so we took a three-pronged approach
- monitoring → tells you something's **wrong**
- observability → tells you **why**
- feature flags → control **blast radius**

💬 we didn't just build alarms, we built safety nets

Controlled Delivery

→ Find bugs before your customers do



headforwards

💬 here's how we turned production
💬 into a low-risk learning space

Solution: 💬 first, we dark-launched the service
- feature flags → old, parallel, new → same input/output → side by side
- hidden

Solution: 💬 next, we added monitoring and observability
- **tracking**: delivery rates → delays → discrepancies → old vs new

Solution: 💬 finally, we started with a safe group of customers
- staff → use product → gradual → confidence grow

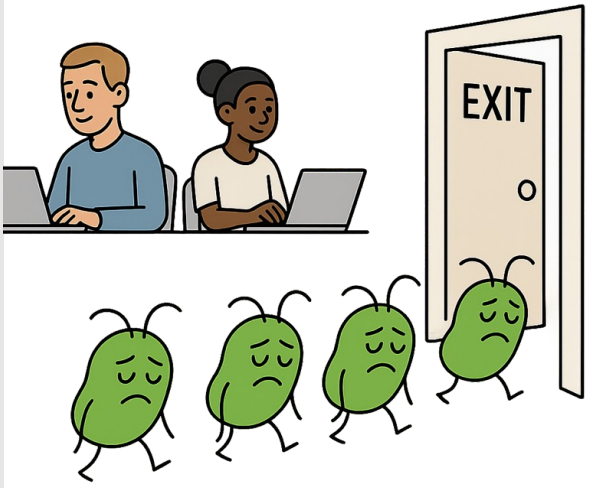
Result: 💬 and the result?
- **delivered**: safely → confidently

Takeaway: 💬 when you're working in a high-stakes context
- fast delivery → not enough → controlled delivery

💬 less **stress**, less **risk**, and nowhere for bugs to hide

Go Beyond Just Try Harder

→ Build systems that make the right thing easy



headforwards

💬 we went from **firefighting** and **fear**
💬 to **feedback** and **flow**

💬 not by pushing **people** harder
💬 but by changing the **system** they work in

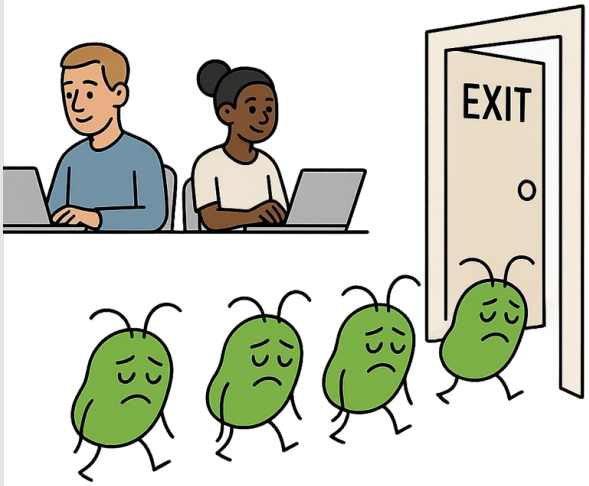
💬 because “Just Try Harder” **doesn’t work**
💬 it **doesn’t fix** broken systems

💬 but the **data** shows us **what does work**
Build **small**. Ship **often**.
Avoid bugs. Fix **fast**.

💬 that’s why these three **strategies matter**
Fast feedback — make **bugs cheaper** to catch
Smaller, safer changes — make **changes easier** to **ship**
Controlled delivery — make **production** a **safe place** to learn

Go Beyond Just Try Harder

→ Build systems that make the right thing easy



headforwards

💬 when your **system works**
- overhead → **drops**
- batch size → **shrinks**
- systems → **simpler**
- bugs → **nowhere to hide**

💬 and this isn't just a developer or tester thing
💬 Scrummasters, Product Owners, Engineering Managers
💬 the whole organisation **shapes** the **system**

💬 **better software** isn't about trying harder
💬 it's about **shaping systems** that **help you succeed**

💬 so let's build **better software**

💬 let's go beyond "Just Try Harder"

Go Beyond
Just Try Harder

→ Explore the ideas, tools, and resources behind the talk

Illustrations generated with ChatGPT + DALL·E (OpenAI)

headforwards

💬 Thanks for listening 🙏