💬 *lets talk about the most popular **non-strategy** in software **delivery**…*

💬 *Just Try Harder*

you've heard it → you've said it

bugs escape, **releases** break
CTO → **incident management** → CTO things

fire out → retro says
devs JTH **not** write → testers JTH **catch**, devs JTH **not** write

get → approvals → checklists → friction

why care?
break features
break flow, **shake** trust, **kill** momentum

here's the thing
JTH **isn't** strategy → **sign** broken system → **system** easy fail

💬 *so let's stop blaming people…*

💬 *and start fixing the system*

Hi, I'm Andy
Software Engineer

💬 *hi, I'm Andy — I'm a software engineer by trade*

💬 *I thought…*

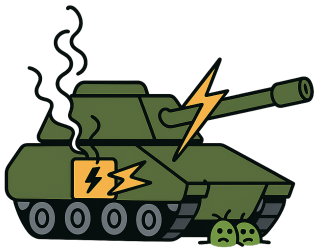
💬 *since we're spending the next 30 or 40 minutes together…*

💬 *we should get to know each other a little*

💬 *I didn't start out in software*

Hi, I'm Andy

I started out in electronics
fixing tanks



💬 *I **started out** in **electronics**, when i joined the Army…*

💬 *and ended up **fixing** tanks*

Hi, I'm Andy

I got a degree
and job designing electronics
for oil and gas

💬 ***after that**, I got a degree…*

💬 *and started **designing** electronics for the oil and gas industry*

Hi, I'm Andy

Bought a guesthouse.
Building a website...
How hard can it be?

💬 ***next***, ***we*** *bought a guesthouse*

💬 *back then, having your own website was still new*

💬 *we needed one…*

💬 *and I thought, "how hard can it be?"*

Hi, I'm Andy

And here I am

💬 ***turns out*** *— not that hard*

💬 *one thing led to another…*

💬 *and here I am…*

💬 *on* ***stage***

💬 *at* ***Agile on the Beach****…*

💬 *talking about* ***Software Delivery***

💬 *but enough about me — what about* ***all of you****?*

Hands Up

Technical people

💬 *quick show of hands…*

💬 ***who here** is technical?*

💬 *so, developers, testers, data, ops, platform…*

💬 *the people who build and run software?*

Hands Up

Product people

💬 ***how about*** *product people?*

💬 *UX/UI, Product Owners, Business Analysts…*

💬 *the people who look after our users and customers?*

Hands Up

Leadership & business people

💬 ***and finally***, *anyone from the leadership or business side?*

💬 *Scrum Masters, Engineering Managers…*

💬 *and the people who look after stakeholders and finances?*

💬 *and make sure we all get paid*

—

💬 *great — a good mix*

—

💬 *hopefully there's something here for everyone…*

💬 *and I'll do my best to bring you all along on the journey*

Beyond
Just Try Harder

Three effective strategies to tackle bugs

💬 *one **quick note** before we **dive in**…*

💬 *this talk isn't a **recipe**, a **framework**, or a **model***

💬 *it's a **story**…*

💬 *one based on **well-documented techniques**…*

💬 *that we **adapted** to our context…*

💬 *and that **worked well** for us*

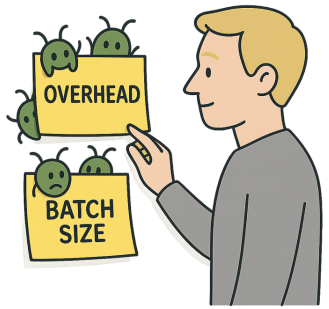
💬 *I hope you'll find something useful in the journey*

💬 *and with that…*

💬 *back to the story*

What The Research Says

From Bohem to DORA
creating systems where bugs can't hide



💬 *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

💬 *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*

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

💬 *and what did the team do?*

avoided running → batch changes → push everything → cross fingers

💬 *and when the pipeline failed?*

quick and dirty → make it pass

💬 *the result?*

bugs slipped through
context lost → stress levels roof

💬 *we ended up debugging code from days ago*

ask what?

delayed feedback **lead** → bigger batches
bigger batches **created** → complexity
complexity **increased** → risk

💬 *so we needed a reset, we made a new rule*

💬 *“Get feedback in the time it takes to make a cup of tea”*

💬 *not a… netflix → lunch → just tea*

💬 *we set out to make it real*

💬 *we set some targets*

feedback…
locally → 1 min
pipeline → 5 min

local runs → analyse slow tests → prioritise → turn off

pipeline target → optimise → no framework → no database
hunt anything slow

💬 *and the result?*

hit targets

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

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

💬 *Let's start looking at what we **inherited***

💬 *because **context** shapes **strategy***

Quality Relied on Instinct: 💬 the day I joined

- no automated tests or quality checks
- new QA → visual regression POC → fast coverage
- relied on manual testing
- call center → QA → "spidey-sense"

Tests Were Broken

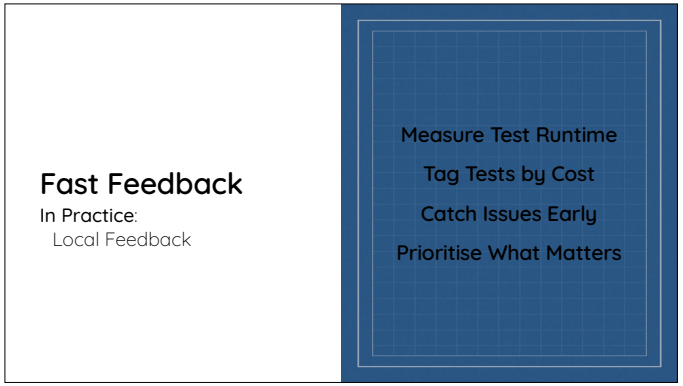
- **were** some tests in codebase - half broken/failing
- once the broken/failing tests were fixed up…
- they ran sloooow (30-minutes)

Pipeline Ran Nothing

- the pipeline did nothing except run deployment scripts
- no quality, no security, no guardrails

- **Framework 100x Overhead**
- cause of 30-minutes
    - Laravel ≈ 10x slower than raw PHP
    - database ≈ 10x slower again (100x overhead total)

💬 *that was where we were starting from*



💬 *so we started to tackle this…*

💬 *with the one thing developers touch all day - local*

Measure Test Runtime: 💬 we…

- used IDE to profile tests - file time, individual test time
- profiling led to Laravel/Database discovery
- results → identify and tag worst offenders

Tag Tests by Cost: 💬 we…

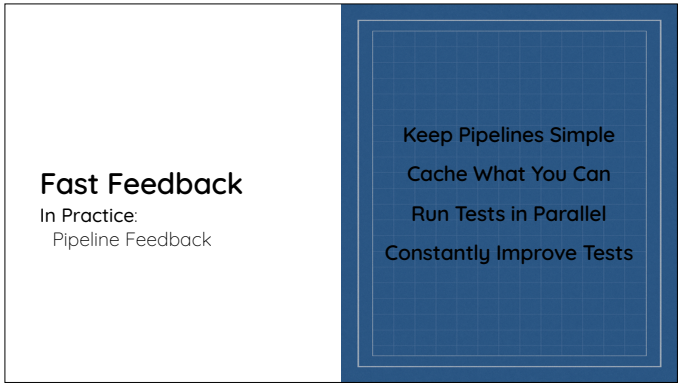
- **@unit**: pure PHP - **fast**
- **@integration**: Laravel, no DB - **medium**
- **@e2e**: Laravel + DB - **slow**
- **@slow** - **really slow** - set threshold – balance of speed and coverage

Catch Issues Early: 💬 we…

- **git hooks** → **pre-commit** checks — configured in the **repo**
- **@unit** ran constantly for **development**
- **@integration** ran **pre-commit**
- **codestyle** & **linting** also added to **pre-commit**

Prioritise What Matters: 💬 we…

- **@unit** during dev - **@integration** pre-commit
- **@e2e** in pipeline
- **@slow** when working with tested code



💬 **next**, we brought that same fast-feedback mindset into pipeline

Keep pipelines simple: 💬 we…

- standard **Bitbucket Pipelines** setup
- no custom workers, runners
- just out-of-the-box caching and parallelisation

Cache What You Can: 💬 we…

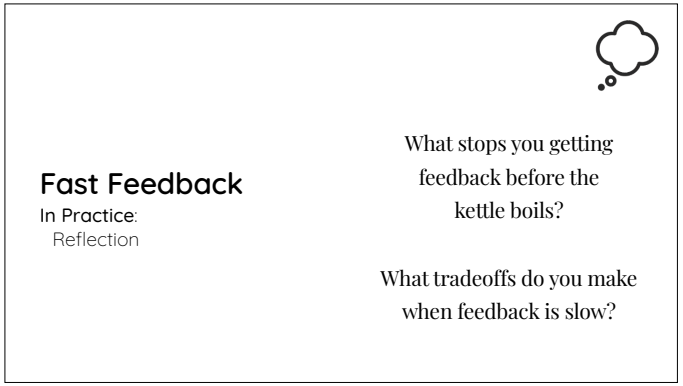
- cached build dependencies and Sonar runs to reduce cold starts

Run Tests in Parallel: 💬 we…

- **shared install step** cached dependencies before split
- pipeline ran **5 jobs in parallel**: codestyle, linting, unit, integration, e2e

Constantly Improve Tests: 💬 we…

- follow "**boy-scout rule**" – fix or move tests down the hierarchy
- @e2e → @unit
- **backlog** @slow - fix when in the area - cost vs benefit
- change hard → eng. meetings → docs → code reviews
- shift **culture** toward fast feedback



💬 *before we move on, let's pause for a moment…*

💬 *here's a couple of questions for you to think about*

—

💬 *What stops **you getting** feedback **before** the kettle boils?*

💬 *What tradeoffs **do you make** when feedback is slow?*

—

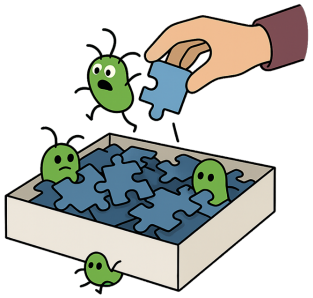
(monologue)

💬 *for us, 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



💬 *fast feedback had helped*

💬 *but it didn't solve the fear of touching our legacy systems*

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

💬 *and it was a full-time job*

triage → firefight
bugs
customers → staff

💬 *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

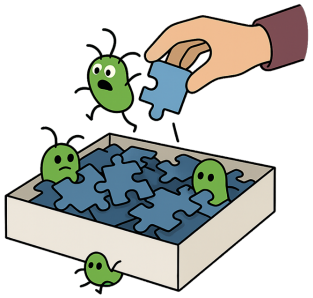
💬 *you can’t inspect a fourth leg onto a three-legged table*

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



💬 *so we took a different approach*

💬 *we followed the strangler fig pattern*

replace legacy → chunk by chunk
rebuild → independent services
good tests → fast pipeline → from day one

ready → switch → remove
Expand. Migrate. Contract.

💬 *and the results?*

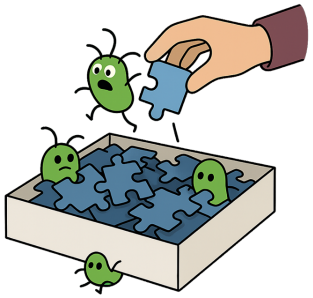
smaller scope → **safer** releases
fewer approvals → **faster** flow
release on demand

💬 *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



💬 *we were deploying the new services multiple times a day*

💬 *but the legacy? still took two weeks (if we were lucky)*

💬 *every sprint finished with a scramble*

cherry-pick → extract → feature → last minute
cross fingers → hope…
ship

💬 *so we shifted our mindset*

move towards **trunk-based** → always releasable
automate what we could → stop **manual** releases

💬 *and the result?*

business → trust
firefighting → reduce

duty engineer → triage bugs
fixes → features
fires

release → 2 weeks (maybe) → 2 days (if needed)

not more approval → know → worse quality

💬 before we could make safer changes…

💬 we had to understand why it all felt so risky

Architecture With No Edges
- high complexity - if/eles, switch/case 10s, method 100s, class 1000s
- two portals, staff/customer, copy-paste-modify - two databases
- 4 repos - distributed monolith - test/deploy together
- early am deploys - hide sync - code-data - frontend-backend

Legacy on Legacy
- insource-outsource - framework customisation - not idiomatic
- old and new approaches side by side
- multiple unfinished “v2” efforts
- API naming arbitrary - ignored out-of-the-box RESTful

Every Change Hurt
- 6-weeks work (e.g. RU99) took 6-months+ - finding bugs year later
- missed timelines → norm → eroded business trust
- quarterly planning became a ‘get it all in now’ rush

Safety Theatre & Burnout
- incidents added checklist items - sign-off process longer, not safer
- support engineer - full-time job
- big-bang releases - frequent high-risk hot fixes

💬 We needed change, but rewrites weren't realistic…

💬 so we carved the legacy out, one piece at a time

Replace Piece by Piece: 💬 we…

- started with Results → after 6-weeks → 6-months → year
- Strangler Fig pattern — migrate one service at a time, not big bang
- break the modular monolith - independently deployable services

Bake in Safety: 💬 we…

- new services…
- good fast tests → contract tests
- pipelines gate releasability — green means go
- pre-commit hooks + SAST tools (Sonar, Snyk) caught issues early
- move ownership - author deploys

Create Boundaries: 💬 we…

- ddd & hexagonal architecture — designed for change
- event-driven design enabled loose coupling
- anti-corruption layers isolated legacy logic

Simplify legacy: 💬 we…

- deleted clean, flagged code
- cautious with messy areas
- dead code remained, complexity reduced

💬 *but shipping the legacy was still painful…*

💬 *so we tackled that next*

Build Small, Merge Often: 💬 we…

- long-lived branches caused integration headaches
- merges piled up at the end, increasing conflicts and delays
- break work into small commits and merge early

Keep Moving Forward: 💬 we…

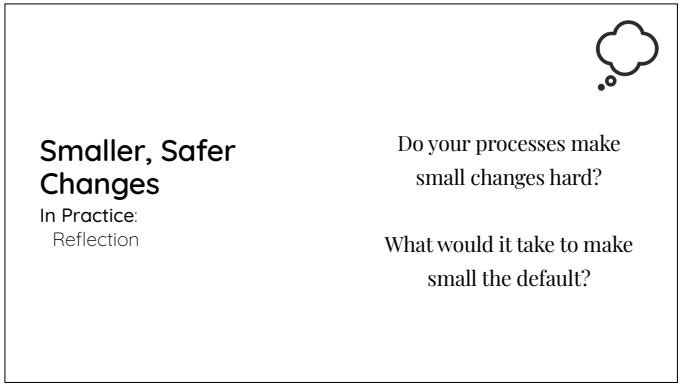
- big-bang releases created high risk and delayed value delivery
- integration pain was siloed with the release engineer
- build, test, and merge regularly to avoid big-bang releases
- pull in changes frequently - share ownership - reduce risk

Test What You Plan to Ship: 💬 we…

- previously…
- tested features in isolation, merged late and bundled
- test merged code - not isolated branches

Stay ready to release: 💬 we…

- releases were unpredictable due to bundling and hidden conflicts
- maintain a releasable main branch at all times
- enforce quality gates and use fast pipelines for stability and readiness



💬 *let's pause again…*

💬 *and take a moment to think about **this***

—

💬 *Do **your** processes make small changes hard?*

💬 *What would it take to make **small** the **default**?*

—

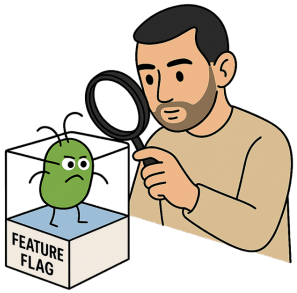
(monologue)

💬 *now we could ship often, the next step?*

💬 *ship without fear*

Controlled Delivery

Find bugs before your customers do



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

💬 *because no matter how good your tests are*

production → final test

real → users → data → load
real → edge cases → test

💬 *and in production*

bugs → not always **stand out** → hide **below surface**

💬 *we were about to launch a new service*

send → sensitive → health
lab results → **heart** measurements → **appointment** reminders
mistakes → embarrassing → regulatory risk → real-world harm

💬 *so we took a three-pronged approach*

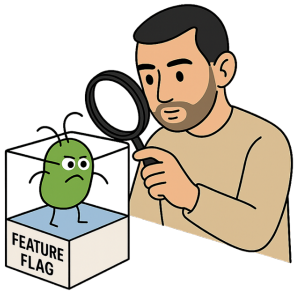
monitoring → tells you something's **wrong**
observability → tells you **why**
feature flags → control **blast** radius

💬 *so we didn't just build alarms…*

💬 *we built safety nets*

Controlled Delivery

Find bugs before your customers do



💬 *here's how we turned production…*

💬 *into a low-risk learning space*

💬 *first, we dark-launched the service*

feature flags → old, parallel, new
same input/output → side by side
hidden users

💬 *next, we added monitoring and observability*

tracking: delivery rates → delays → discrepancies → old vs new

💬 *finally, we started with a safe group of customers*

staff → use product
gradual → confidence grow

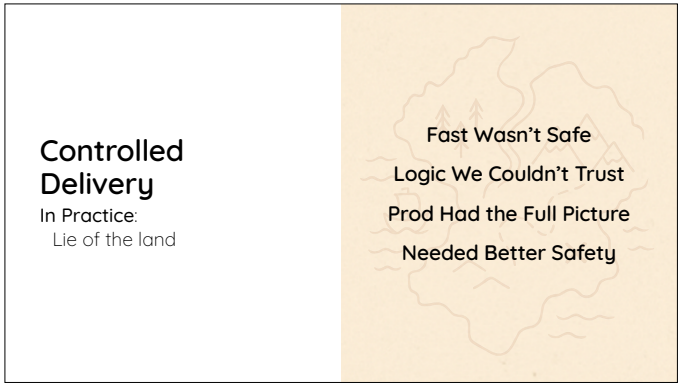
💬 *and the result?*

delivered: safely → confidently

💬 *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*



💬 fast feedback helped — but this was HealthTech…

💬 shipping still felt risky — so we asked…

💬 how do we make production a safe place to learn?

Fast Wasn't Safe
- small, well-tested changes still carried hidden risk
- comms service had real consequences if it broke
- even "it passed testing" didn't nessasarily mean "it worked"

Logic We Couldn't Trust
- old system had high complexity
- edge cases, undocumented and misunderstood
- no way to know if we missed something

Prod Had the Full Picture
- real data, real schedules, real user behaviour
- hard to recreate outside production
- side-by-side comparison became the safest test

Needed Better Safety
- couldn't bet the farm on a big-bang switch
- controlled delivery let us observe before release
- confidence came from production parity, not hope

💬 we started by shipping to production

💬 but hidden from users

Map the Event Flow: 💬 we…

- used event storming to model system behaviour
- defined command → event chains up front
- designed for traceability before writing logic

Track Requests End to End: 💬 we…

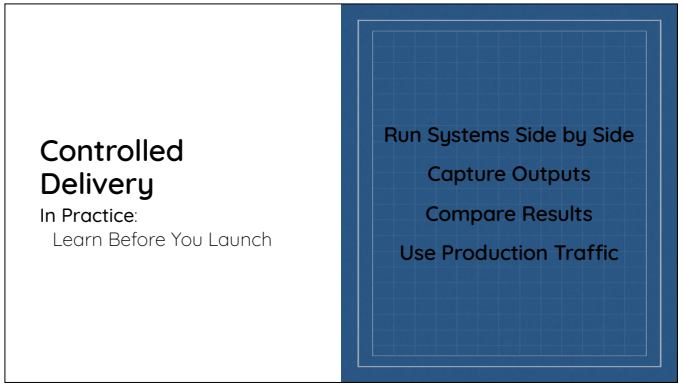
- each workflow started with a unique ID - api, automated process
- ID flowed through all emitted events
- enabled reliable end-to-end traceability

Log Failures as Events: 💬 we…

- exceptions triggered structured events
- failures weren't silent — they were observable
- made it possible to track unknown failures in prod

Metrics From Day One: 💬 we…

- measured message volumes and types (old vs new)
- used metrics to validate system behaviour pre-launch



💬 so we'd built a safety net..

💬 now we needed answers

Run Systems Side by Side: 💬 we…

- identified two key trigger points in code
- added new events to also trigger the new service
- old system kept running as-is

Capture Outputs: 💬 we…

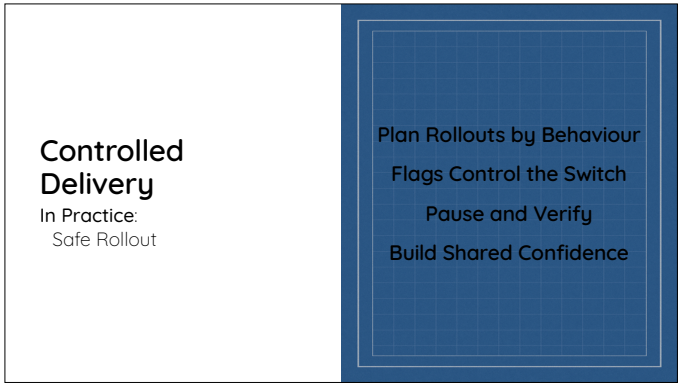
- new service ran full logic but didn't send messages
- sending disabled via feature flags
- let us observe behaviour safely

Compare Results: 💬 we…

- monitored message counts and types across systems
- investigated mismatches before going live
- parity gave us confidence to proceed

Use Production traffic: 💬 we…

- inputs were live — not synthetic
- surfaced edge cases we'd never have predicted
- made production safe to learn from



💬 when we were confident…

💬 we didn't press launch…

💬 we eased into it

Plan Rollouts by Behaviour: 💬 we…

- for comms, we phased by user group and message type
- other services used % rollouts when it made sense
- strategy matched the risk and system shape

Flags Control the Switch: 💬 we…

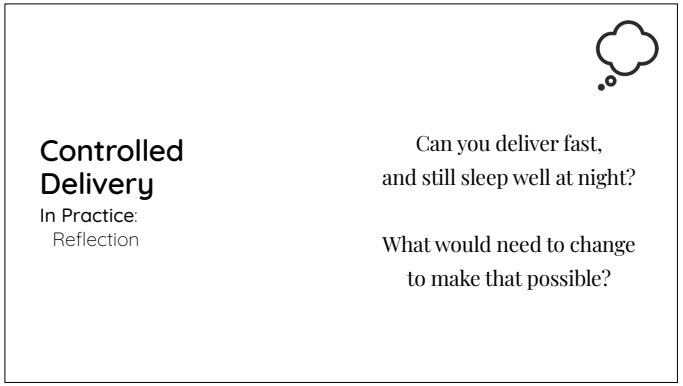
- one flag switched off the old and on the new
- allowed clean, reversible cutovers (sometimes)
- reduce risk of split behaviour

Pause and Verify: 💬 we…

- we watched the data between rollout phases
- used metrics and discrepancies to decide
- no automatic "go" — always a checkpoint

Build Shared Confidence: 💬 we…

- engineers and the business reviewed rollout results
- agreement was needed before progressing
- safety was a team responsibility



💬 lets pause one last time…

💬 and think about these last two questions

—

💬 Can **you** deliver **fast**, and still **sleep** well at **night**?

💬 What would need to **change** to **make** that **possible**?

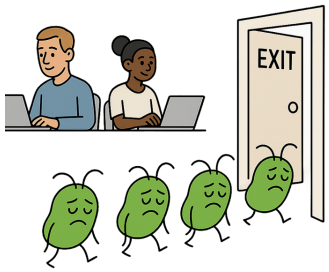
—

(monologue)

💬 here’s what changed for us

Go Beyond
Just Try Harder

Build systems that help
you succeed



💬 *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*

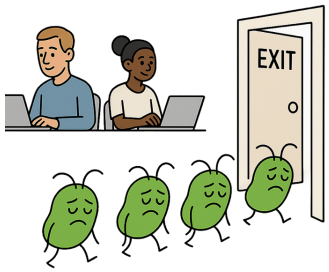
Build **small**. Ship **often**.
Avoid bugs. **Fix** fast.

💬 *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 help
you succeed



💬 *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***

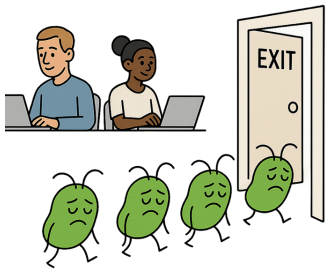
💬 *it's about **shaping systems** that **help** you **succeed***

💬 *so let's build better software*

💬 *let's go beyond "Just Try Harder"*

Go Beyond
Just Try Harder

Questions?



💬 *we've covered a lot today…*

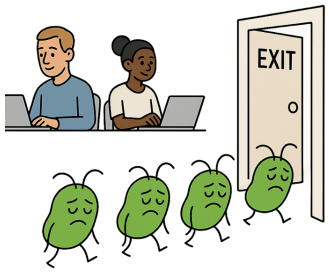
- Fast Feedback
- Smaller, Safer Changes
- Controlled Delivery

💬 *now I'd love to hear from you*

(QUESTIONS)

Go Beyond
Just Try Harder

Wrap Up



💬 *just to reiterate as we wrap up…*

💬 *this isn't a **recipe**, or a **framework**, or a **model***

💬 your **context**…

💬 your **people**…

💬 and your **codebase**…

💬 will all be **different**

💬 *someone asked me last night if I had a bonus strategy*

💬 *I don't, but I do want to leave you with this…*

Small steps → Plan → Execute → Reality…
Iterate

💬 *for me, that's Agile in a nutshell…*

💬 *and this is an Agile conference, after all…*

Two QR codes

QR left: event feedback
QR right: ideas, resources, tools behind the talk

💬 *If you're facing similar challenges…*

💬 *speeding things up safely…*

💬 *or untangling legacy…*

💬 *I'd love to swap ideas*

💬 *I'll be around for the rest of the day…*

💬 and all day tomorrow

💬 *or feel free to drop me a message on LinkedIn*

💬 *thank you*