Extreme Continuous Delivery

at Unruly

Alex Wilson - @pr0bablyfine
Benji Weber - @benjiweber

http://unruly.co/

Marketing Technology

12-120 in under 2 years

3-30 tech team

Talk Structure

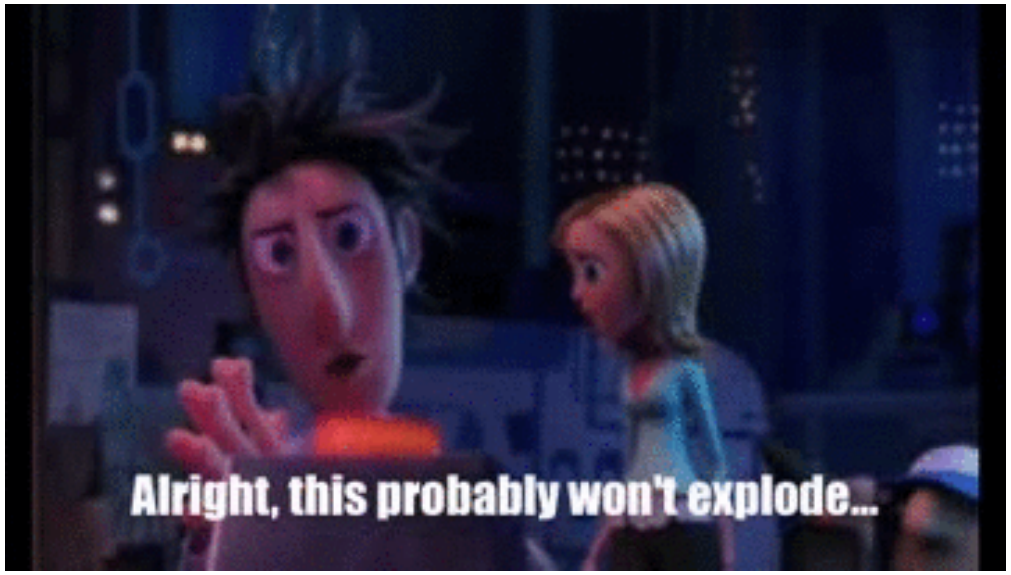
How we work

Why it works

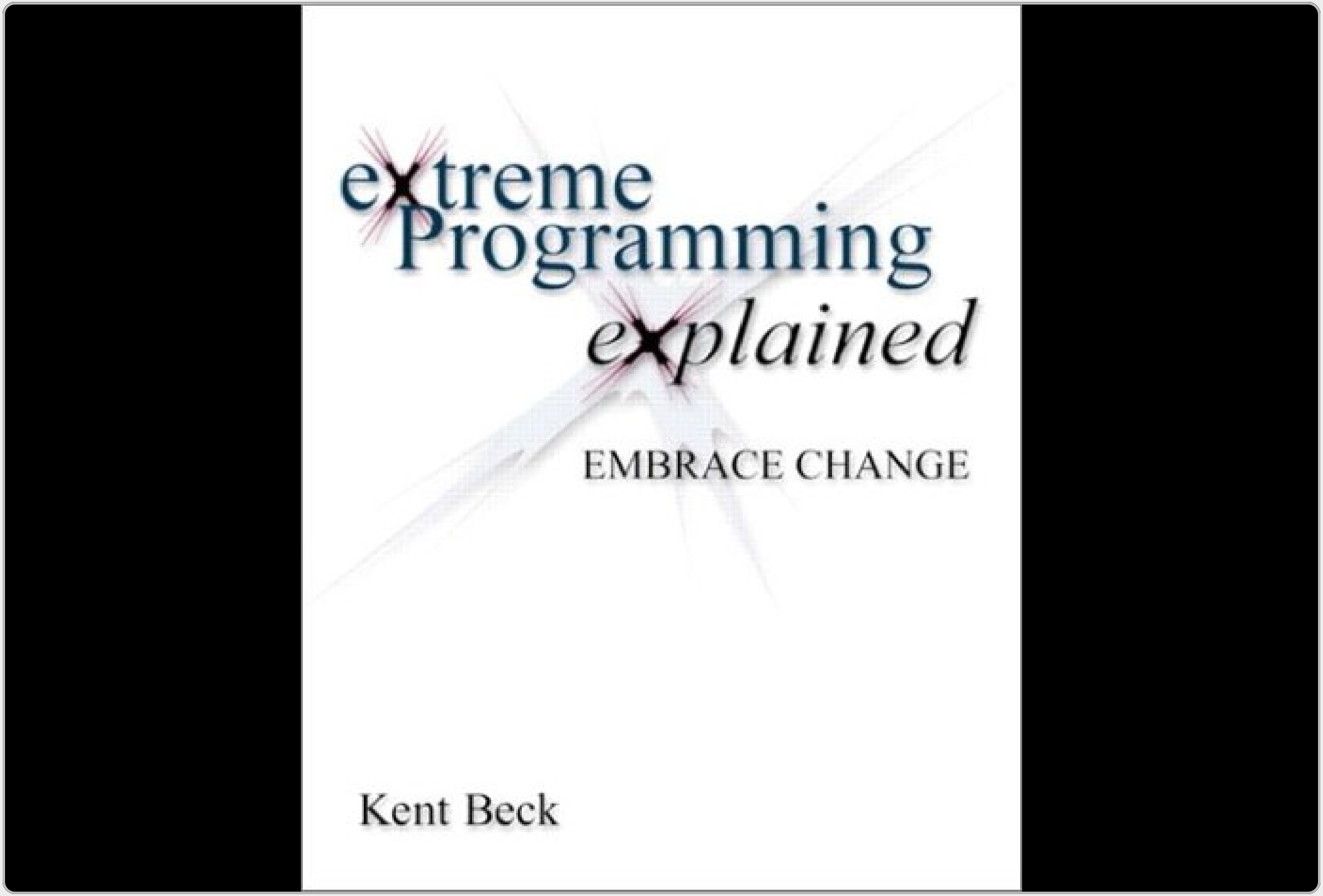
Scaling Infrastructure

Scaling Development

Careful & Considered Approach



How we work...


extreme Programming
explained
EMBRACE CHANGE
Kent Beck

Feedback Loops

Pairing

TDD

Customer

Deploy

Iterations & Release Planning

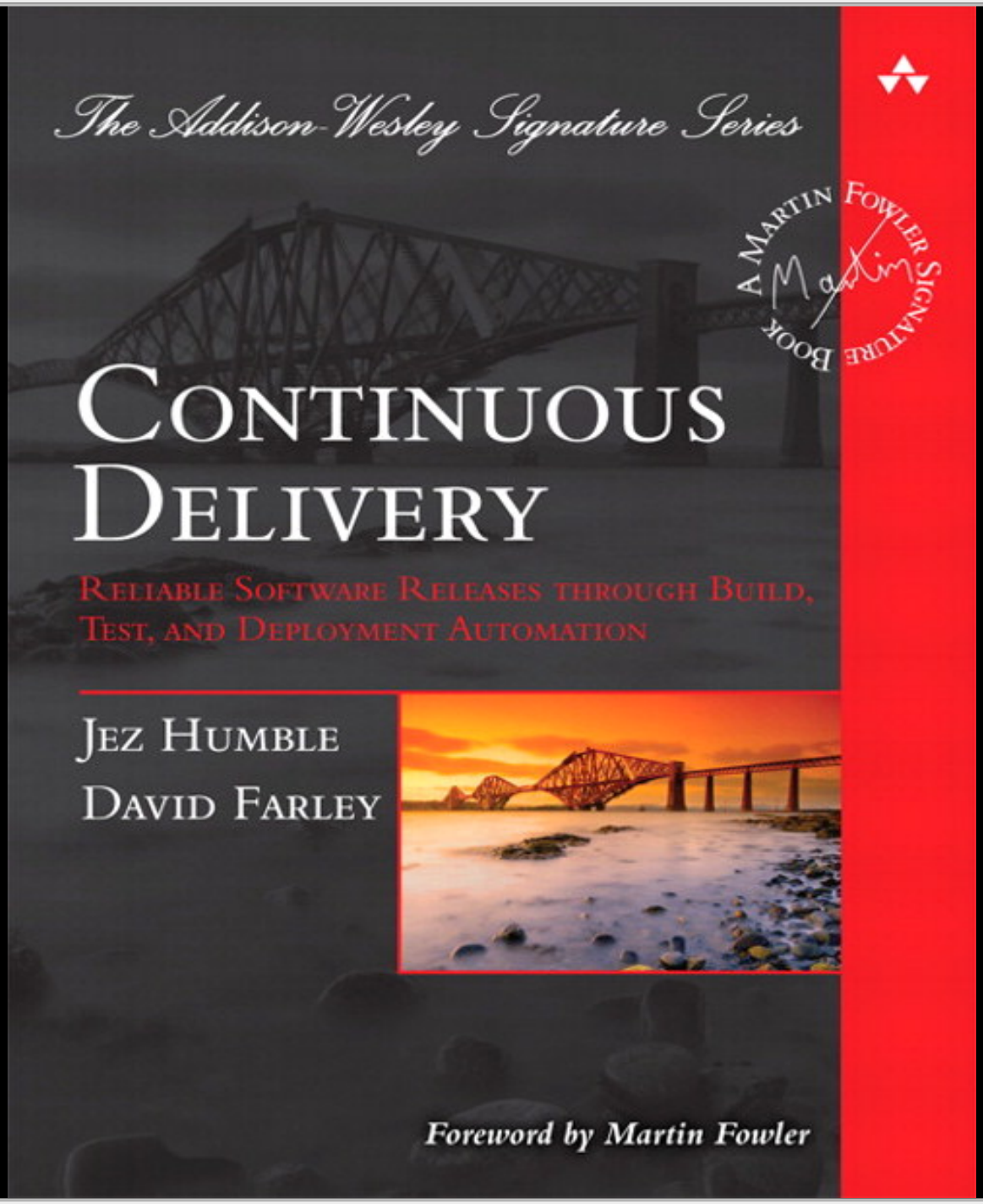
"Plan releases once a quarter. Plan iterations more frequently"
(XP Explained)

We do none of these things

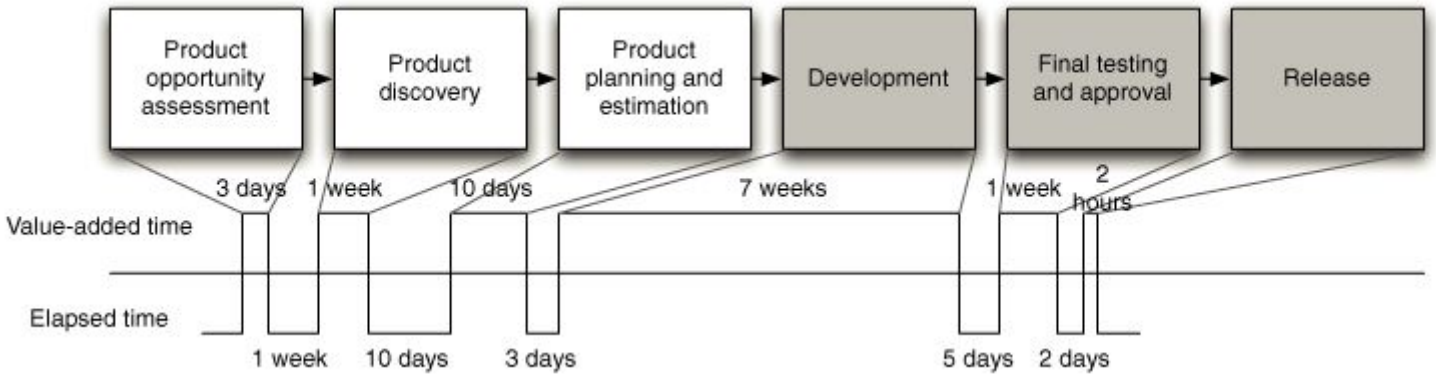
Goal

Deliver value as quickly as possible

Minimise time from conception to value

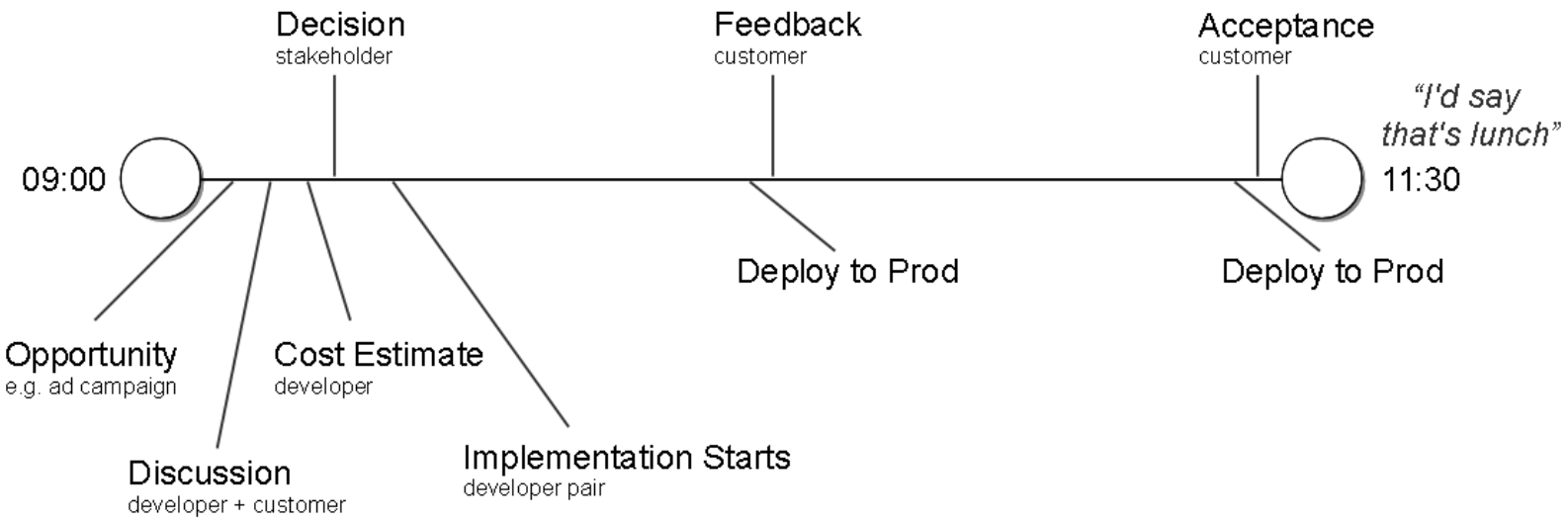

The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Martin
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler

Value Stream

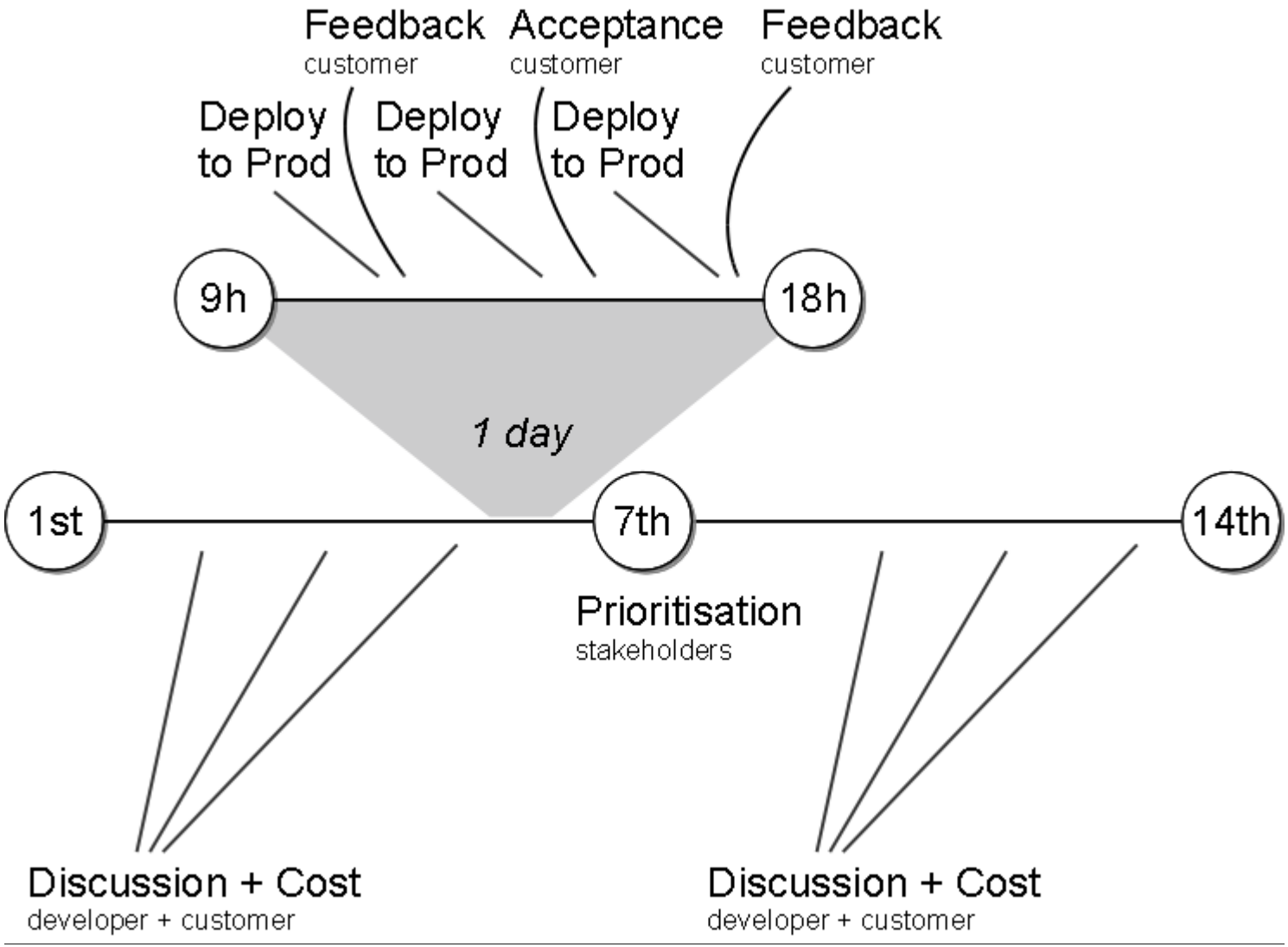


(Continuous Delivery, Jez Humble & David Farley)

Accelerated Value Stream



Normal Value Stream



Definition of Done

~~When tests pass~~

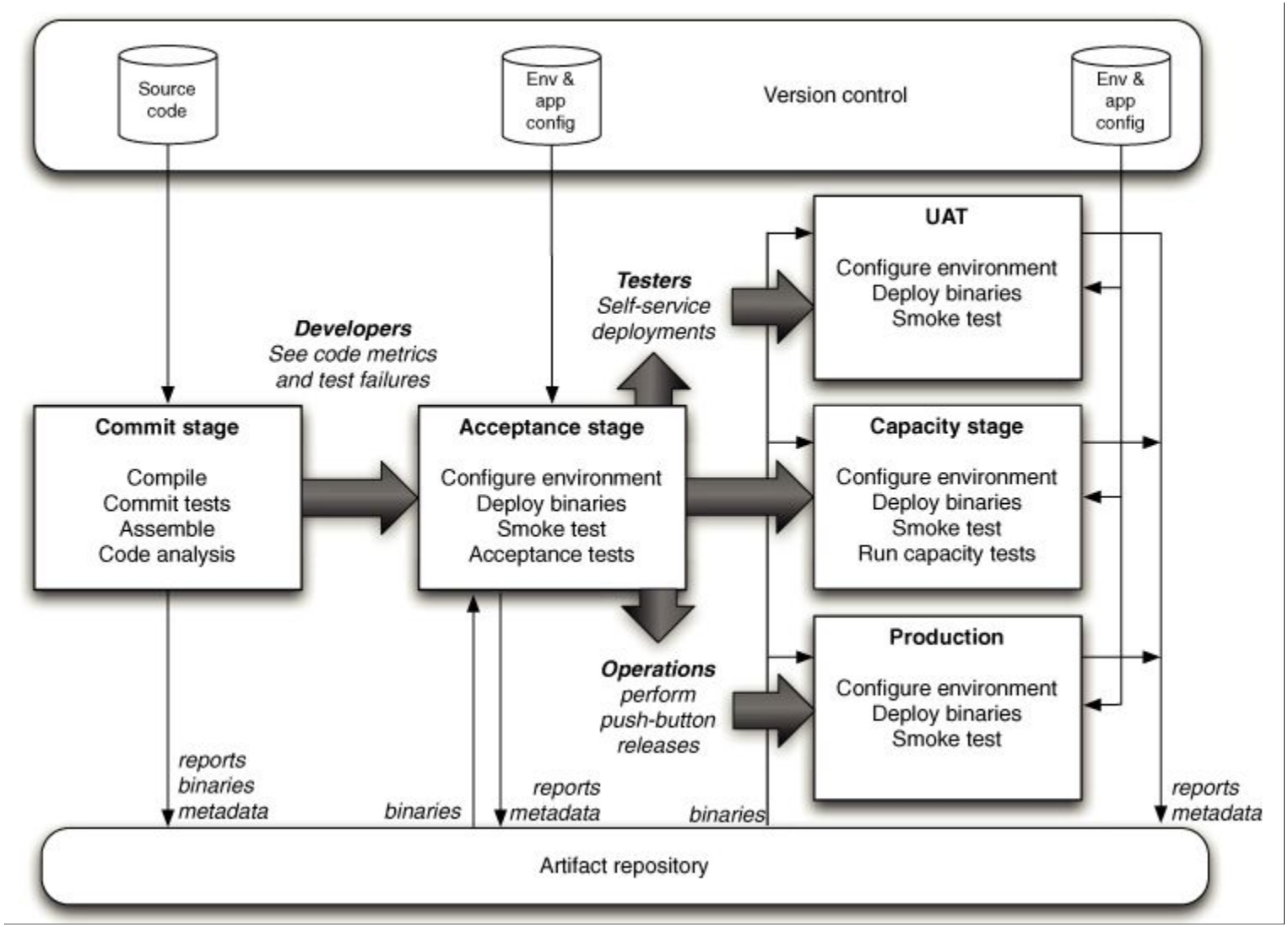
~~When integrated~~

~~When deployable build~~

~~When deployed~~

When measurably delivering value

Deploy Pipeline



(Continuous Delivery, Jez Humble & David Farley)

Waterfall in a Tube

Gate Metaphor

Automate all the Things

Acceptance

Integration

Component/Unit

Performance

...

Move Checks Post-Deploy

Live with broken

Monitoring

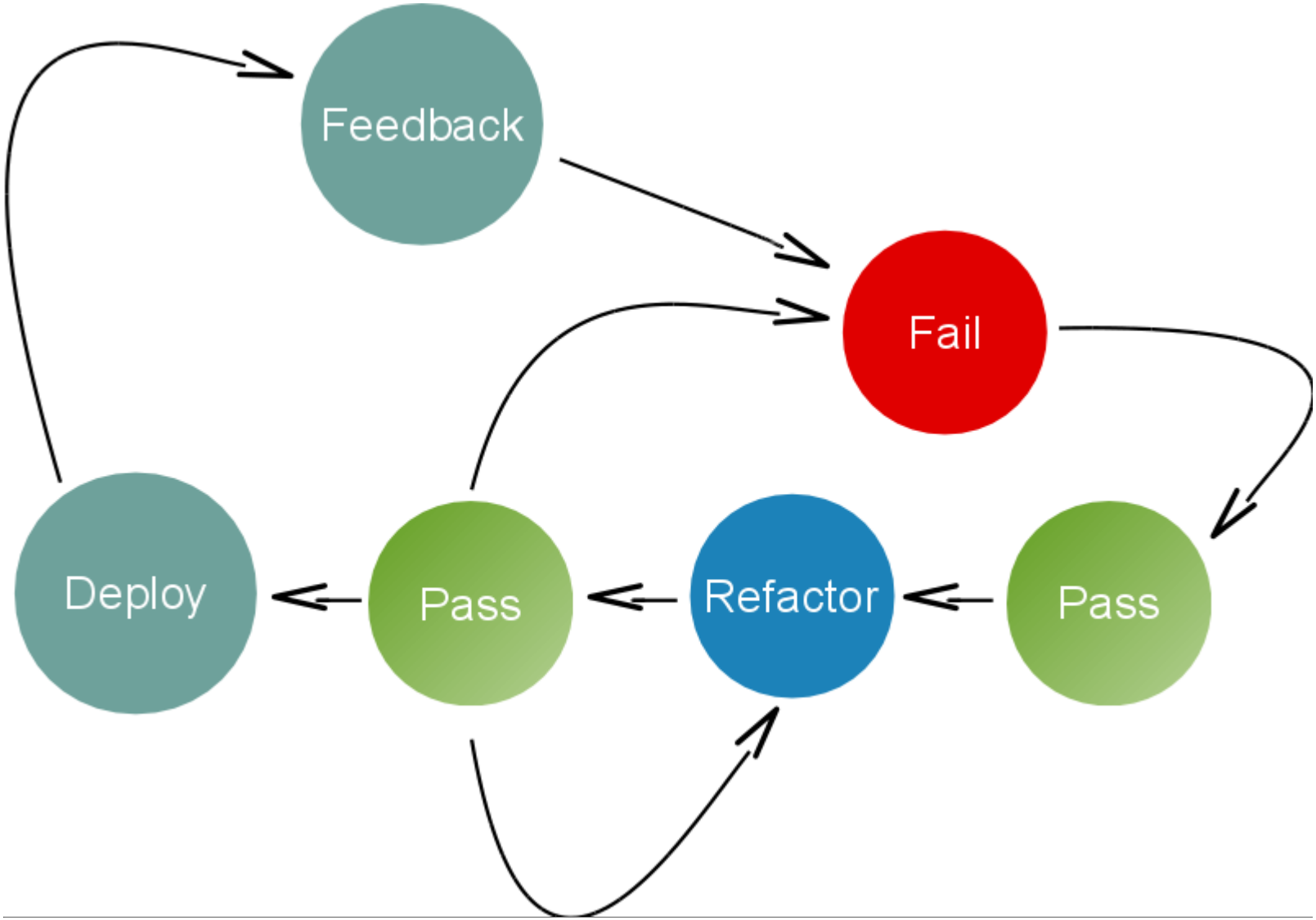
What's Broken & Why?

Fix rapidly

NagDD

Constantly running tests against production

TDD-Deployment



Synchronous

Deploy & Feedback

Inform next change

Motivation to keep deploy fast

Who does CI?

No CI Server

No Branching

No long lived feature branches

No pushed branches

Real CI

Integrate with users and data

Continuous Isolation

Promotion

~~Business~~ Dev decides when to deploy

Feature Toggles

UAT in Production

Global Org

Production-like

Performance feedback early

Canary deploys

Accessible Customer

Essential

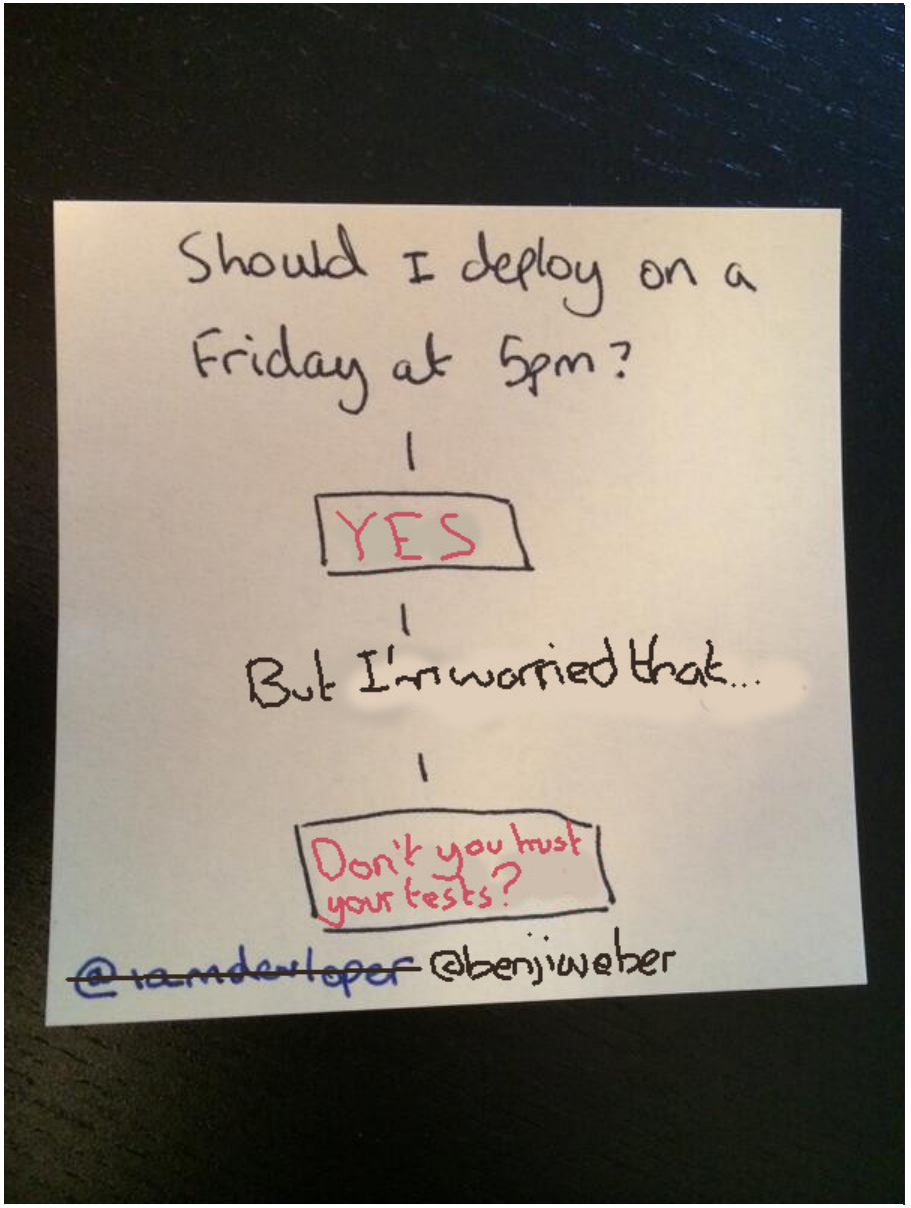
Harder across multiple time-zones

Continuous ~~Delivery~~ Deployment

~~Every build is capable of being deployed~~

Every push is deployed

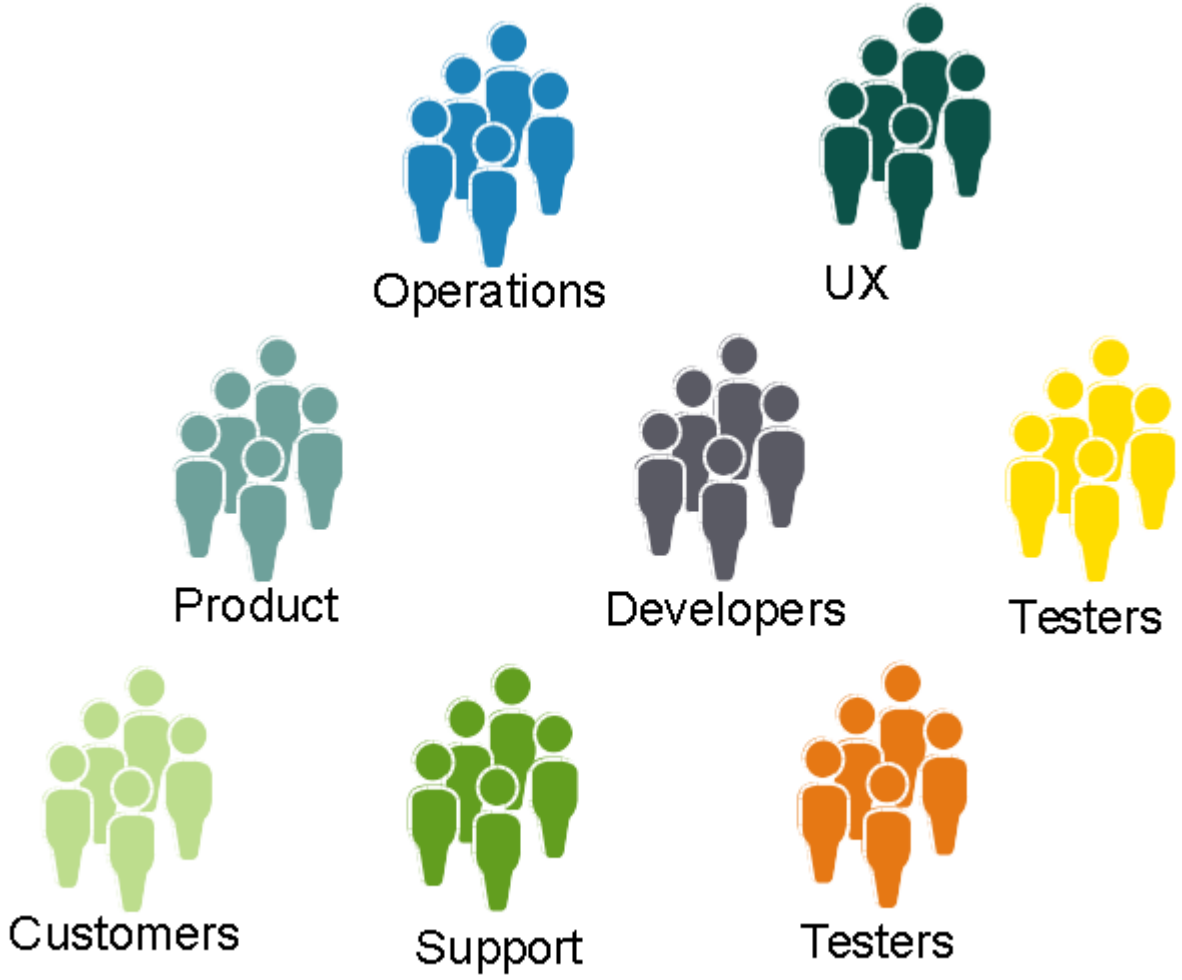
Even after beer o'clock


Should I deploy on a Friday at 5pm?
YES
But I'm worried that...
Don't you trust your tests?
@iamdevloper @benjiweber

Roll ~~Back~~ Forward

People & Culture

Siloing is Bad



Split by ~~Project~~ Product

Generalists over Specialists

Specialists help generalists do better

Product Team

Didn't work - became a bottleneck

Replaced with Product Strands

Collective Ownership

Code

Tests

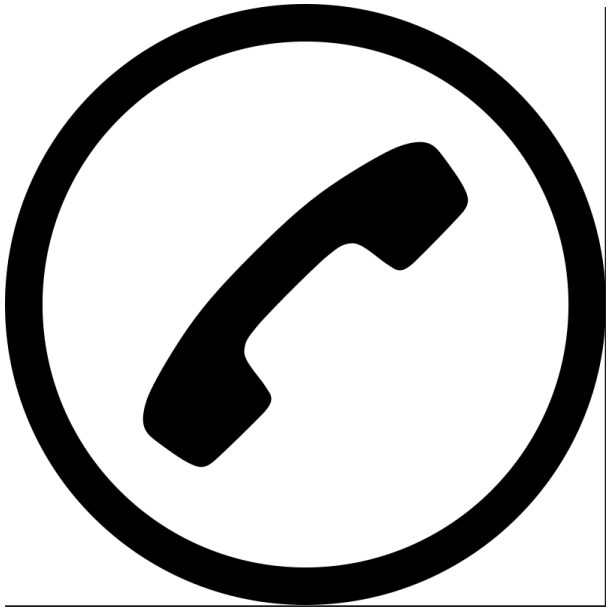
Requirements

Ops

Support

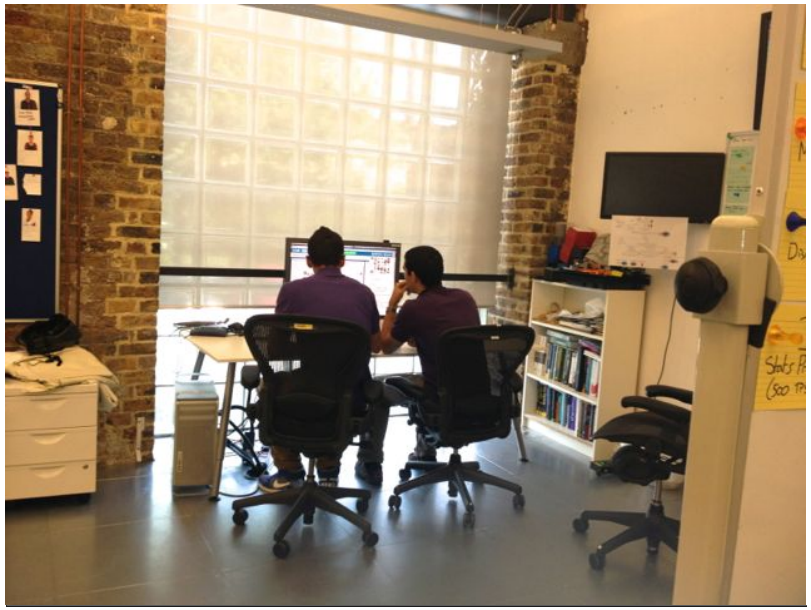
Devs on Call

Freedom and responsibility



Pair Programming

Real-time code review



Self-Improvement

Retrospectives

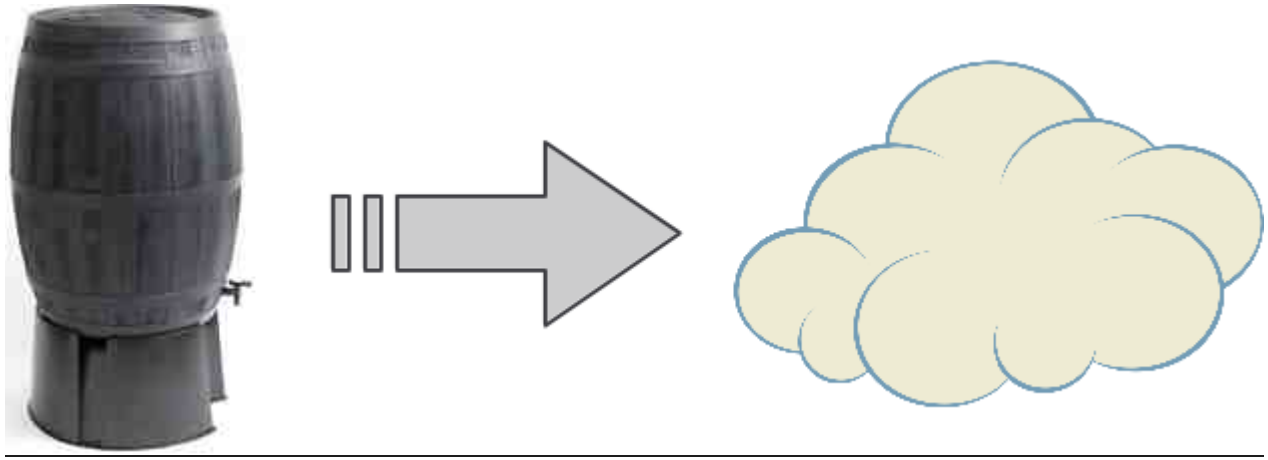
20% Time

Dev Tasks


Questions
so far?

Infrastructure Challenges

1-100 Servers



Infrastructure as Code

TDD

Pairing

Tight feedback loop

Making Snowflakes Disposable

More frequent machine death during growth

Continuous Disposal

Caught out by assuming servers were rebuildable - they weren't

First Day

Any project's first step - deploy "Hello World"

Deploy something, iteratively improve

TDD (sort of)

Unit-testing less useful

Acceptance testing much more informative

Acceptance Testing

```
@test "apache should redirect to https" {

 run curl http://analytics.unrulymedia.com/

[ "$status" -eq 0 ]

 echo "$output" \

| grep -q '< Location: https://analyti…

}
```

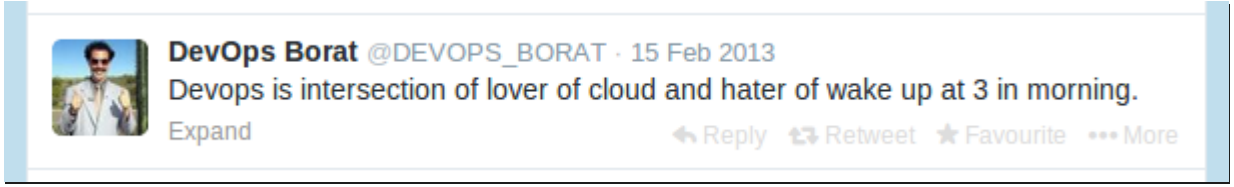
Module Testing

```
@RunWith(ServerSpec.class)

public class AnalyticsWeb {{

describe(service("httpd"), it -> {

        it.should(be.enabled);

        it.should(be.running);

               });


    describe(port(80), it -> {

       it.should(be.listening);

               });


   describe(port(443), it -> {

       it.should(be.listening);

               });

              }}
```

Shared Infrastructure

Assumed care; Ensured suffering

Cross-team collaboration



Reduce Variance, Increase Mean

Homogenous systems are easier to reason about.

MTBF becomes less important than MTTR

Phoenix Workstations

Cronned Code Deletion

Dev Scaling Challenges

Existing Product Boundaries

Conway's Law

"organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations"

Take advantage

Split on Demand

You know least when you start

Monolith vs MicroServices

Deploy speed vs Dependency Hell

Single Repo vs Versioning

Cross-Pollination

Internal tech-talks

Team rotation

Team *lead* rotation

State

Continuous Investment

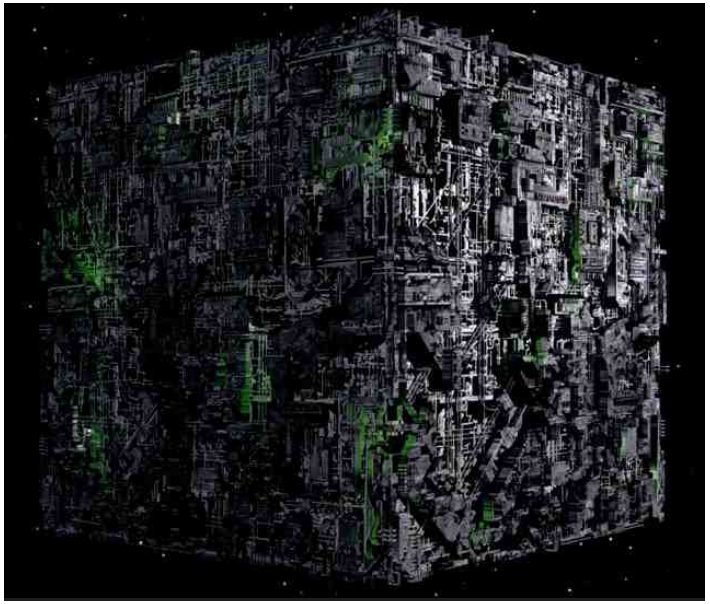
Deploy Speed & Reliability

@QuarantineRule

0.1% failure rate is annoying with 100 tests

0.1% failure rate is impossible with 10,000 tests

Collective Ownership
vs
Freedom



Unusual things

Continuous Deployment
Synchronous Deploy
No CI Server
Cronned Code Deletion

Key Points

Short pipeline
Fast feedback
Early value


Build Quality In
Continuous Delivery
and DevOps
experience reports
edited by Steve Smith and Matthew Skelton

Thanks for Listening

Heckle us on Twitter
@pr0bablyfine @benjiweber

Any Questions?

We're hiring –
talent@unrulymedia.com