# Continuous Delivery for Legacy Code


**Richard Gross** (he/him)


Archaeology + Health Checks


TestDSLs


Hypermedia

speakerdeck.com/richargh

richargh.de/

@arghrich


Lean Agile Scotland

25.09.24

MAIBORNWOLFF

# What are the properties of low-quality software?

> The Quality of a System is Defined by Our Ability to Change it!

Dave Farley

@davefarley77



Dave Farely, author of the best-selling book “Continuous Delivery”

Bad (legacy) software is software where we no longer have any options

# A story of the worst software I have ever and you will never see.

And we still got to Continuous Delivery ☺

# Along comes a customer …

Customer

Please
Health
Check

Me

# ~~5M~~ 4M LoC Delivery


Classic Asp + VBScript

Asp.NET + C#

C#

JavaScript

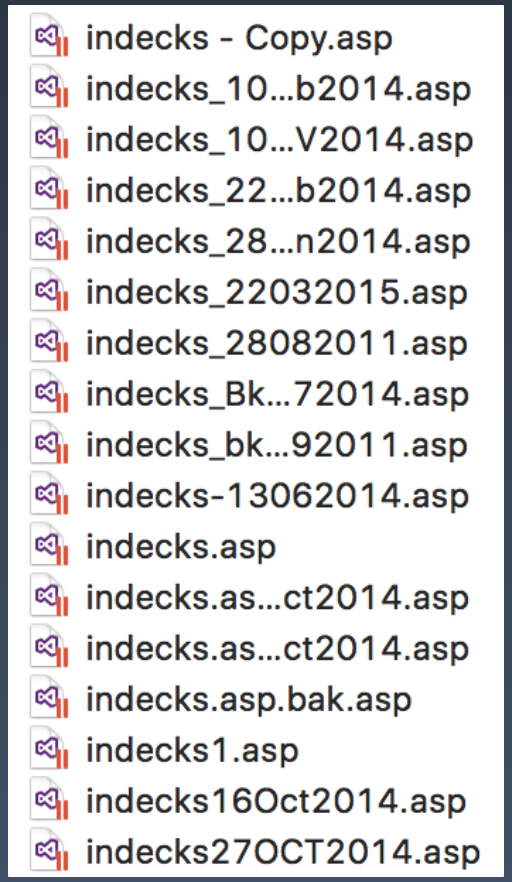
WinForms + C#

VBA

T-SQL

MIA

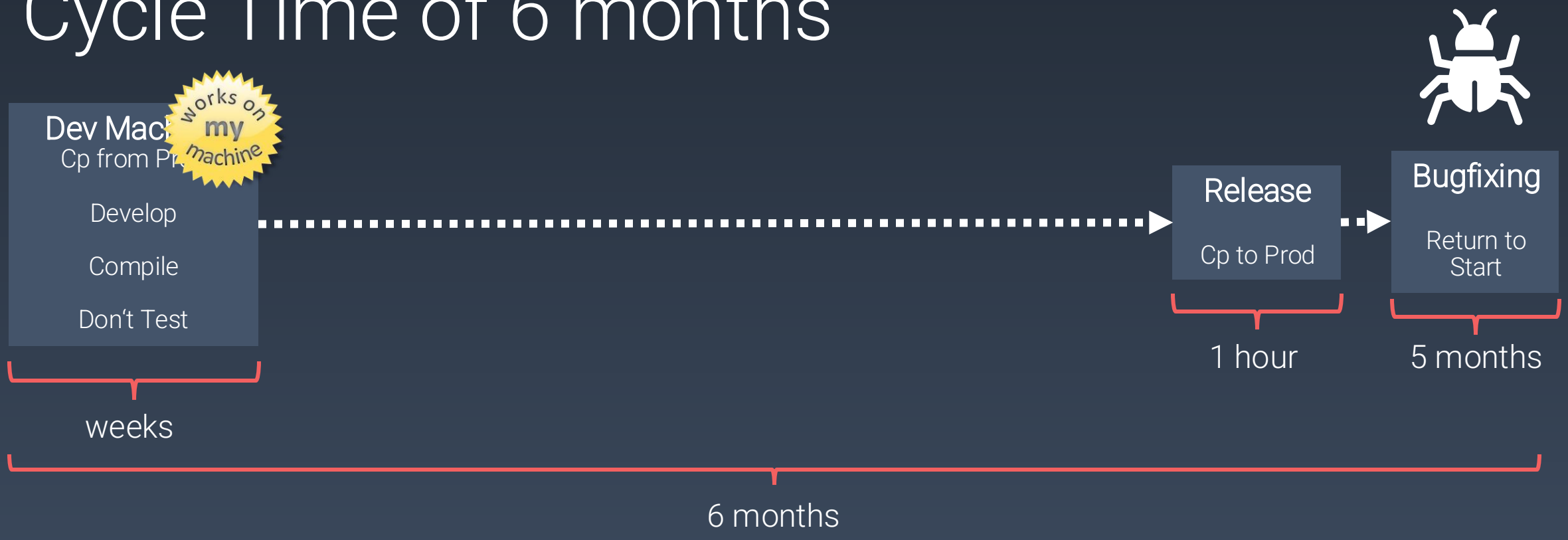
# Copy-Based Version Control™ with production server as origin






File tree from actual project

# Cycle Time of 6 months

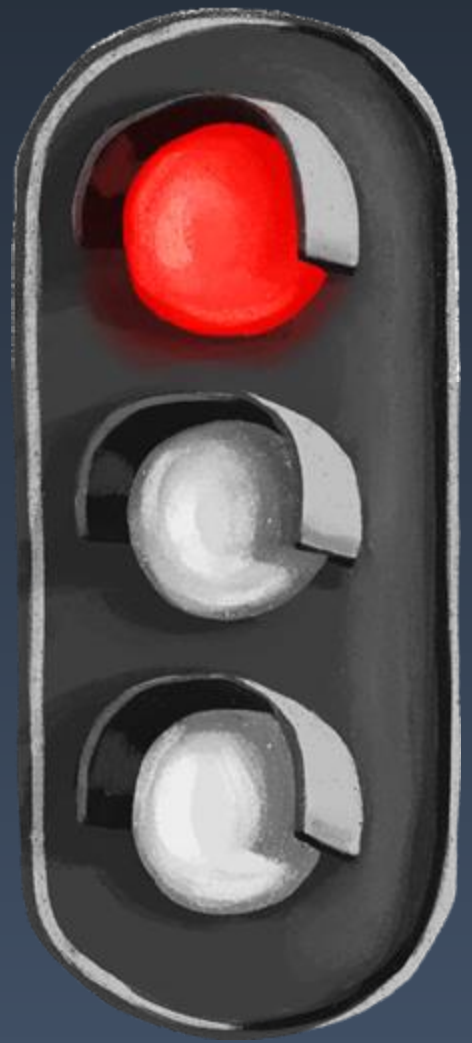


# Too much to fix

15 years of Uncontrolled growth

Not Versioned & Doesn't Compile

„Interesting" Deployment



Conflated Logic, UI & SQL

No Tests

Horrible Cycle Time

Customer

I'll

Buy

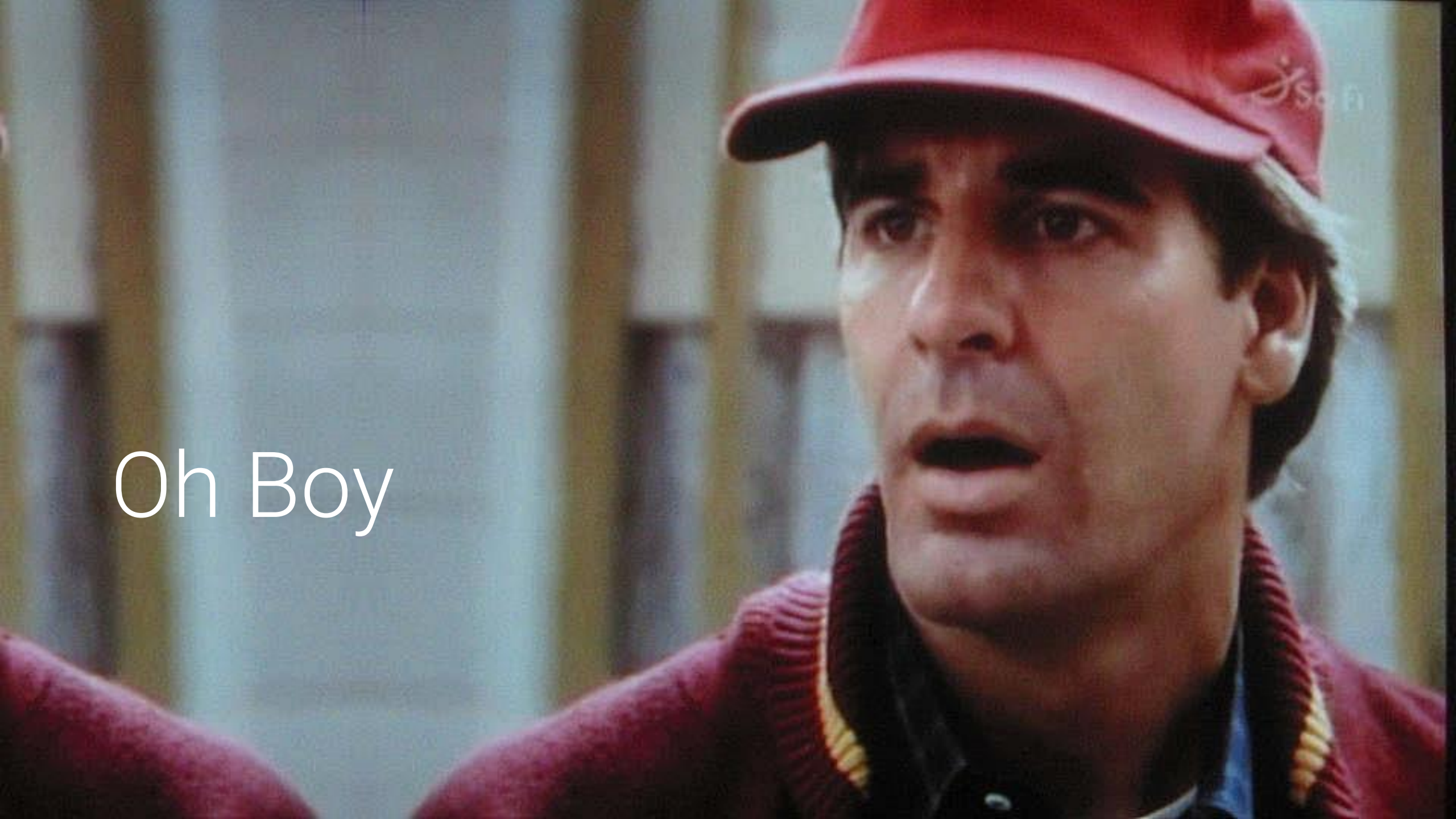
Money

In-house
„Fenix“

Customer

Please
Fix


Me


Oh Boy

> Legacy Code is code that's too scary to update and too profitable to delete.

Dylan Beattie
@dylanbeattie

> It's not CD if you can't deploy to production right now

Ken Mugrage
@kmugrage

# Can we deploy this to production right now?

Nope

15 years of
Uncontrolled growth

# Where would I even start?

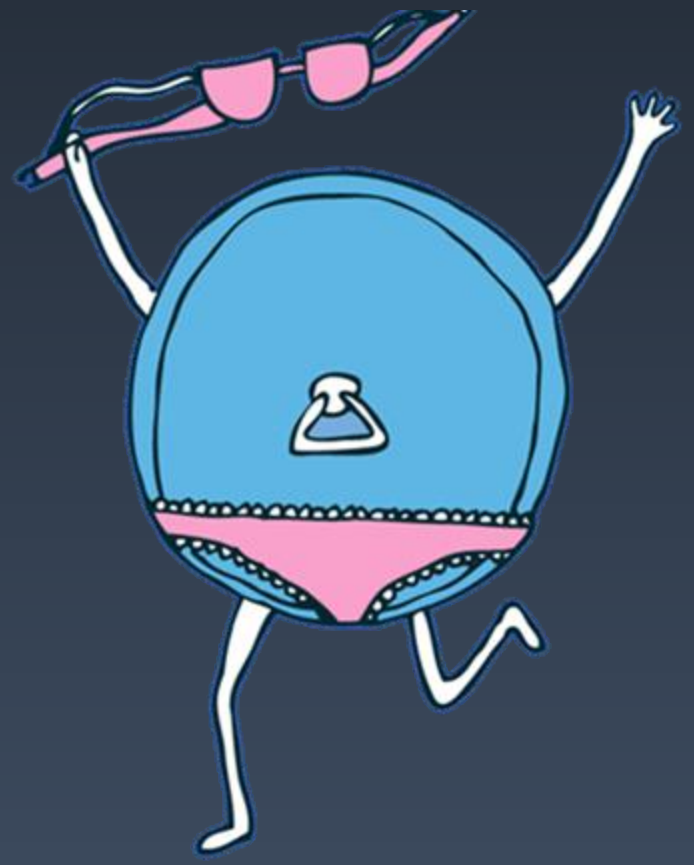
If I’m lucky I have a long list of issues:

- Issue 1 in file x
- Issue 2 in file y
- Issue 3 in file z
- Issue …
- Issue …
- Issue …
- Issue …
- Issue …
- Issue …
- Issue …
- Issue …
- Issue 9001

# Don’t

Fix everything
Everyhwere
All at once

Map your code.
Extract your insights.
Master your Legacy.

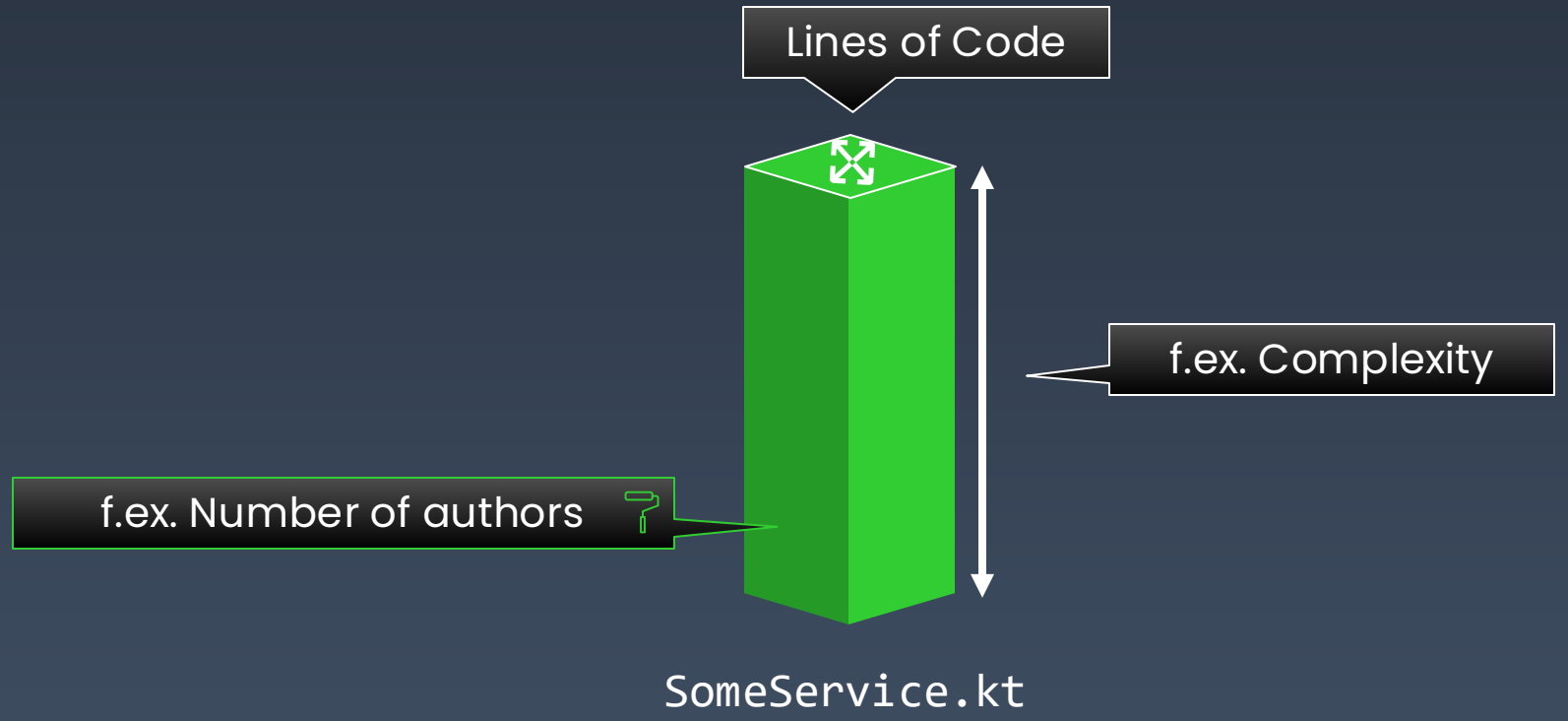


A Shameless Plug.



Art by Zaufishan

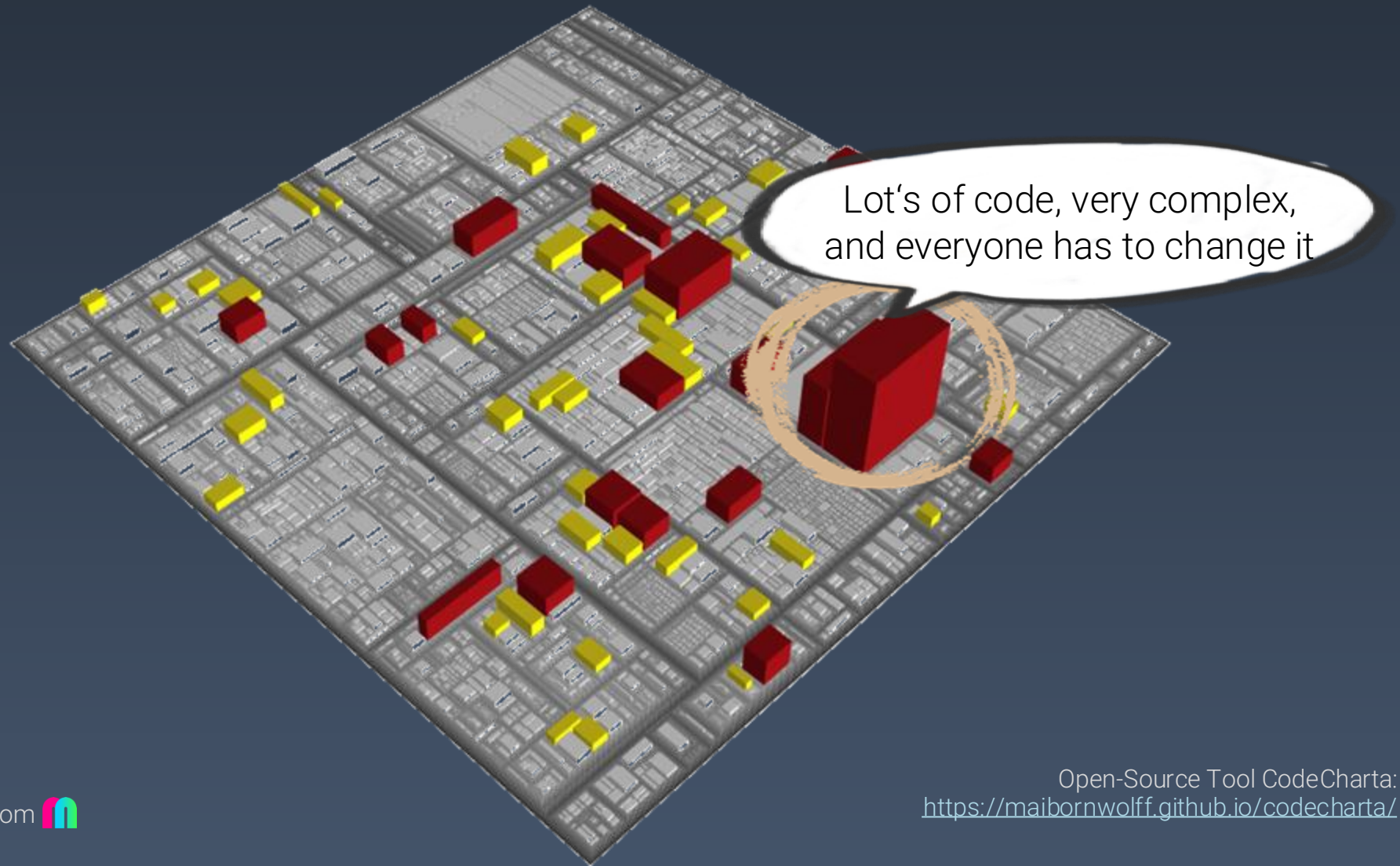
# Visualize metrics with CodeCharta buildings






Open-Source Tool CodeCharta:
https://maibornwolff.github.io/codecharta/

# Find potential hotspots



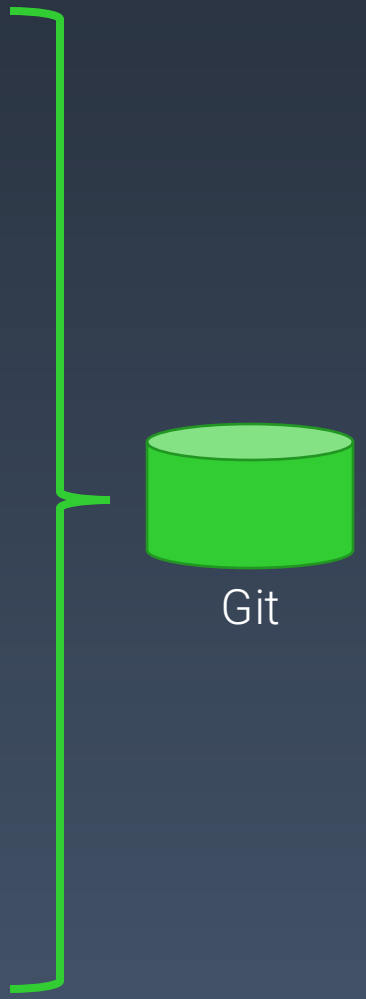

Open-Source Tool CodeCharta:
https://maibornwolff.github.io/codecharta/

We know what to focus on 

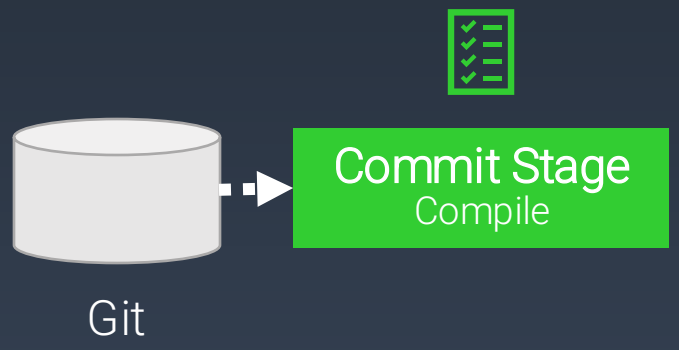
Not Versioned
& Doesn‘t
Compile 

# Put everything into version control

- Code
- Configuration
- Deployment scripts
- Infrastructure Provisioning
- Alerting
- Monitoring
- Documentation
- Onboarding.sh Scripts
- …



# Establish a Pipeline



We know what to focus on 

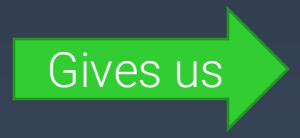
Versioned &
Automated Compilation 

„Interesting“
Deployment 

# Scripts to Provision Infrastructure

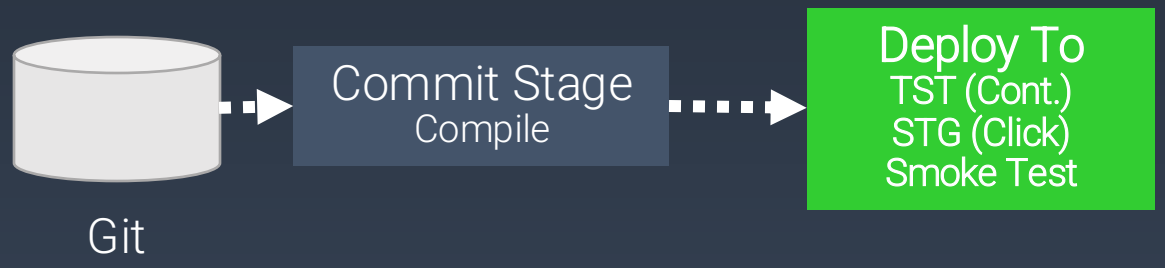
Same but parameterized Scripts for TST, STG and PRD

- Machine
- Web Server
- Application Server
- Database
- Email Server
- FTP Server
- Pipeline Server




Phoenix Infra: With one click we can (re)create any infrastructure

# Deploy via Pipeline to Test Env



# Can we deploy this to production right now?

We have a button to do it, but do we want to?

We know what to focus on 

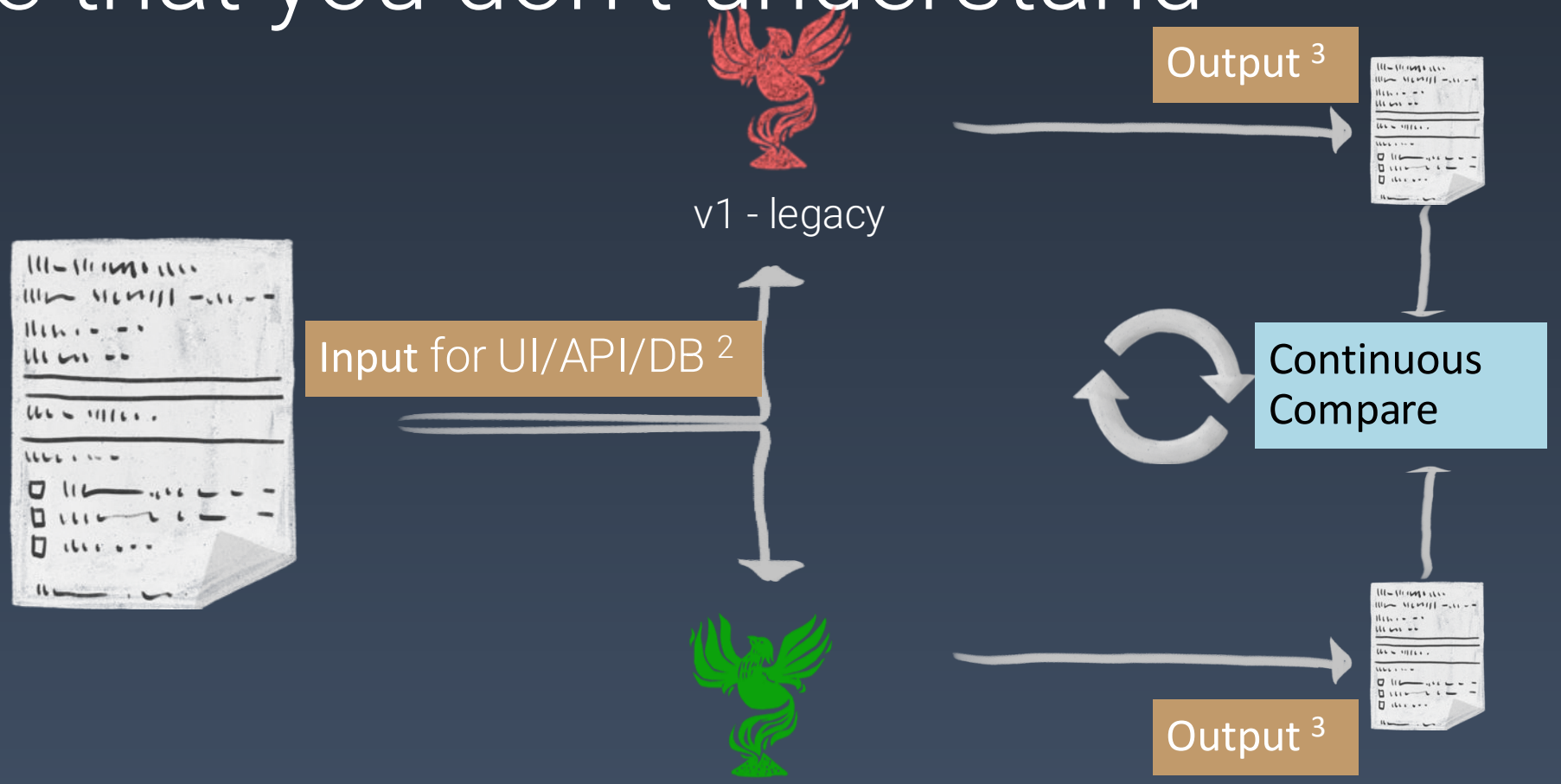
Versioned &
Automated Compilation 

„Interesting“
Deployment 

Conflated
Logic, UI & SQL

No Tests

# Characterization test[1]: Write Tests for code that you don't understand




[1] Also called approval or golden master test.
[2] Put system in a known (database) state is also an input.
[3] Log messages you add are also an output.

# Start with tests for critical use cases and hotspots

- ~10 E2E Tests
  For critical use cases
- DB-Integration Tests
  For direct-db-access hotspots that you will refactor
- Unit tests (should give 80% confidence)
  For refactored hotspots
  For no-direct-db-access hotspots
  For new code

# The tests guide the way

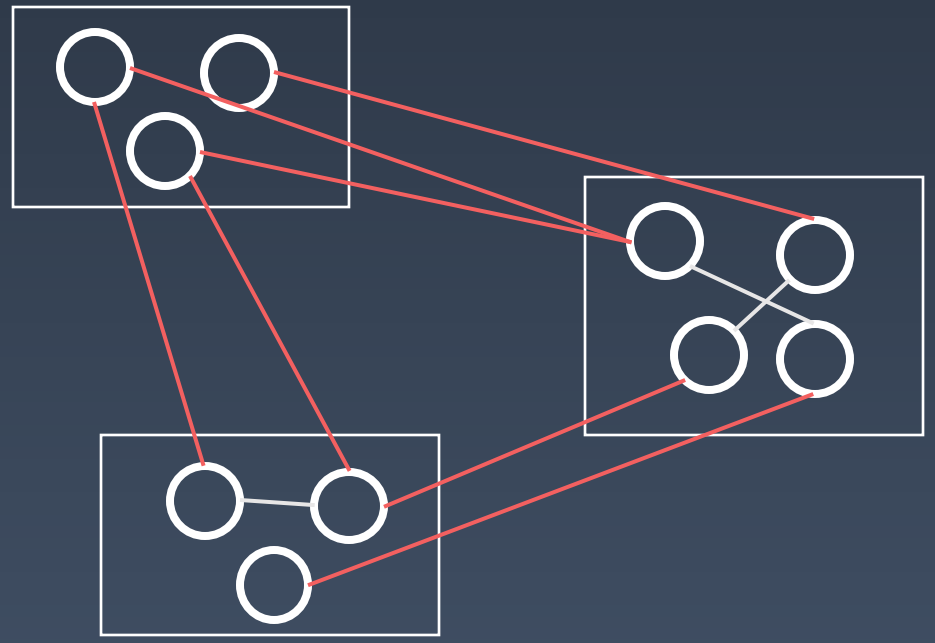
1. Establish a test dsl[1,2] (⚠ the easy part)
2. Try to unit test legacy element and … fail
3. Realize that coupling prevents you from proper testing
4. Write a too-big integration test
5. Fix coupling problems (⚠ the hard part)
6. Use Dsl to switch integration to unit test



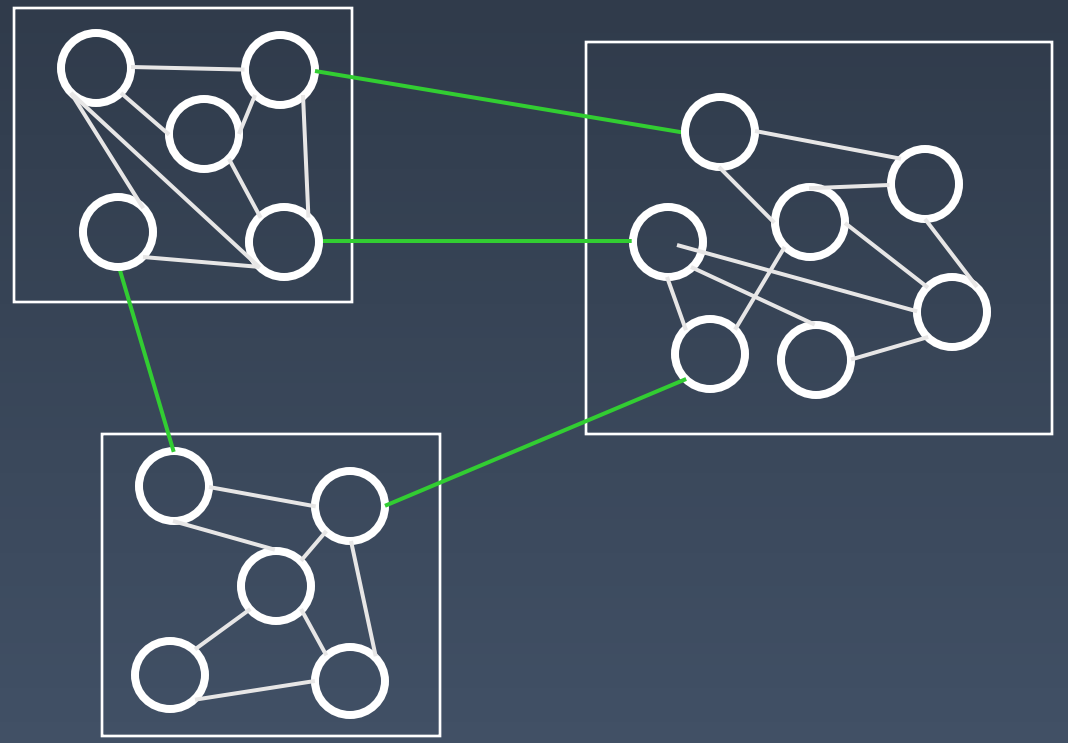
1 https://github.com/Richargh/testdsl
2 Structure-cementing tests and how to avoid them (in German DE)

# Understand Coupling and Cohesion

High Coupling
Low Cohesion



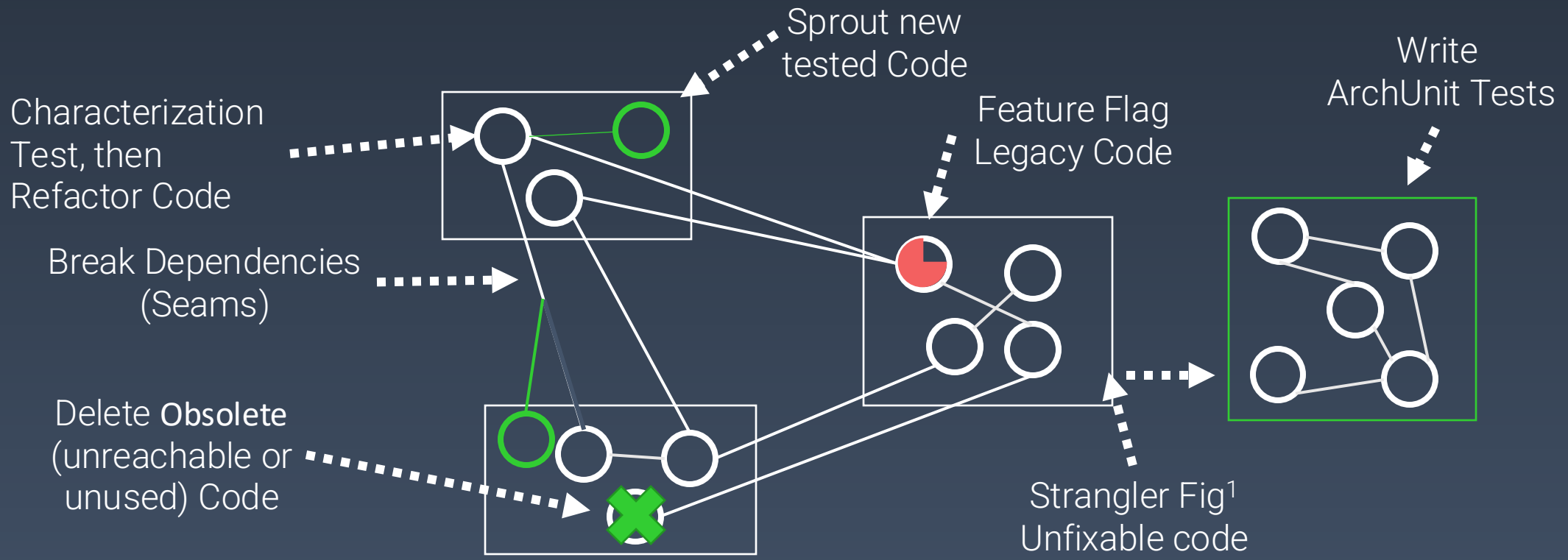
Low Coupling
High Cohesion





Unknown source

# Uncouple your Code




1 https://www.maibornwolff.de/en/know-how/strangler-fig-pattern/

# Can we deploy this to production right now?

Hmm, it’s somewhat tested but can we really be sure nothing breaks in production?

We know what to focus on

Separated Logic, UI & Data
in Hotspots

Versioned &
Automated Compilation

Critical Use Case Tests
Growing Unit Tests
Unit tested new code

„Interesting“
Deployment

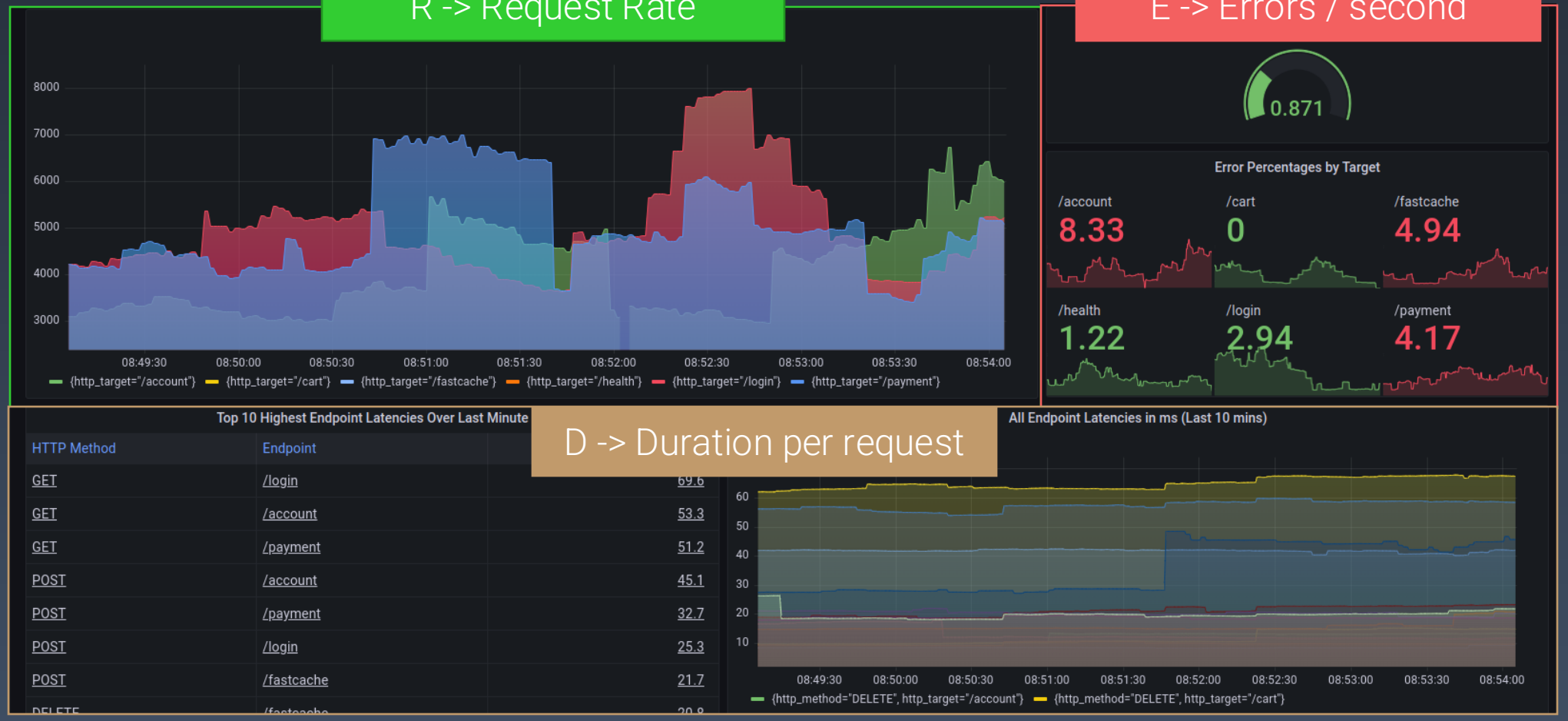
Horrible Cycle Time

# Do Monitor in Production

# RED Metrics

R -> Request Rate

E -> Errors / second



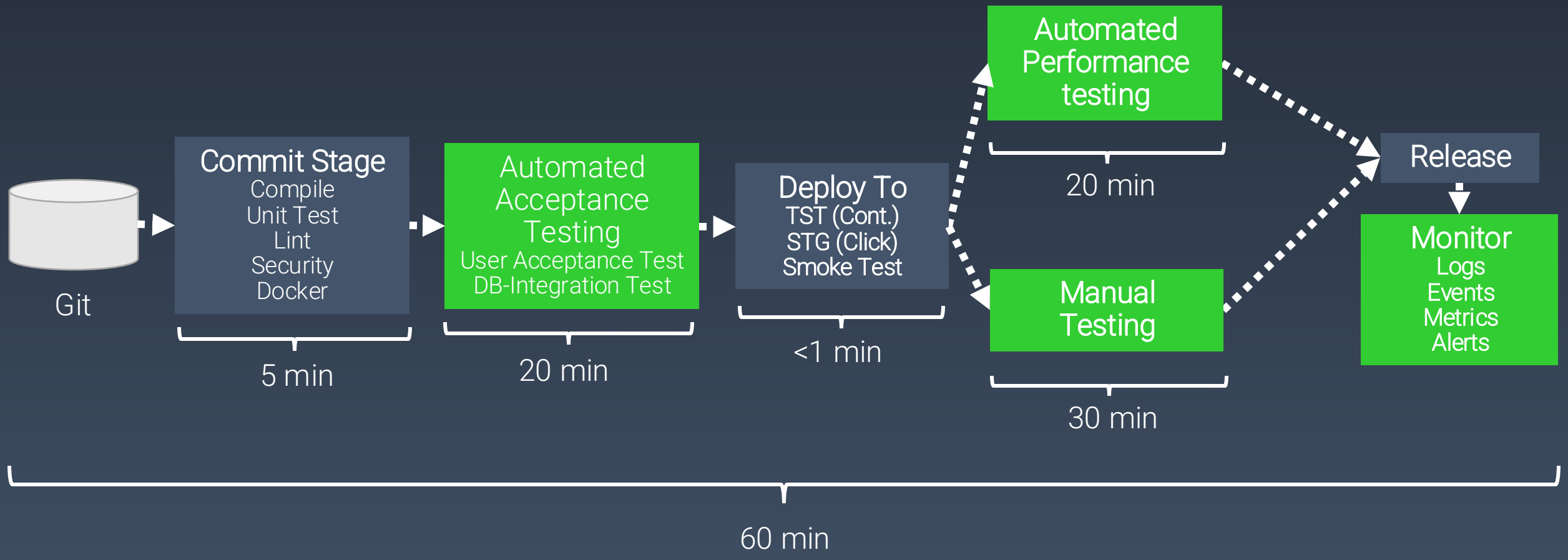

D -> Duration per request

[1] https://thenewstack.io/monitoring-microservices-red-method/

Community Dashboard https://community.grafana.com/t/how-can-i-configure-red-dashboard/69885

Slide 39 by richargh.de from

# New Cycle Time, without bug-cycle






Slide adapted from Dave Farley

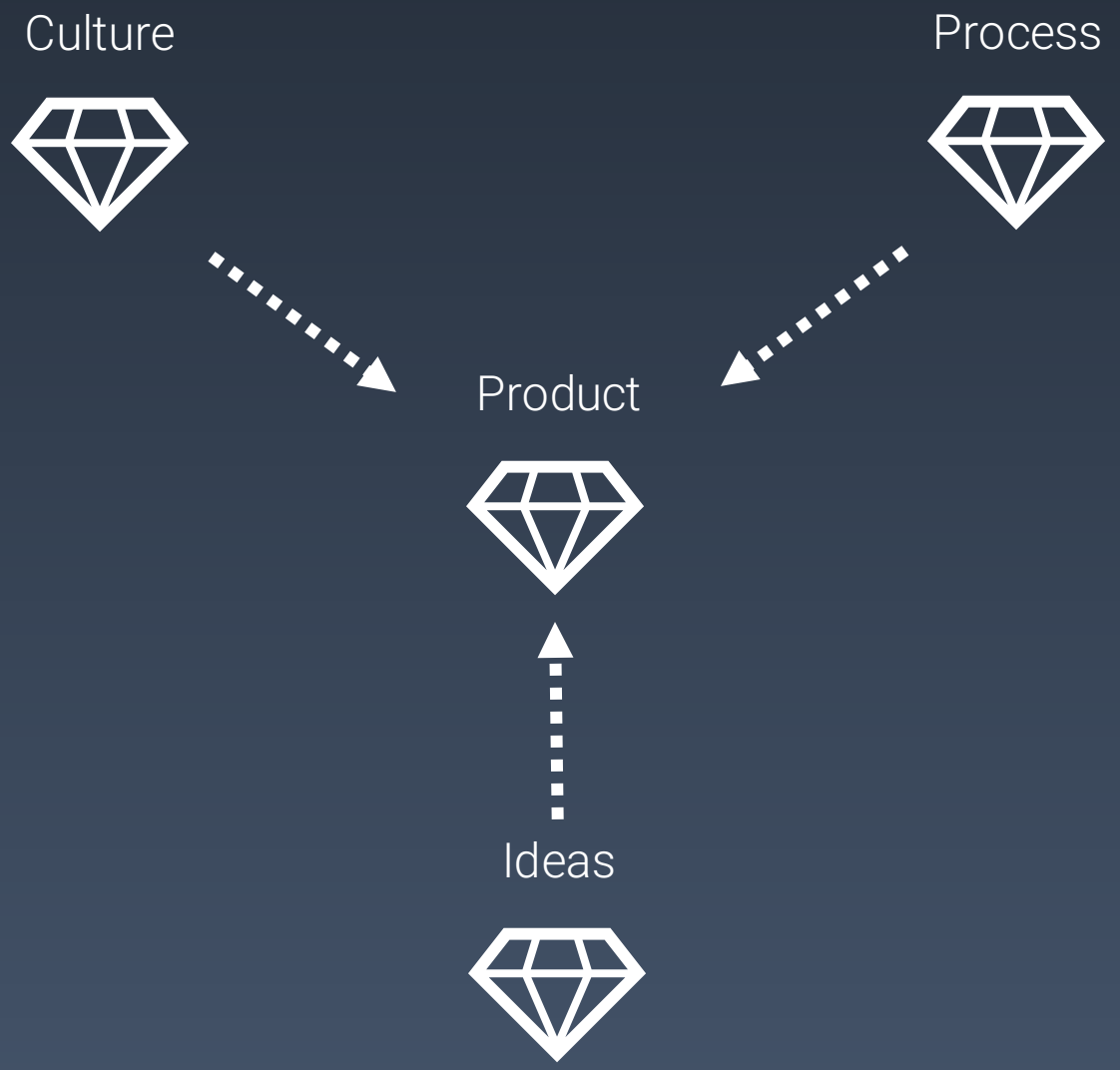
# Can we deploy this to production right now?

Yes. Depending on where we make the change, even within 30min.

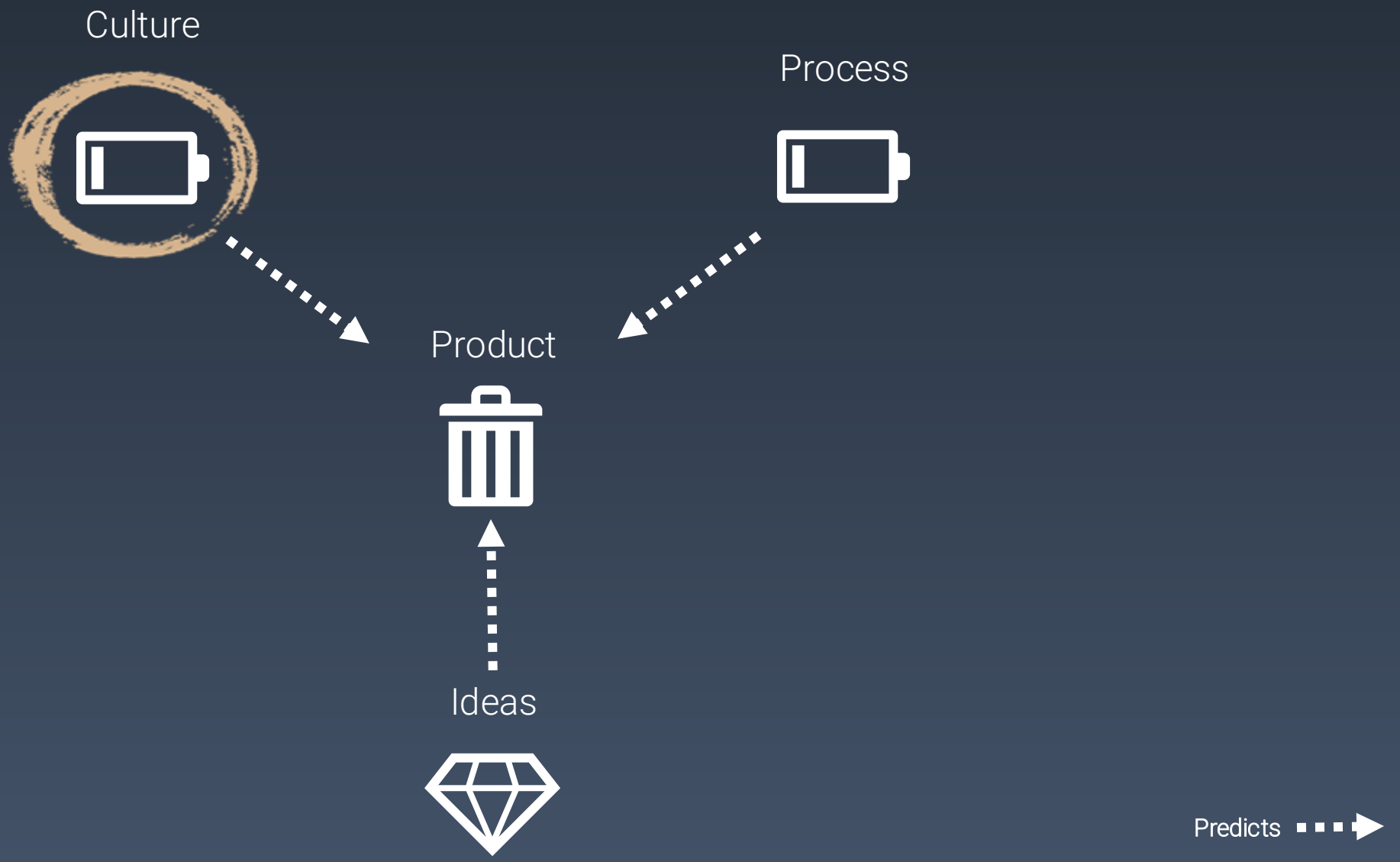
# But there is still a problem…

# How did the previous developers get here?

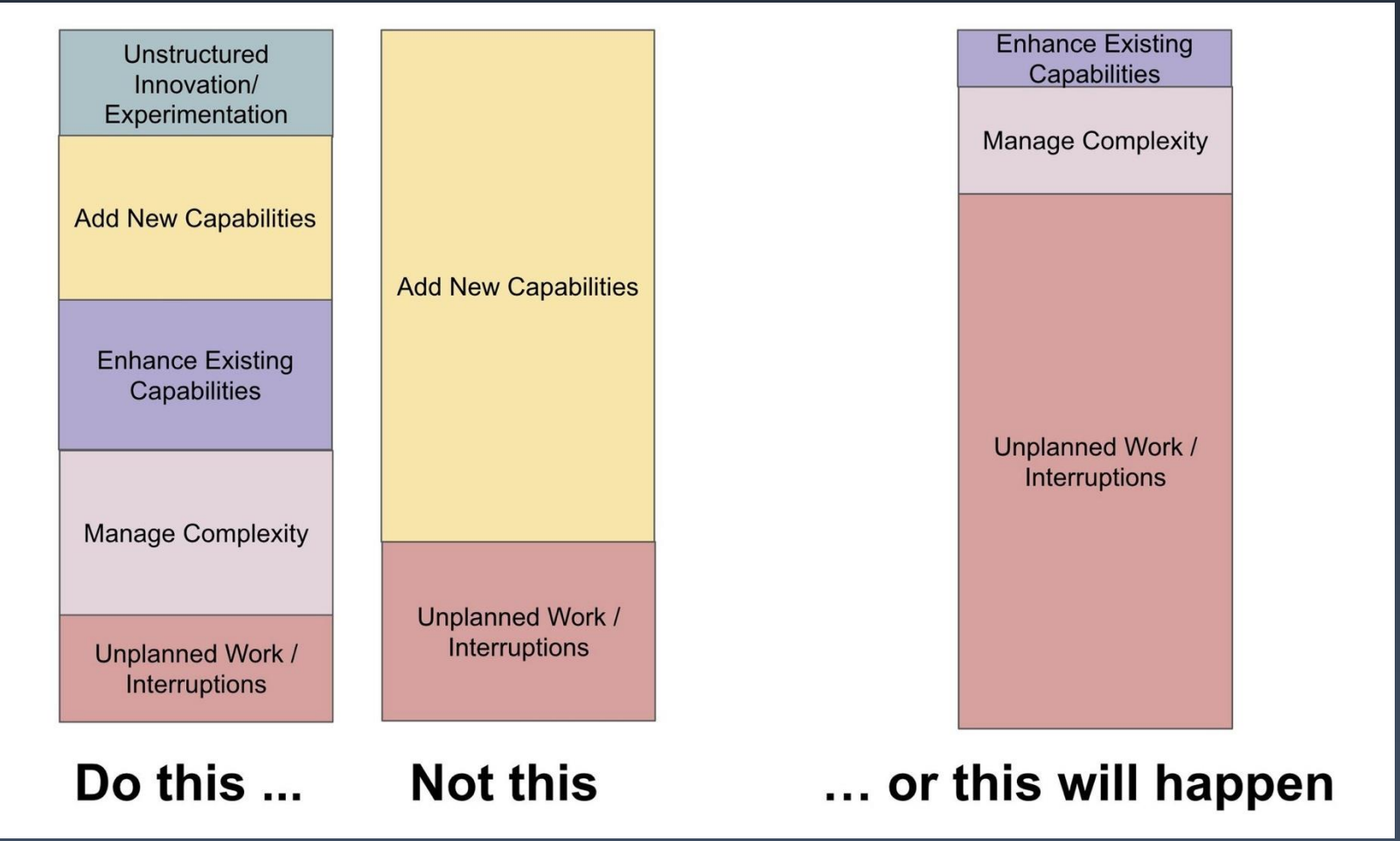
# Healthy Software



# GIGO



# A Feature Factory helps no one




https://cutlefish.substack.com/p/tbm-4952-your-calendar-your-priorities

> Culture is the sum of everyone's attitude and actions, so model the behaviour you want to see!

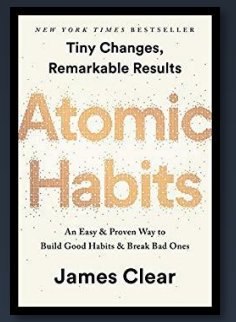
Henrik Kniberg

@henrikkniberg

Observe behavior:

- Who does the management promote?
- Is there fear of failure?
- Is quality appreciated or story points or …?
- What is the delivery „process“?
- Is there some kind of learning exchange?

# Make the desired behavior …




Atomic Habits

4 Laws of behavior change

## Obvious

- Define Principles together
- Shared purpose

## Attractive

- Reduce (time) pressure
- (Refactoring) score

## Easy

- Tech-coach
- Reduce friction

## Satisfying

- Pair up
- Never break the chain (twice)

# CD is a change in mindset

# CD = Working in a way that software is always in a releasable state

It’s not about the tools, even though they are getting better all the time

We know what to focus on

In case you need some help here...

Separated logic, UI & data in hotspots

Versioned & Automated Compilation

Critical E2E Tests;
Growing unit test suite;
New code always unit tested

„Interesting“ Deployment

Nice Cycle Time

# Code Quality Insights

Scan to get your insights

# Thank you

Ask me about these topics ☺

**Richard Gross** (he/him)

IT Archaeology + Health Checks

TestDSLs

Hypermedia

richargh.de/

@arghrich

rigross

Works for maibornwolff.de/

Code Quality Insights

Contact

Slides, Code, Videos

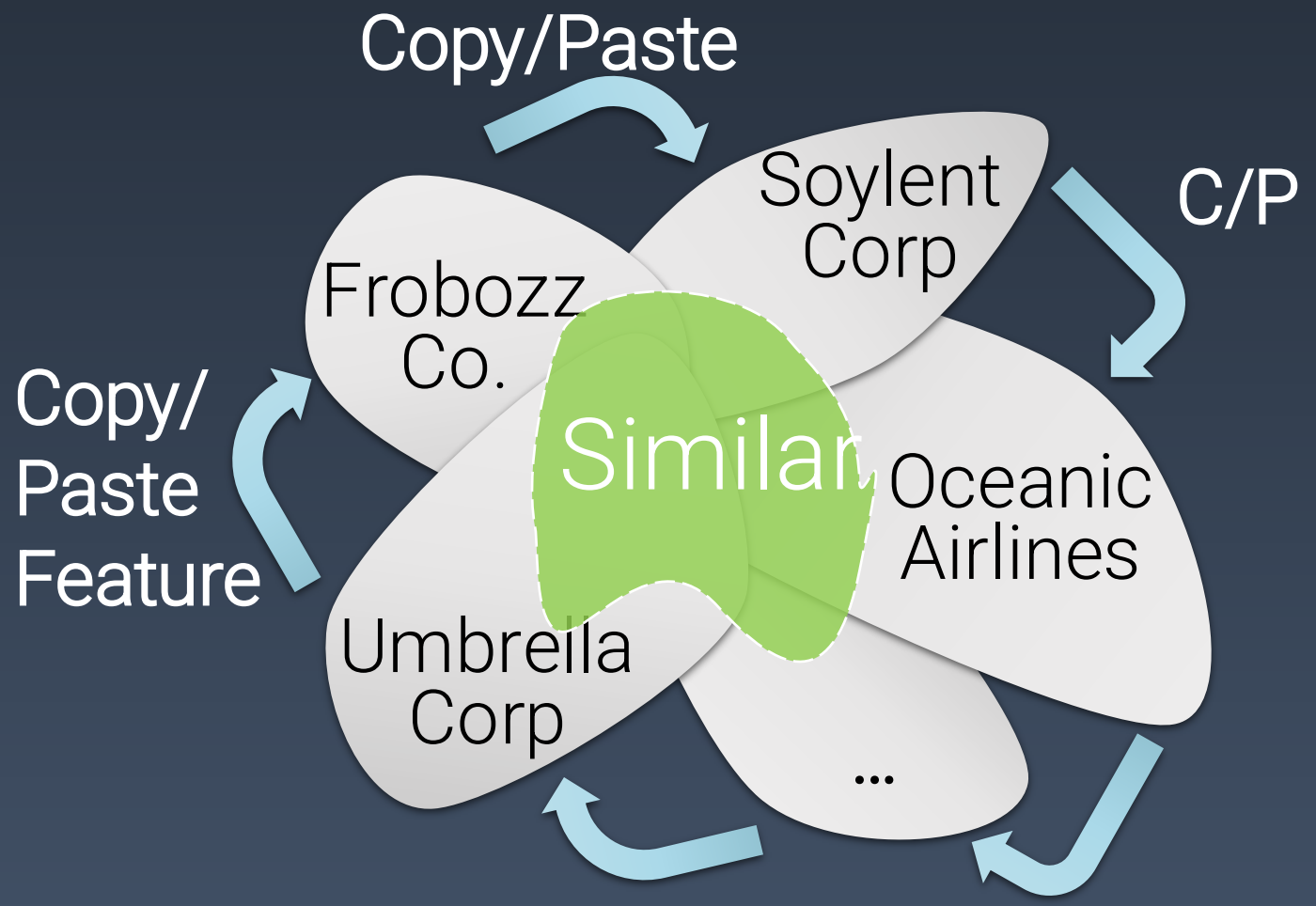
Slide 54 by richargh.de from 

# Backup

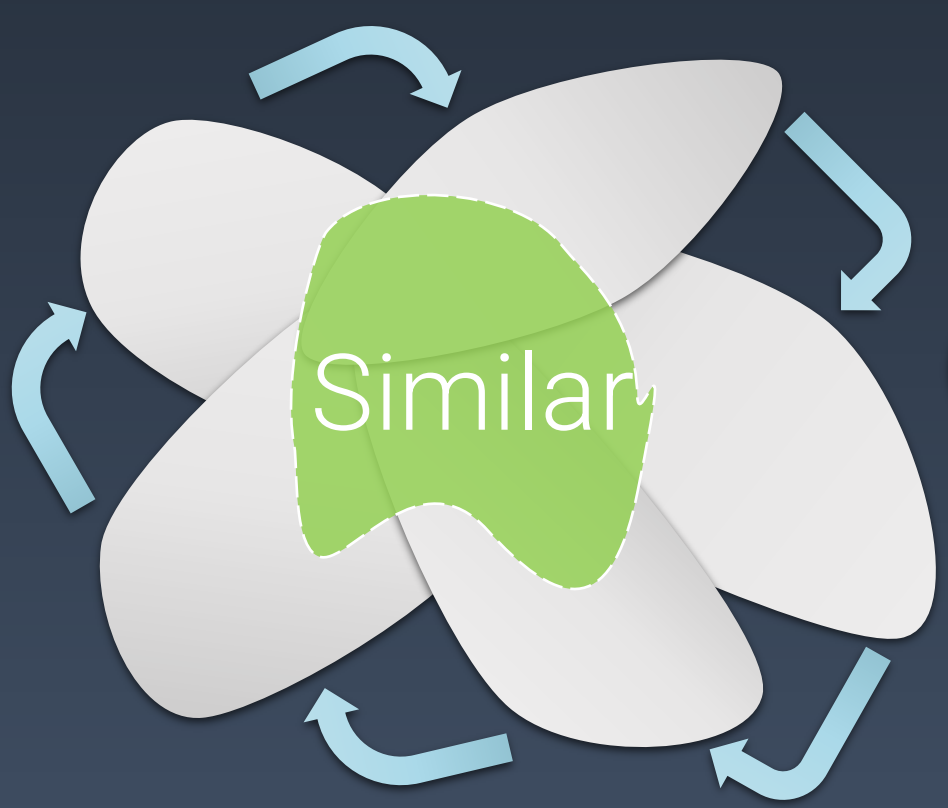

WARNING
IF YOU DON'T SCHEDULE TIME FOR MAINTENANCE, YOUR EQUIPMENT WILL SCHEDULE IT FOR YOU.
BRADY #110753 www.bradyid.com/visualworkplace Y922359

# 10 customer installations

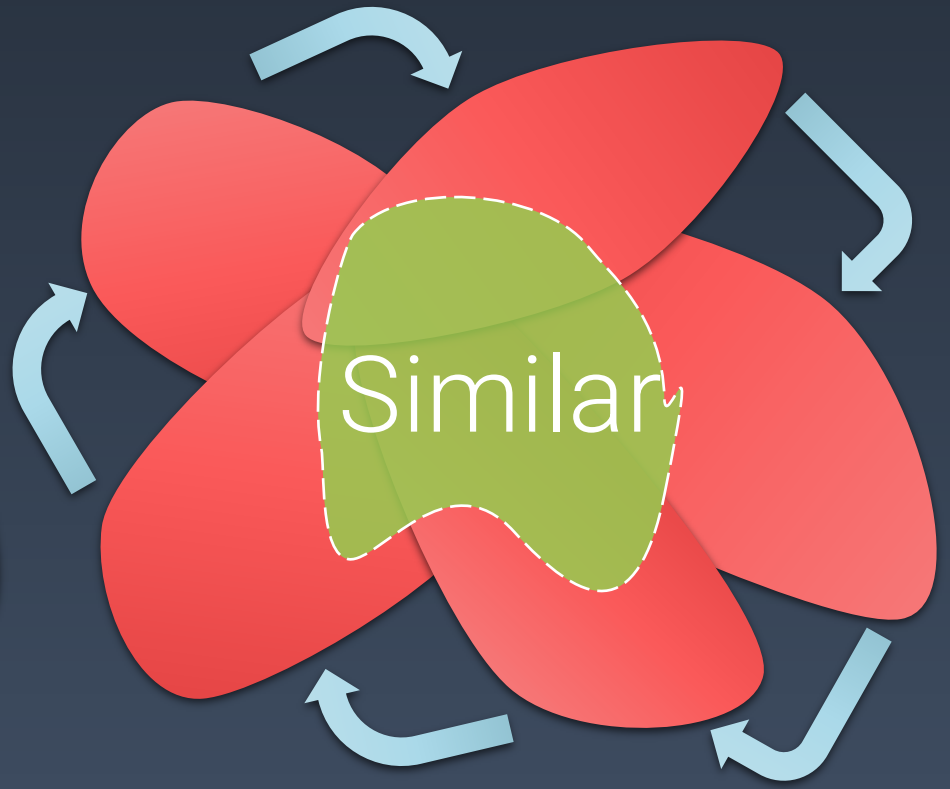
# And also 10 separate codebases



10 codebases




And 10 database schemata



# Our Legacy CD Principles

- Build quality in
- Focus on current tree, not on the forest
- Tackle fear head on
- Focus on reducing batch size
- Decouple Deployment and Release
- Computers perform repetitive tasks, people solve problems

CD Principles by Jez Humble

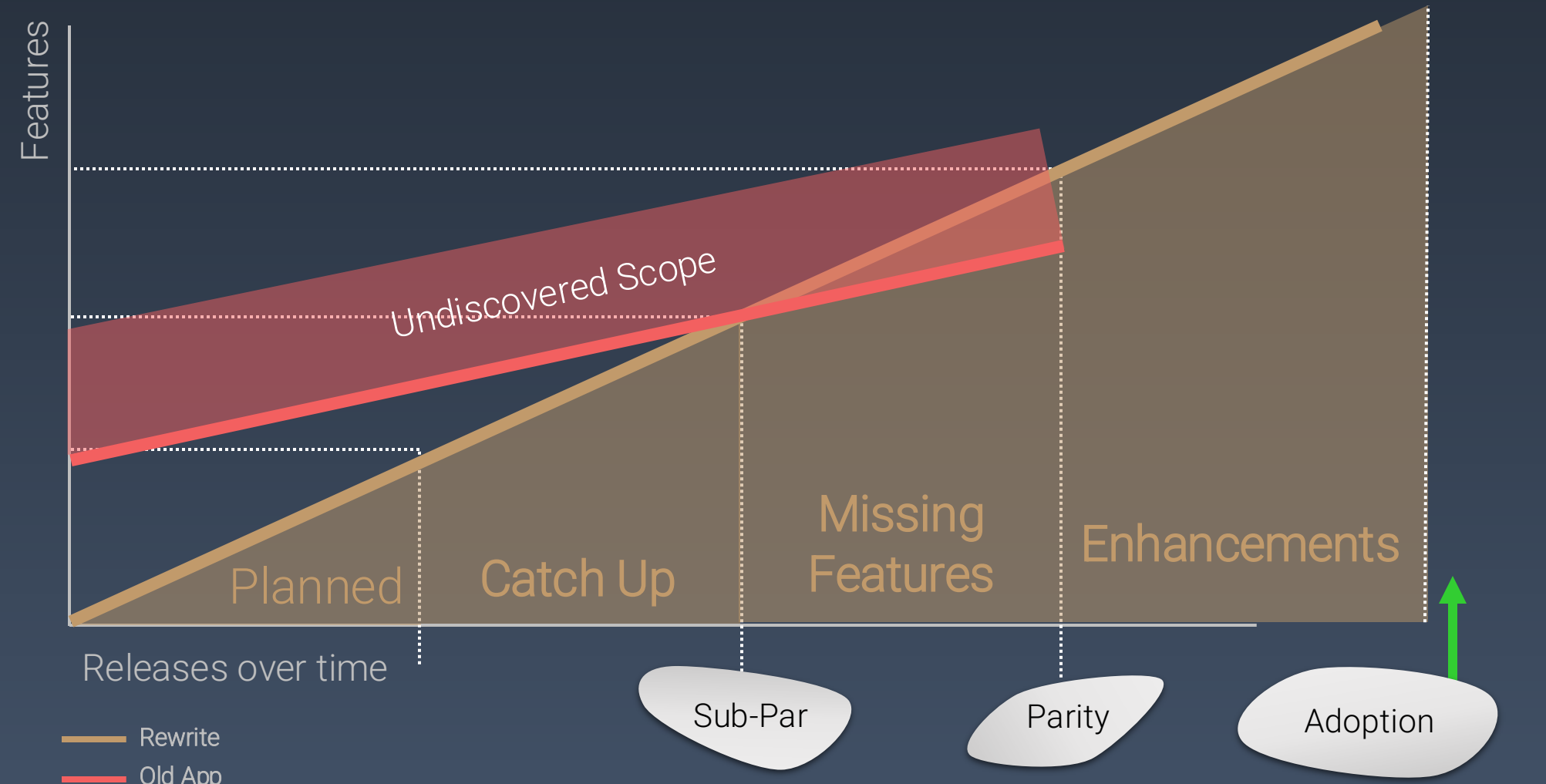
Four Principles of Low-Risk Software Releases by Jez Humble

# Our branching strategy

- Don‘t Branch
- Don‘t Even Branch
- Branch and you’re screwed
- Never Branch to Refactor

- Branches indicate $_{Ci}$=Continuous Isolation

# The True Cost of Rewrites




Original Article by Doug Bradbury [The True Cost of Rewrites]