scrum Tisch Darmstadt

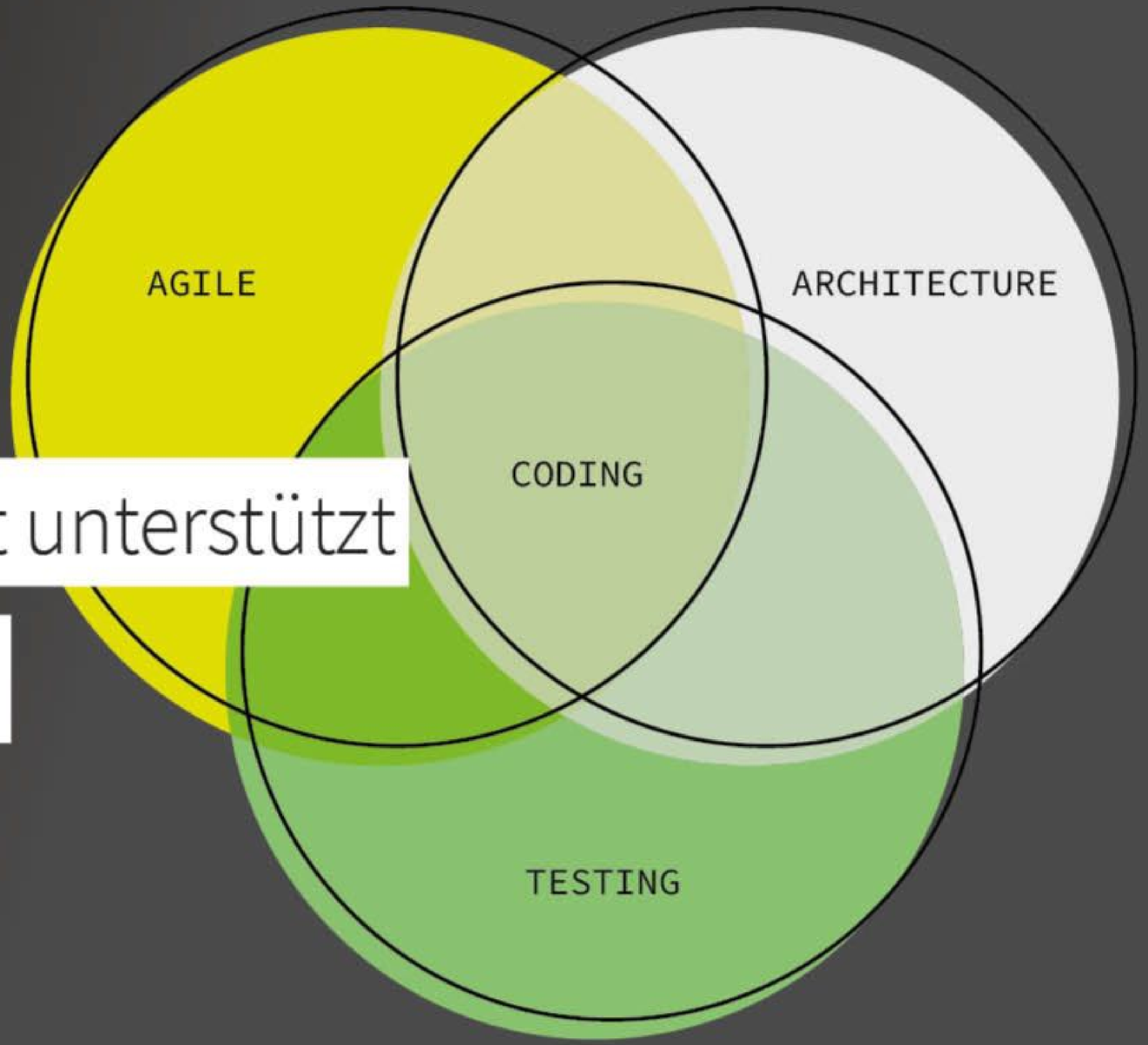
[18.10.2023 | 19 Uhr]

Wie Testbarkeit eure Agilität unterstützt

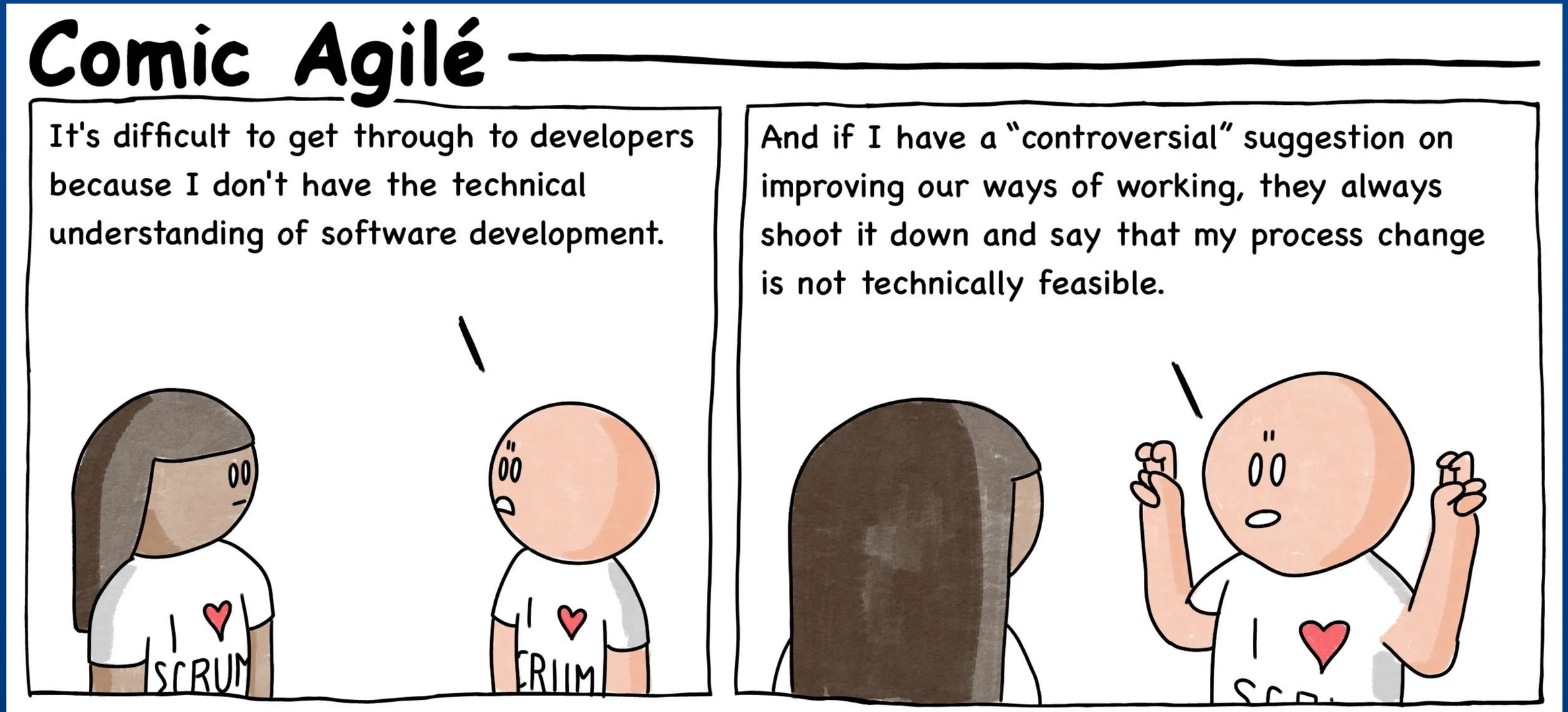
- Grundlagen und Beispiele

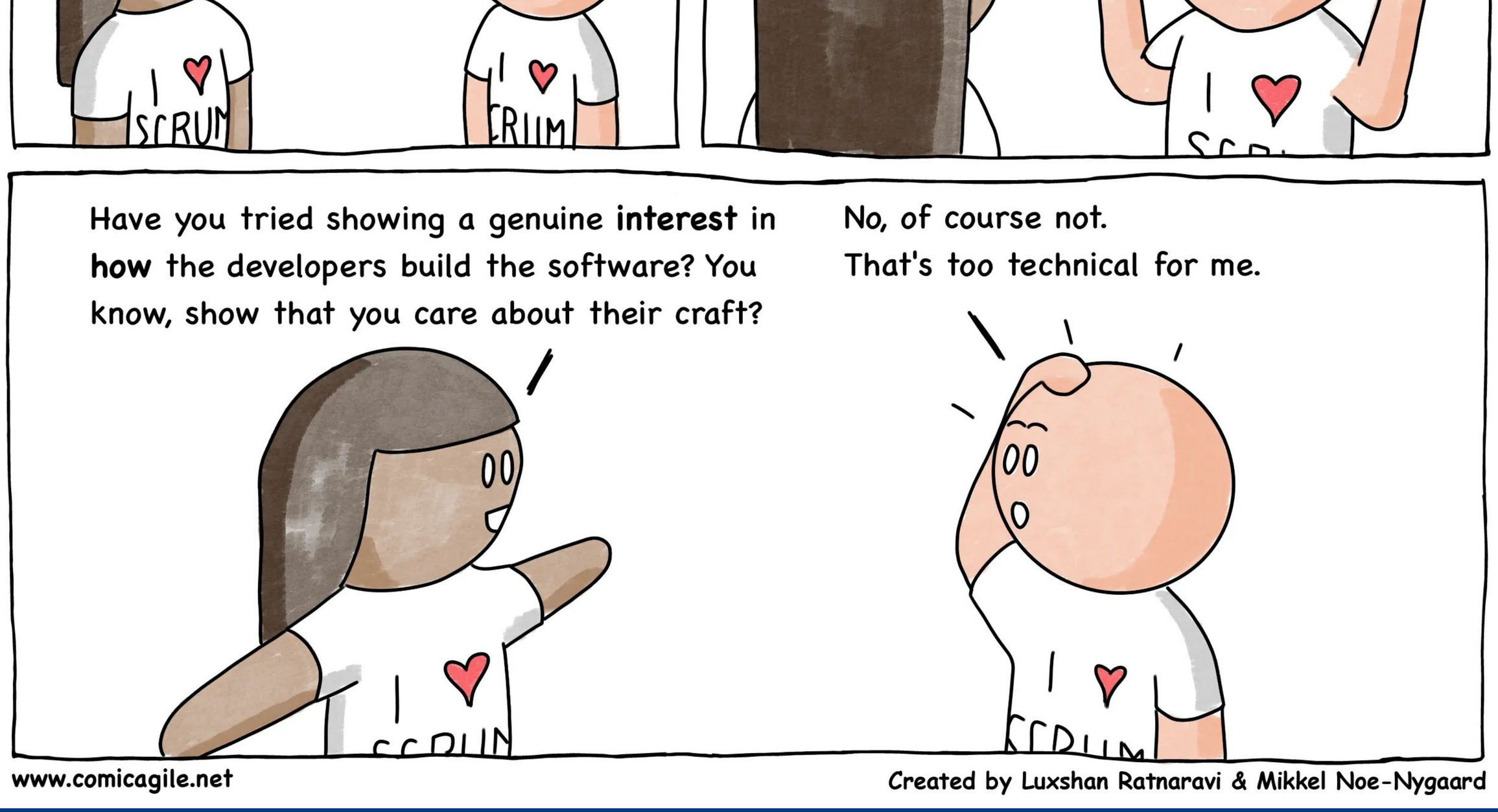
mit Thomas Much

@thmuch



Warum dieser Talk?





https://www.comicagile.net/comic/having-technical-knowledge/

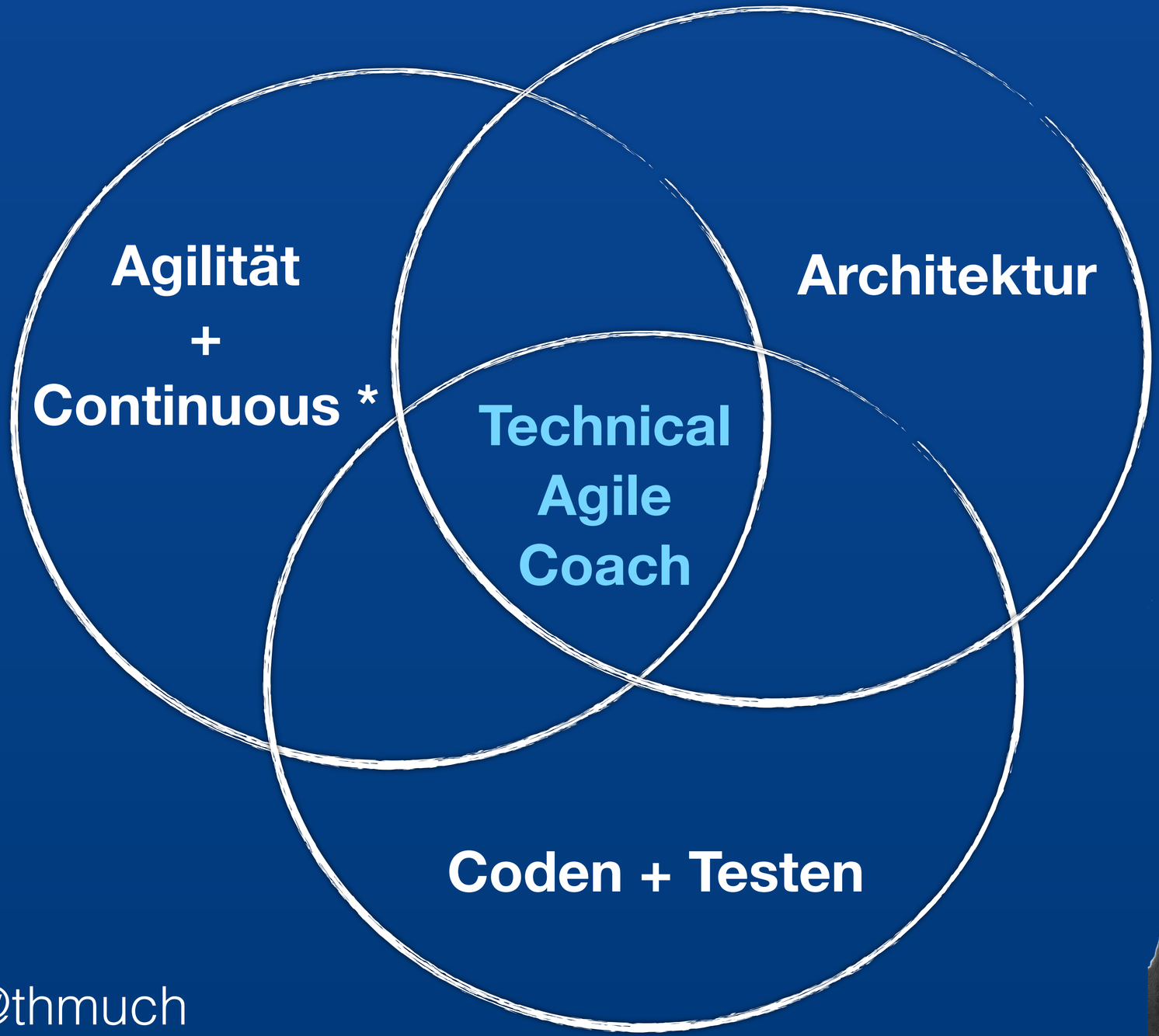
Hinweis

Die gleich gezeigten Ideen sind **nicht neu**.

Ab und zu müssen wir über solche **Grundlagen** reden.

Diese (alten) Grundlagen sind immer noch **fundamental wichtig**.

Vielleicht sogar wichtiger als vor 20 Jahren,
weil immer mehr Unternehmen **agil** sein
und Software **kontinuierlich testen** und ausliefern wollen.



Die Techniker

www.tk.de/IT

@thmuch

Coden + Testen

Typische Situationen:

Unser Team arbeitet nicht „Agile“ genug.

Kunden unzufrieden, Ergebnisse kommen zu spät.

„Wir sehen keinen Fortschritt“

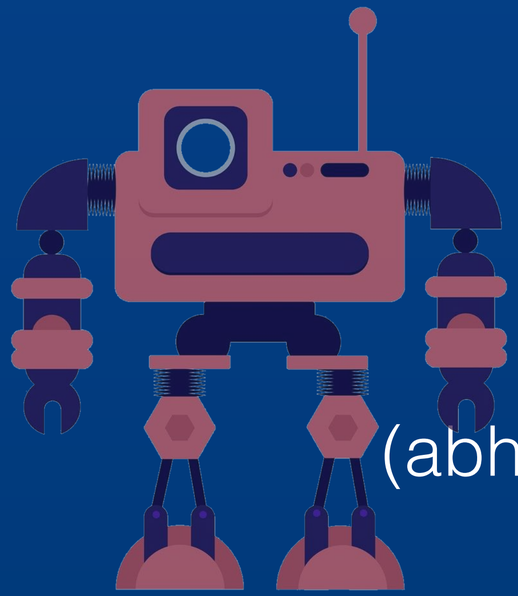
Agilität
+
Continuous *

Architektur

Typische Symptome

Keine Tests
(nur ein bisschen Herumgeklicke hier und da)

Weitestgehend manuelle Tests
(langsam!)



Langsame automatisierte Tests
(üblicherweise End-to-end via UI)

Fragile, instabile, wackelige Tests
(abhängig von der Umgebung bzw. externen Systemen)

https://pixabay.com/de/illustrations/roboter-abbildung-technologie-3124412/

https://pixabay.com/de/illustrations/nagel-hammer-mei%C3%9Fel-beitel-1816362/

Software testen – warum?

Funktionieren neuer **und alter** Features sicherstellen

neue Features

Änderungen 😳

Anpassungen bestehender Features

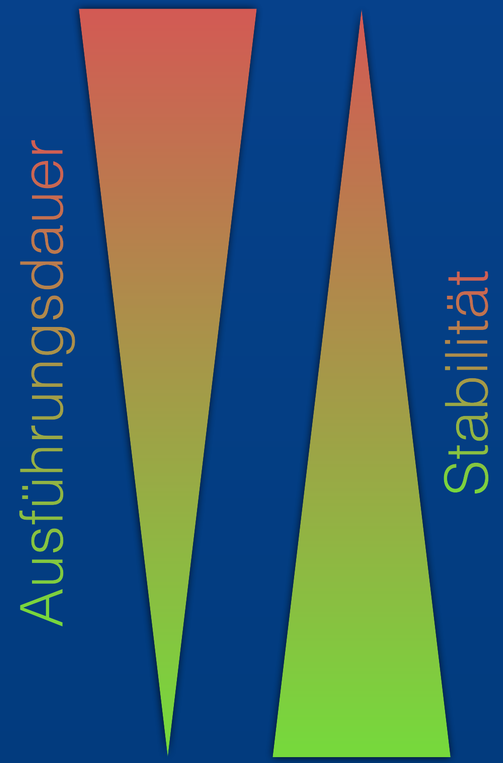
Bugfixes

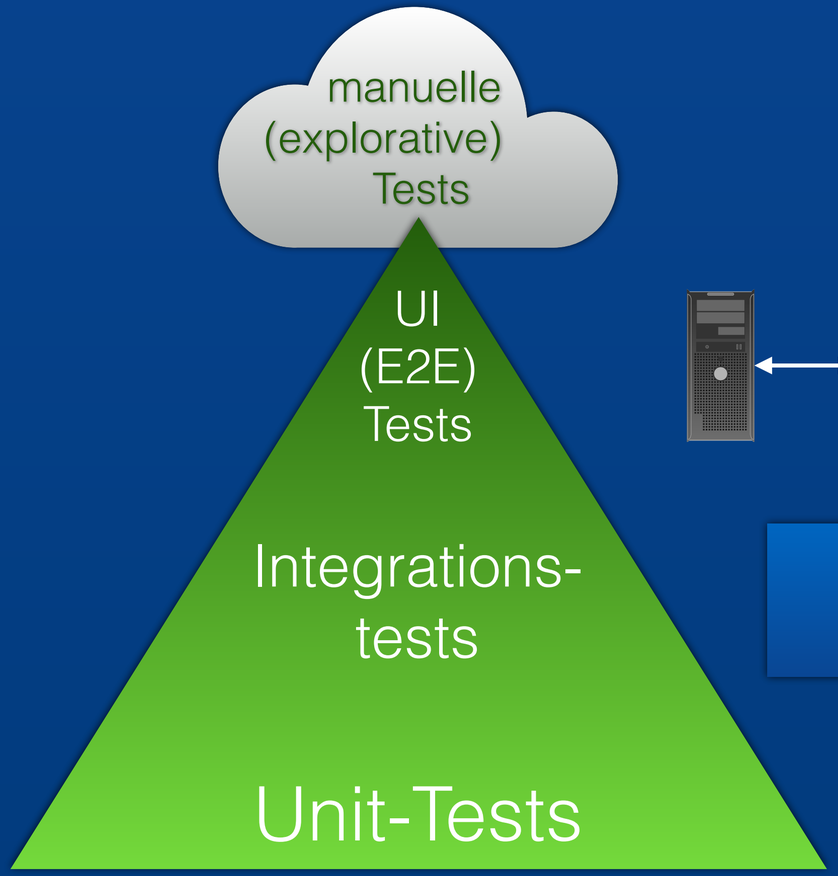
Deshalb
bauen wir
Software! 😀

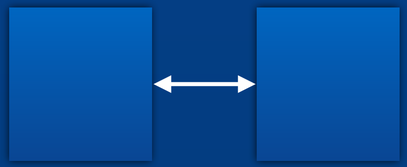
Risiko managen

Klassische „agile“ Testpyramide

Mike Cohn (2009):
„test automation pyramid“







```
class Service {
  function tuWas() {
    …
  }
}
```

https://pixabay.com/de/vectors/rechner-desktop-netzwerk-server-158776/

Testen im Wandel der Zeit

manuelle Tests

manuelle (explorative) Tests

UI (E2E) Tests

Integrations-tests

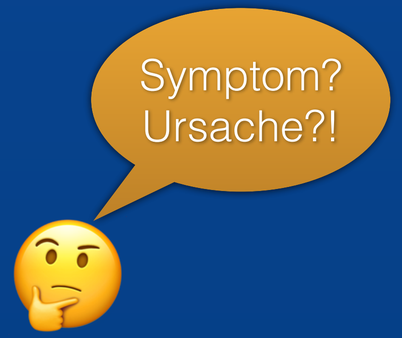
Unit-Tests

Testbarkeit

Containerisierung & Cloud

Ice cream cone emoji inspired by Neal Brooks ;-)

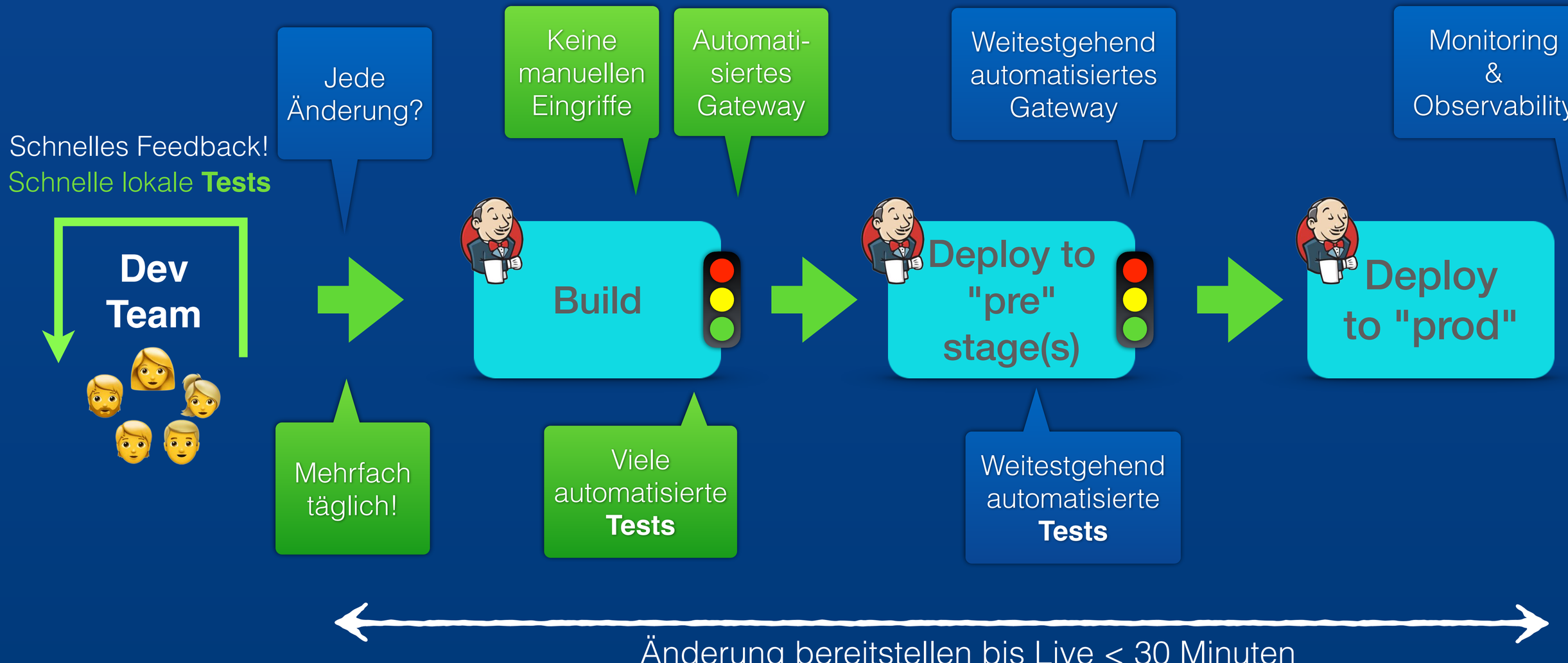
Meine Beobachtung



Teams mit **schneller Test-Automatisierung**
sind **agiler***
als Teams ohne oder mit nur langsamen Tests

*) liefern häufiger aus, liefern kleinere Einheiten, haben ein gemeinsames Verständnis ihrer Software, können sich gegenseitig besser unterstützen – und sie können einfacher Tests schreiben

Was macht solche Teams agiler?



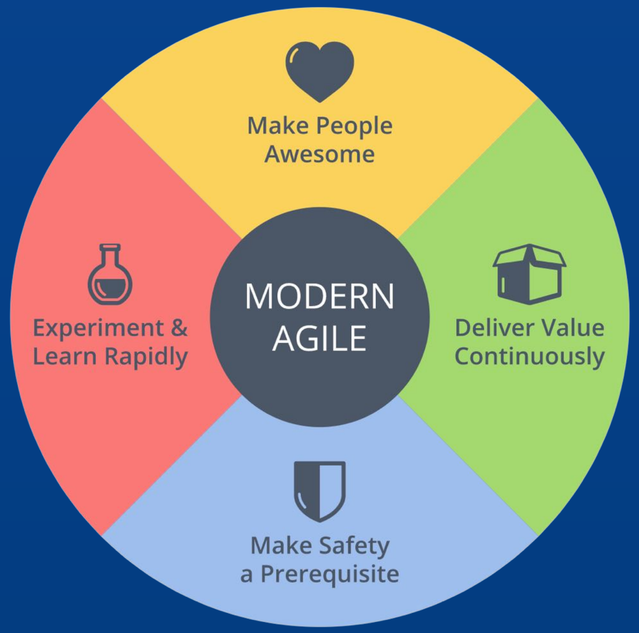
Wie können die das wagen?

Schnelle, verlässliche, umfassende Tests als Sicherheitsnetz

Sie **liefern kontinuierlich Wertvolles aus**
(und falls mal was Schlimmes™ passiert, war es nur
eine kleine Änderung, einfach zurückzudrehen)

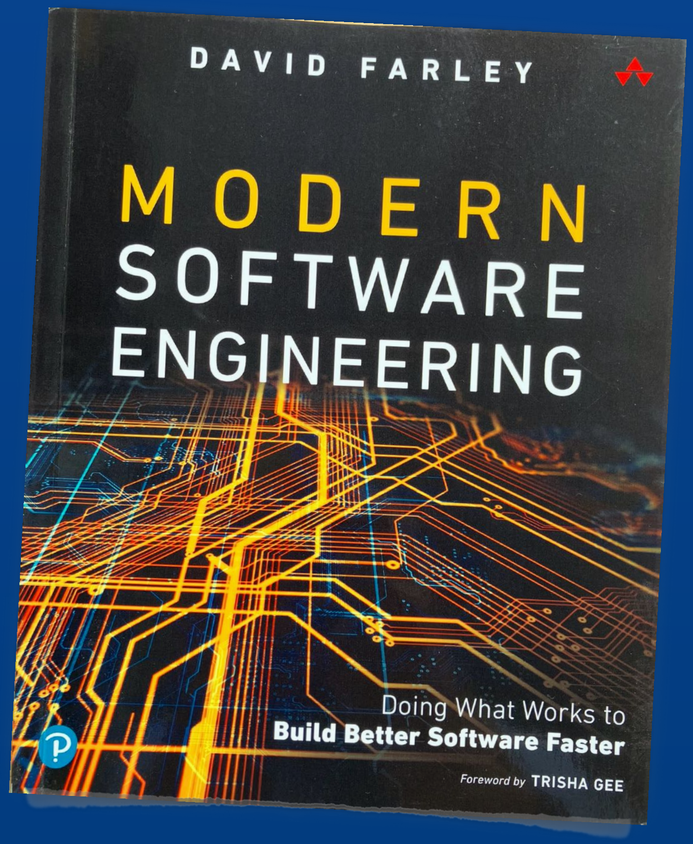
Glückliche Entwickler
(und vermutlich auch glückliche **Kunden**)

Sie trauen sich, Änderungen vorzunehmen,
trauen sich auszuprobieren, was den Kunden hilft

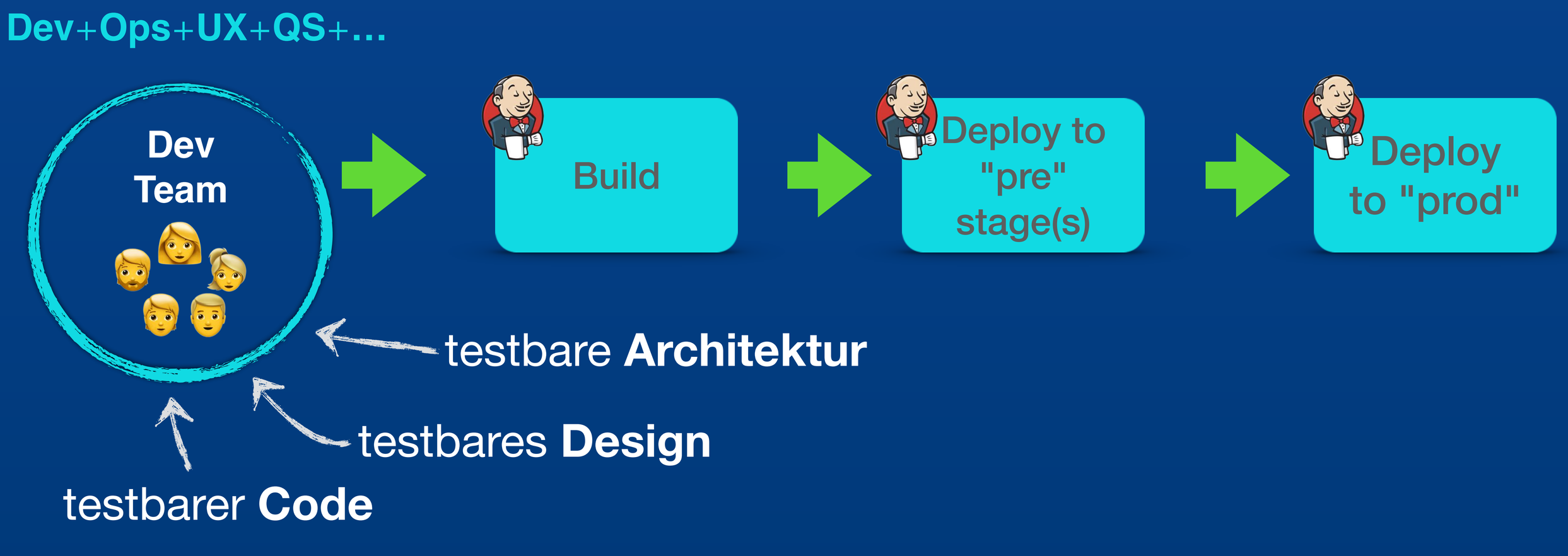


Das ist schön **kontinuierlich** und ziemlich **agil!**

😃



Kontinuierlich testen: Voraussetzungen?

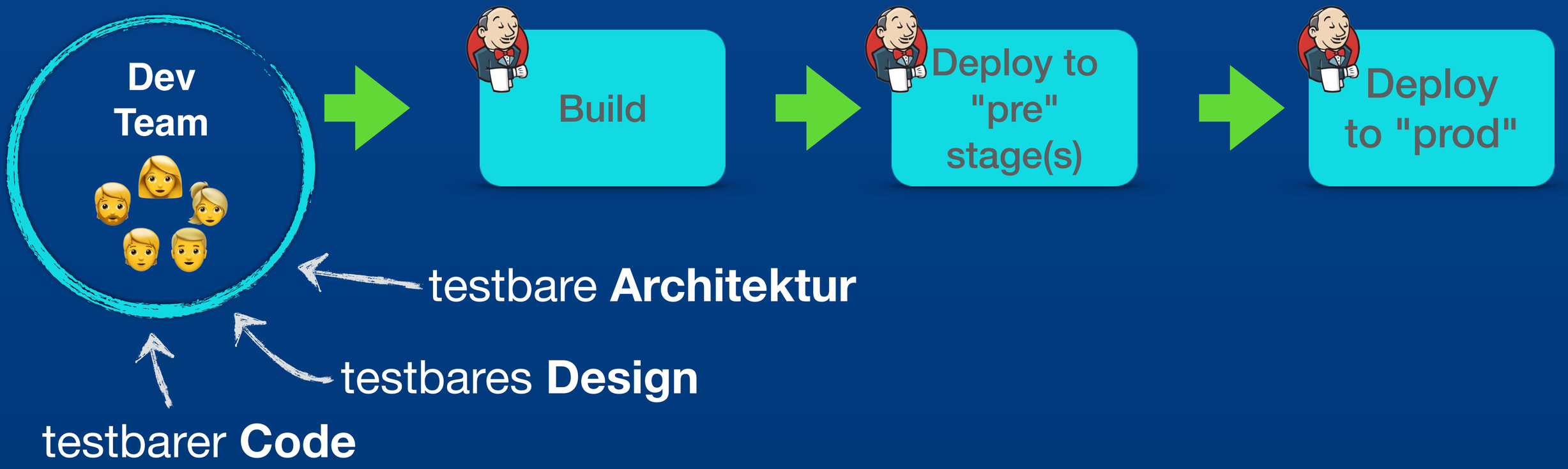


Warum geht das **alle Rollen** an?

Testbarkeit & kontinuierliches Testen

- ist nicht „magisch“ da
- muss **begleitet** und **moderiert** werden
- dabei hilft **Verständnis**

Dev+Ops+UX+QS+…



https://pixabay.com/photos/brick-brickwork-ruin-broken-decay-2205882/

Testbarkeit

Automatisierung **praktikabel**

Testdaten **unter Kontrolle**

Schnelles Feedback

Schnelle Tests

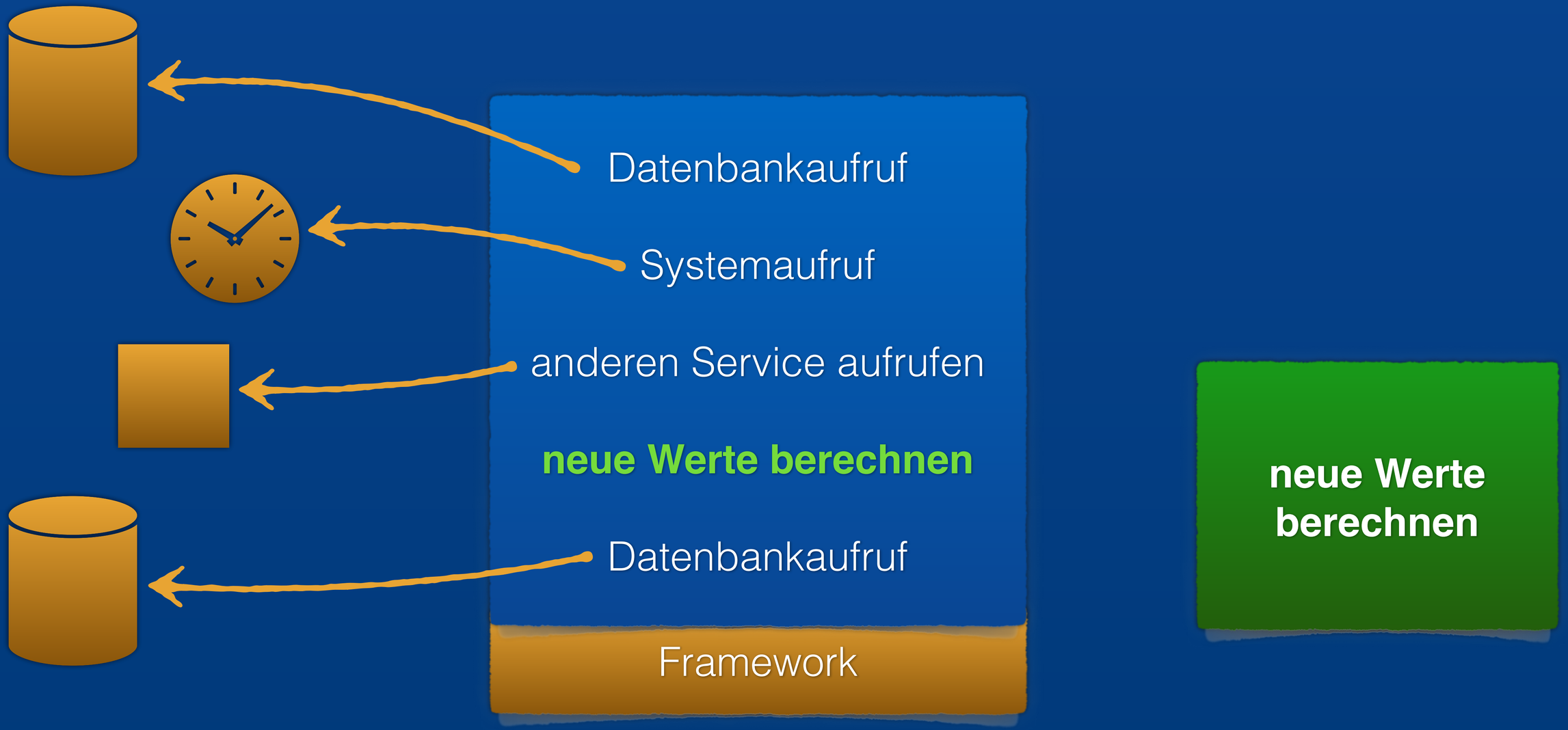
Viele
umfassende
schnelle
stabile
Tests

Schnelle Tests

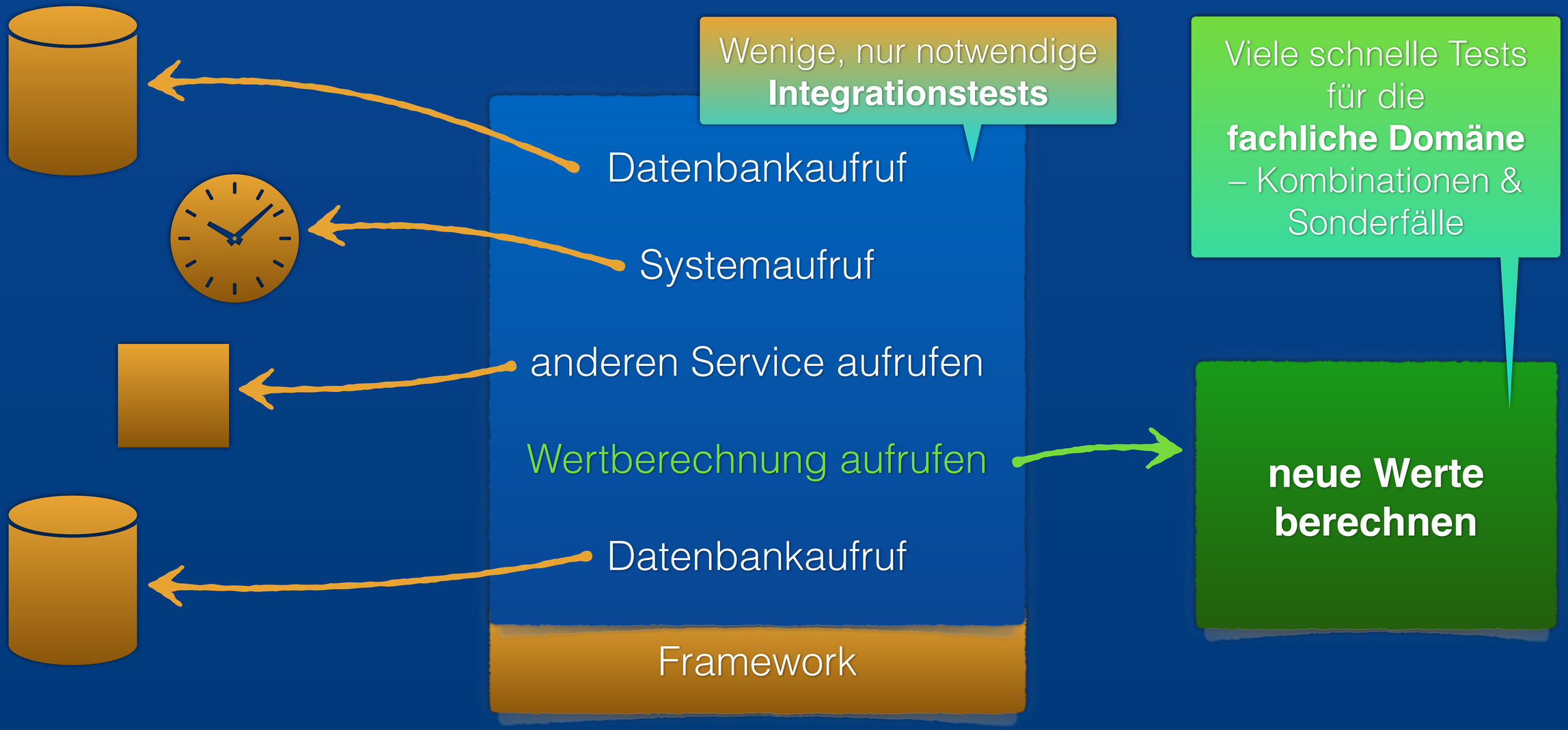
Code und **Architektur** müssen
für **Testbarkeit** designed sein

Abhängigkeiten
machen das schwierig

Abhängigkeiten



Abhängigkeiten



Abhängigkeiten erkennen

Tester:innen (Qualitätsspezialisten) können im Dev-Team **helfen**, schwierig zu testende **Abhängigkeiten zu erkennen**

Können Diskussionen der Entwickler:innen **moderieren** und mit **geeigneten Fragen** in Richtung Testbarkeit lenken

Dafür ist **Verständnis** hilfreich, wo überall Abhängigkeiten lauern und welche **Muster (Patterns)** es als mögliche Lösungen gibt

Scrum Master & Coaches können hier unterstützen!

Trennung von
Integrationscode
und
Domänencode

… für wenige bzw. **gut beherrschbare (testbare!) Abhängigkeiten**

Code-Design

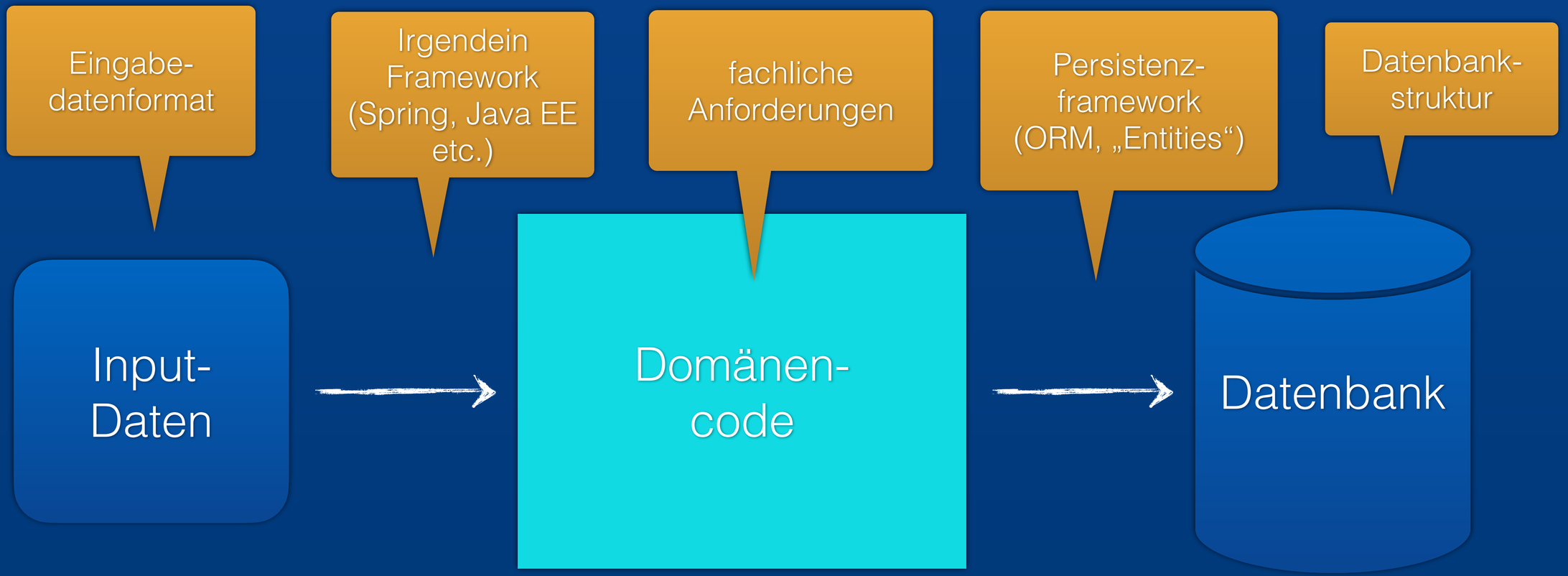
Ein
mögliches
Muster
(Pattern)

Architektur

Typisches Beispiel

Viele, viele Abhängigkeiten

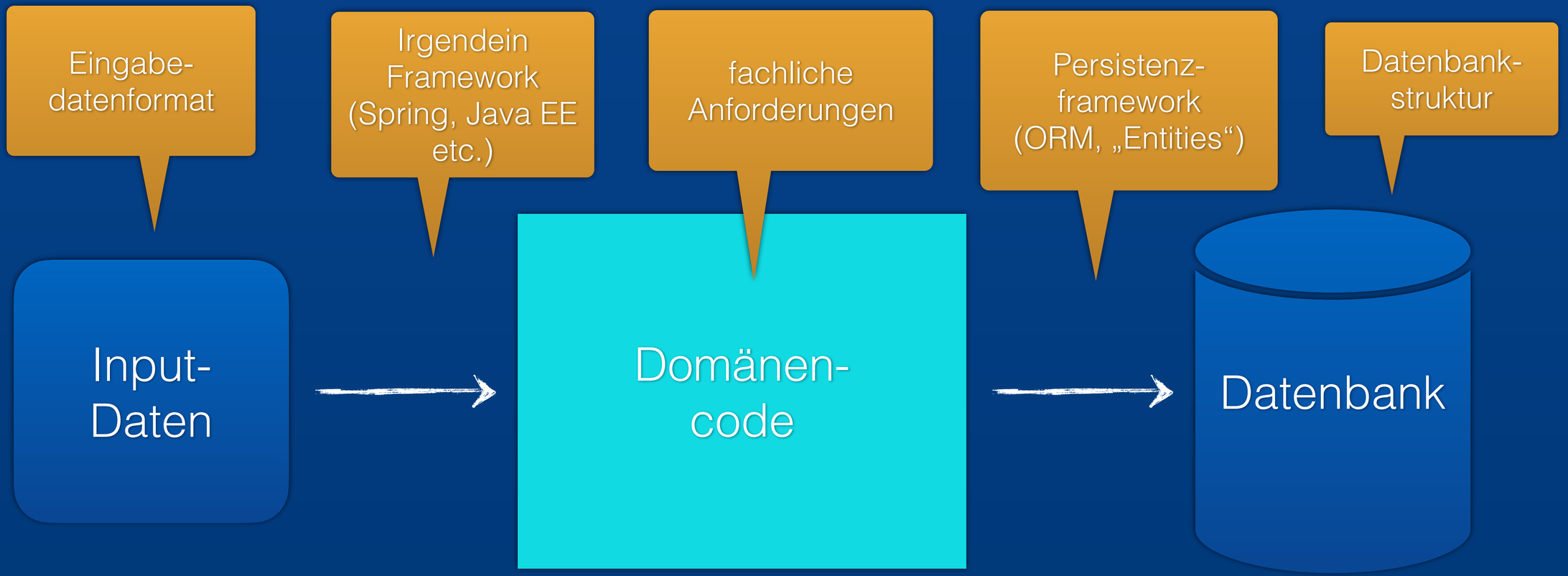
Abhängigkeiten sind oft **nicht unter unser Kontrolle**



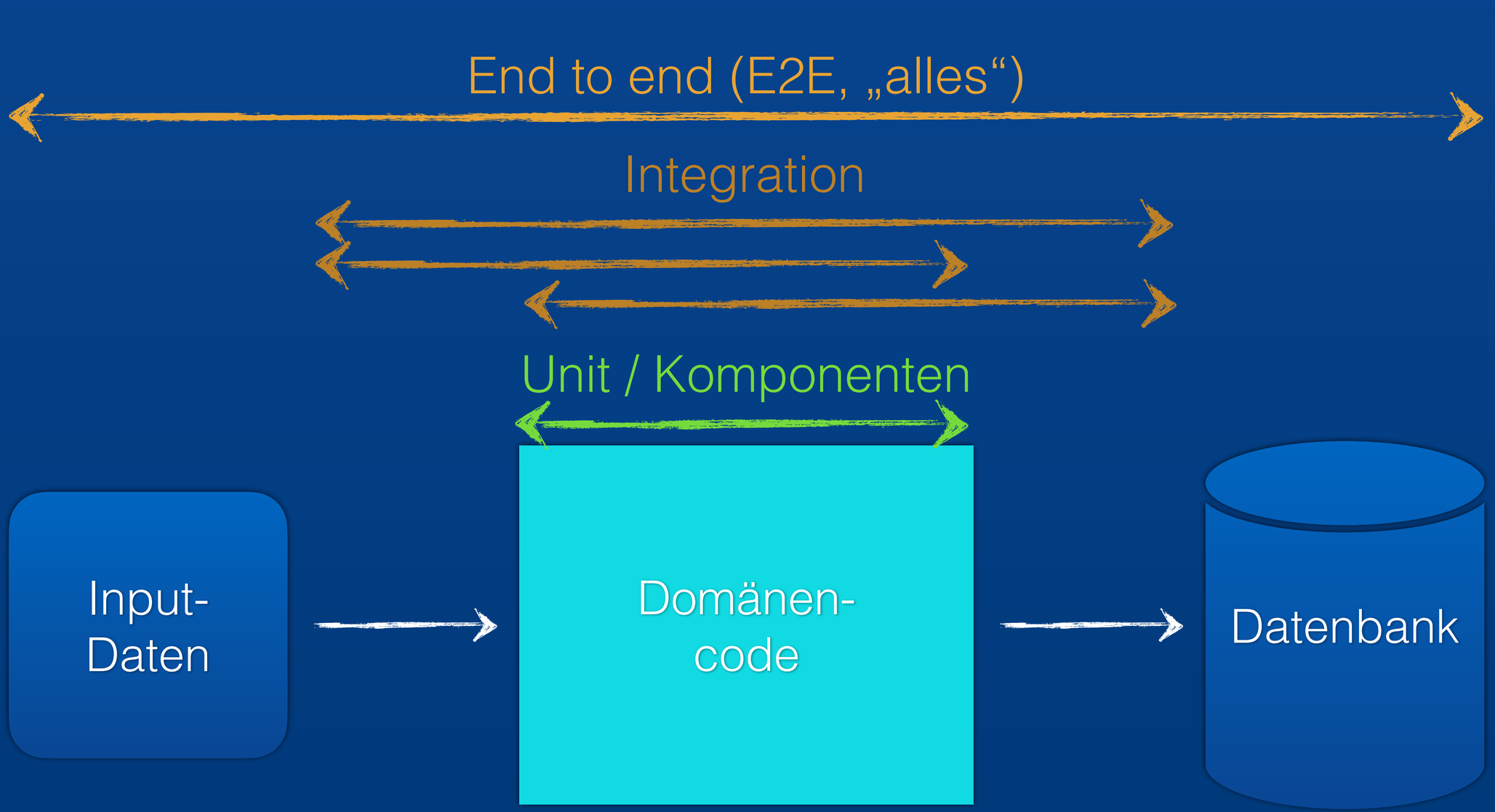
Abhängigkeiten – Gründe für Änderungen

… Abhängigkeiten, die den Code kaputtmachen

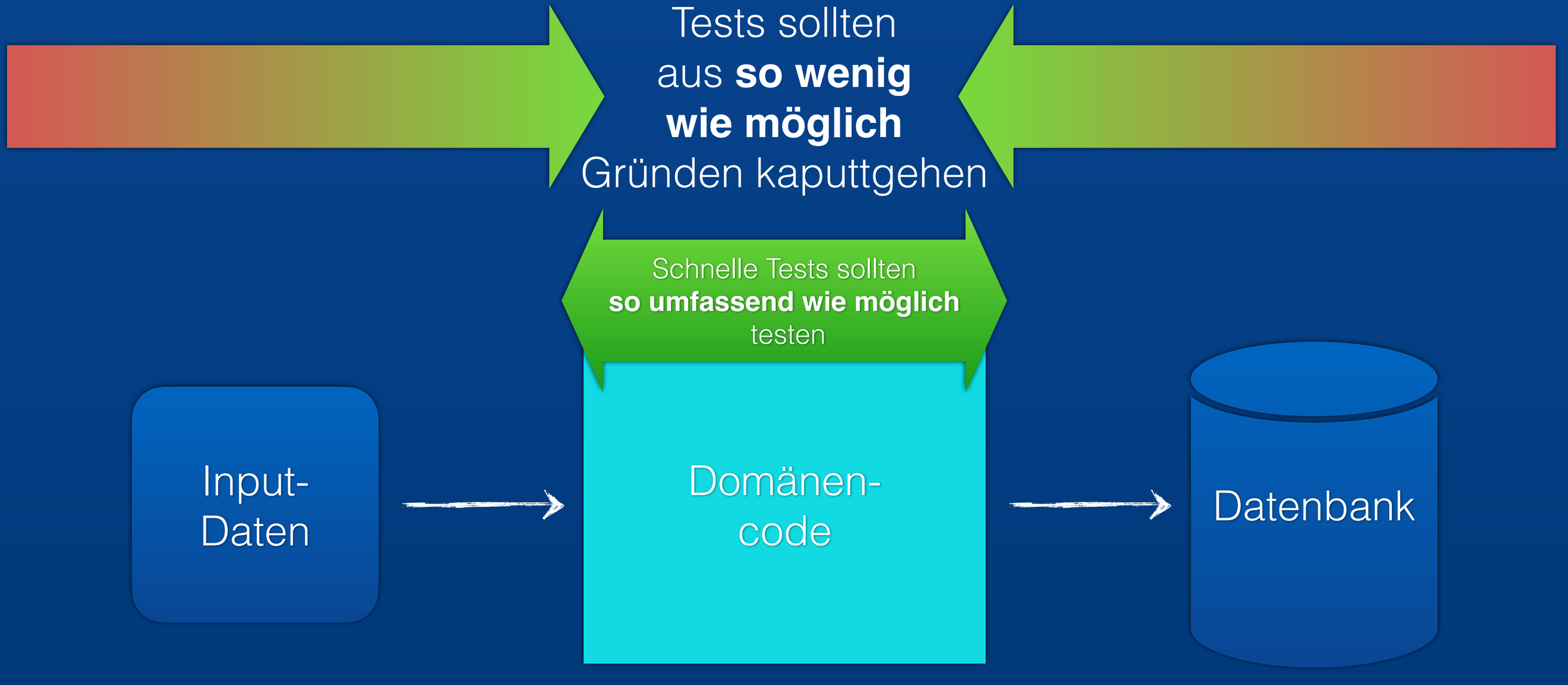
… Abhängigkeiten, die die Tests kaputtmachen



Was wird wo getestet?

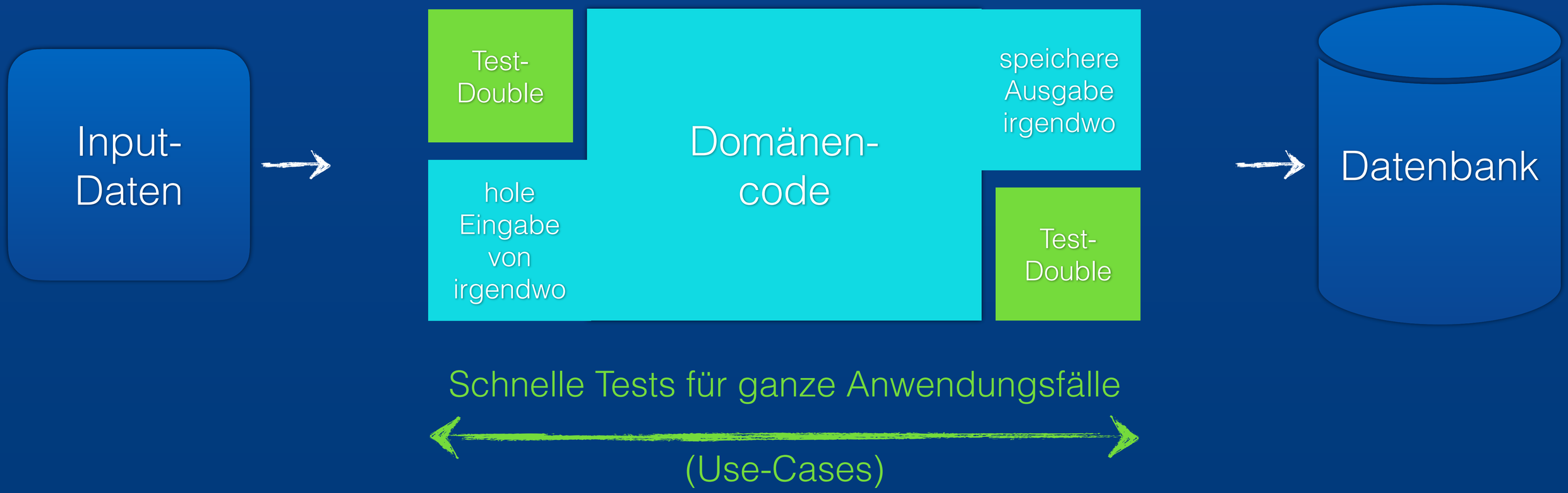


Wenige, beherrschbare Abhängigkeiten



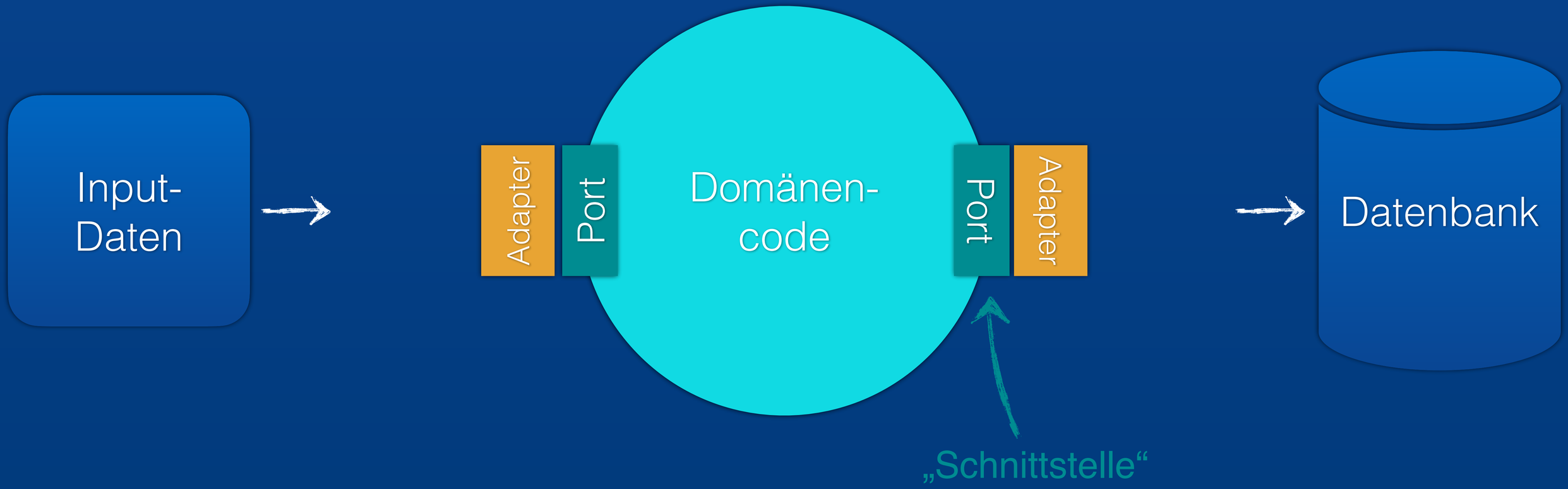
Abhängigkeiten nach außen …

Architekturmuster



Architekturmuster

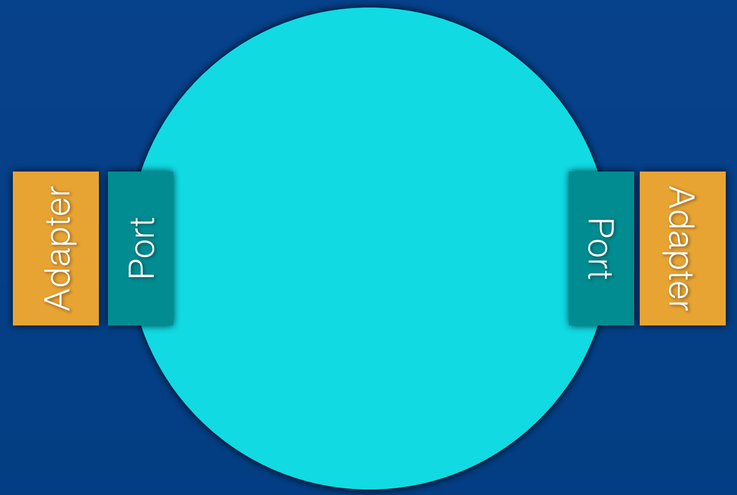
Architekturstile



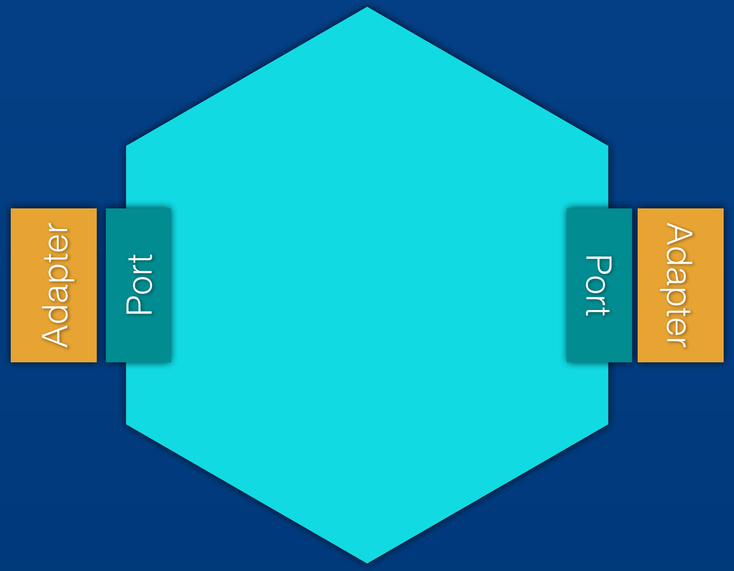
Architekturstile

Beispiele!

Ports & Adapters

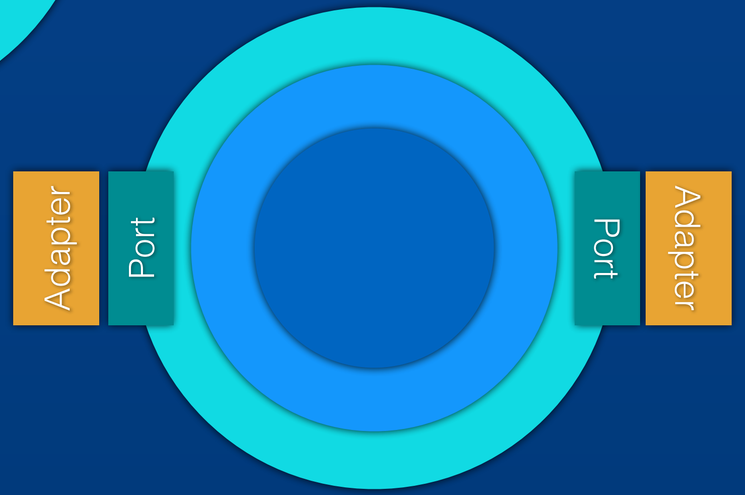


Hexagonal

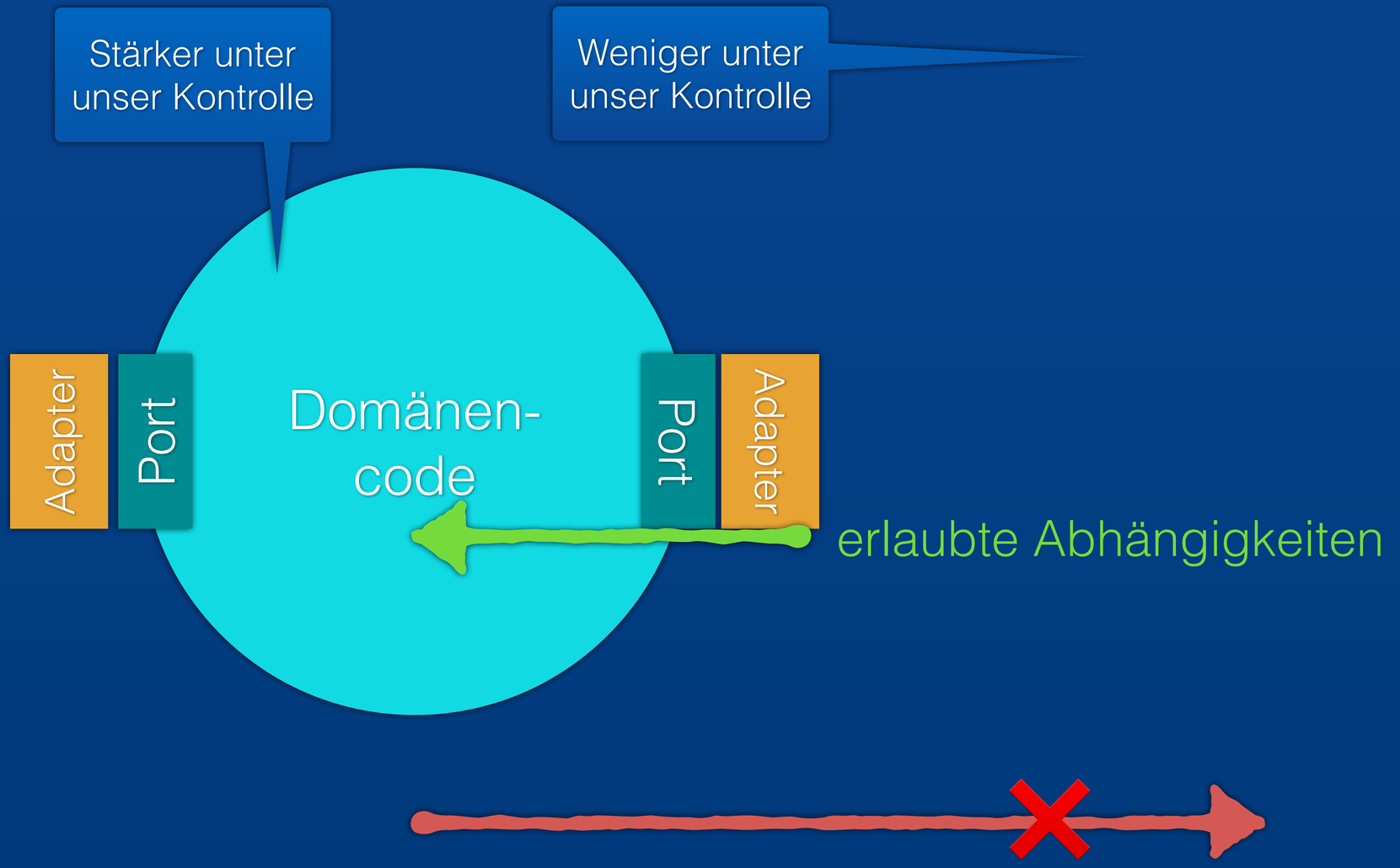


Onion (Ringe)

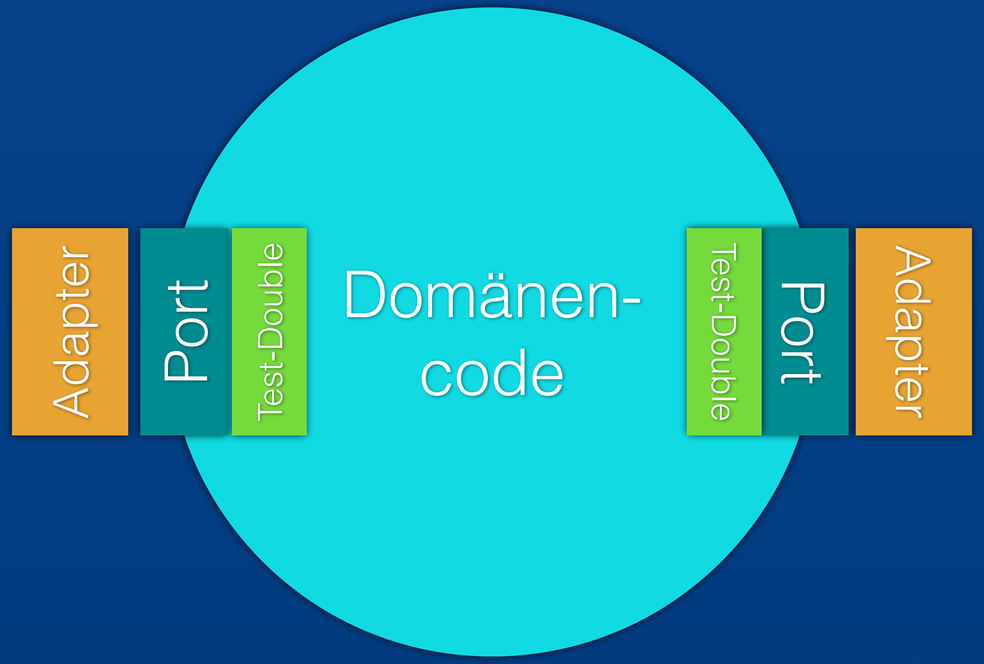
u.a.



Innen & außen


Stärker unter unser Kontrolle
Weniger unter unser Kontrolle
Adapter
Port
Domänen-code
Port
Adapter
erlaubte Abhängigkeiten

Schnelle Tests



Schnelle, stabile,
umfassende **Tests**

Fazit & Ausblick

Wie Testbarkeit eure Agilität unterstützt

Schnelle Tests liefern
kontinuierlich schnelles Feedback

Je **umfassender** die schnellen Tests sind,
desto **sicherer** lassen sich Änderungen vornehmen und ausliefern.

Dafür müssen Code-Design und Architektur auf **Testbarkeit**
ausgelegt sein, z.B. durch das **Beherrschen von Abhängigkeiten**.

Wie Testbarkeit eure Agilität unterstützt

Moderation gefragt

Welche Abstraktionen lohnen sich? Was ist **angemessen**?

Wie können **schnelle Tests** erreicht werden?

Alle Rollen sind **von Anfang an** beteiligt

Testbarkeit wird **von Anfang an** mitgedacht

Die richtigen **Fragen** fragen

„**Wie** werden wir das testen?“

„**Warum** ist das schwer zu testen?“

In unterschiedlichen Rollen:
Entwickler. Architekt. Tester. **Coach.**

„Was müssten wir **ändern**,
um bessere Testbarkeit zu erreichen?“

„Wäre es **gut genug**, … zu testen,
wenn wir … separieren würden?“

Testbarkeit **von Anfang an**

User Story & Akzeptanzkriterien

Aufgabengröße

Aufgabenschnitt

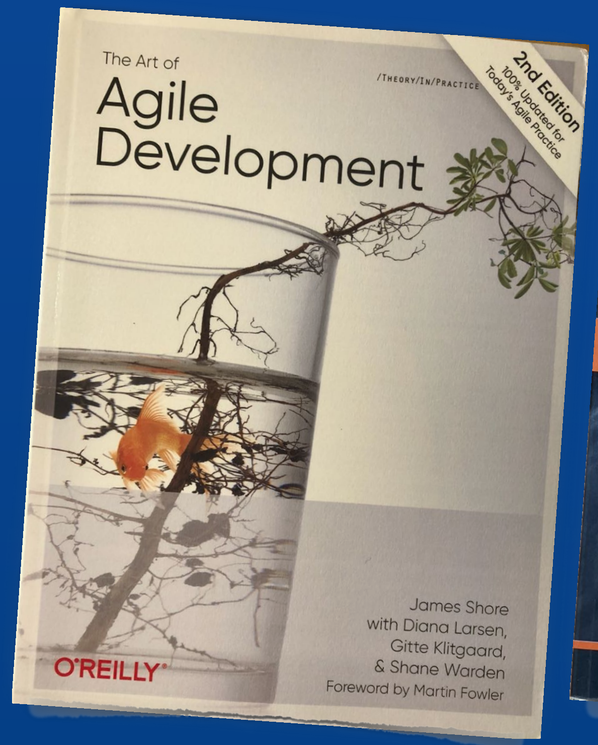
…

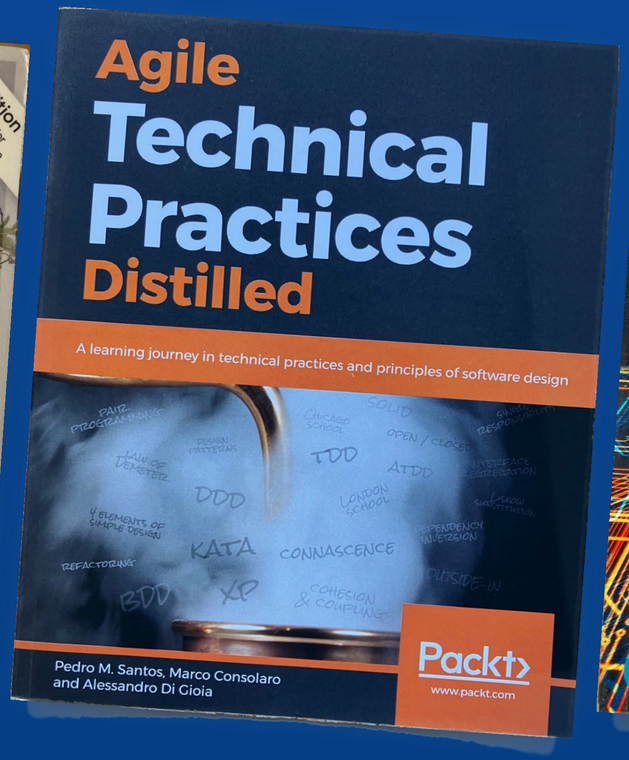
Zusammen arbeiten.

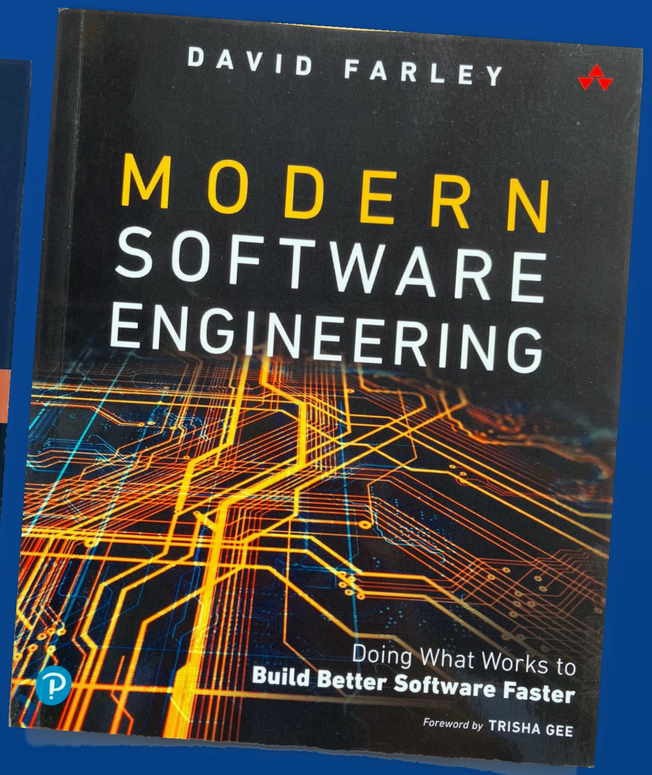
Zusammen coden.

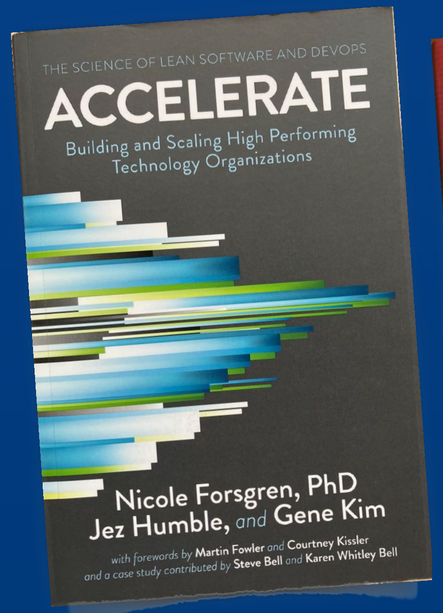
Zusammen testen.

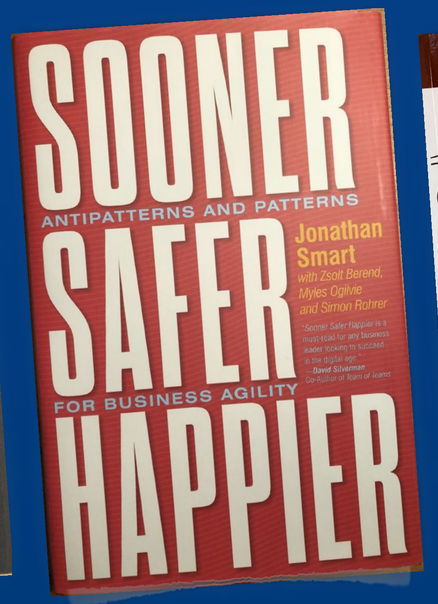
Voneinander lernen.

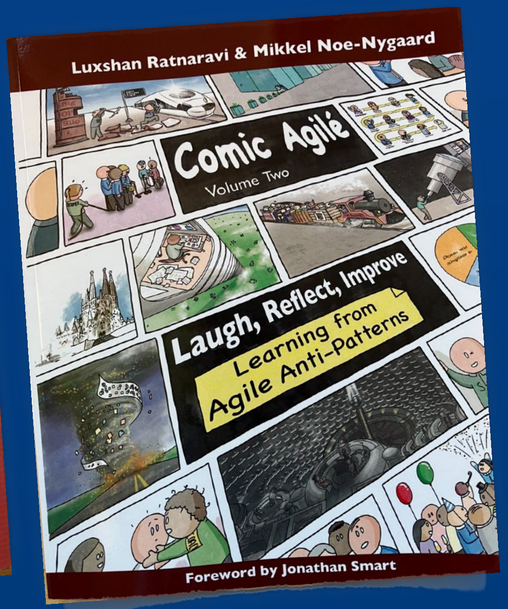

The Art of
Agile Development
/Theory/In/Practice
2nd Edition
100% Updated for Today's Agile Practice
James Shore
with Diana Larsen,
Gitte Klitgaard,
& Shane Warden
Foreword by Martin Fowler
O'REILLY


Agile
Technical
Practices
Distilled
A learning journey in technical practices and principles of software design
Pedro M. Santos, Marco Consolaro
and Alessandro Di Gioia
Packt
www.packt.com


DAVID FARLEY
MODERN
SOFTWARE
ENGINEERING
Doing What Works to
Build Better Software Faster
Foreword by TRISHA GEE


THE SCIENCE OF LEAN SOFTWARE AND DEVOPS
ACCELERATE
Building and Scaling High Performing
Technology Organizations
Nicole Forsgren, PhD
Jez Humble, and Gene Kim
with forewords by Martin Fowler and Courtney Kissler
and a case study contributed by Steve Bell and Karen Whitley Bell


SOONER
SAFER
HAPPIER
ANTIPATTERNS AND PATTERNS
FOR BUSINESS AGILITY
Jonathan
Smart
with Zsolt Berend,
Myles Ogilvie
and Simon Rohrer


Luxshan Ratnaravi & Mikkel Noe-Nygaard
Comic Agile
Volume Two
Laugh, Reflect, Improve
Learning from
Agile Anti-Patterns
Foreword by Jonathan Smart

Scrum Tisch Darmstadt

Vielen Dank 😊

Die Techniker

www.tk.de/IT

@thmuch

Testgeschwindigkeitspyramide?!