Abhängigkeiten beherrschbar machen:

Bessere **Testbarkeit** durch Code-Design- und Architekturmuster

Thomas Much
@thmuch

24. Mai 2023, Frankfurt/Main

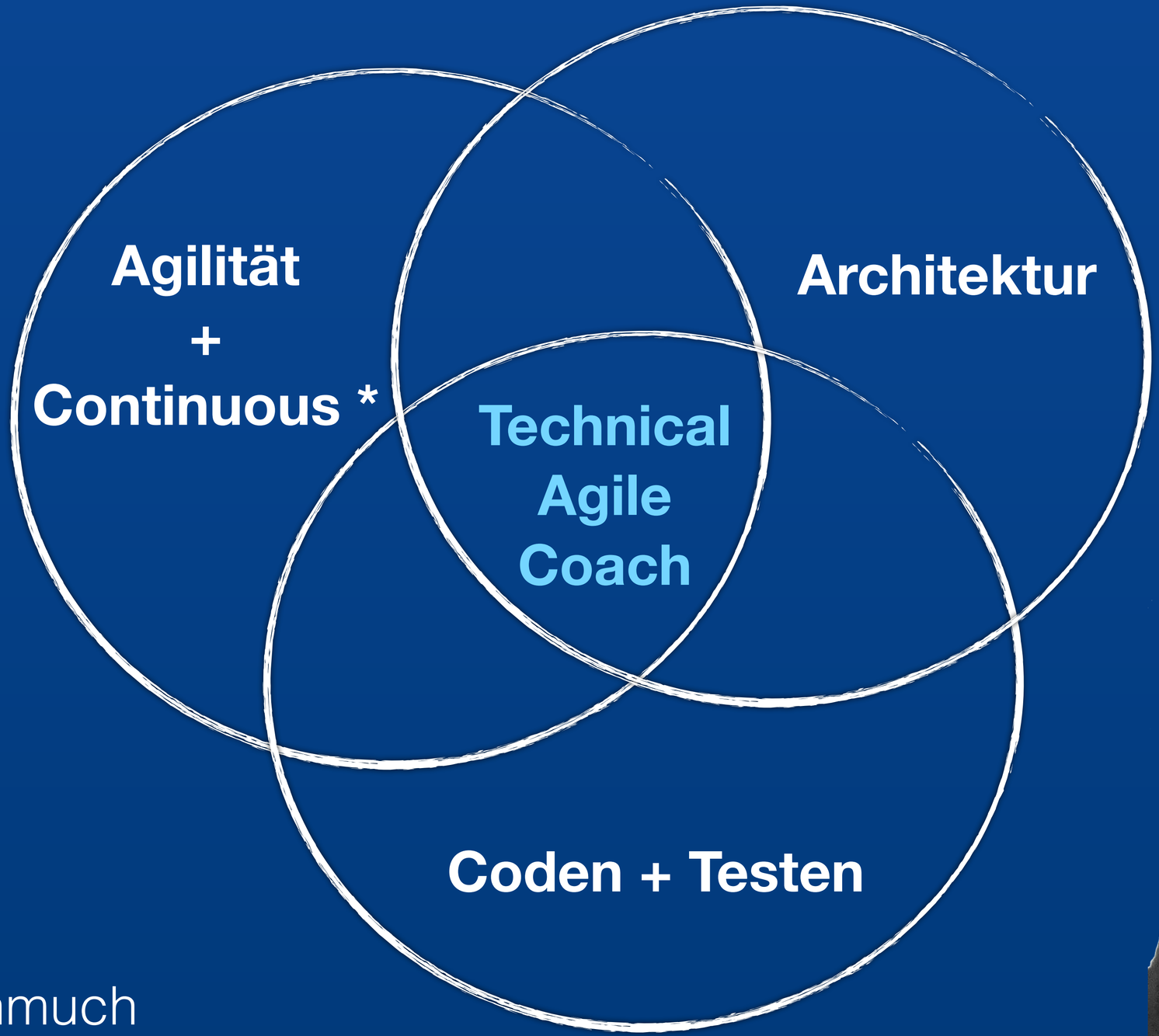
Hinweis

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.

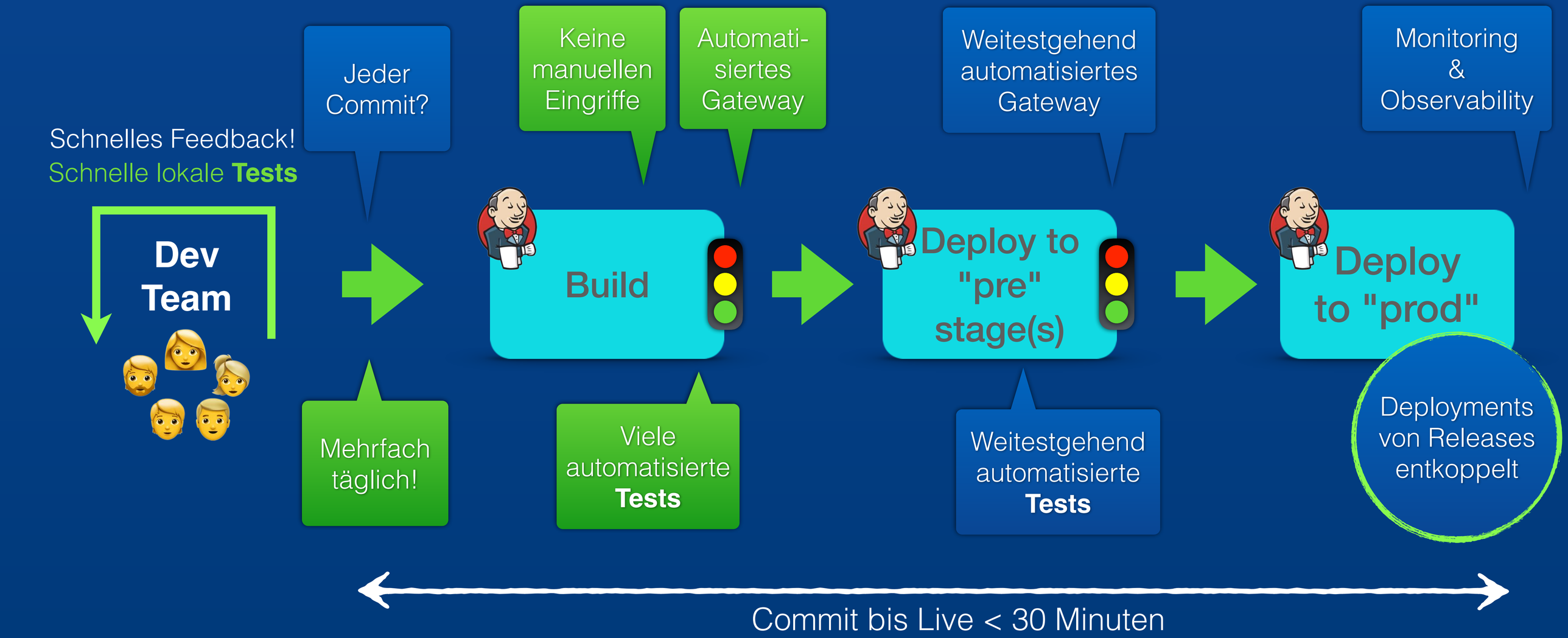


Die Techniker

www.tk.de/IT

@thmuch

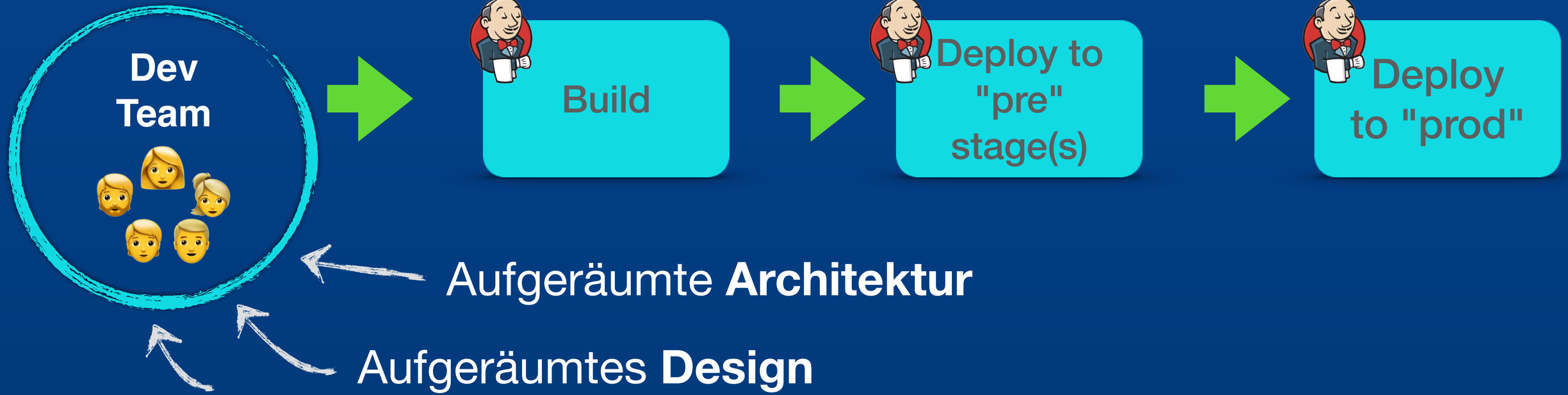
Kontinuierlich testen



Kontinuierlich testen: Voraussetzungen?

„Clean Code“ „Clean Architecture“ **„testbar“**

Dev+Ops+UX+QS+…



Aufgeräumte **Architektur**

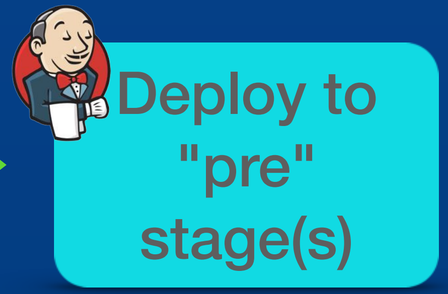
Aufgeräumtes **Design**

Aufgeräumter **Code**

Alles automatisiert? Wo sind die Tester?

Qualitätsspezialisten!

Dev+Ops+UX+QS+…



- moderieren das Testen
- vermitteln Test-Wissen (was / wo / wie getestet wird)

➡ **Entwickler:innen müssen Test-Wissen aufbauen**

➡ **Tester:innen sollten Programmierwissen aufbauen**

Testbarkeit

Automatisierung **praktikabel**

Testdaten **unter Kontrolle**

Schnelles Feedback

Testpyramide & Co.

manuelle Tests

manuelle (explorative) Tests

UI (E2E) Tests

Integrations-tests

Unit-Tests

Testbarkeit

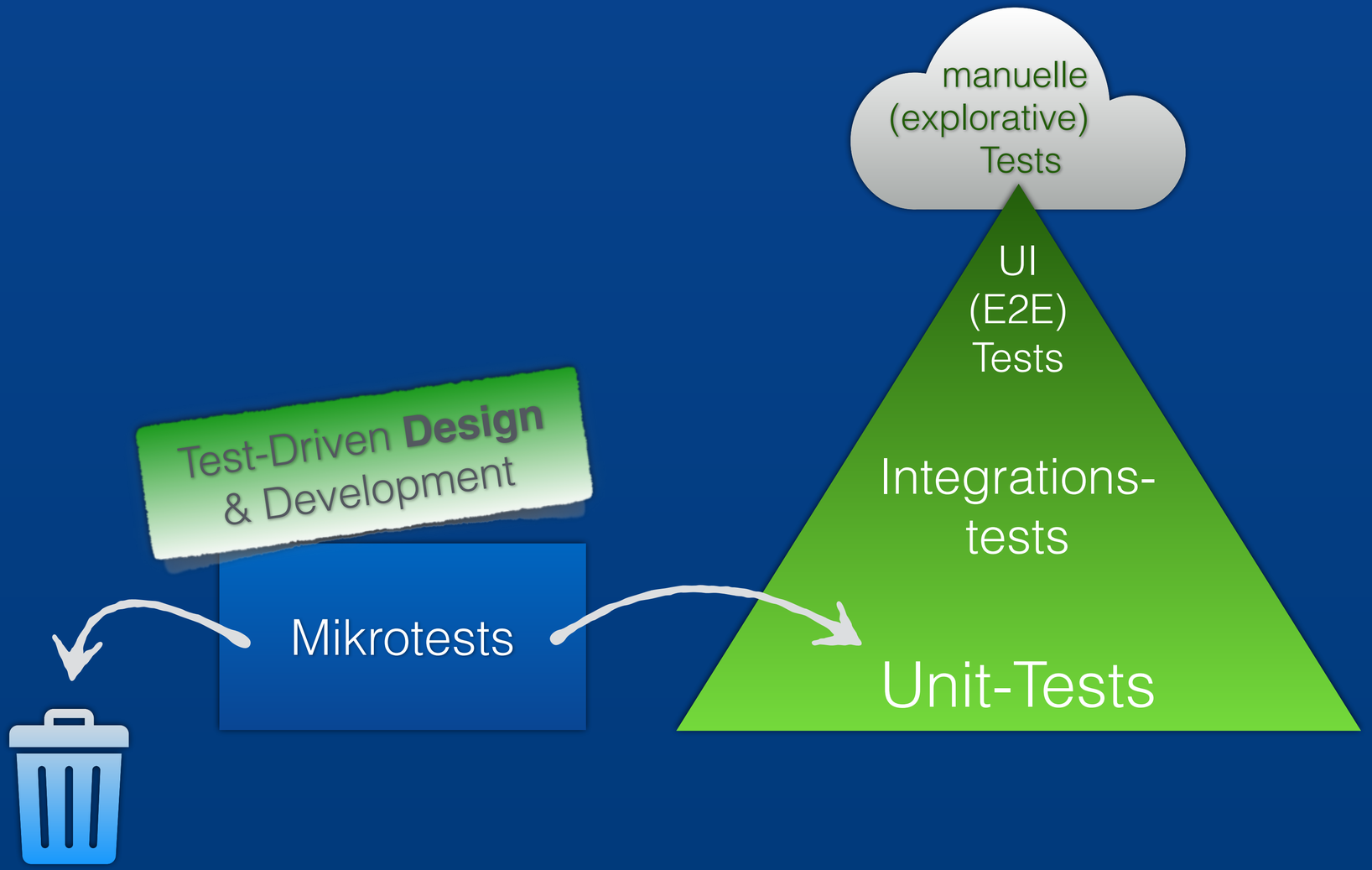
Containerisierung

Ice cream cone emoji inspired by Neal Brooks ;-)

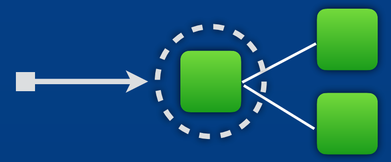
Schnelle Tests

Viele
umfassende
schnelle
stabile
Tests

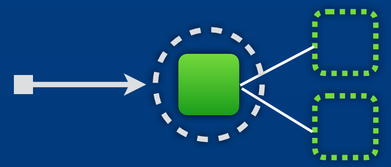
Units oft größer als Mikrotests!



„Sociable Tests“



„Solitary Tests“



https://martinfowler.com/articles/2021-test-shapes.html

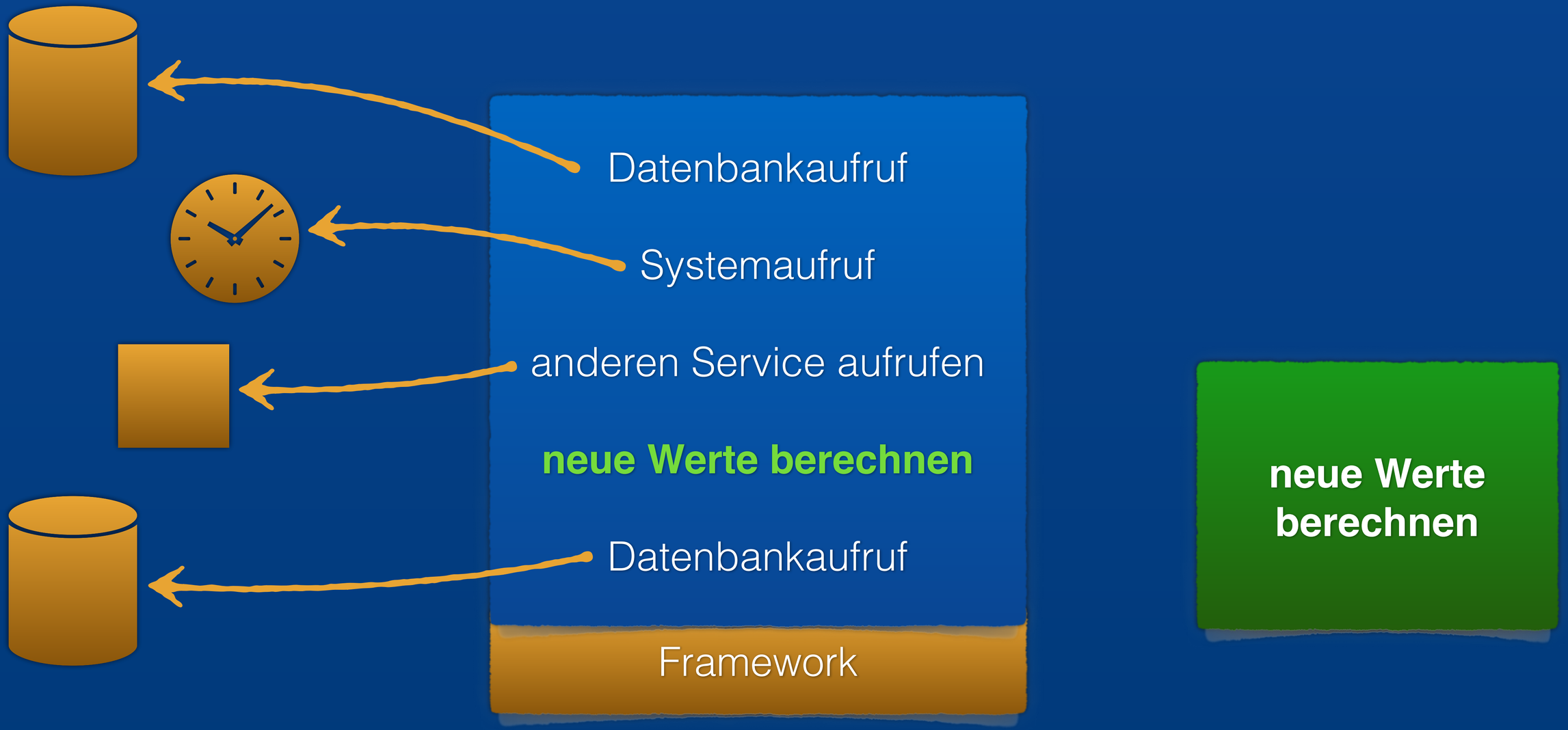
Schnelle Tests*

*) ca. 10 bis 1000 Tests pro Sekunde (Hardware-Stand: 2015)

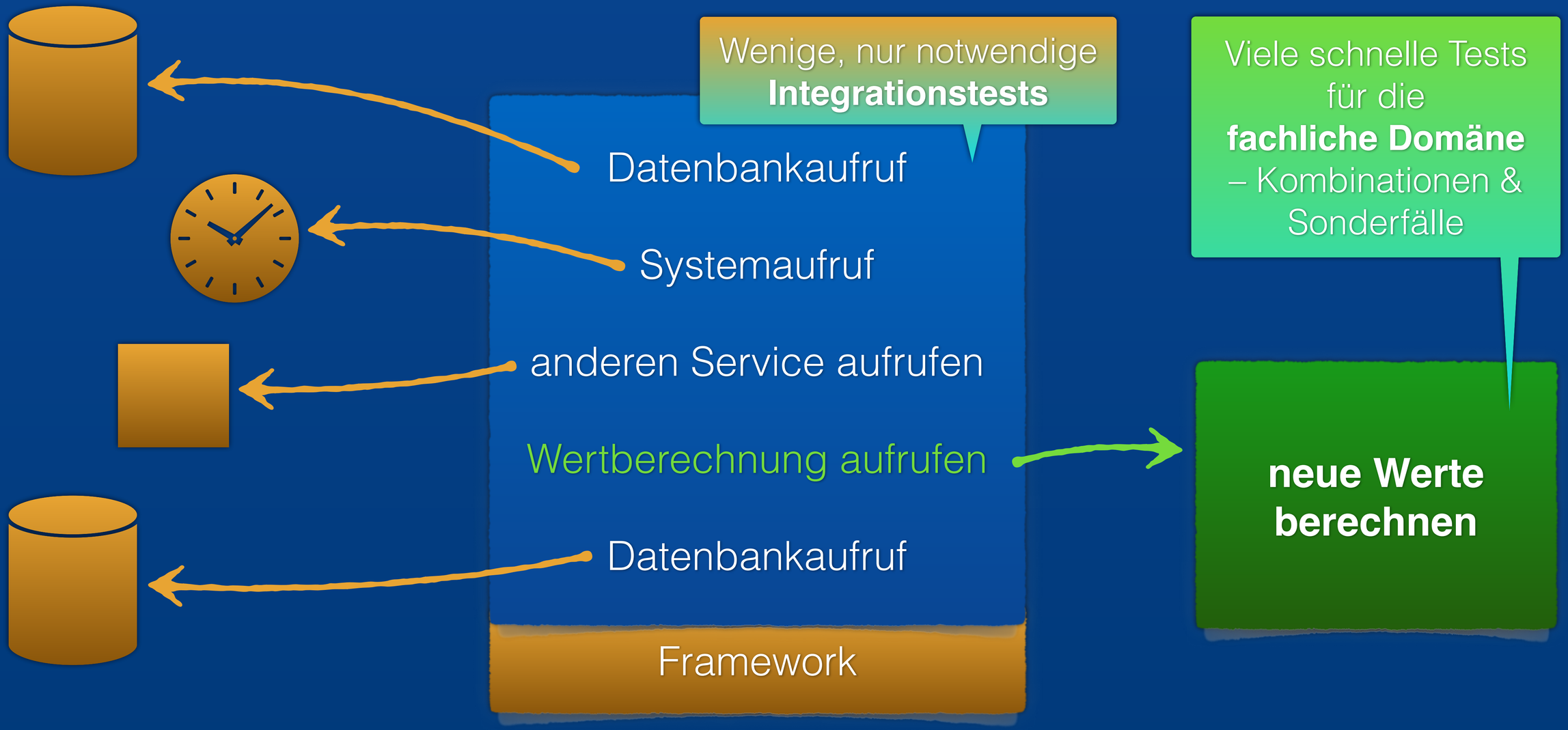
Code und **Architektur** müssen
für **Testbarkeit** designed sein

Abhängigkeiten
machen das schwierig

Abhängigkeiten



Abhängigkeiten



Trennung von
Integrationscode
und
Domänencode

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

Muster (Patterns) & Stile

(Clean-)Code-Design-Patterns

"Integration Operation Segregation Principle" (IOSP)

"Single Layer of Abstraction" (SLA)

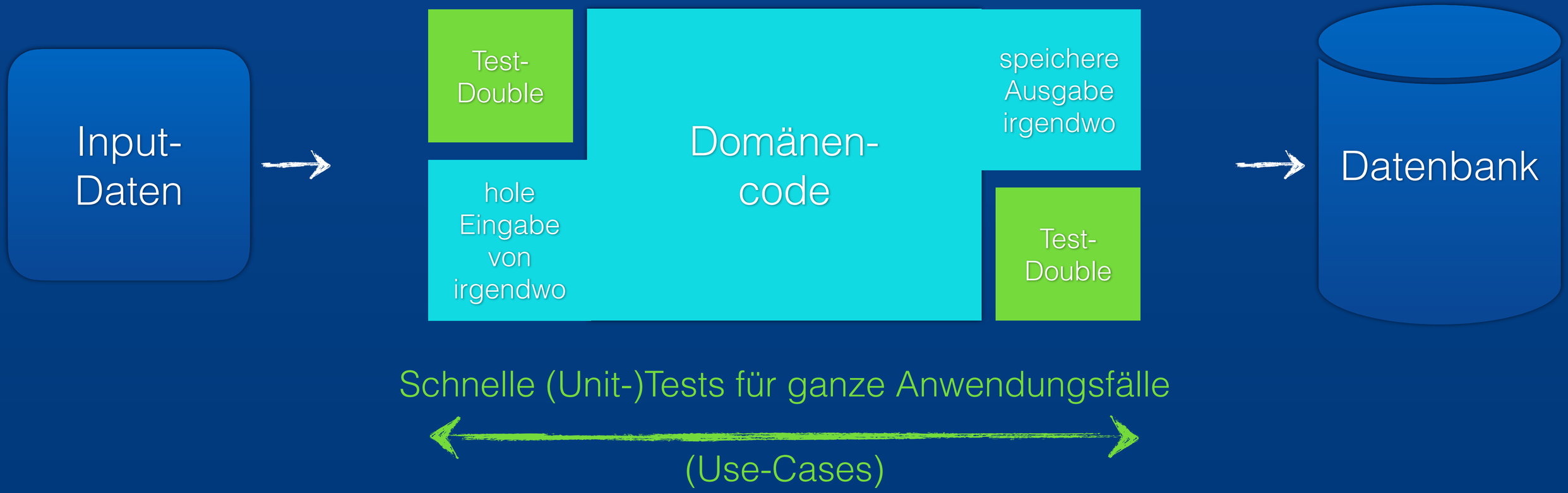
etc.

Vergleichbare **Muster & Stile** für die **Architektur**!

Typisches Beispiel

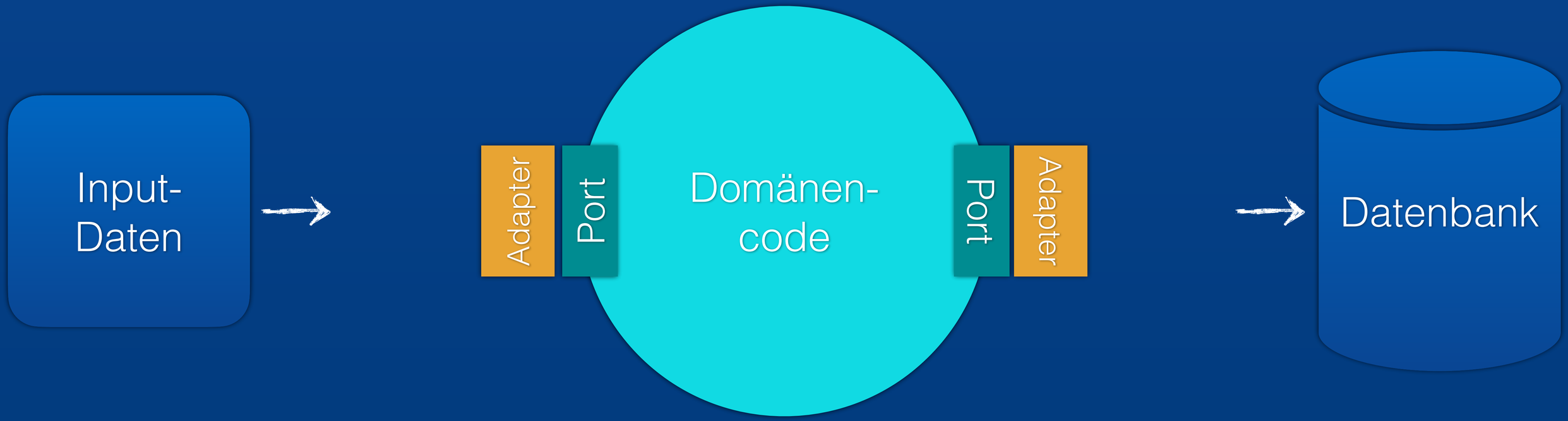
Abhängigkeiten nach außen …

Architekturmuster



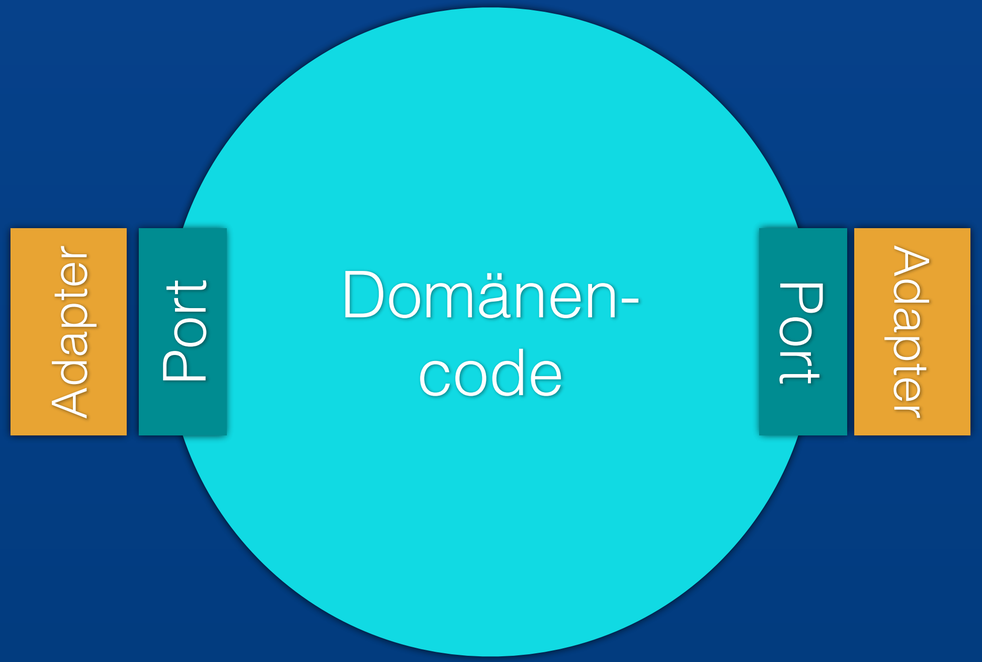
Architekturmuster

Architekturstile

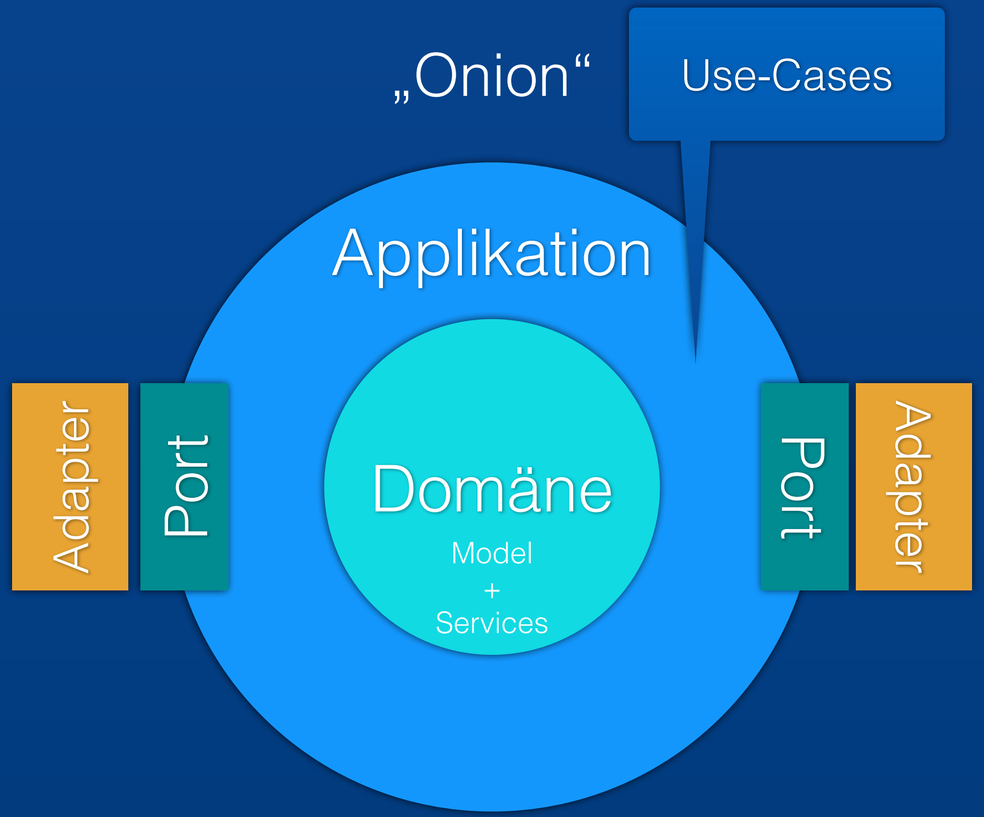


Architekturstile

Ports & Adapters, „Hexagonal“

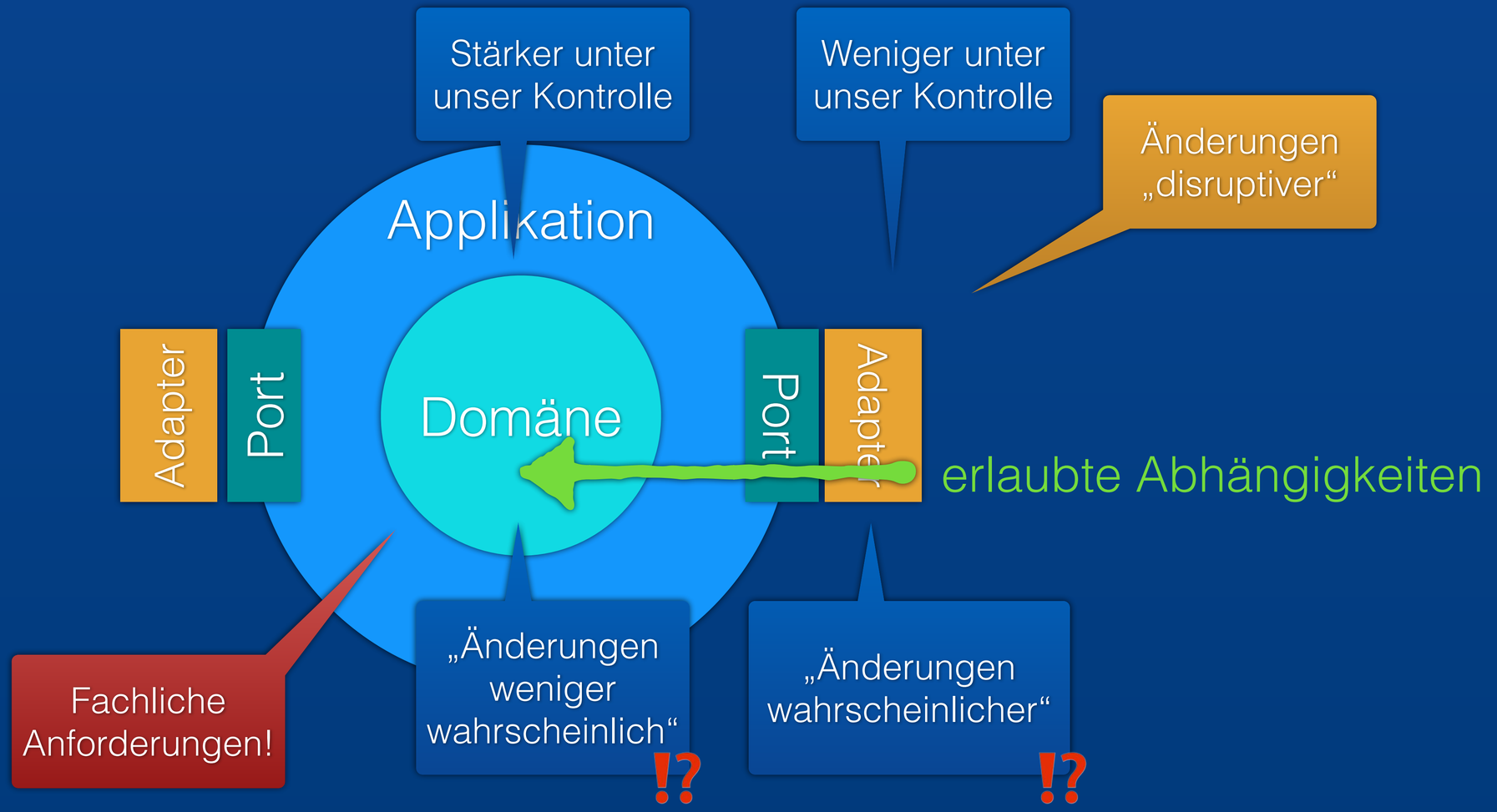


„Onion“

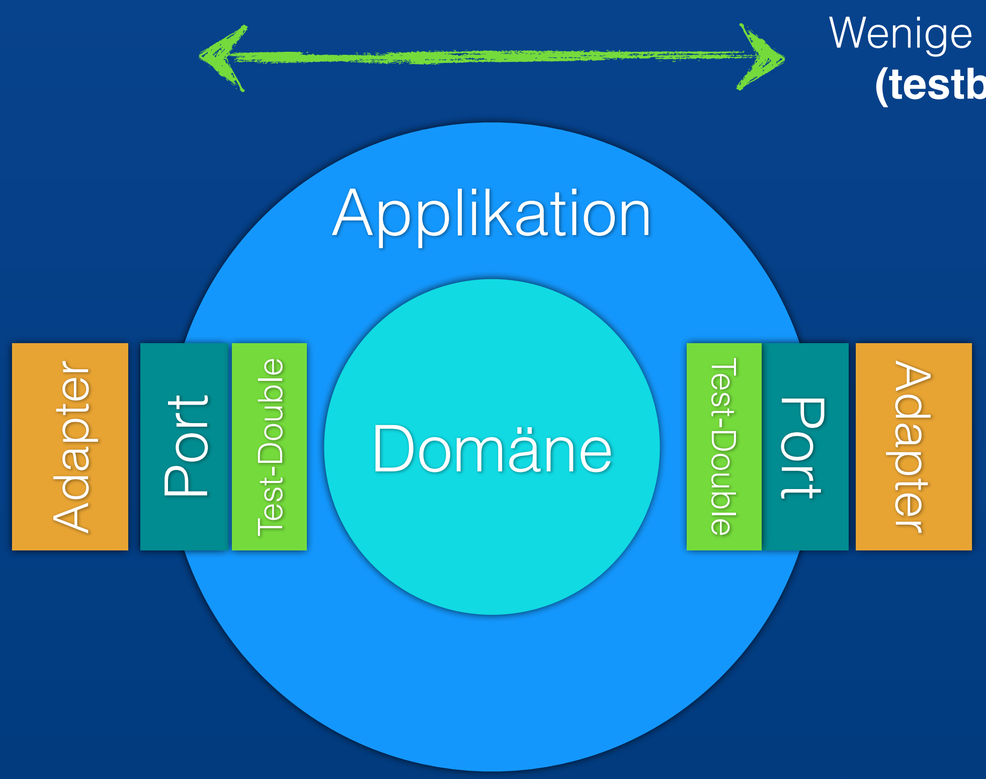


„Clean“ (irgendwo dazwischen)

Statt Schichten: Innen & außen!



Schnelle Tests für ganze Use-Cases



Fazit & Ausblick

Testbarkeit
in Code-Design und Architektur

… **von Anfang an** in unterschiedlichen Rollen mitdenken

… **Abhängigkeiten** auf schwer zu Testendes identifizieren

… **schnelle Tests** ermöglichen

… auch für **größere Units** (> 1 Methode, > 1 Klasse)

… z.B. durch Trennung von **Integrationscode** und **Domänencode**

Tester als Moderatoren

Moderation gefragt – **welche Abstraktionen lohnen sich?**

Was ist **angemessen**? Wieviel Architekturarbeit ist nötig?

Oder können **schnelle Tests** anders erreicht werden?

Tester sind **von Anfang an** mit dabei

Testbarkeit wird **von Anfang an** mitgedacht

Die richtigen Fragen fragen

„**Wie** werden wir das testen?“

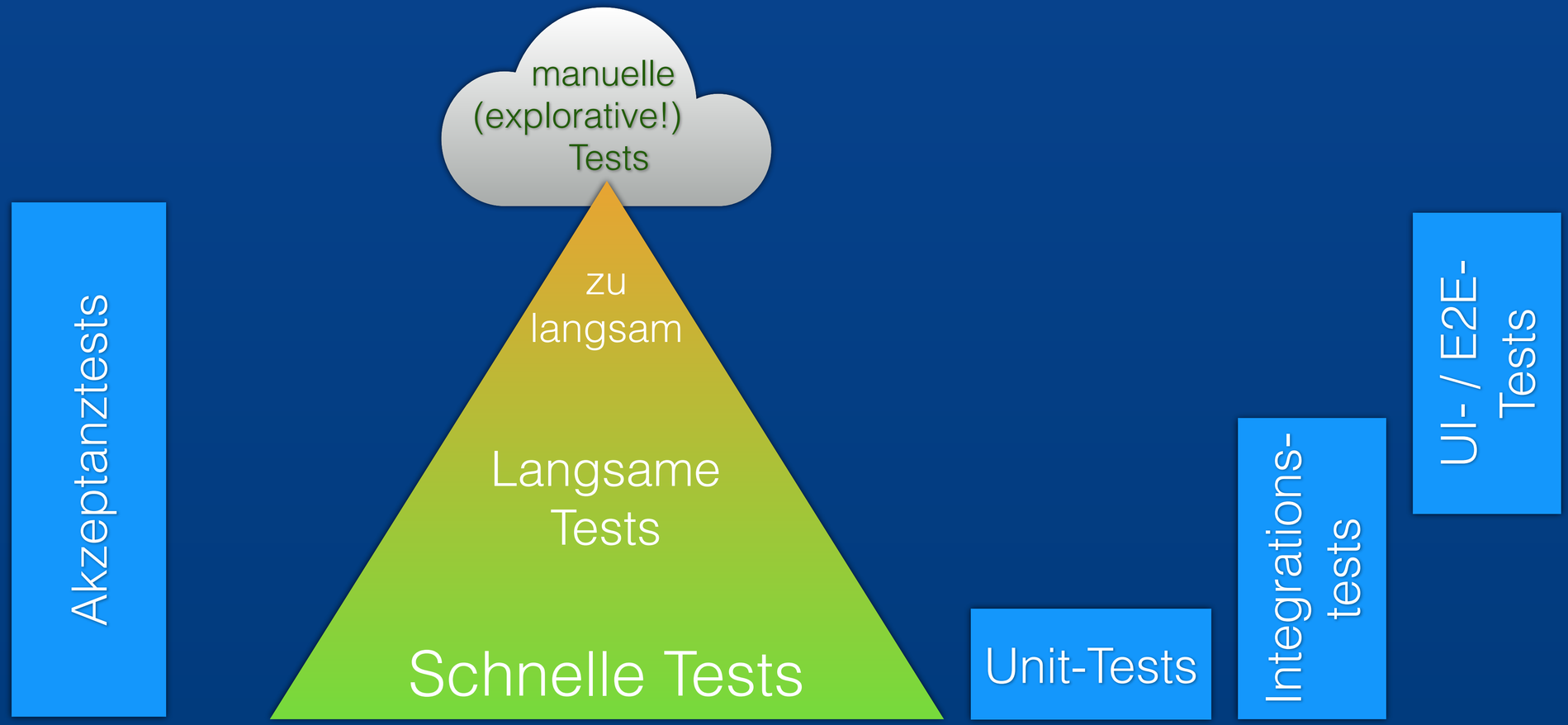
„**Warum** ist das schwer zu testen?“

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?“

Testgeschwindigkeitspyramide?!



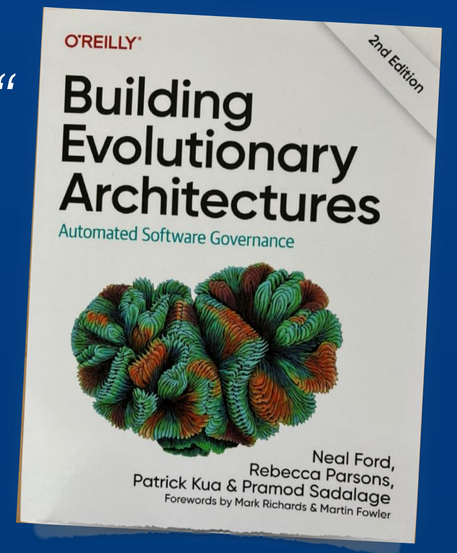
Testbarkeit behalten – wie?

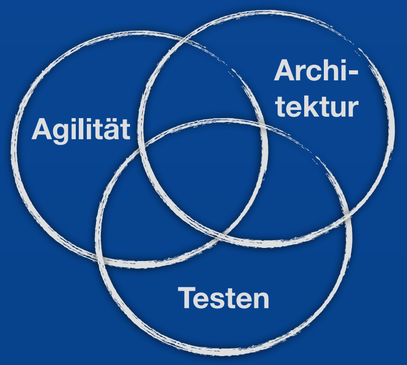
Testet Eure Architektur!

Fitness Functions für die Kernideen der Architektur aufstellen & automatisieren:

„Abhängigkeiten nur von außen nach innen“
„keine Framework-Abhängigkeiten im Domain-Kern“
etc.

ArchUnit Passende Werkzeuge nutzen, z.B. **ArchUnit**





Zusammen arbeiten.

Zusammen coden.

Zusammen testen.

Voneinander lernen.


AGILE TESTING CONDENSED
A Brief Introduction by
Janet Gregory & Lisa Crispin
DAVID FARLEY
MODERN SOFTWARE ENGINEERING
The Addison Wesley Signature Series
A Kent Beck Book
GROWING OBJECT-ORIENTED SOFTWARE, GUIDED BY TESTS
STEVE FREEMAN
NAT PRYCE
WORKING EFFECTIVELY WITH LEGACY CODE
Michael C. Feathers
Unit Testing
Principles, Practices, and Patterns
Vladimir Khorikov
MANNING
O'REILLY
Full Stack Testing
A Practical Guide for
Delivering High Quality Software
Gayathri Mohan
Foreword by Dr. Rebecca Parsons
MANNING
Effective Software Testing
A developer's guide
Maurício Aniche
Forewords by Arie van Deursen and Steve Freeman

Agilität

Testbarkeit

Architektur

Code-Design

Kontinuierliches Testen

Schnelle Tests

Kohäsion

Entkopplung

Abhängigkeiten

Continuous Delivery

Schnelle Feedback-Schleifen

Fragen?

(Q & A)

@thmuch

GERMAN TESTING DAY

Vielen Dank 😊

Die Techniker

www.tk.de/IT

www.sigs.de/autor/thomas.much

@thmuch

"Do not depend on volatile things"
(Robert C. Martin)

**"Make the change easy (this can be hard!),
then make the easy change"**
(Kent Beck)

"Many More Much Smaller Steps"
(GeePaw Hill)