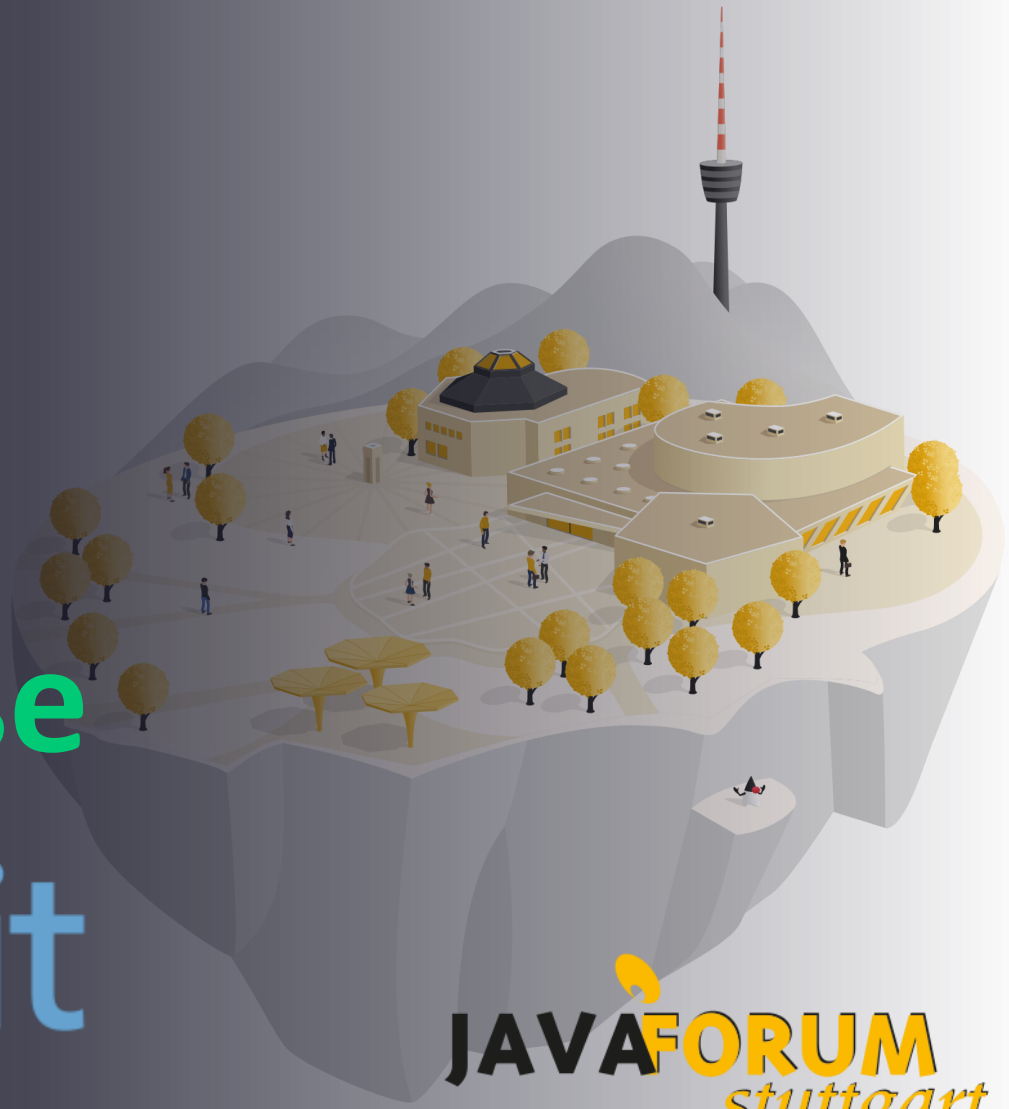
Automatisierte Architekturtests und statische Codeanalyse mit ArchUnit



JAVA FORUM stuttgart

MARTIN LEHMANN, DR. KRISTINE SCHAAL

JAVA FORUM STUTTGART, 13. JULI 2023

ACCSO
ACCELERATED SOLUTIONS

v.3.18
v.3

Martin Lehmann
Accso - Accelerated Solutions GmbH

Cheftechnologe

Martin Lehmann ist Diplom-Informatiker und arbeitet als Cheftechnologe bei der Accso - Accelerated Solutions GmbH. Seit Ende der 90er-Jahre arbeitet er als Softwarearchitekt in Individualentwicklungsprojekten für Kunden verschiedener Branchen. Er interessiert sich besonders für Software-Architektur und Entwicklungsmethodik, gerne auf der Java-Plattform.

martin.lehmann@accso.de @mrtnlhmnn xing.to/mle

Kristine Schaal
Accso - Accelerated Solutions GmbH

Softwarearchitektin

Dr. Kristine Schaal ist als Softwarearchitektin bei der Accso - Accelerated Solutions GmbH tätig. Sie arbeitet seit mehr als 20 Jahren in der Softwareentwicklung und ist in Projekten der Individualentwicklung für Kunden verschiedener Branchen unterwegs, technisch überwiegend im Java-Umfeld.

kristine.schaal@accso.de @krschaal www.xing.com/profile/Kristine_Schaal

https://www.archunit.org

Open-Source

Apache License 2.0

für Java und andere JVM-Sprachen
V1.0.1 vom 21. November 2022

https://www.archunit.org/
https://github.com/TNG/ArchUnit

für DOT.NET und C#
V0.10.5 vom 10. November 2022

https://github.com/TNG/ArchUnitNET

- **Architektur testen mit ArchUnit**
- Statische Code-Analyse mit ArchUnit
- Fachliche Säulen, technische Schichten, Zyklen
- Integration mit anderen Tools
- Fazit

Warum automatisierte Architektur-Tests mit ArchUnit?

Schnelles Feedback für
alle Entwickler:innen

läuft einfach in jedem
Build mit

Dokumentiert
Architekturentscheidungen,
macht sie explizit

automatisiert und/statt (nur) Code-Reviews

Programmieren statt deklarieren –
Man arbeitet im gleichen Code, in der
gleichen Programmiersprache (Turing-vollständig)

Refactoring schließt auch die Tests ein

Schnell erlernbar und einführbar,
gut lesbar und wartbar

Ausdrucksstarke und typsichere API

damit mächtiger als Checkstyle, PMD,
Findbugs, JDepend, Sonar*,
Structure101, jqAssistant

Wie schreibt man ArchUnit-Tests?

Regeln schreiben

- Regeln prüfen einen Aspekt der Architektur.
- Regeln definiert man über `ArchRule`, bzw. spezifische Subklassen.
- API vereinfacht das Erstellen der Regeln.

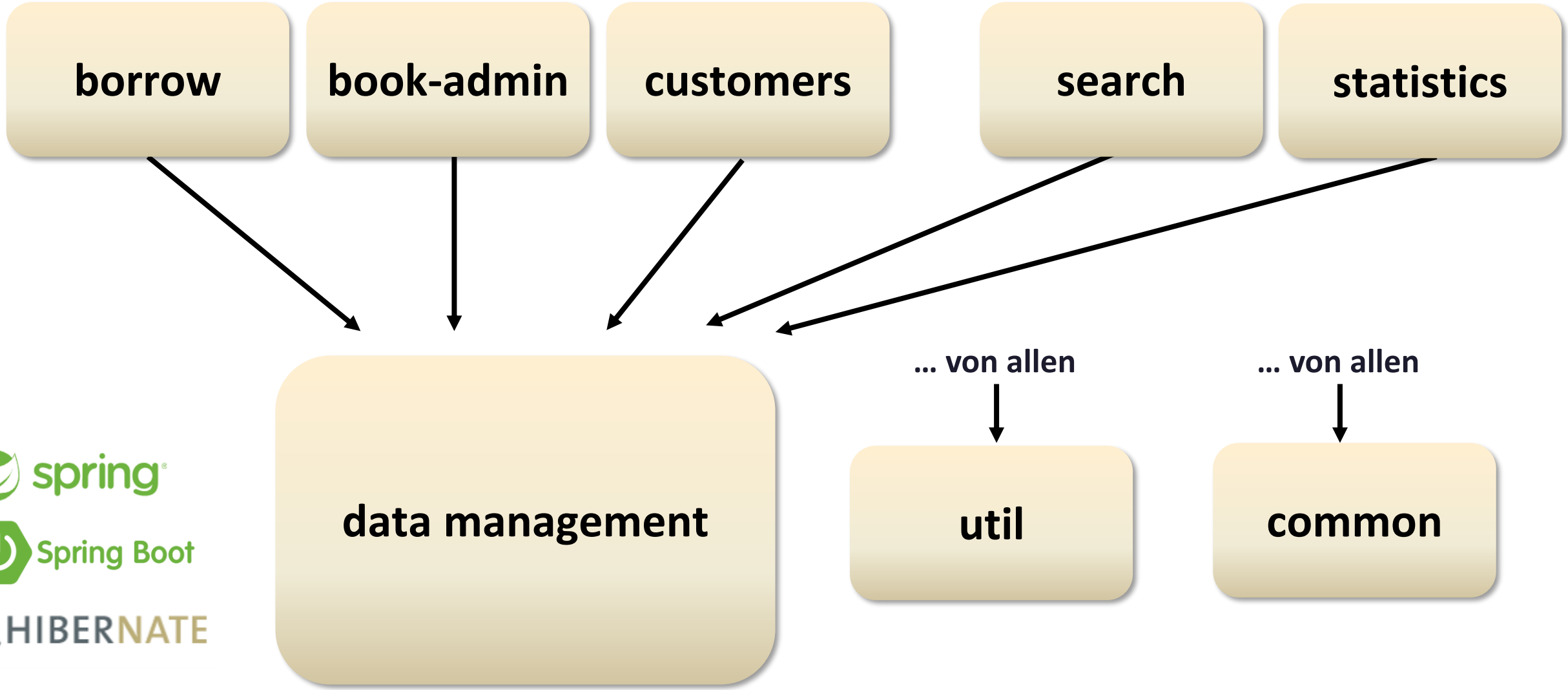
Was kann man prüfen?

- Package Dependency
- Class Dependency
- Class and Package Containment
- Inheritance
- Annotation
- Layer
- Cycle
- …

Tests ausführen

- ArchUnit-Tests können als JUnit4- oder JUnit5-Tests geschrieben und ausgeführt werden.
- Sie können im normalen Build-Zyklus mit-laufen.
- Wenn die Architektur-prüfung fehlschlägt, wird eine Exception geworfen → Der Test schlägt fehl.

Fachliches Bücherei-Beispiel „library“



spring
Spring Boot
HIBERNATE
Maven

LiveCoding

Beispiel 1

Check von
Namenskonventionen

Beispiel 2

Check von
Abhängigkeiten

Beispiel 3

Sind alle Daos
auch JpaRepositories?

Sind nur Daos
JpaRepositories?

https://github.com/accso/static-code-analysis-archunit

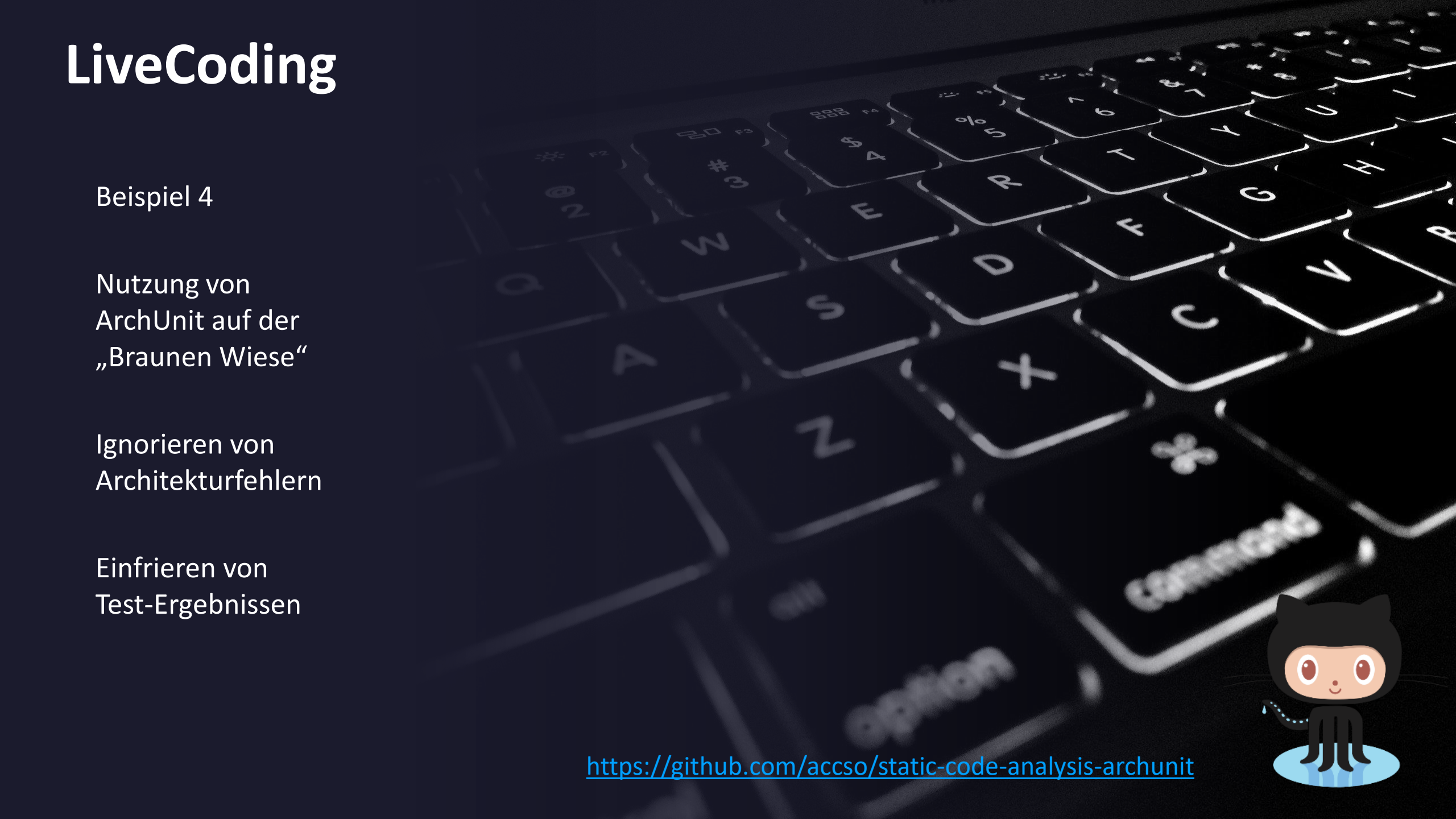
LiveCoding

Beispiel 4

Nutzung von
ArchUnit auf der
„Braunen Wiese“

Ignorieren von
Architekturfehlern

Einfrieren von
Test-Ergebnissen



https://github.com/accso/static-code-analysis-archunit

- Architektur testen mit ArchUnit
- **Statische Code-Analyse mit ArchUnit**
- Fachliche Säulen, technische Schichten, Zyklen
- Integration mit anderen Tools
- Fazit

LiveCoding

Nochmal Beispiel 3

diesmal mit
Fokus auf statischer
Code-Analyse

Beispiel 5

Finde transitive
Abhängigkeiten
für eine Klasse.

Relevant z.B. für
Migrations-
planung und
-schätzung

https://github.com/accso/static-code-analysis-archunit

Dependency

Represents a dependency of one Java class on another Java class. Such a dependency can occur by either of the following:

- a class accesses a field of another class
- a class calls a method of another class
- a class calls a constructor of another class
- a class inherits from another class (which is in fact a special case of constructor call)
- a class implements an interface
- a class has a field with type of another class
- a class has a method/constructor with parameter/return type of another class

Note that a `Dependency` will by definition never be a self-reference, i.e. `origin` will never be equal to `target`.

```
public class Dependency implements HasDescription, Comparable<Dependency>, HasSourceCodeLocation {
```

ArchUnit basiert auf statischer Codeanalyse

Byte Code

- ArchUnit prüft Byte Code, nicht den Source Code
- ArchUnit baut ein statisches Abhängigkeits-modell auf.

Abhängigkeiten

- Aus- und eingehend
- Vererbung, Interfaces, Generics, Annotationen

ArchUnit nutzt diese Informationen:

- Konstruktoren
- Feldern
- Annotationen
- Typen

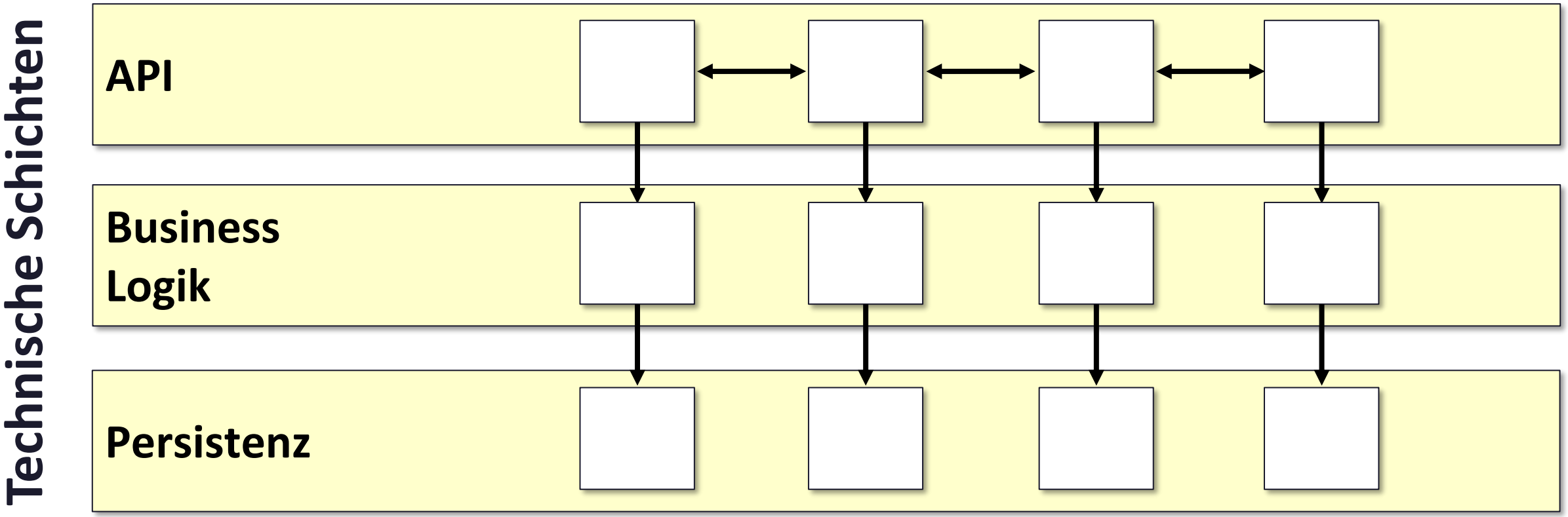
Modell

- `ClassFileImporter` scannt den Classpath.
- Einschränkung auf eigene Packages möglich. Mit/ohne Test

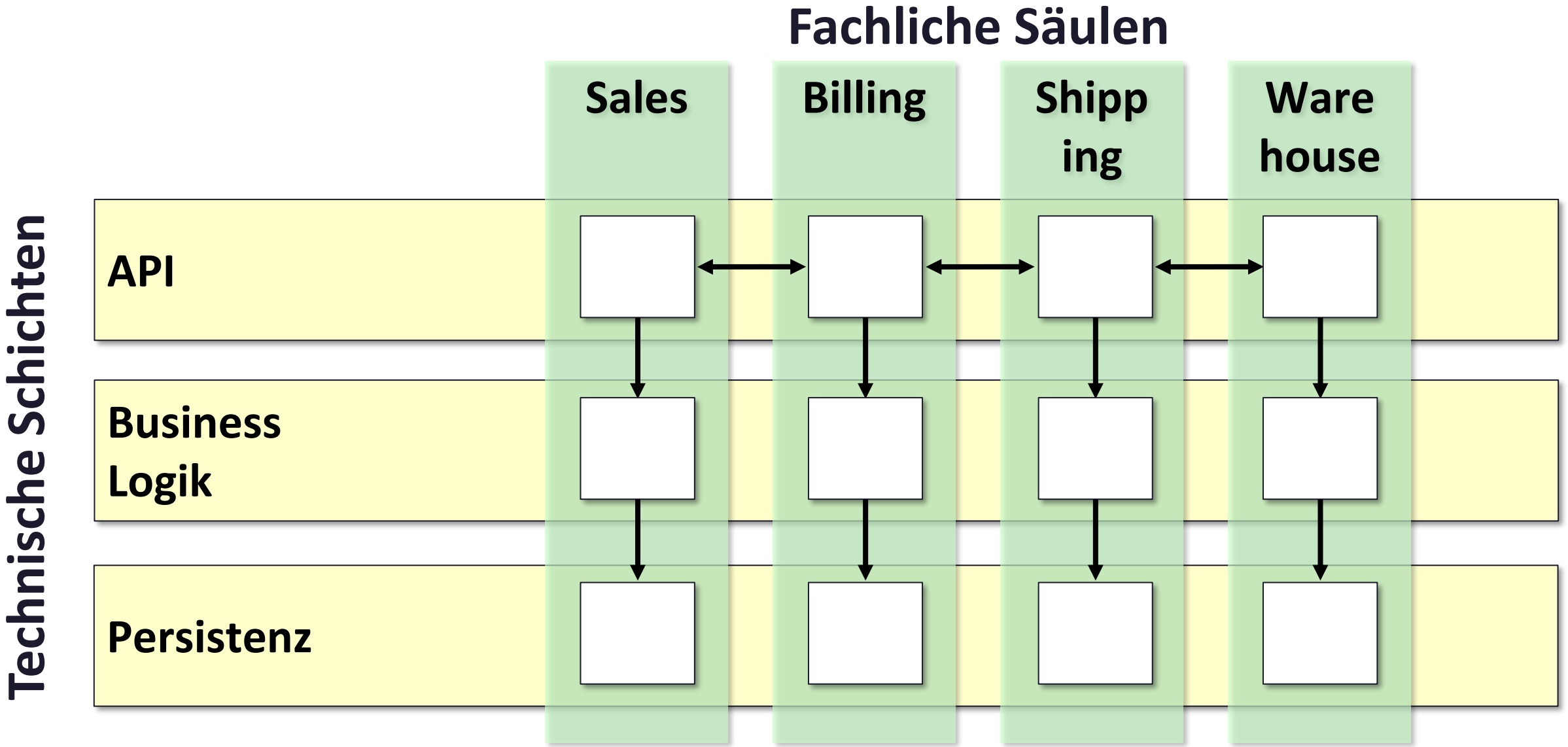
- `JavaClass`
- `JavaClasses`
- `Dependency`
- `JavaAccess`

- Architektur testen mit ArchUnit
- Statische Code-Analyse mit ArchUnit
- **Fachliche Säulen, technische Schichten, Zyklen**
- Integration mit anderen Tools
- Fazit

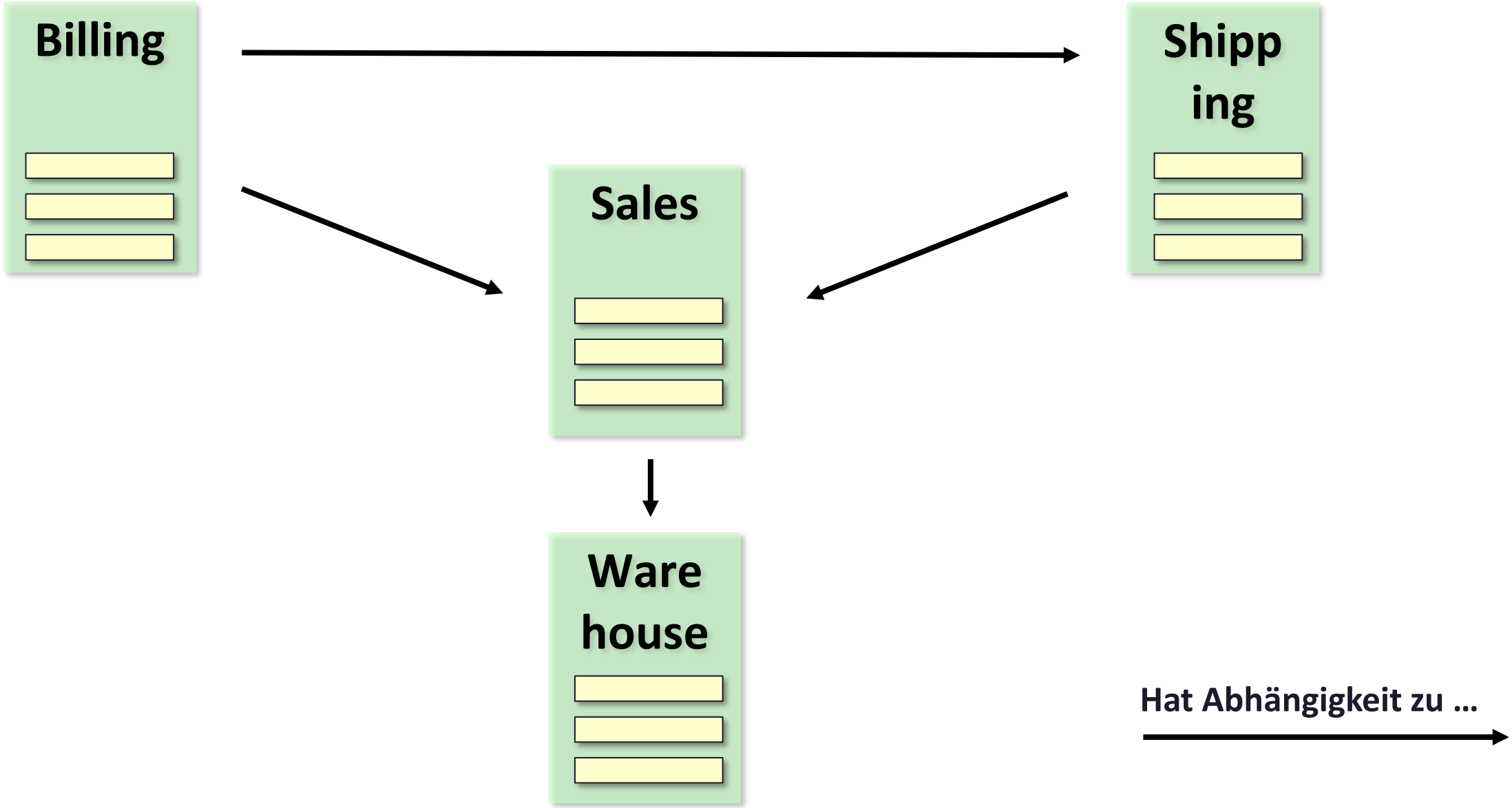
Komponentenschnitt in Säulen und Schichten



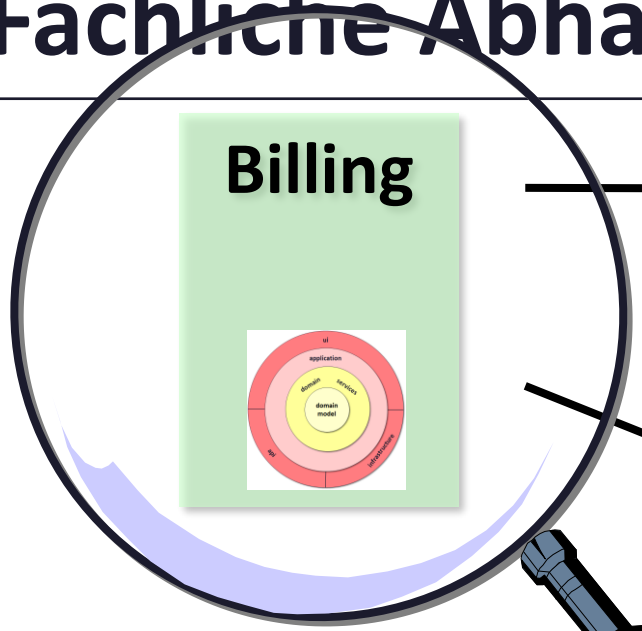
Komponentenschnitt in Säulen und Schichten



Fachliche Abhängigkeiten

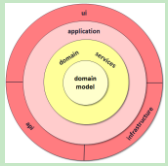


Fachliche Abhängigkeiten

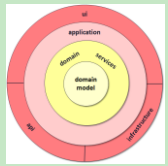


Billing

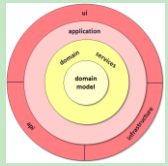
Shipping



Sales

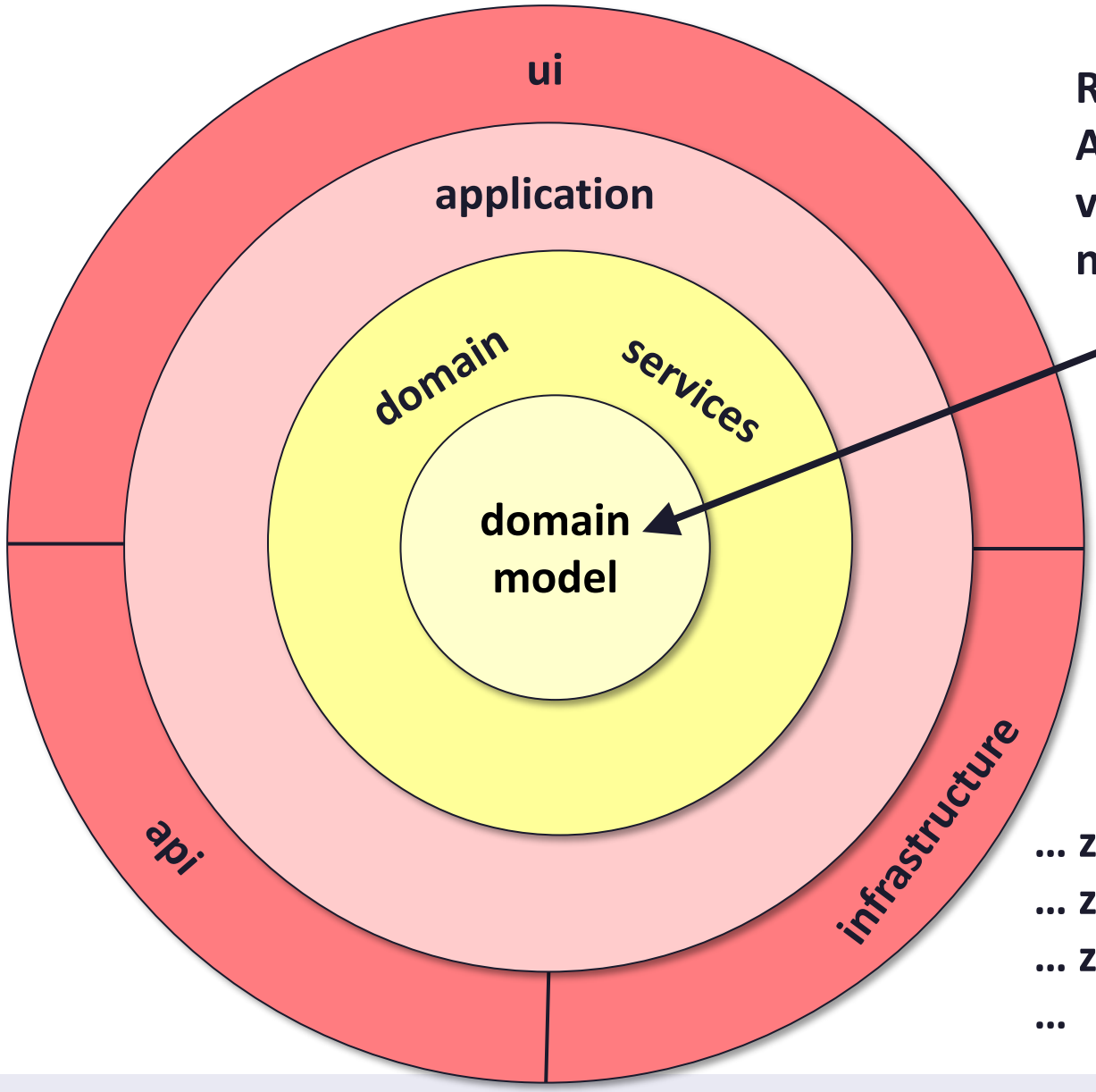


Warehouse



Hat Abhängigkeit zu …

Onion-Architektur



Richtung der Abhängigkeiten von außen nach innen

… zu Persistenz
… zu Messaging
… zu Monitoring
…

Billing

- de.accso.ecommerce
  - billing
    - api
    - core
      - application
        - services
          - BillingApp
          - BillingConfig
          - BillingEventConsumer
          - BillingEventProducer
          - BillingMessaging
          - BillingMetricsProvider
          - BillRepository
          - PaymentRepository
        - domain
          - model
          - services
      - infrastructure
        - config
        - messaging
        - monitoring
        - persistence
    - ui

LiveCoding

Beispiel 6

Zyklen?

Check auf
erlaubte
Abhängigkeiten

… zwischen den
fachlichen
Komponenten

… auf Onion-
Architektur

https://github.com/accso/static-code-analysis-archunit

- Architektur testen mit ArchUnit
- Statische Code-Analyse mit ArchUnit
- Fachliche Säulen, technische Schichten, Zyklen
- **Integration mit anderen Tools**
- Fazit

My Favorites | All

Search by project name or key

Create Project

3 projects

Perspective: Overall Status

Sort by: Name

Filters

Quality Gate

- Passed 3
- Failed 0

Reliability (Bugs)

- A 2
- B 0
- C 1
- D 0
- E 0

Security (Vulnerabilities)

- A 3
- B 0
- C 0
- D 0
- E 0

Security Review (Security Hotspots)

- A ≥ 80% 2
- B 70% - 80% 0
- C 50% - 70% 0
- D 30% - 50% 0
- E < 30% 1

Maintainability (Code Smells)

- A 3
- B 0
- C 0
- D 0
- E 0

Coverage

- ≥ 80% 1
- 70% - 80% 0
- 50% - 70% 0
- 30% - 50% 0
- < 30% 2
- No data 0

ecommerce-example — Passed

Last analysis: 5 minutes ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
0 A	0 A	– A	52 A	0.0%	0.0%	526 XS Java, XML

library-example — Passed

Last analysis: 3 minutes ago

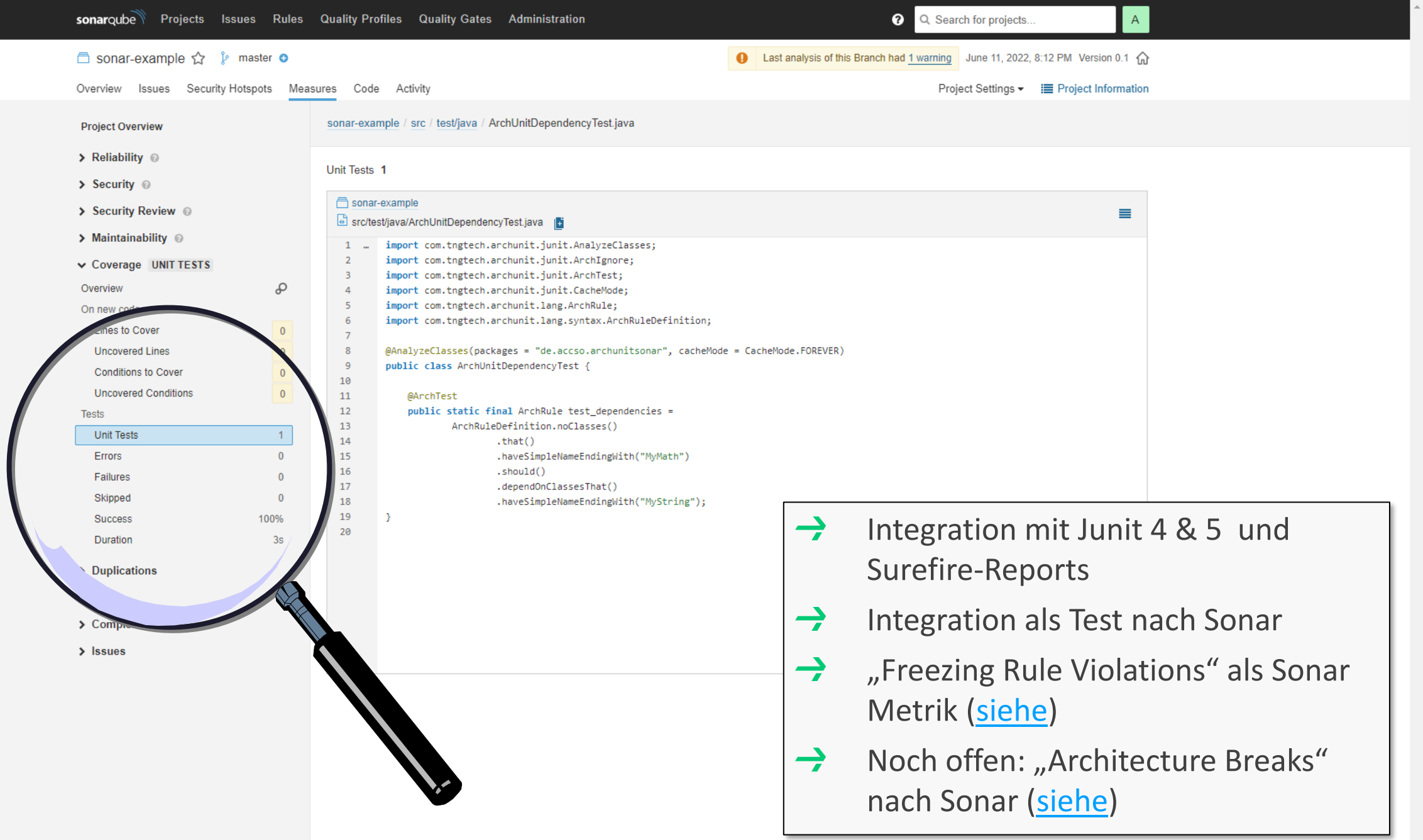
Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
1 C	0 A	0.0% E	37 A	0.0%	0.0%	1.4k S Java, XML

sonar-example — Passed

Last analysis: 31 seconds ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
0 A	0 A	– A	4 A	100%	0.0%	74 XS XML, Java

3 of 3 shown



- Integration mit Junit 4 & 5 und Surefire-Reports
- Integration als Test nach Sonar
- „Freezing Rule Violations“ als Sonar Metrik (siehe)
- Noch offen: „Architecture Breaks“ nach Sonar (siehe)

- Architektur testen mit ArchUnit
- Statische Code-Analyse mit ArchUnit
- Fachliche Säulen, technische Schichten, Zyklen
- Integration mit anderen Tools
- **Fazit**

Fazit – einfach mal einsetzen!

- Leichtgewichtig einführbar
- Regeln per API sehr einfach zu schreiben.
- Refactoring möglich (außer Package-Namen als String)
- Filter über Importer, Ignore, Freeze
- Tests über JUnit in jeden Build integrierbar
- Wird stetig weiterentwickelt (Beispiel: Type Erasure, Generics)
- Gute Basis für eigene statische Code-Analyse-Werkzeuge

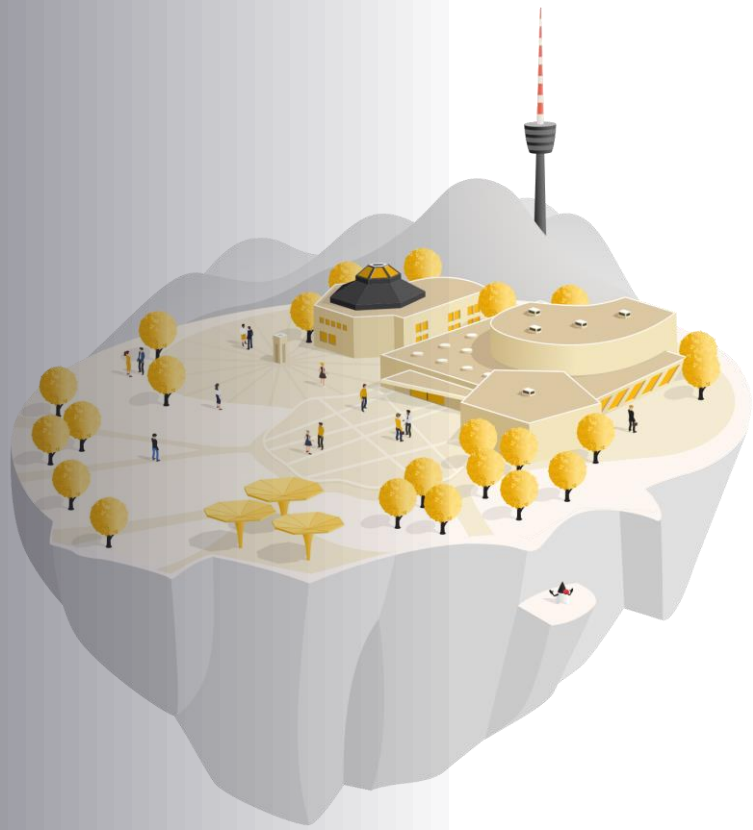
- Semantik von Depend, Access etc. nicht immer intuitiv
- Strings statt Typen bei Klassen, Packages, Wildcards
- Ungewohnte Wildcard-Notation

- Architekturtests sind Tests für die Code-Architektur (nicht mehr, aber auch nicht weniger).
- Kann andere Tools wie Checkstyle, Sonar & Co ergänzen, in Teilen ersetzen.

https://accso.de/publikationen/

https://speakerdeck.com/mrtnlhmnn

https://github.com/accso/static-code-analysis-archunit



@mrtnlhmnn
@krschaal
@accso

ACCSO
ACCELERATED SOLUTIONS

Accso – Accelerated Solutions GmbH
T | +49 6151 13029-0
E | info@accso.de
@ | www.accso.de

Hilpertstraße 12 | 64295 Darmstadt
Rahmhofstraße 2 | 60313 Frankfurt a. M.
Im Mediapark 6a | 50670 Köln
Balanstraße 55 | 81541 München
Clocktower, 302 | CV&A Waterfront, Cape Town 8002, ZA