Modul 11 - ArchUnit

Architekturregeln als ausführbare Tests

CC BY-NC-SA 4.0, Alexander Erben

Lernziele

- ArchUnit als Werkzeug für automatisierte Architektur-Governance kennen
- Architekturregeln als JUnit-Tests schreiben
- Clean-Architecture-Regeln mit ArchUnit durchsetzen
- Namenskonventionen und Annotation-Regeln formulieren
- Onion Architecture als vordefinierte Form nutzen
- Architektur-Baseline für bestehenden Code einsetzen

CC BY-NC-SA 4.0, Alexander Erben

Warum automatisierte Architektur-Governance?

Das Problem

```
Woche 1:  Architektur sauber, alle halten sich dran        [OK]
Woche 4:  "Nur diese eine Abkürzung..."                    [WARNUNG]
Woche 12: Domain importiert JPA, Controller hat Logik      [FEHLER]
Woche 24: "Wir müssen die Architektur neu aufsetzen"       [GAME OVER]
```

CC BY-NC-SA 4.0, Alexander Erben

Die Lösung

- Architekturregeln als ausführbare Tests formulieren
- Laufen bei jedem Build - Verstöße brechen die Pipeline
- Keine zusätzliche Infrastruktur nötig - normaler JUnit-Test

> Architekturregeln, die nicht automatisch geprüft werden, werden früher oder später gebrochen.

CC BY-NC-SA 4.0, Alexander Erben

Was ist ArchUnit?

- Java-Bibliothek von TNG Technology Consulting
- Architekturregeln als JUnit 5 Tests formuliert
- Prüft: Paketabhängigkeiten, Annotationen, Vererbung, Namenskonventionen
- Läuft als normaler Unit-Test - `mvn test` genügt
- Open Source, aktiv gepflegt (aktuell 1.3.x)

Maven Dependency

```
<dependency>
    <groupId>com.tngtech.archunit</groupId>
    <artifactId>archunit-junit5</artifactId>
    <version>1.3.0</version>
    <scope>test</scope>
</dependency>
```

CC BY-NC-SA 4.0, Alexander Erben

Grundlegende API

```
@AnalyzeClasses(packages = "de.realestate")
class ArchitectureRulesTest {

    @ArchTest
    static final ArchRule domain_has_no_spring_imports =
        noClasses()
            .that().resideInAPackage("..domain..")
            .should().dependOnClassesThat()
                .resideInAnyPackage(
                    "org.springframework..",
                    "jakarta.persistence..",
                    "jakarta.transaction..")
            .as("Domain must not have Spring/JPA dependencies");
}
```

- `@AnalyzeClasses` - welche Packages werden analysiert
- `@ArchTest` statt `@Test` - ArchUnit erkennt die Regel automatisch
- Fluent API: `classes().that()...should()...`

CC BY-NC-SA 4.0, Alexander Erben

Die API im Überblick

```
ArchRuleDefinition
   ├── classes()        → Klassen selektieren
   ├── noClasses()      → Negativprüfung ("keine Klasse darf...")
   ├── methods()        → Methoden selektieren
   └── fields()         → Felder selektieren

Selektoren (.that())
   ├── resideInAPackage("..domain..")
   ├── areAnnotatedWith(Service.class)
   ├── haveSimpleNameEndingWith("Controller")
   ├── implement(SomeInterface.class)
   └── areNotInterfaces()

Bedingungen (.should())
   ├── onlyDependOnClassesThat(...)
   ├── notBeAnnotatedWith(...)
   ├── onlyBeAccessed().byAnyPackage(...)
   ├── haveSimpleNameEndingWith(...)
   └── resideInAPackage(...)
```

CC BY-NC-SA 4.0, Alexander Erben

Regel 1: Domain ist Framework-frei

```
@ArchTest
static final ArchRule domain_is_framework_free =
    noClasses()
        .that().resideInAPackage("..domain..")
        .should().dependOnClassesThat()
            .resideInAnyPackage(
                "org.springframework..",
                "jakarta.persistence..",
                "jakarta.transaction..",
                "com.fasterxml.jackson..")
        .as("Domain must not have framework dependencies");
```

- Sichert die Dependency Rule der Clean Architecture ab
- Domain ist der innerste Ring - kennt nur `java.*` und sich selbst
- Verstöße brechen den Build sofort

CC BY-NC-SA 4.0, Alexander Erben

Regel 2: Domain kennt keine äußeren Ringe

```
@ArchTest
static final ArchRule domain_does_not_know_outer_rings =
    noClasses()
        .that().resideInAPackage("..domain..")
        .should().dependOnClassesThat()
            .resideInAnyPackage(
                "..application..",
                "..infrastructure..",
                "..adapter..")
        .as("Domain must not access outer rings");
```

```
@ArchTest
static final ArchRule application_does_not_know_infrastructure =
    noClasses()
        .that().resideInAPackage("..application..")
        .should().dependOnClassesThat()
            .resideInAnyPackage(
                "..infrastructure..",
                "..adapter..")
        .as("Application must not access Infrastructure/Adapter");
```

CC BY-NC-SA 4.0, Alexander Erben

Regel 3: Annotationen am richtigen Ort

```
@ArchTest
static final ArchRule rest_controller_only_in_adapter_web =
    noClasses()
        .that().resideOutsideOfPackage("..adapter.web..")
        .should().beAnnotatedWith(RestController.class)
        .as("@RestController only allowed in adapter.web");

@ArchTest
static final ArchRule entity_only_in_infrastructure =
    noClasses()
        .that().resideOutsideOfPackage("..infrastructure..")
        .should().beAnnotatedWith(
            jakarta.persistence.Entity.class)
        .as("@Entity only allowed in infrastructure");

@ArchTest
static final ArchRule transactional_only_in_application =
    noClasses()
        .that().resideOutsideOfPackage("..application..")
        .should().beAnnotatedWith(Transactional.class)
        .as("@Transactional only allowed in application");
```

CC BY-NC-SA 4.0, Alexander Erben

Regel 4: Namenskonventionen

```
@ArchTest
static final ArchRule controllers_named_controller =
    classes()
        .that().areAnnotatedWith(RestController.class)
        .should().haveSimpleNameEndingWith("Controller")
        .as("REST controllers should end with 'Controller'");

@ArchTest
static final ArchRule request_dtos_named_request =
    classes()
        .that().resideInAPackage("..adapter.web..")
        .and().haveSimpleNameEndingWith("Request")
        .should().beRecords()
        .as("Request DTOs should be records");

@ArchTest
static final ArchRule services_implement_port =
    classes()
        .that().resideInAPackage("..application.service..")
        .and().areAnnotatedWith(Service.class)
        .should().implement(
            resideInAPackage("..application.port.."))
        .as("Application services should implement a port");
```

CC BY-NC-SA 4.0, Alexander Erben

Vordefinierte Architekturform: Onion Architecture

```
@ArchTest
static final ArchRule onion_architecture =
    Architectures.onionArchitecture()
        .domainModels("..domain.model..", "..domain.event..")
        .domainServices("..domain.port..")
        .applicationServices("..application..")
        .adapter("web", "..adapter.web..")
        .adapter("persistence", "..infrastructure.persistence..")
        .adapter("config", "..infrastructure.config..");
```

- ArchUnit bietet vordefinierte Architekturformen an
- `onionArchitecture()` prüft automatisch alle Abhängigkeitsregeln
- Weniger Code als einzelne Regeln - aber weniger granulare Fehlermeldungen
- Alternative: `layeredArchitecture()` für klassische Schichtarchitekturen

CC BY-NC-SA 4.0, Alexander Erben

Layered Architecture - Alternative

```
@ArchTest
static final ArchRule layered_architecture =
    Architectures.layeredArchitecture()
        .consideringOnlyDependenciesInLayers()
        .layer("Adapter").definedBy("..adapter..")
        .layer("Application").definedBy("..application..")
        .layer("Infrastructure").definedBy("..infrastructure..")
        .layer("Domain").definedBy("..domain..")
        .whereLayer("Adapter")
            .mayNotBeAccessedByAnyLayer()
        .whereLayer("Infrastructure")
            .mayNotBeAccessedByAnyLayer()
        .whereLayer("Application")
            .mayOnlyBeAccessedByLayers("Adapter")
        .whereLayer("Domain")
            .mayOnlyBeAccessedByLayers(
                "Application", "Infrastructure", "Adapter");
```

> `onionArchitecture()` ist passender für Clean Architecture,
> `layeredArchitecture()` für traditionelle Schichten.

CC BY-NC-SA 4.0, Alexander Erben

Cross-BC-Regeln: Bounded Context Isolation

```
@ArchTest
static final ArchRule brokerage_does_not_access_acquisition_domain =
    noClasses()
        .that().resideInAPackage("..brokerage.domain..")
        .should().dependOnClassesThat()
            .resideInAPackage("..acquisition.domain..")
        .as("Brokerage domain must not directly access "
            + "Acquisition domain");

@ArchTest
static final ArchRule bcs_communicate_only_via_events =
    slices().matching("de.realestate.(*).domain..")
        .should().notDependOnEachOther()
        .as("Domain layers of different BCs "
            + "must not depend on each other");
```

- Stellt sicher, dass Bounded Contexts isoliert bleiben
- Kommunikation zwischen BCs nur über Events oder definierte APIs
- `slices()` prüft alle BC-Kombinationen auf einmal

CC BY-NC-SA 4.0, Alexander Erben

Architektur-Baseline: Legacy-Code schrittweise verbessern

Problem: 47 bestehende Verstöße - Build bricht sofort

```
// FreezingArchRule: "freeze" existing violations
@ArchTest
static final ArchRule domain_framework_free =
    FreezingArchRule.freeze(
        noClasses()
            .that().resideInAPackage("..domain..")
            .should().dependOnClassesThat()
                .resideInAPackage("org.springframework.."));
```

- Erster Lauf: alle Verstöße werden in `archunit_store/` gespeichert
- Folgende Läufe: nur neue Verstöße brechen den Build
- Bestehende Verstöße werden schrittweise abgebaut
- Datei `archunit_store/` in `.gitignore` aufnehmen oder committen (Team-Entscheidung)

> Ideal für Legacy-Projekte: Regeln sofort einführen,
> ohne alle bestehenden Verstöße auf einmal fixen zu müssen.

CC BY-NC-SA 4.0, Alexander Erben

Integration in CI/CD

```
# azure-pipelines.yml (excerpt)
- task: Maven@4
  inputs:
    mavenPomFile: 'pom.xml'
    goals: 'verify'
    publishJUnitResults: true
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
```

- ArchUnit-Tests laufen mit jedem Build
- Fehlgeschlagene Regeln brechen die Pipeline
- Ergebnisse in den normalen Surefire Test-Reports sichtbar
- Keine zusätzliche Konfiguration nötig

> ArchUnit-Tests sind schnell (< 1 Sekunde) - sie analysieren Bytecode, starten keinen Spring-Kontext.

CC BY-NC-SA 4.0, Alexander Erben

Hands-on: Lab-08

ArchUnit-Tests für das Immobilien-CRM

CC BY-NC-SA 4.0, Alexander Erben

Diskussion

> Welche Architekturregeln würdet ihr in eurem Projekt einführen?

- Habt ihr heute schon Architekturregeln - und werden sie eingehalten?
- Welche Namenskonventionen gelten in euren Projekten?
- Wie unterscheidet sich ArchUnit von Code-Review in der Praxis?
- Wo ist die Grenze zwischen sinnvoller Governance und Over-Engineering?

CC BY-NC-SA 4.0, Alexander Erben