

Clean Architecture

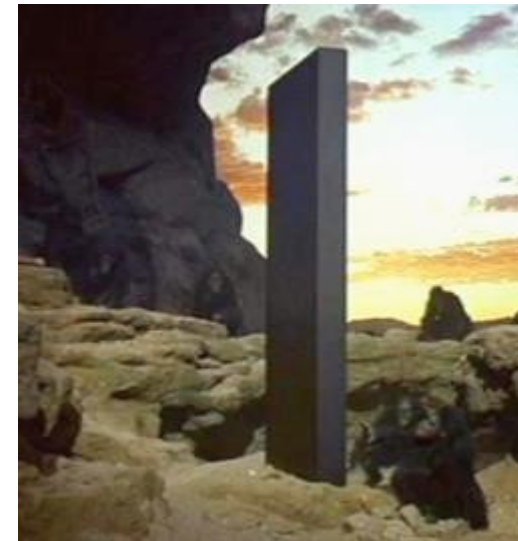
Software für die Ewigkeit



Pont du Gard, erbaut ca. 1. Jh. v. oder n. Chr.

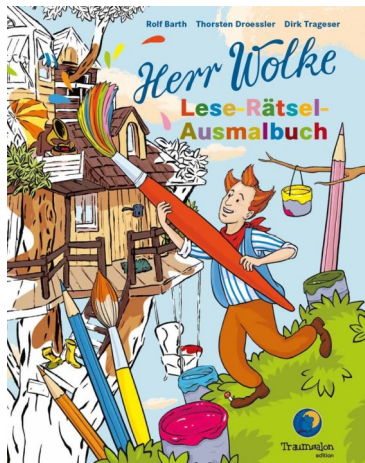
Softwareentwicklung

- Beständiger Wandel alle fünf Jahre
 - Kein zentraler Takt
 - Unterschiedliche Zykluslänge von Produkten
- Keine monolithischen Systeme mehr
 - Zusammenstöpseln von Bausteinen (Building Blocks)
 - Fundament bildet häufig ein Framework



Framework vs. Library

- Framework (Rahmenstruktur)
 - Semi-vollständige Anwendung
 - Kohärente Struktur
 - Entwickler vervollständigen „nur“ die leeren Bereiche - wie in einem Malbuch



Framework vs. Library

- Library (Programmbibliothek)
 - Sammlung von nützlichen Klassen und Methoden
 - Keine/kaum Anforderungen an Restprogramm
 - Keine Unterstützung für Strukturierung
 - Entwickler „kleben“ Bibliotheken aneinander



Framework vs. Library

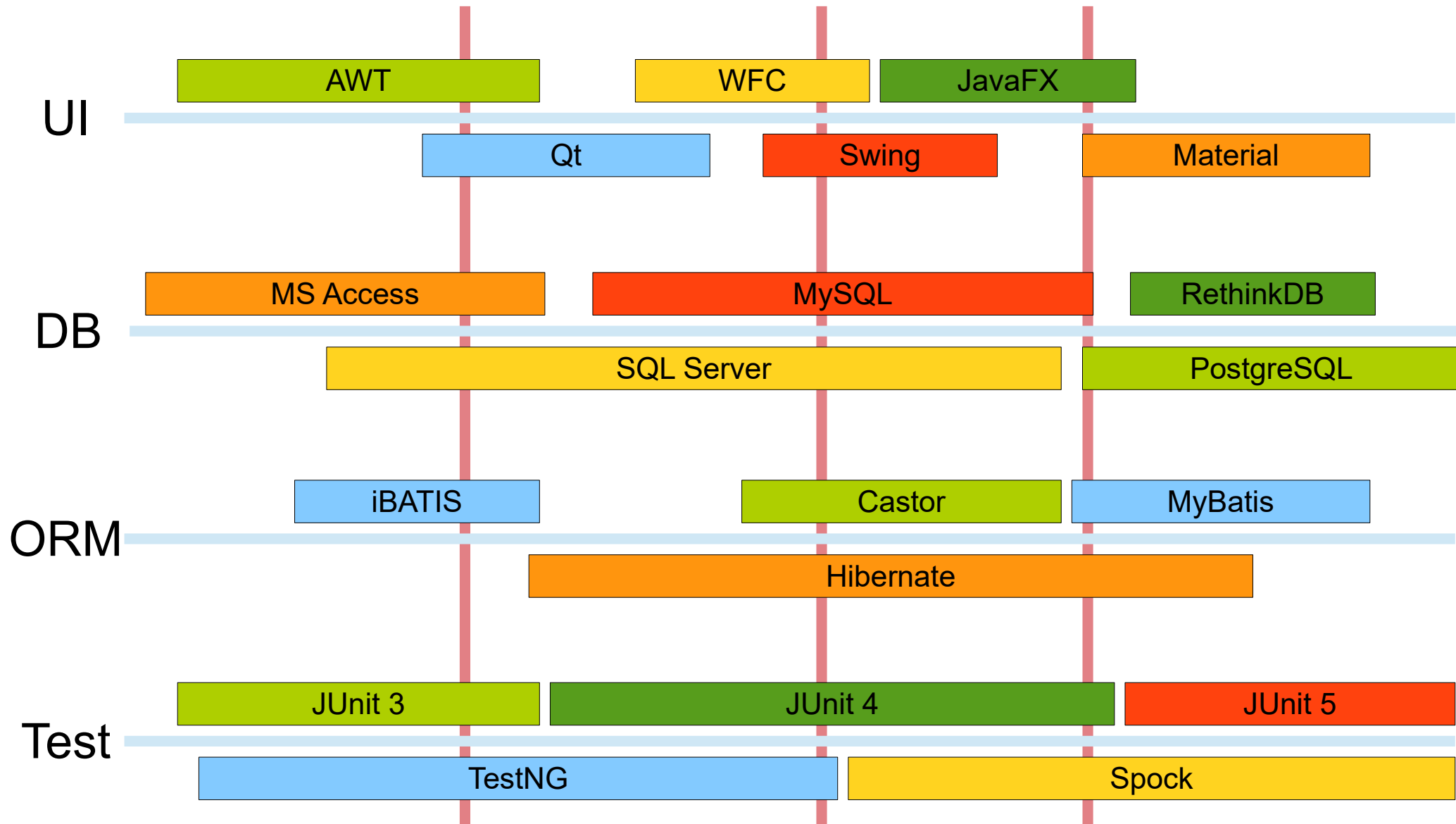
- Frameworks binden die Anwendung an sich
 - Starke Kopplung (Vendor lock-in)
 - Im Zweifel muss das „ausgemalte“ Malbuch weggeworfen werden
 - Kopplung auch an den Lebenszyklus
 - Häufig ändern sich Frameworks stark im Zuge eines Major Version Release
 - Libraries lassen mehr Freiheiten
 - Starke Kopplung vermeidbar
- bevorzugt Libraries anstatt Frameworks verwenden
 - Frameworks nicht „wie gedacht“ einsetzen

Vendor Lock-In

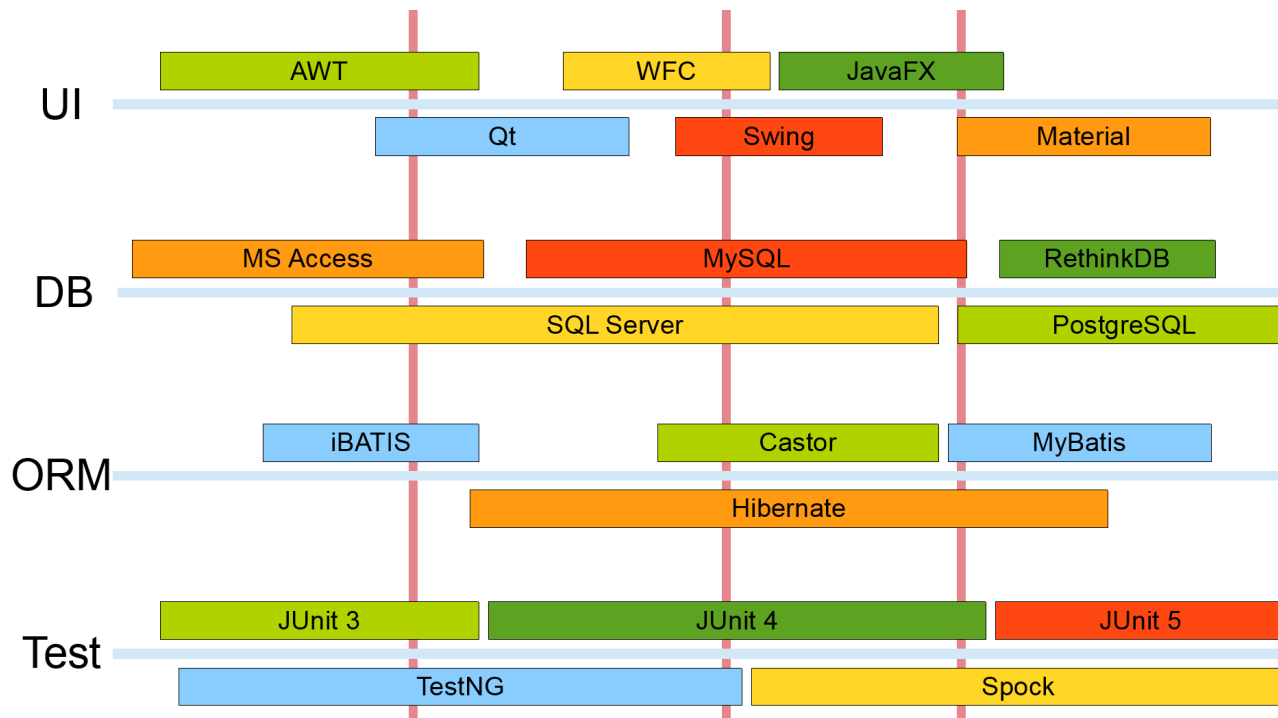
„I started development of a product with Angular 1.0, after some time Google released Angular 2, so we had to write the client side code and retest the entire product. When Google released Angular 4 (there is no 3...), backward compatibility was partial and again we had to re-write some portion of code, but retest the entire applications. We found some bugs in the Angular 4 & reported the same. Google has not fixed these bugs in 4 and has released Angular 5 along with bugs reported in 4, but again lot of compatibility issues. Hence again I have to resolve the compatibility issues and retest the entire application.“

<https://www.quora.com/What-is-the-reason-behind-no-backward-compatibility-in-angular-versions-Is-it-a-right-choice-for-client-side-development>
, abgerufen 25.03.2019

Technologiewahl für Projekte



Technologiewahl für Projekte



UI	AWT	Swing	JavaFX
DB	MS Access	MySQL	RethinkDB
ORM	iBATIS	Hibernate	MyBatis
Test	JUnit	TestNG	Spock

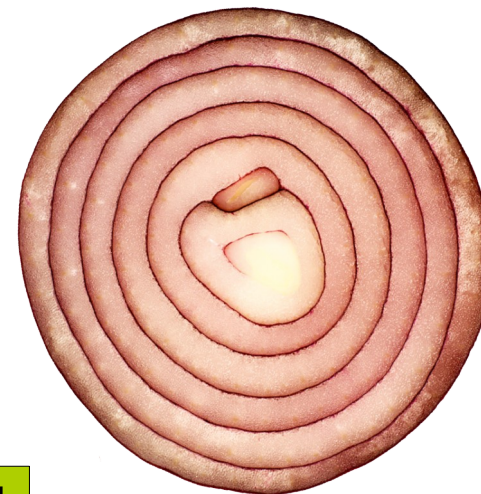
- Stark vom Zeitpunkt abhängig
- Bei gleichen Anforderungen trotzdem unterschiedlich
- Früh zu treffende Entscheidung
- Immer ein Kompromiss

Nachhaltige Technologiewahl

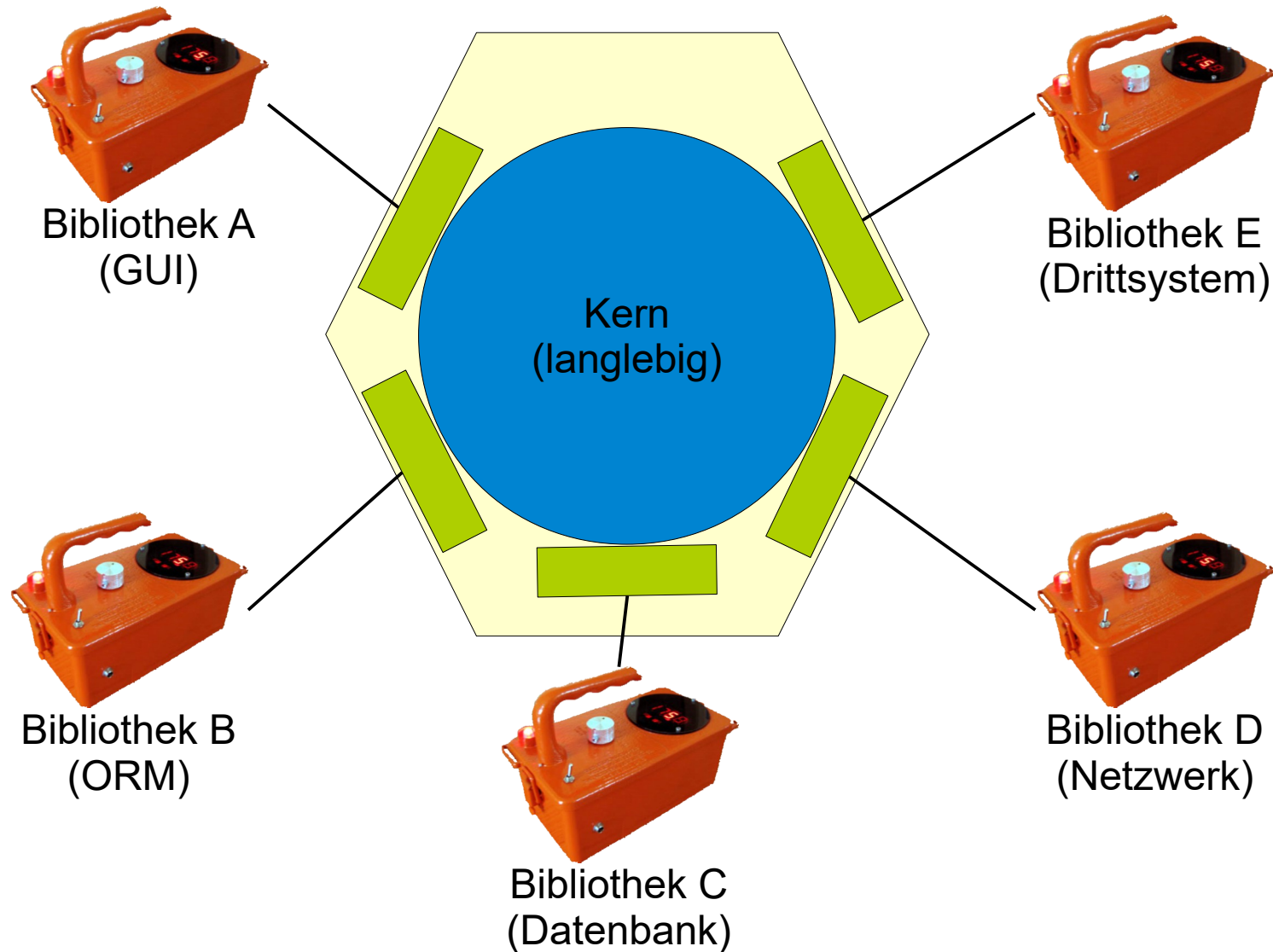
- Gute Entscheidungen werden spät getroffen
- Strukturen (Architektur) so wählen, dass Entscheidungen verzögert werden können
 - Ohne negative Folgen
- Minimalziel: Entscheidungen revidieren können
 - Mit möglichst geringen negativen Folgen

Kriterien für nachhaltige Architektur

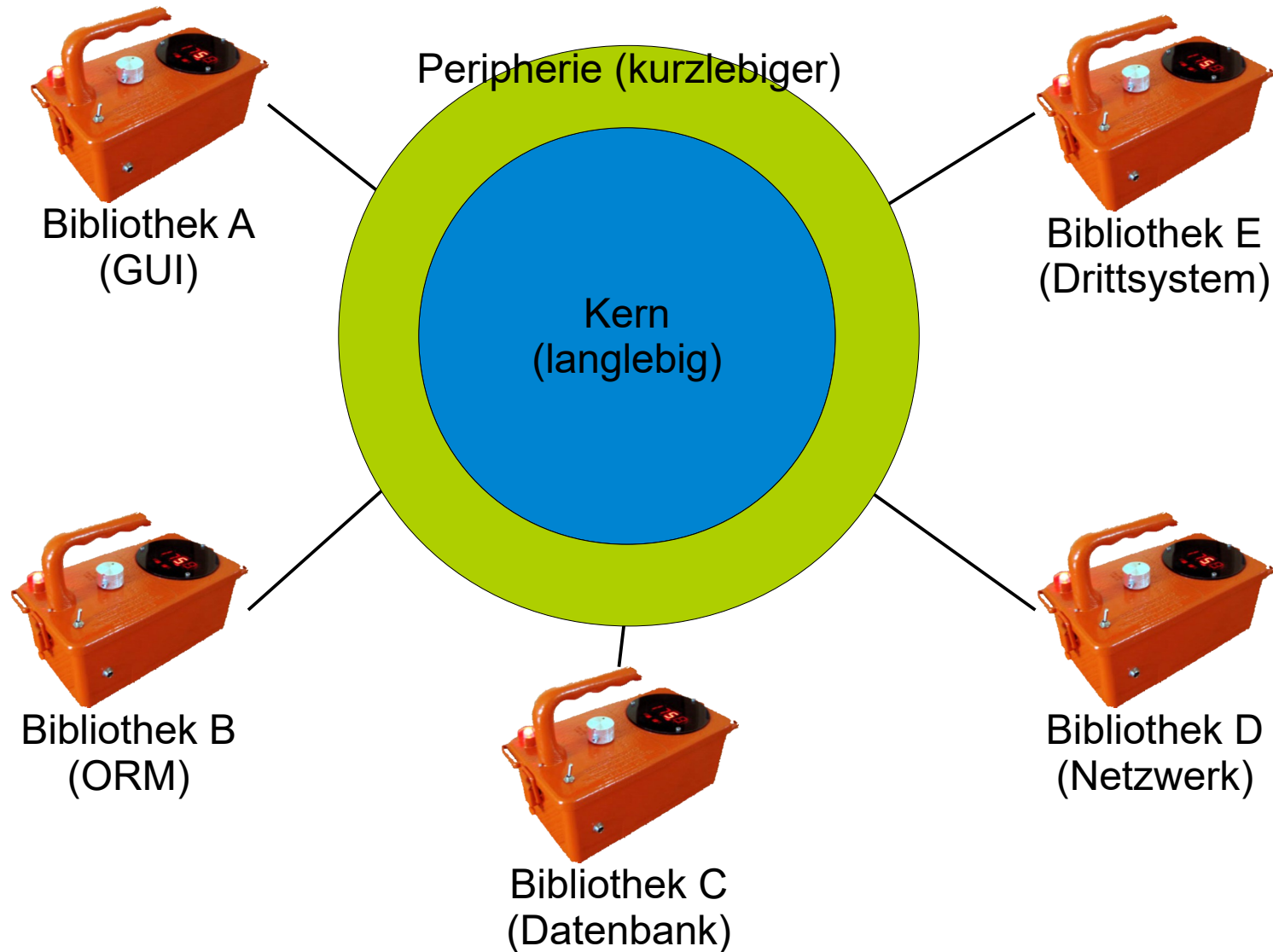
- Eine langfristige Architektur
 - Besitzt einen technologieunabhängigen Kern
 - Die eigentliche Anwendung
 - Behandelt jede Abhängigkeit als temporäre Lösung
 - Unterscheidet zwischen zentralem (langlebigem) und peripherem (kurzlebigerem) Sourcecode
- Metapher: Die Zwiebel
 - „Onion Architecture“
(siehe Vorlesung im letzten Semester)



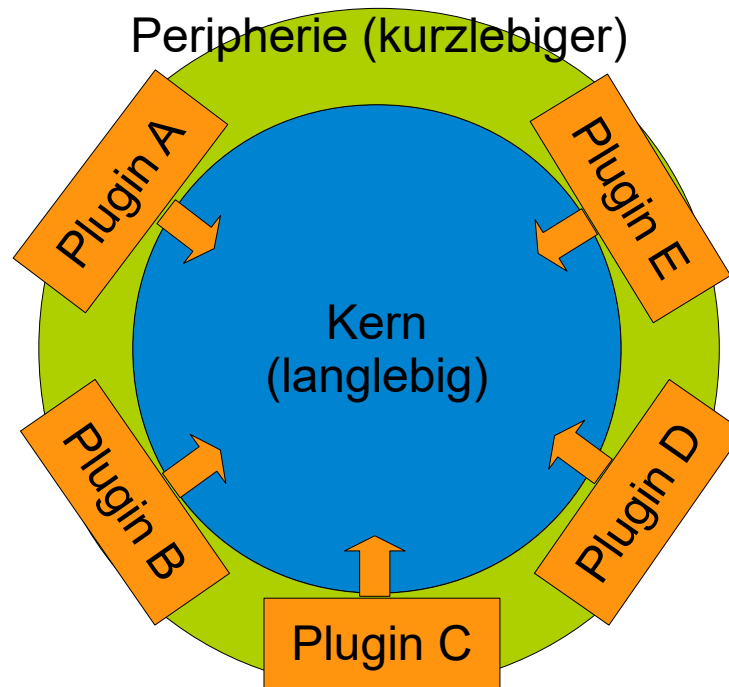
Struktur der Clean Architecture



Struktur der Clean Architecture



Struktur der Clean Architecture



- Abhängigkeit immer von außen nach innen
- Kern-Code hängt nie von Plugins ab

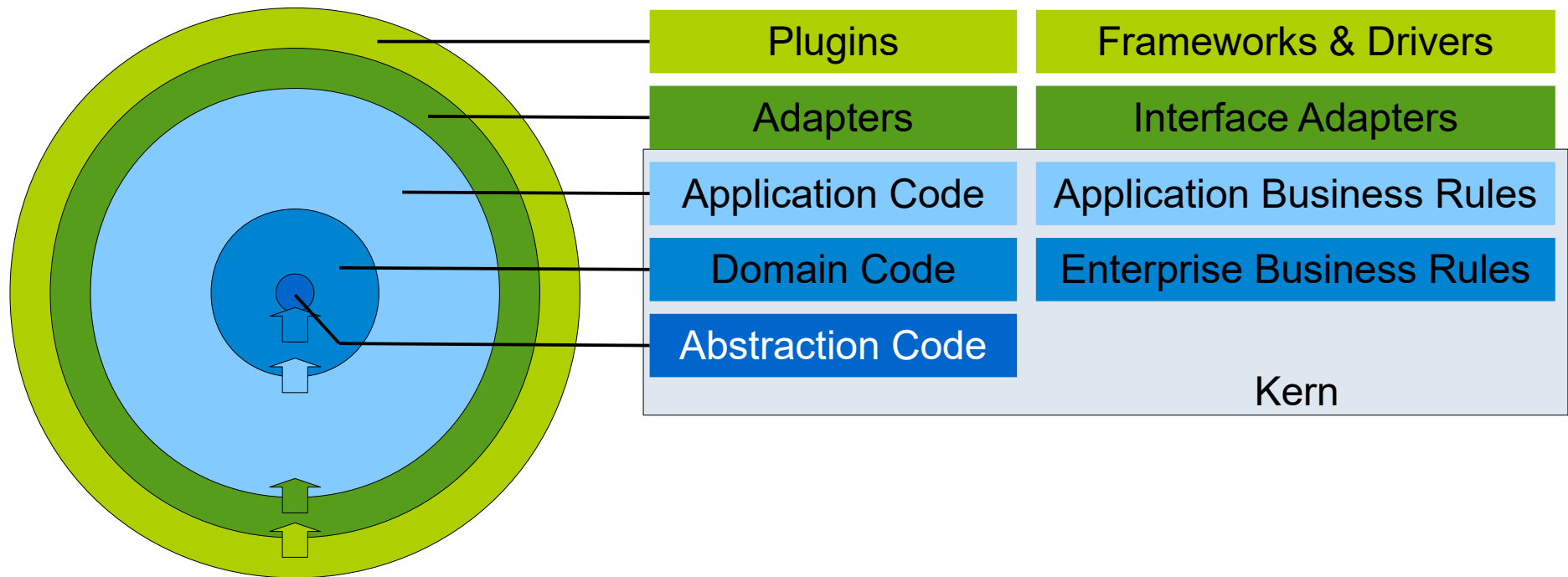
Die Dependency Rule

- Zentrale Regel für Abhängigkeiten

Abhängigkeiten immer von außen nach innen

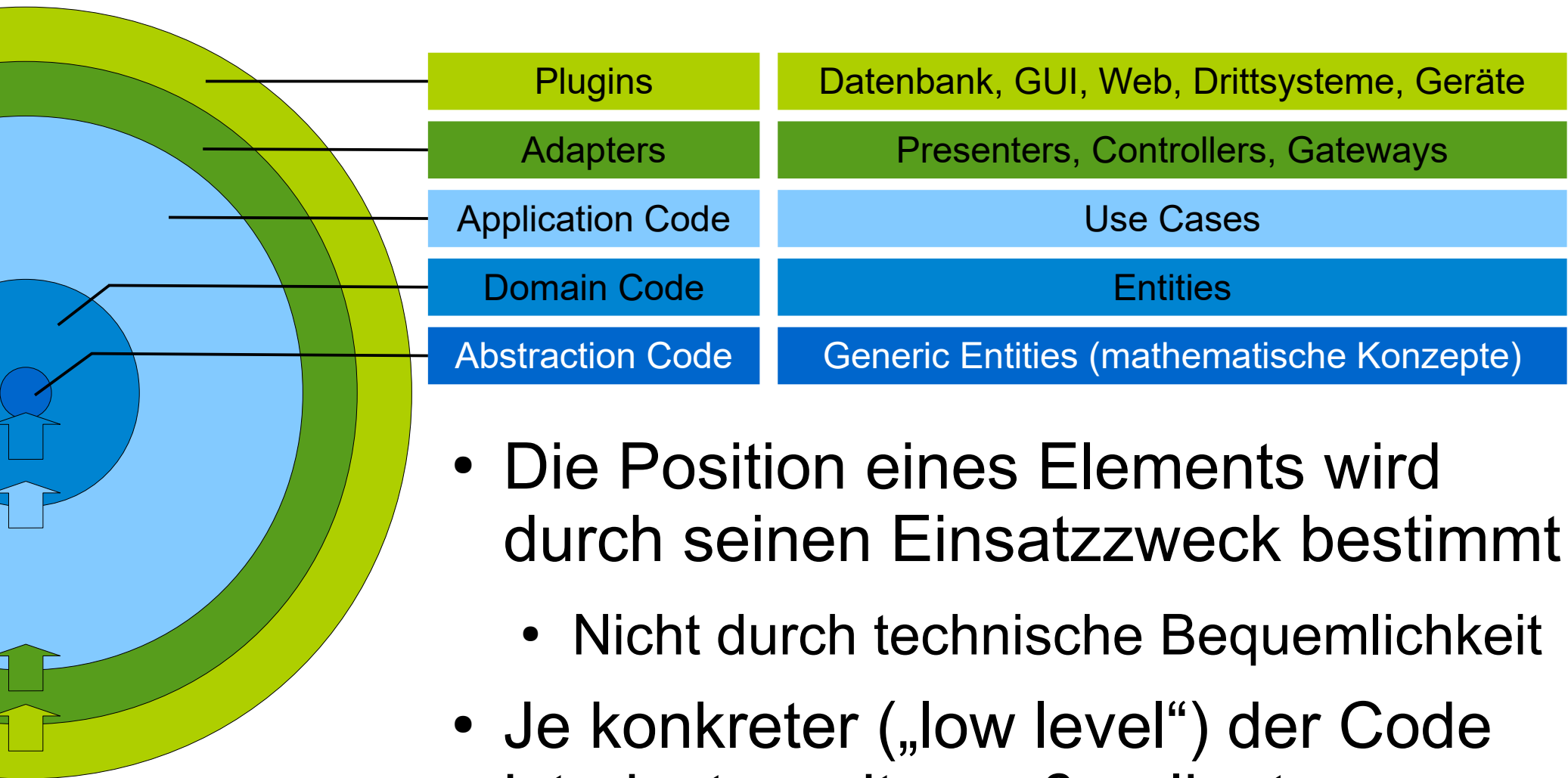
- Erfordert für jede Klasse eine klare Positionierung
- Abhängigkeitspfeile gehen immer von außen nach innen
 - Aufrufpfeile können in beide Richtungen gehen

Struktur der Clean Architecture



- Innere Schichten wissen nichts von den Äußeren
 - Abhängigkeiten immer von außen nach innen
- Beliebig viele innere Kern-Schichten (oft drei)

Position der Elemente



Grundregeln der Clean Architecture

- Der Anwendungs- und Domaincode ist frei technischen Details
 - Sämtlicher Code kann eigenständig verändert werden
 - Sämtlicher Code kann unabhängig von Infrastruktur kompiliert und ausgeführt werden
- Innere Schichten definieren Interfaces, äußere Schichten implementieren diese
 - Die äußeren Schichten koppeln sich an die inneren Schichten (Richtung Zentrum)
 - Dependency Inversion

Schicht 4: Abstraction Code

- Enthält domänenübergreifendes Wissen
 - Grundbausteine, die nicht domänenspezifisch sind
 - Allgemeine Konzepte und Algorithmen
 - „nachgerüstete“ Libraries
- Sehr abstrakt
- Ändert sich selten/nie → sehr stabil

Schicht 4: Abstraction Code

Domänenübergreifende Grundbausteine:

- Beim Schreiben von Software müssen wir nicht bei Null beginnen
- Meistens existieren Grundbausteine, die nicht domänenspezifisch sind, sondern in vielen oder gar allen Anwendungen verwendet werden, z.B.
 - Strings
 - Zahlen und mathematische Operationen
- Normalerweise über die verwendete Programmiersprache verfügbar

Schicht 4: Abstraction Code

Allgemeine Konzepte und Algorithmen

- Mathematische Konzepte (z.B. Matrizen)
- Algorithmen (z.B. QuickSort, Breadth First Search)
- Datenstrukturen (z.B. Baum, Stack, geordnete Liste)
- Abstrahierte Muster (z.B. Quantitäten)

Schicht 4: Abstraction Code

„nachgerüstete“ Libraries

- Häufig werden bestimmte Funktionalitäten in vielen/allen Schichten benötigt:
 - Validierung
 - Fehlende Standards:
 - Spring für Dependency Injection anstatt CDI
 - JPA
- In diesen Fällen können die entsprechenden Libraries als Teil des „Abstraction Code“ betrachtet werden
- Achtung: je mehr Abhängigkeiten im Abstraction Code, desto weniger „langlebig“ ist diese Schicht!
 - Auf das Nötigste beschränken

Schicht 4: Abstraction Code

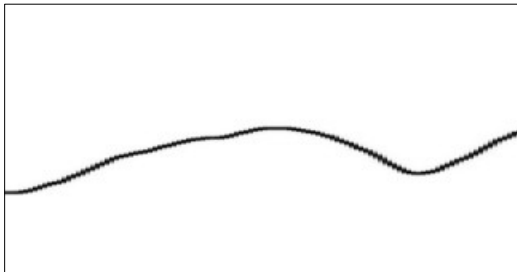
Abstraction Code in der Praxis

- Explizite Anlage als eigene Schicht häufig nicht notwendig
- Fehlende Elemente wahrscheinlich bereits als Library verfügbar
- Kann auch nachträglich extrahiert und eingeführt werden
- Nur anlegen wenn wirklich benötigt

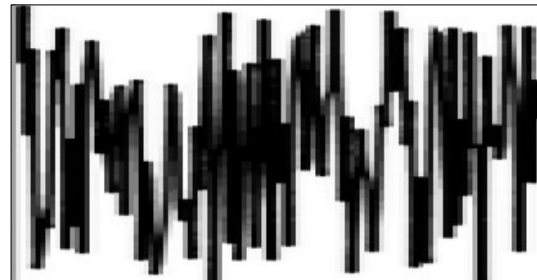
Beispiel für Abstraction Code

Perlin Noise

- Rauschfunktion
- Erzeugt „natürlicher“ verlaufende Pseudo-Zufallszahlen

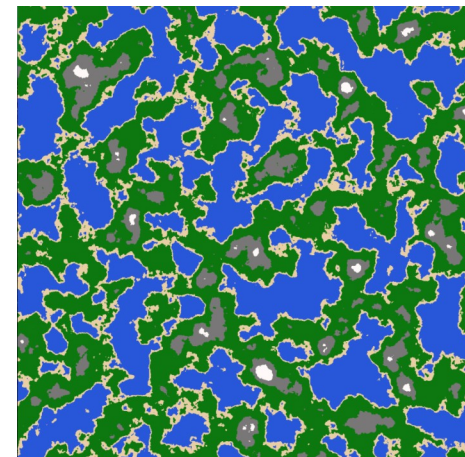
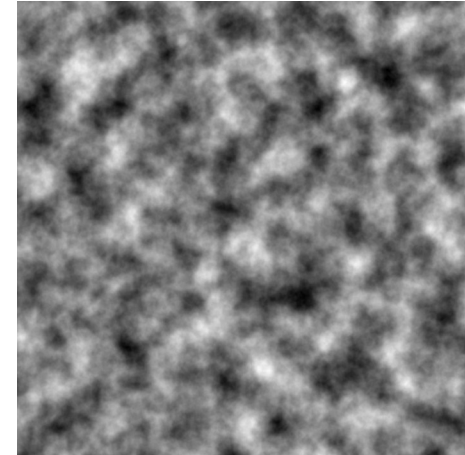


Perlin



Echter Zufall

- Wird häufig für zufallsgenerierte Inhalte verwendet (procedural content generation)
- Beispiel: generierte Karten



Schicht 3: Domain Code

- Enthält v.a. Entities (Business Objects)
- Implementiert organisationsweit gültige Geschäftslogik (Enterprise Business Rules)
- Der innere Kern der Anwendung bzw. Domäne
- Sollte sich am seltensten ändern
 - Immun gegen Änderungen an Details wie Anzeige, Transport oder Speicherung
 - Unabhängig vom konkreten Betrieb der Anwendung

Schicht 3: Domain Code

Enterprise Business Rules

- Fachliche Regeln, die innerhalb einer Organisation immer gelten (vgl. Vorlesung DDD)
- Existieren immer, unabhängig davon, ob diese Regeln in einer Anwendung nachmodelliert wurden
- Beispiele:
 - ein Student kann genau eine Erstklausur, eine Nachklausur und eine mündliche Prüfung pro Modul ablegen
 - Eine Bestellung muss mindestens eine Position und eine Rechnungsanschrift haben

„[...] business rules are rules or procedures that make or save the business money. [...] they are critical to the business itself, and would exist even if there were no system to automate them“

-Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston, MA: Prentice Hall. ISBN: 978-0-13-449416-6

Beispiel für Domain Code

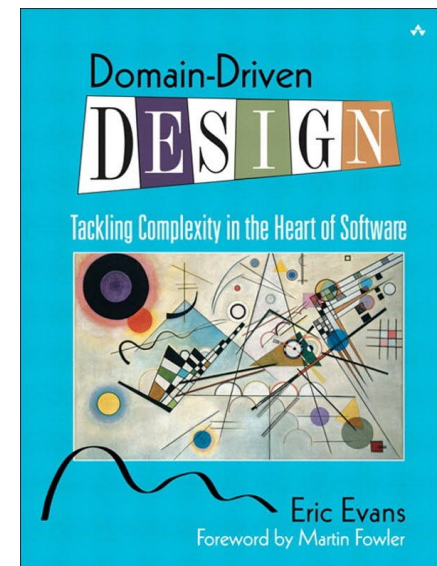
- Domäne: Bankkonto-Verwaltungssoftware
- Ein zentraler Begriff ist das „Konto“
- Jedes Konto muss der zentralen Regel genügen:

Die Summe der Zubuchungen, Abbuchungen und des inversen Kontostands ergibt immer 0

- Das Konto ist eine Klasse im Domain Code
- Die Regel ist eine Invariante in der Konto-Klasse
 - Jede Methode der Klasse Konto muss die Regel beachten

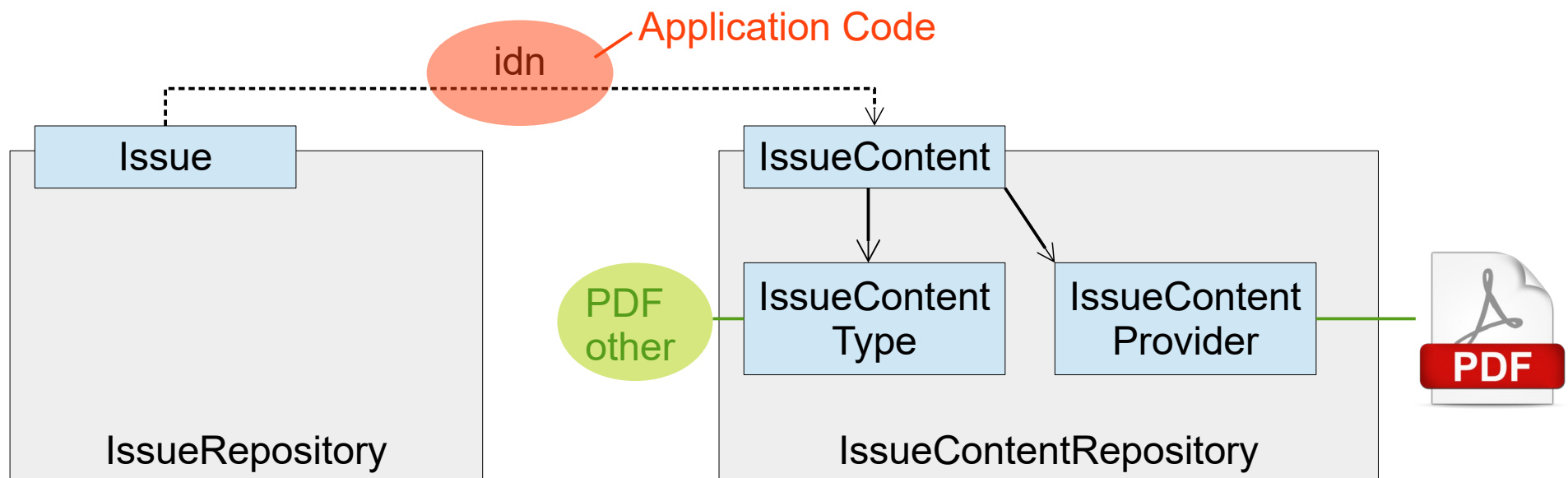
DDD und Domain Code

- Aggregate (Entities, Value Objects), Domain Services, Repositories und Factories sind typische Strukturen im Domain Code
- Aggregat-übergreifendes Verhalten ist Bestandteil der Use Cases (Application Code)



Beispiel für DDD und Domain Code

- Projekt zur Bereitstellung von E-Books



- Issue und IssueContent sind über die idn konzeptionell miteinander verbunden
- Diese Verbindung wird aber erst im Application Code umgesetzt

Schicht 2: Application Code

- Enthält die Anwendungsfälle (Use Cases)
 - Resultiert direkt aus den Anforderungen
 - „was kann die Anwendung“
- Implementiert die anwendungsspezifische Geschäftslogik (Application-specific Business Rules)
- Steuert den Fluss der Daten und Aktionen von und zu den Elementen des Domain Codes
 - Orchestriert die Elemente des Domain Codes, um den jeweiligen Anwendungsfall umzusetzen

Schicht 2: Application Code

Application-specific Business Rules

- Regeln, die nicht organisationsweit, sondern spezifisch innerhalb der Anwendung gelten
- Häufig Regeln für den Ablauf eines Workflows
- Beispiele:
 - Registrierung per Double Opt-In
 - Nutzer registriert sich
 - Nutzer erhält Bestätigungs-Mail an angegebene Email-Adresse mit Bestätigungs-Link
 - Erst nach Klick des Links ist die Registrierung abgeschlossen
 - Ändern eines Passwortes erfordert Angabe & Prüfung des alten Passwortes

Schicht 2: Application Code

- Änderungen an dieser Schicht beeinflussen die Domain-Schicht (v.a. die Entities) nicht
- Isoliert von Änderungen an der Datenbank, der graphischen Benutzeroberfläche, etc.
 - Der Use Case weiß nicht wer ihn aufgerufen hat oder wie das Ergebnis präsentiert wird
- Wenn sich Anforderungen ändern, hat das wahrscheinlich Auswirkungen auf diese Schicht

Beispiel für Application Code

- Bankkonto-Verwaltungssoftware
- Zentraler Use Case: Überweisungen
 - Abbuchung von Konto 1, Zubuchung auf Konto 2
 - Auch hier muss eine wichtige Regel gelten:

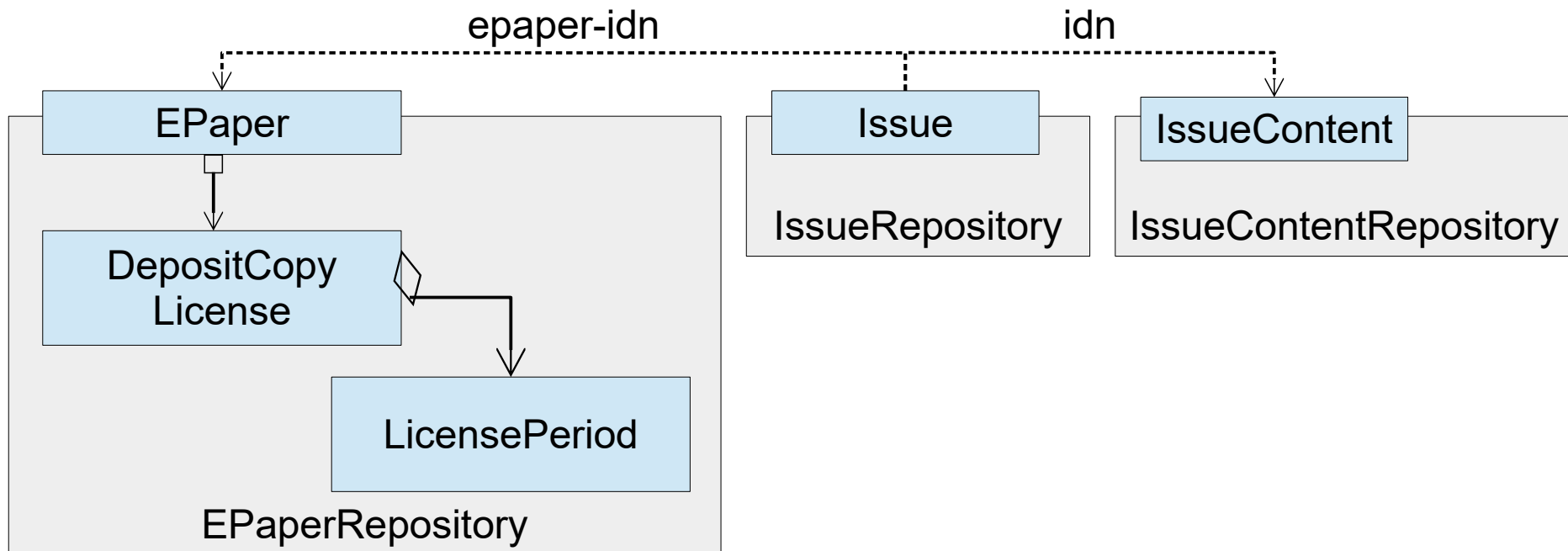
Die Summe aus Abbuchung und Zubuchung ergibt immer 0

- Kann sich ändern, beispielsweise bei Einführung von Transaktionsgebühren
 - Hat aber keine Auswirkungen auf die Domäne!

Beispiel für DDD und Application Code

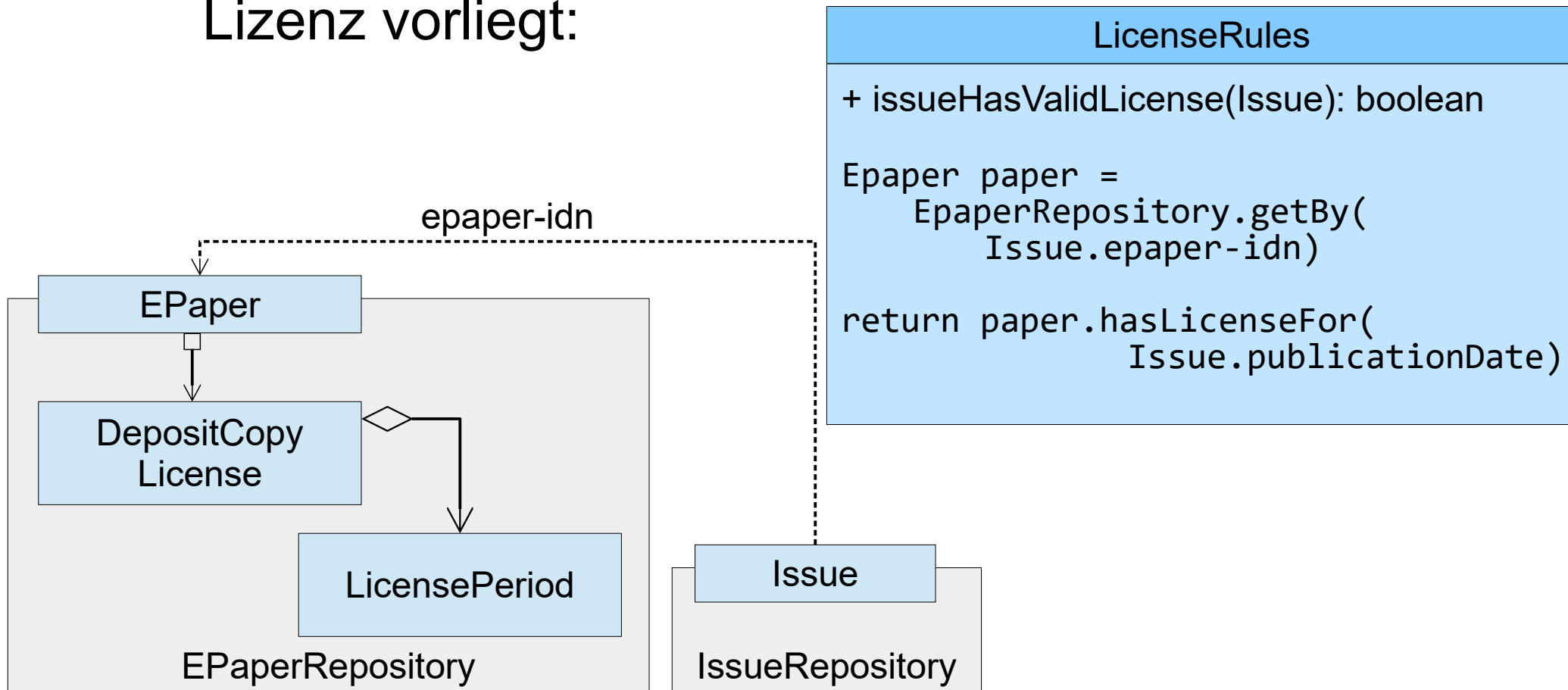
- Projekt zur Bereitstellung von E-Books
 - Prüfen, ob für eine konkrete Ausgabe eine gültige Lizenz vorliegt:

LicenseRules
+ issueHasValidLicense(Issue): boolean



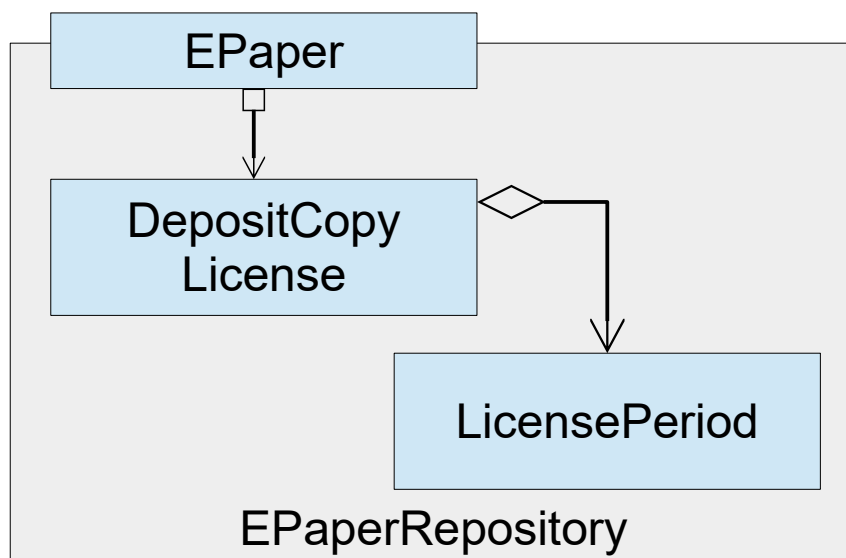
Beispiel für DDD und Application Code

- Projekt zur Bereitstellung von E-Books
 - Prüfen, ob für eine konkrete Ausgabe eine gültige Lizenz vorliegt:



DDD und Application Code

- Application Code (Use Cases) formuliert aggregatübergreifende Funktionalität
 - Steigt nicht in die Details eines Aggregats ab
 - Verwendet nur Methoden der Aggregate Root Entity
 - Law of Demeter light



+ `Epaper.hasLicenseFor(publicationDate): boolean`

Schicht 1: **Adapters**

- Diese Schicht vermittelt Aufrufe und Daten an die inneren Schichten
 - Formatkonvertierungen
 - Externes Format wird so umgewandelt, dass die Applikation gut zurecht kommt
 - Internes Format wird so umgewandelt, dass die externen Plugins gut zurecht kommen
- Oftmals nur einfache Datenstrukturen, die hin- und hergereicht werden
- Ziel: Entkopplung von „innen“ und „außen“
 - „hässlicher“ Mapping-Code liegt in dieser Schicht
 - Macht die „Drecksarbeit“

Schicht 1: **Adapters**

- Anti-Corruption Layer
- Beispiele:
 - alle Klassen einer MVC-Struktur für eine bestimmte UI (z.B. Vaadin)
 - direkt verwendbares Render-Model für serverseitiges Rendering, z.B. Reports oder Websites
 - Key-Value-Paket
 - Abbildung von Eingabe-Formularen

Beispiel für **Adapters**

- Bankkonto-Verwaltungssoftware
- Anzeige auf Webseite (HTML) vorbereiten
- Alle veränderlichen Inhalte der Seite unzweideutig berechnen (RenderModel)
 - Geldbeträge als Zeichenketten im Format 1234,56 €
 - Die Anzeigeschicht benötigt keine numerischen Werte
 - Farben als HTML-Hexcodes
 - Attribute (z.B. checked="checked" für Checkboxes)
- Ziel: Keine Umsetzungslogik in der Plugin-Schicht notwendig

Beispiel für **Adapters**

- Alle Werte „mundfertig“ im RenderModel

FinanceStatusRenderModel
(Map<String, String>)

blz	94059421
user.name	Herrn Max Muster
messages.new	Sie haben neue...
debit	15.207,16 EUR
debit.color	#10141D
credit	-22,85 EUR
credit.color	#B9354C
overview.debit	114.497,45 EUR
...	...
...	...
accounts	List<AccountRenderModel>

AccountRenderModel
(Map<String, String>)

The screenshot displays the Sparkasse Musterstadt online banking portal. The main section, titled 'Finanzstatus', shows a summary of account balances. On the left, a sidebar menu lists various services like 'Internet-Banking', 'Abmelden', and 'Konten und Karten'. The main content area is divided into sections for 'Giro**', 'Geldanlage', 'Depot*', and 'Darlehen'. Each section lists specific accounts with their respective balances and transaction history. For example, the 'Giro**' section shows a balance of 15.207,16 EUR and a debit of -22,85 EUR. The 'Geldanlage' section shows a balance of 31.019,51 EUR. The 'Depot*' section shows a balance of 68.270,78 EUR. The 'Darlehen' section shows a balance of -33.876,95 EUR. At the bottom, there are buttons for 'Druckansicht' and 'Aktualisieren'.

Konto-Nr.	Saldo (EUR)
75432	4.619,52
10023844	582,68
10023851	10.004,96
10037505	-22,85

Konto-Nr.	Saldo (EUR)
75432	4.619,52
10023844	582,68
10023851	10.004,96
10037505	-22,85

Konto-Nr.	Saldo (EUR)
75432	4.619,52
10023844	582,68
10023851	10.004,96
10037505	-22,85

Konto-Nr.	Saldo (EUR)
75432	4.619,52
10023844	582,68
10023851	10.004,96
10037505	-22,85

Beispiel für DDD und **Adapters**

- Projekt zur Bereitstellung von E-Books
 - Ausgabe der Daten als REST-Webservice

Adapters

```
public class EPaperResource {  
    private Link selfReferringLink;  
    private String idn;  
    private String title;  
    private List<String> formerTitles;  
    private List<String> laterTitles;  
    private Integer yearOfFirstPublication;  
    private Integer yearOfLastPublication;  
    private List<PublisherResource> publishers;  
  
    @XmlJavaTypeAdapter(Link JAXBAdapter.class)  
    public List<Link> getLinks() {  
        return Collections.singletonList(  
            selfReferringLink);  
    }  
}
```

Domain Code

```
public class EPaper {  
    private String idn;  
    private String title;  
    private final List<String> formerTitles;  
    private final List<String> laterTitles;  
    private Year yearOfFirstPublication;  
    private Year yearOfLastPublication;  
    private final List<Publisher> publishers;  
}
```

Implementierungs-
details für das Plugin
Hier:
Serialisierung mit JAXB

Warum Umkopieren für **Adapters**?

- „Warum nochmal ein Mapping von Domaindaten auf Adapterdaten?“
 - „Vor allem, wenn sich an den Daten nichts ändert?“
- Antwort: Weil dieser Zustand **temporär** und **zufällig** ist!
 - Domain und Adapter sind **momentan** sehr ähnlich
 - Sie werden sich in Zukunft **unabhängig voneinander** verändern
 - Die Auswirkungen von Änderungen sollten möglichst lokal gehalten werden → Ähnlich zu Law of Demeter

Aber ich will trotzdem nicht!

- „Es ist unnütze Arbeit ohne unmittelbaren Wert“
- Das ist eine momentan korrekte Einschätzung
- Wie wäre es mit einem Kompromiss:
 - Aktuell kein Mapping einbauen
 - Sourcecode so strukturieren, dass späteres Trennen der Ebenen durch ein Mapping einfach eingebaut werden kann
 - Die Möglichkeit des Trennens immer als Werkzeug parat haben
- Arbeit dann erledigen, wenn sie einen Wert hat

Schicht 0: Plugins

- Diese Schicht greift hauptsächlich auf die Adapter zu
- Enthält Frameworks, Datentransportmittel und andere Werkzeuge
 - v.a. Datenbank, Benutzeroberfläche, Web
 - Alle „Pure Fabrication“-Entscheidungen
- Wir versuchen, hier möglichst wenig Code zu schreiben
 - Hauptsächlich Delegationscode, der an den Application Code und die Adapter weiterleitet

Schicht 0: Plugins

- Auf gar keinen Fall enthält diese Schicht Anwendungslogik
 - Die Daten fallen mundfertig aus dem Adapter
 - Alle Entscheidungen sind bereits gefallen
 - Anfragen werden nicht uminterpretiert (das machen die Adapter)
- Keine emotionale Bindung an diesen Code
 - Jederzeitige Änderung möglich
 - Auswirkungen nur auf die Adapterschicht
 - Übersichtlicher Aufwand

Beispiel für Plugins

- Bankkonto-Verwaltungssoftware
- HTML-Rendering mit Velocity-Template

Sparkasse Märkisch-Oderland

Einkaufsgutscheine im Online-Banking.
[Mehr erfahren](#)

BLZ 17054040 | BIC WELADED1MOL

Home Ihre Sparkasse Service Übersicht Kontakt

Suchbegriff

Online-Banking
Max Mustermann
[Neue Nachrichten](#)
[Abmelden](#)

direkt zu:
- Bitte auswählen -

Startseite
Finanzstatus
Kontodetails
Termingeld
Umsätze
Banking
PIN/TAN-Verwaltung
Brokerage
Deka
Kreditkarte

Finanzstatus

Nach Kontoarten sortieren | Giro-Detail-Übersicht | Termingeldübersicht

Mustermann, Max [Seitenanfang](#)

Konto	Kontonummer	Kontostand	
Privatgirokonto ** Lebensmittel	123456	1.000,00 EUR	Info Liste Transfer Zahlung Karte
Tagesgeld Rücklage	200000	10.235,00 EUR	Info Liste Transfer Zahlung Karte
Girokonto (USD) **	654321	1.000,00 USD (734,65 EUR)	Info Liste Transfer
Termingeld	223344556	15.000,00 EUR	Info Liste Karte
Deka *	000100000	47.472,85 EUR	Info Liste Transfer Zahlung Karte

Info-Box
Zu Ihrer Sicherheit erfolgt die automatische Abmeldung in 2 Min.

Online-Banking-Hotline
03341 340-4444
03346 150-4444
03344 33440-4444

[E-Mail schreiben](#)
[Filiale finden](#)
[Notfallnummern](#)

[f](#) [t](#)

Beispiel für Plugins

- Sourcecode der HTML-Seite
- Serverseitig generiert bei jedem Request

```
<tr class="tablerowodd">
  <td>Privatgirokonto<em>**</em><br>Lebensmittel<br></td>
  <td class="right" title="IBAN: DE89 1705 4040 0000 1234 56">123456<br></td>
  <td class="right"><span class="plus">1.000,00&nbsp;EUR</span><br></td>
  <td class="right">
    <input name="juhWEH" value="Kontodetails" onclick="return do();"
      src="6.gif" title="Kontodetails" type="image">
    <input name="yjSUpS" value="Umsatzabfrage" onclick="return do();"
      src="2.gif" title="Umsatzabfrage" type="image">
    <input name="ikqdyo" value="Überweisung" onclick="return do();"
      src="3.gif" title="Überweisung" type="image">
    <input name="cjYcZR" value="Dauerauftrag" onclick="return do();"
      src="5.gif" title="Dauerauftrag" type="image">
    <input name="gzZfjB" value="Weitere Funktionen" onclick="return do();"
      src="if5_i_aktionen.gif" title="Weitere Funktionen" type="image">
  </td>
</tr>
```

Beispiel für Plugins

- Veränderliche Inhalte als benannte Variablen
- Velocity setzt die Werte des RenderModel ein

```
<tr class="tablerowodd">
  <td>                                $account_title                                </td>
  <td class="right" title="            $iban            "> $number </td>
  <td class="right"><span class="$sgn">    $balance    </span><br></td>
  <td class="right">
```



AccountRenderModel
(Map<String, String>)

account_title	Privatgirokonto** Lebensmittel
iban	IBAN: DE89 1705 4040 0000 1234 56
number	123456
sgn	plus
balance	1.000,00 EUR

Beispiel für Plugins

- Die entstandene Webseite enthält keinen Hinweis auf Variablen oder das Rendering

 Sparkasse Märkisch-Oderland

Einkaufsgutscheine im Online-Banking.
 Mehr erfahren

BLZ 17054040 | BIC WELADED1MOL

Home Ihre Sparkasse Service Übersicht Kontakt

A A A Suchbegriff

▼ Online-Banking
Max Mustermann
☒ [Neue Nachrichten](#)
 Abmelden


direkt zu:
- Bitte auswählen - 


Startseite
Finanzstatus
Kontodetails
Termingeld
Umsätze
Banking
PIN/TAN-Verwaltung
Brokerage
Deka
Kreditkarte

Finanzstatus

 Nach Kontoarten sortieren  Giro-Detail-Übersicht  Termingeldübersicht

➔ Mustermann, Max ▲ Seitenanfang

Konto	Kontonummer Kontoname	Kontostand	
Privatgirokonto ** Lebensmittel	123456	1.000,00 EUR	    
Tagesgeld ** Rücklage	200905	18.235,00 EUR	    
Girokonto (USD) **	654321	1.000,00 USD (734,65 EUR)	  
Termingeld	223344556	15.000,00 EUR	  
Deka *	000100000	47.472,85 EUR	  

 Info-Box
Zu Ihrer Sicherheit erfolgt die automatische Abmeldung in 2 Min.

 Online-Banking-Hotline
03341 340-4444
03346 150-4444
03344 33440-4444

☒ E-Mail schreiben
 Filiale finden
 Notfallnummern

 DHBW
Duale Hochschule
Baden-Württemberg

Beispiel für DDD und Plugins

- Projekt zur Bereitstellung von E-Books
 - REST-API mit JAX-RS

```
@Path("/epapers")
public class EPapersEndpoint {
    @Inject private EPaperAccess ePaperAccess;
    @Inject private EPaperToEPaperResourceMapper ePaperToEPaperResource;

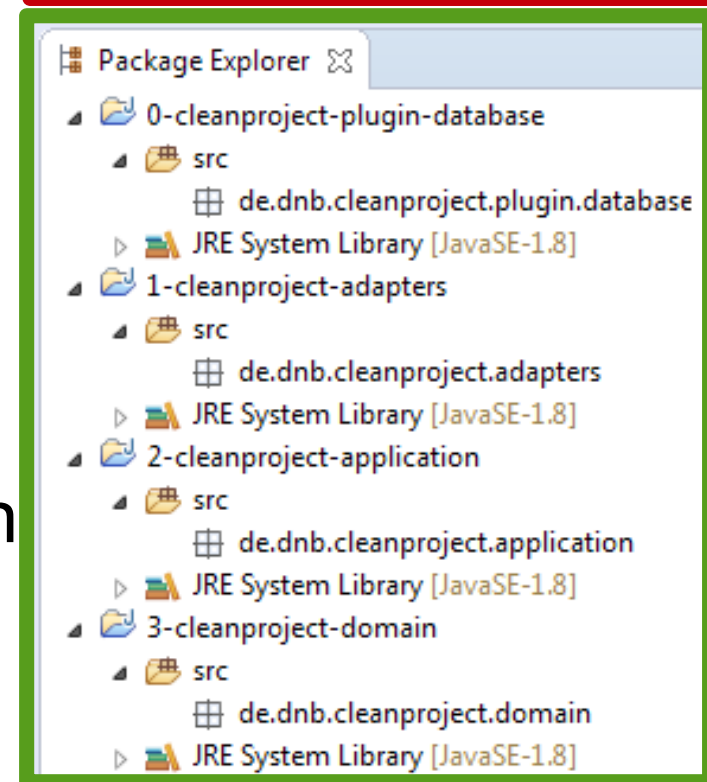
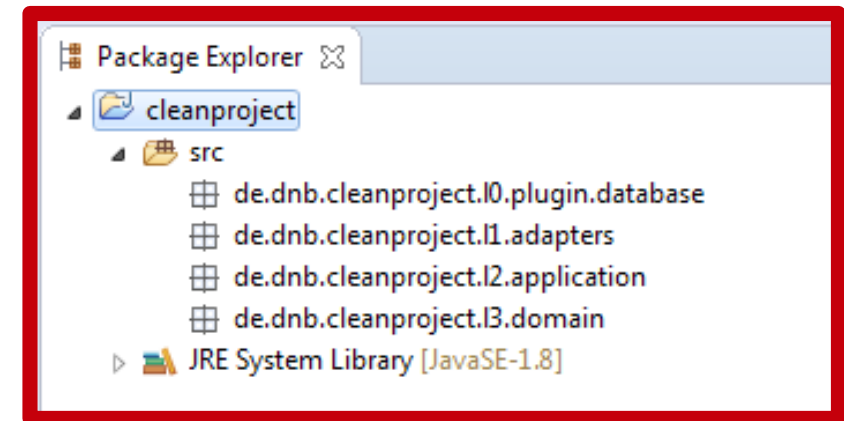
    @GET
    @Path("")
    @Produces(MediaType.APPLICATION_JSON + "; charset=utf-8")
    public Response findeEPapers(
        @QueryParam("iln")
        @NotNull(message = "add required query parameter iln") String holdingsIlIn) {
        return Response.ok(
            ePaperAccess.findEPapers(holdingsIlIn).stream()
                .map(ePaperToEPaperResource)
                .collect(Collectors.toList())
        ).build();
    }
}
```

Application Code (points to `ePaperAccess.findEPapers(holdingsIlIn).stream()`)

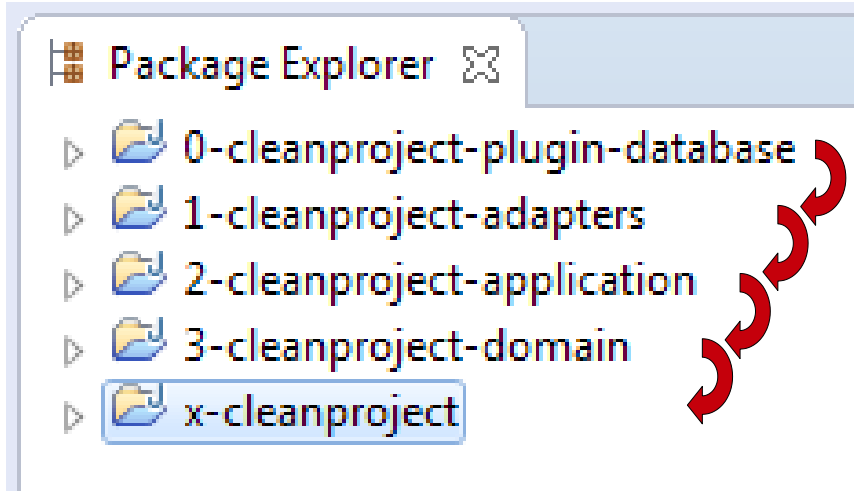
Adapters (points to `ePaperToEPaperResource`)

Konkrete Umsetzung

- Nicht alle Klassen in einem Projekt
 - Schichtenbildung über Packages ist in Ordnung
 - Aber: keine Überprüfung durch den Compiler
- Lieber mehrere Projekte („Multi-Projekt“)
 - Compiler findet nur Klassen
 - im eigenen Projekt
 - in referenzierten Projekten



Konkrete Umsetzung: Maven



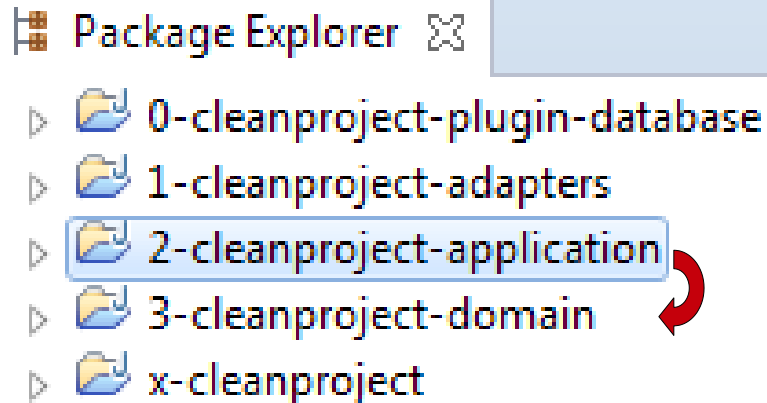
- Ein „Klammerprojekt“ für globale Einstellungen
 - Maven „Parent-POM“
- Enthält alle anderen Projekte als Module

```
<project>
  <groupId>de.dnb</groupId>
  <artifactId>x-cleanproject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>3-cleanproject-domain</module>
    <module>2-cleanproject-application</module>
    <module>1-cleanproject-adapters</module>
    <module>0-cleanproject-plugin-database</module>
  </modules>

  [...]
</project>
```

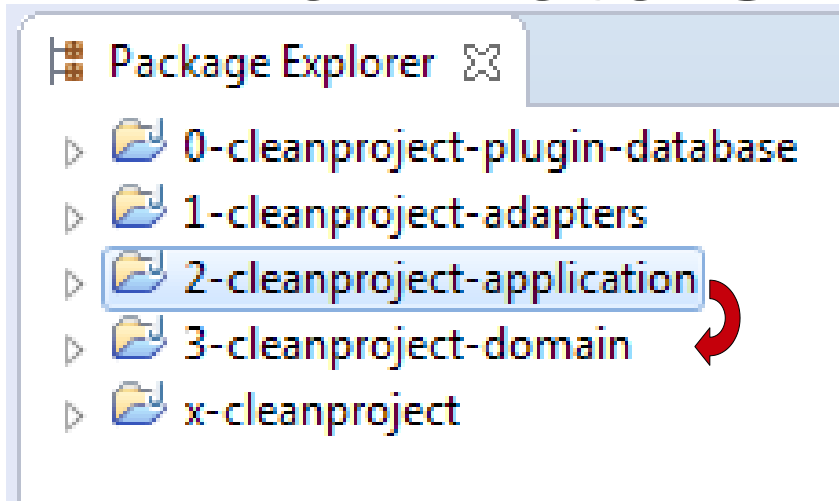
Konkrete Umsetzung: Maven



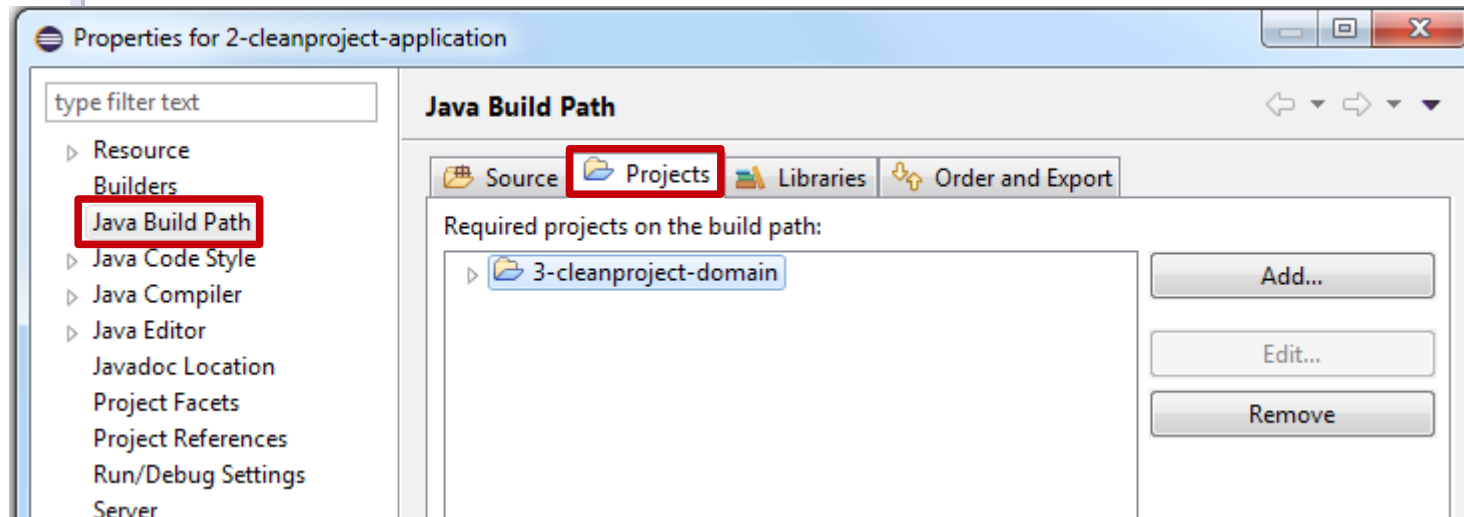
- Projekt 2 soll von Projekt 3 abhängen
- Im pom.xml als Dependency eintragen

```
<project>
  <artifactId>2-cleanproject-application</artifactId>
  <dependencies>
    <dependency>
      <artifactId>3-cleanproject-domain</artifactId>
      <groupId>de.dnb</groupId>
      <scope>compile</scope>
    </dependency>
  </dependencies>
  <parent>
    <artifactId>x-cleanproject</artifactId>
    <groupId>de.dnb</groupId>
  </parent>
  [...]
</project>
```

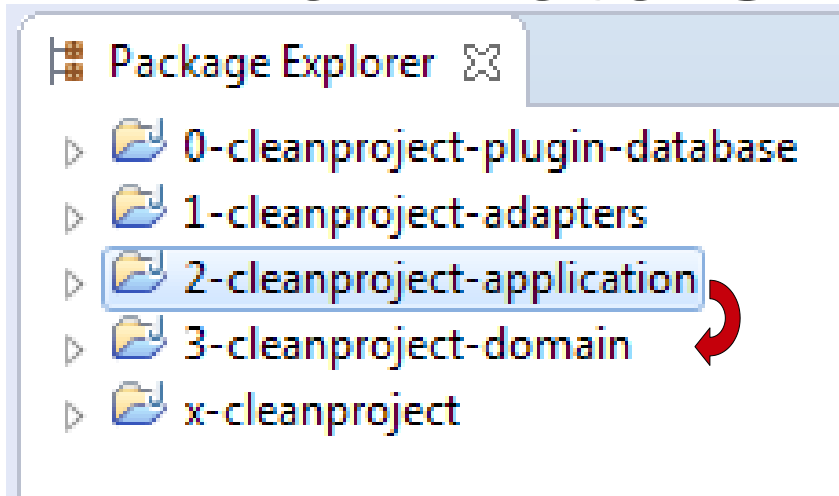

Konkrete Umsetzung: Manuell



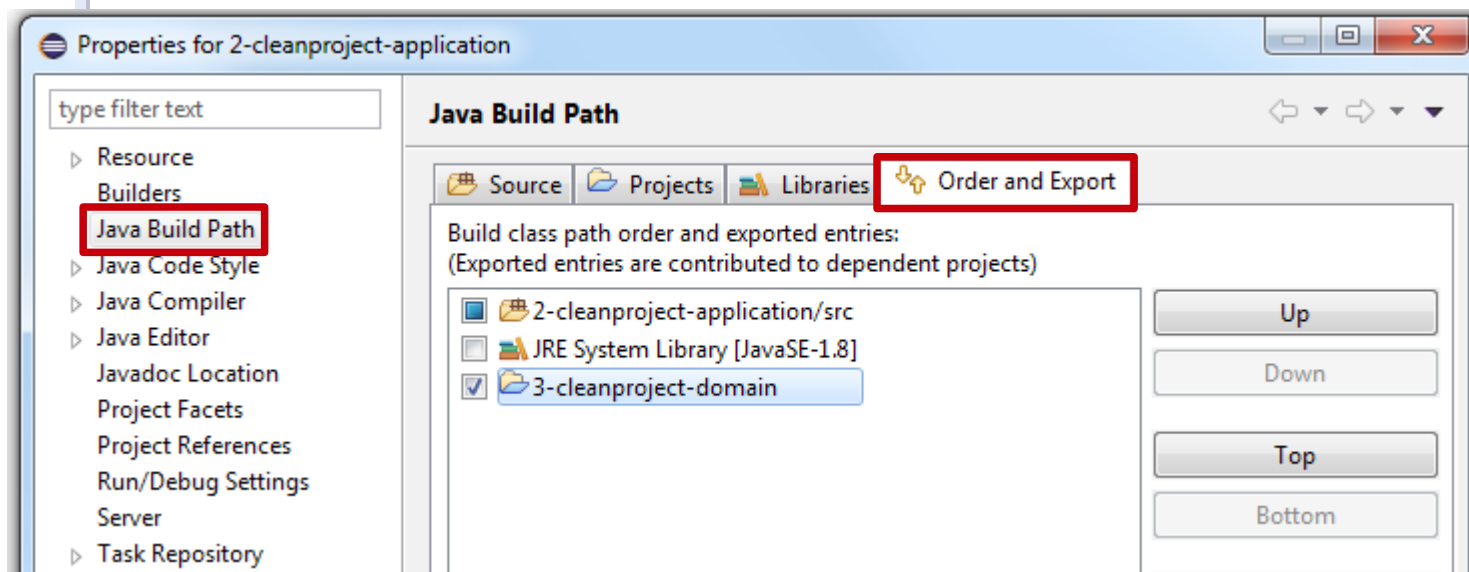
- Projekt 2 soll von Projekt 3 abhängen
- In den Eclipse-Projekteinstellungen angeben



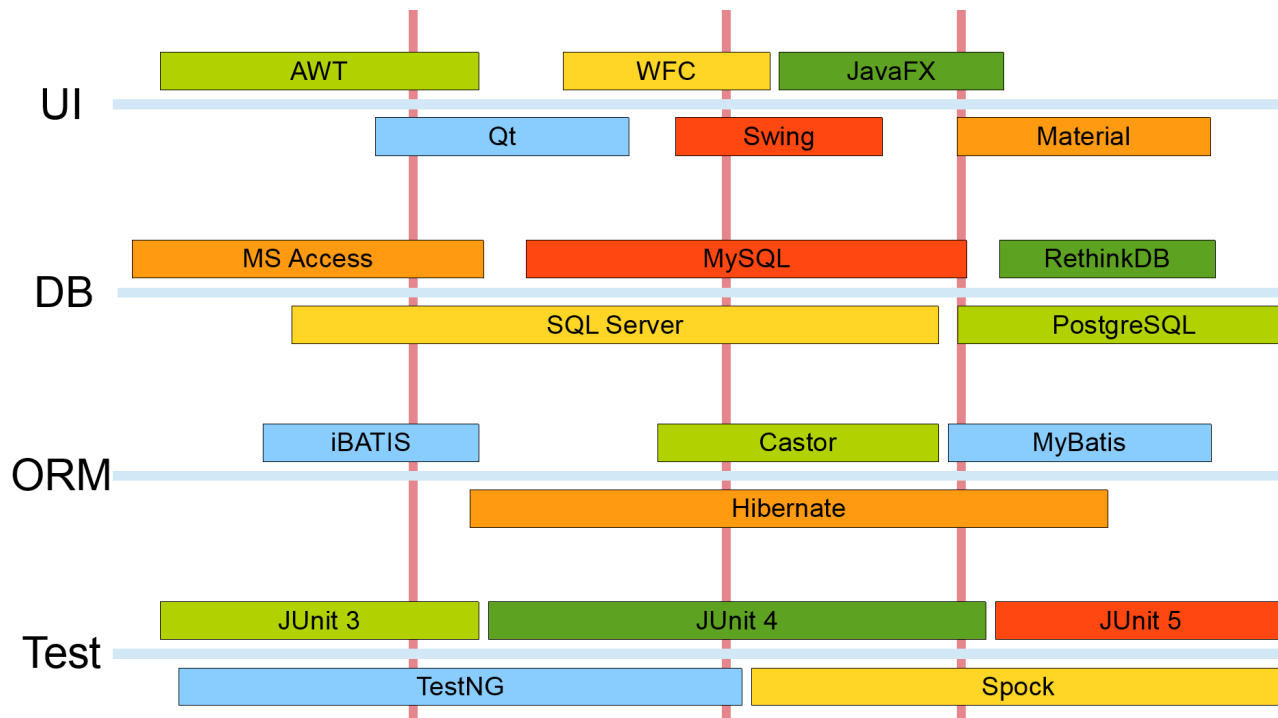
Konkrete Umsetzung: Manuell



- Transitive Abhängigkeiten freigeben
- In den Eclipse-Projekteinstellungen



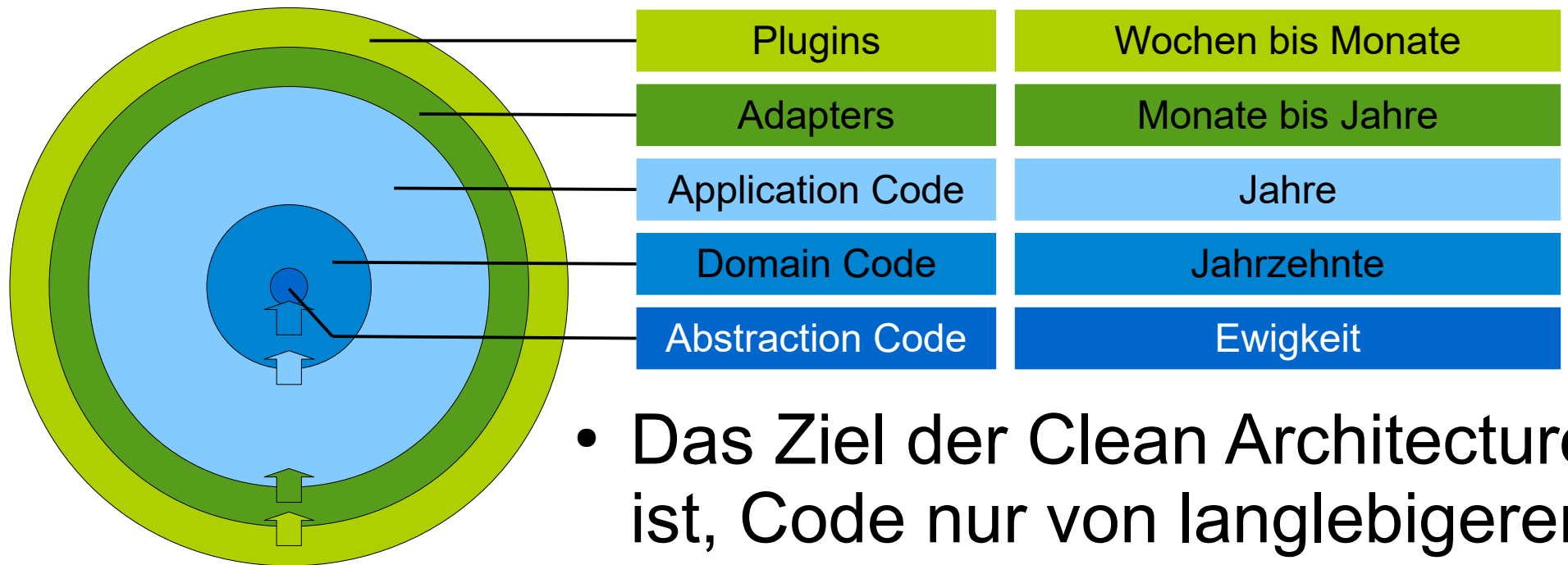
Review der Technologiewahl



UI	AWT	Swing	JavaFX
DB	MS Access	MySQL	RethinkDB
ORM	iBATIS	Hibernate	MyBatis
Test	JUnit	TestNG	Spock

- Auswahl der Technologie zu Beginn eines Projekts
- Hat starken Einfluß auf die Entwicklung
- Altert und veraltet zusammen mit der Anwendung

Ziel der Clean Architecture



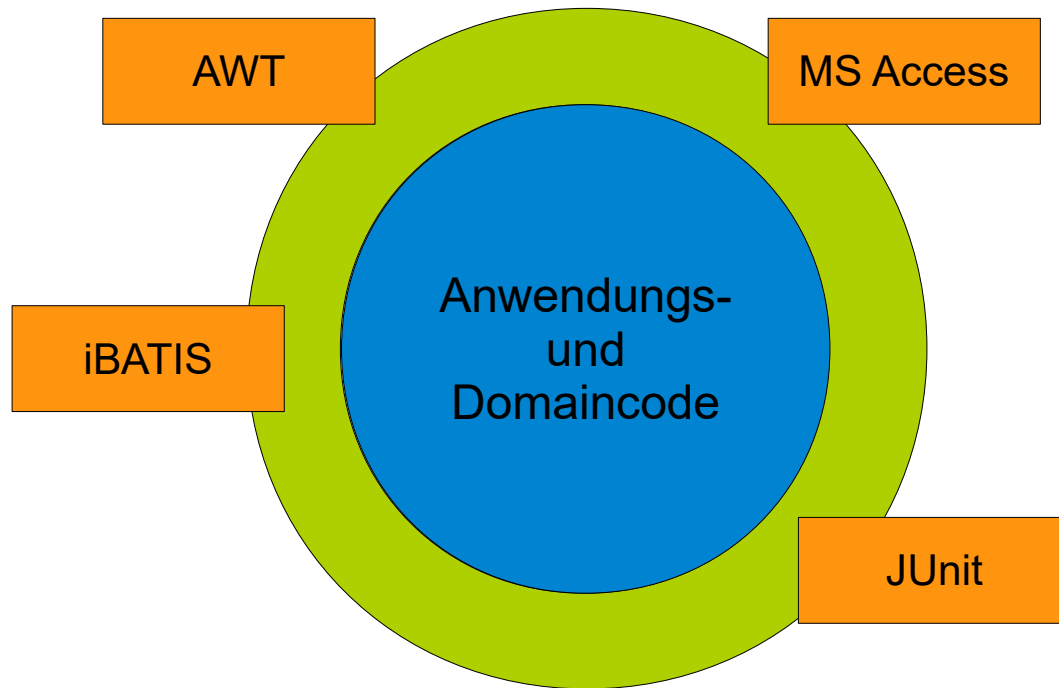
- Das Ziel der Clean Architecture ist, Code nur von langlebigerem Code abhängig zu machen
- Wenn sich Technologien ändern müssen, kann die Anwendung unverändert bleiben

Grenzen der Clean Architecture

- Technische Grundlagen müssen stabil* bleiben
 - Plattform SDK (bei Java das JDK)
 - Programmiersprache (bei Java die Java-Syntax)
 - Compiler (bei Java der javac)
 - Laufzeitumgebung (bei Java die JVM)
- Auch Betriebssystem und Hardware benötigen ausreichende Stabilität
- Das ist ein Grund, warum immer noch Cobol auf Mainframes produktiv betrieben wird

*) stabil = mindestens abwärtskompatibel

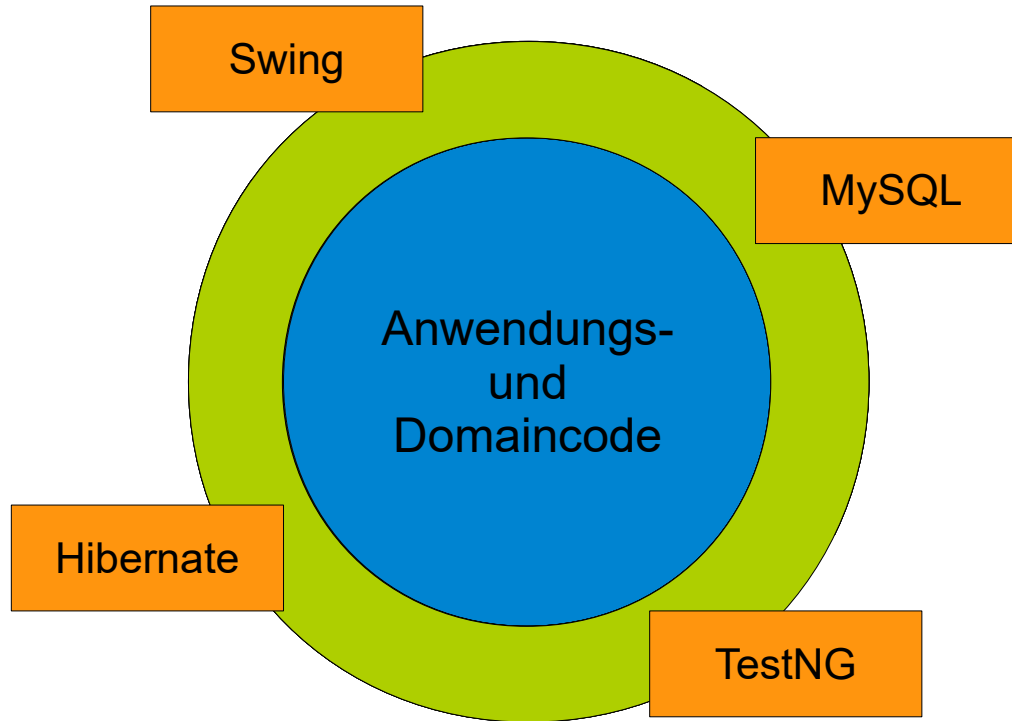
Clean Architecture Technologiewahl



UI	AWT	Swing	JavaFX
DB	MS Access	MySQL	RethinkDB
ORM	iBATIS	Hibernate	MyBatis
Test	JUnit	TestNG	Spock

- Anwendung von Technologiewahl nicht betroffen
- Konkrete Technologien sind nur noch Plugins
 - „Details“
- Können einzeln ersetzt werden

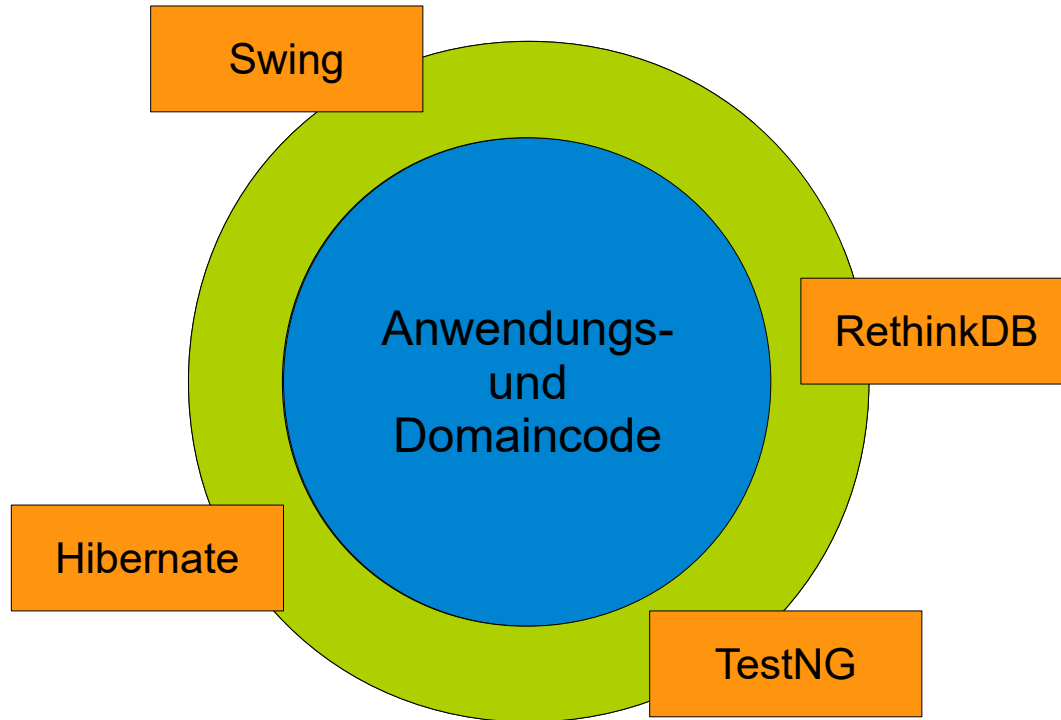
Clean Architecture Technologiewahl



- Ersetzen einer Technologie ändert die Anwendung nicht
- Adapter müssen wahrscheinlich angepasst werden
- Alle Anforderungen bleiben erhalten

UI	AWT	Swing	JavaFX
DB	MS Access	MySQL	RethinkDB
ORM	iBATIS	Hibernate	MyBatis
Test	JUnit	TestNG	Spock

Clean Architecture Technologiewahl



- Jedes Plugin kann einzeln ersetzt werden
- Keine oder nur minimale Abhängigkeiten zwischen Plugins

UI	AWT	Swing	JavaFX
DB	MS Access	MySQL	RethinkDB
ORM	iBATIS	Hibernate	MyBatis
Test	JUnit	TestNG	Spock

- Separation of Concerns

Positionierung: Beispiel 1

Use Case

Plugins

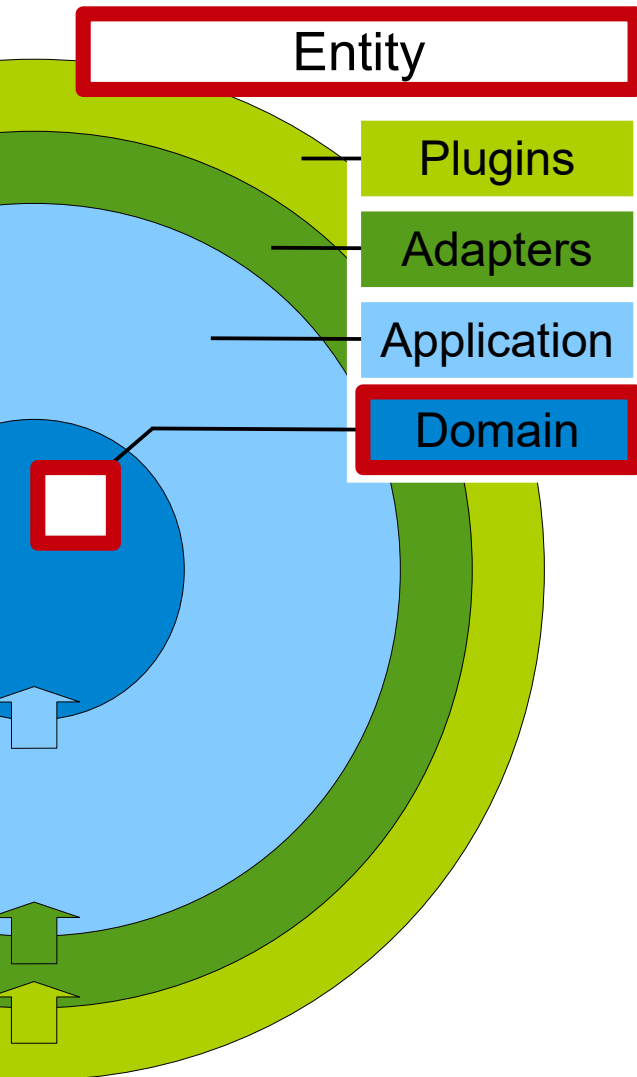
Adapters

Application

Domain

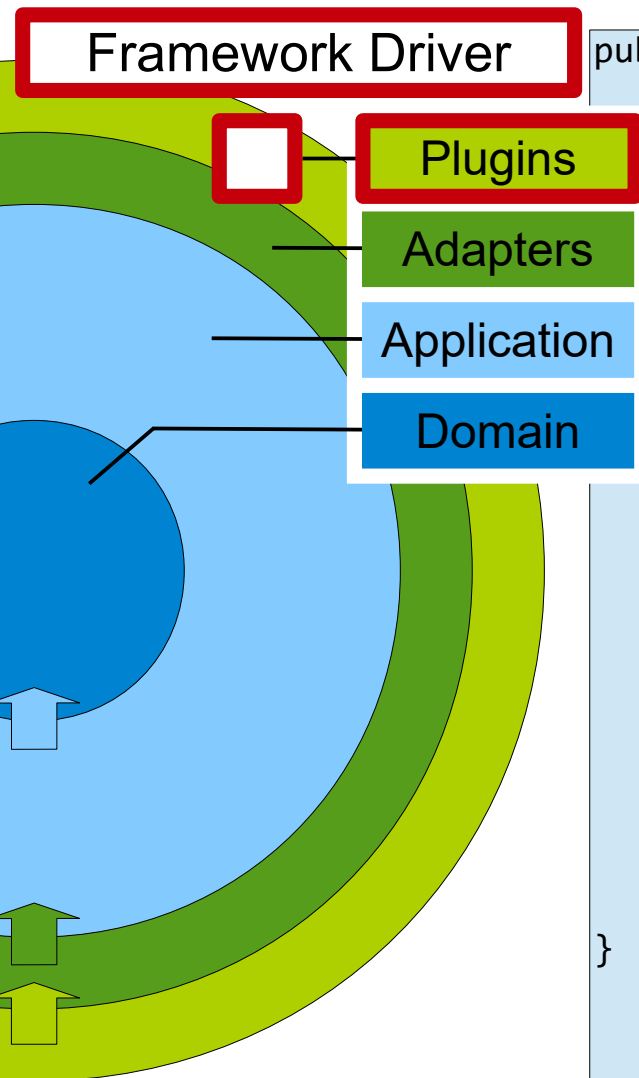
```
public class ChangeUserPassword {  
    private final AuthenticationService authenticationService;  
  
    @Inject  
    public ChangeUserPassword(  
        final AuthenticationService authenticationService) {  
        this.authenticationService = authenticationService;  
    }  
  
    @Transactional  
    public boolean changeUserPassword(  
        @NotNull final Authentication authentication,  
        @NotNull final String oldPassword,  
        @NotNull final String newPassword) {  
        final boolean oldPasswordIsValid =  
            authenticationService.checkPassword(  
                authentication,  
                oldPassword);  
  
        if (oldPasswordIsValid) {  
            authenticationService.setUserPassword(  
                authentication.getLoginName(),  
                newPassword);  
        }  
        return oldPasswordIsValid;  
    }  
}
```

Positionierung: Beispiel 2



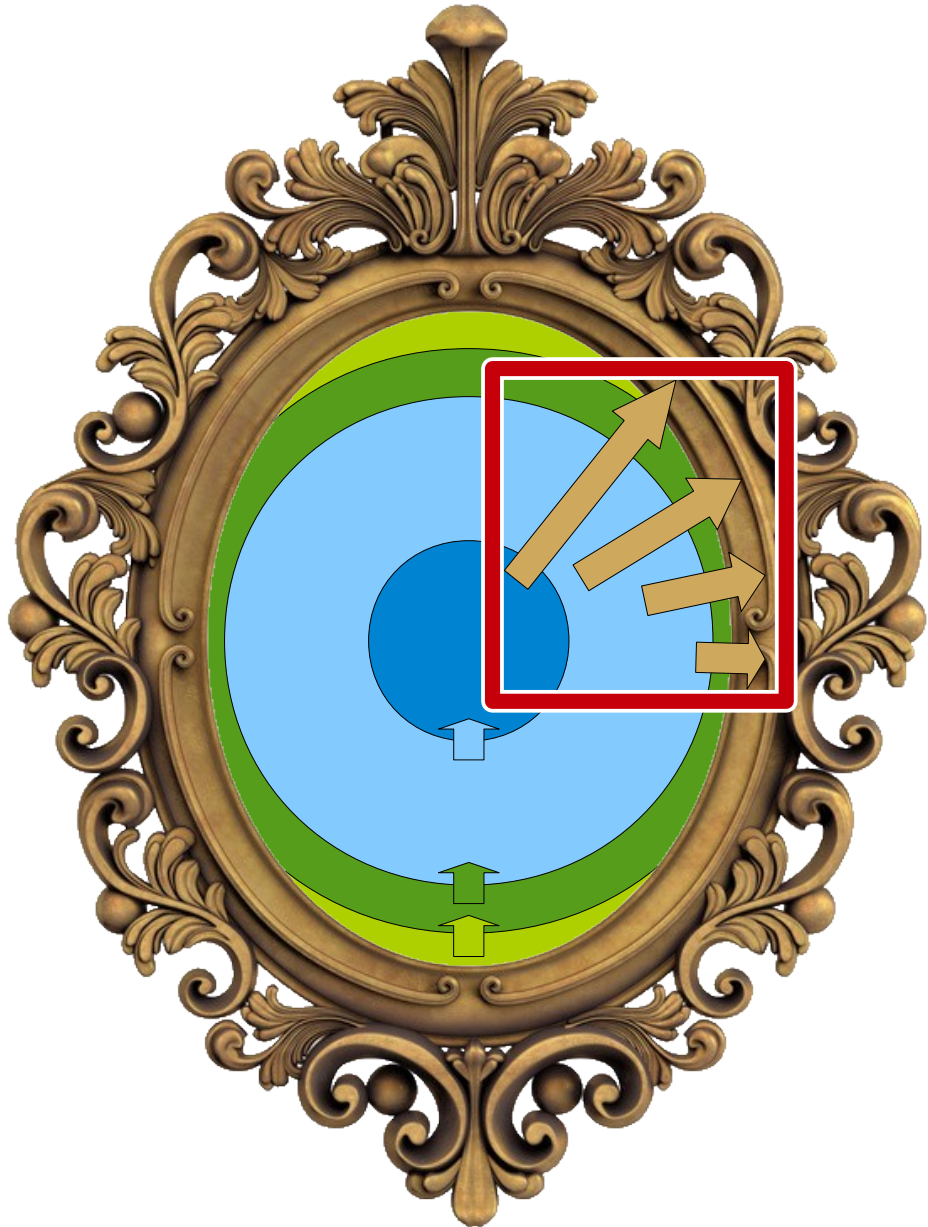
```
public class User {  
  
    @NotNull private String loginName;  
    @NotNull private String fullName;  
    @NotNull private String emailAddress;  
  
    protected User() {  
    }  
  
    public static UserBuilder create() {  
        return new UserBuilder();  
    }  
  
    public String getLoginName() {  
        return loginName;  
    }  
  
    public String getFullName() {  
        return fullName;  
    }  
  
    public String getEmailAddress() {  
        return emailAddress;  
    }  
    [...]  
}
```

Positionierung: Beispiel 3



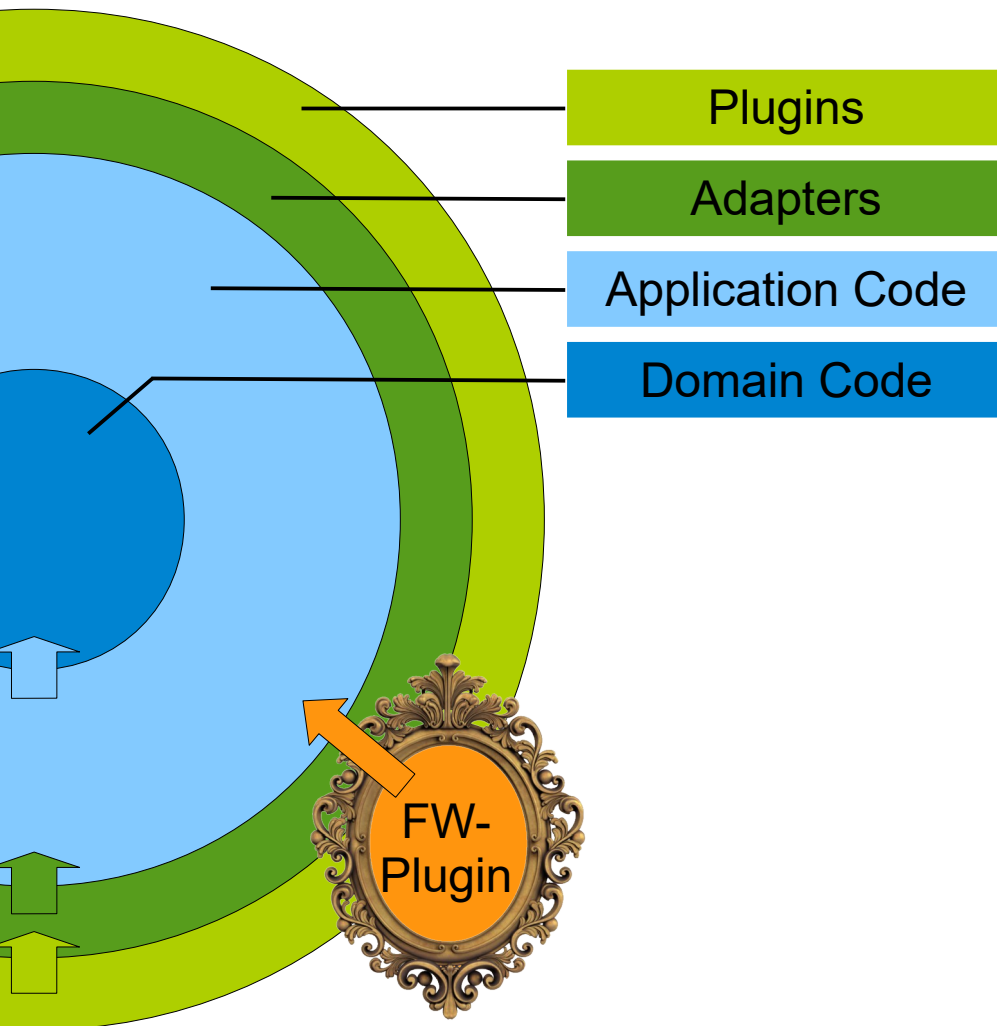
```
public class JPAAuthenticationService implements AuthenticationService {  
  
    private JPAUserEntityRepository userEntityRepository;  
    private JPAAuthenticationEntityRepository authenticationRepository;  
  
    @Inject  
    public JPAAuthenticationService(  
        final JPAUserEntityRepository userEntityRepository,  
        final JPAAuthenticationEntityRepository repository) {  
        this.userEntityRepository = userEntityRepository;  
        this.authenticationRepository = repository;  
    }  
  
    @Override  
    public boolean checkPassword(  
        @NotNull final Authentication authentication,  
        @NotNull final String password) {  
        return authenticationRepository  
            .findAuthenticationEntityByLoginNameAndPassword(  
                authentication.getLoginName(),  
                password)  
            .isPresent();  
    }  
    [...]  
}
```

Clean Architecture und Frameworks



- Frameworks streben oft die Alleinherrschaft an
- Abhängigkeiten zeigen oft vom Anwendungscode in das Framework
 - Das ist die falsche Richtung
- Abhängigkeiten immer von außen nach innen

Frameworks positionieren



- Frameworks sind Details
- Details sind Plugins
- Plugins gehören „an den Rand“ der Anwendung
- Das Framework ausfüllen heißt, Aufrufe an die Anwendung zu delegieren
- Problematisch bei Frameworks mit Metaprogrammierung

Frameworks separieren

- Am besten den Code inkl. Framework in eigenes Projekt auslagern
 - Framework-Projekt referenziert Anwendungs-Projekt
- Kern-Anwendung muss unabhängig vom Framework sein
- Die Schnittstelle für das Framework-Projekt wird eventuell sehr spezifisch ausfallen
 - Versuchung widerstehen, eine universelle Schnittstelle zu entwickeln
 - „Throwaway“-Adapter, d.h. Code, der verzichtbar ist

Beispiel: Clean Spring Boot

- Framework für die Erstellung von Standalone JVM-Anwendungen
- Basiert auf Spring-Standard („Konkurrenz“ zu JEE)
- Stellt diverse Starter-Module für verschiedene Anforderungen bereit, z.B.
 - spring-boot-starter-web
 - Spring-boot-starter-data-jpa
 - Spring-boot-starter-security
- Convention over configuration



Beispiel: Clean Spring Boot

- Problem:
 - Spring und Spring Boot bieten viel „Magie“ und neben dem Entwickler viel Arbeit ab
 - Je mehr „Magie“ man benutzt, desto mehr Kontrolle gibt man aus der Hand und desto abhängiger wird man
 - Lackmustest: kann ich spring boot entfernen, ohne dass es zu Kompilierfehlern in den Kernschichten kommt?

Beispiel: Clean Spring Boot

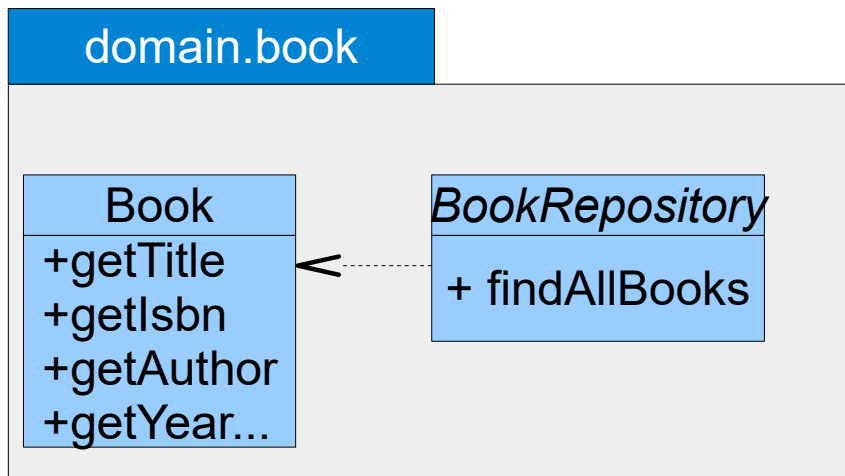
- <https://github.com/mirkobeine/cleanproject>
- Sehr einfach gehaltenes Beispiel für die Vorlesung – kein best practice-Projekt für die Praxis!
- Problemdomäne: Buchverwaltung
- (einzigster) Use Case: alle gespeicherten Bücher als JSON per REST abfragen

Beispiel: Abstraction Code

- Definiert Abhängigkeiten auf allgemein genutzte Libraries
 - **JPA 2.2:** Standard für OR-Mapping mit Java
 - **Spring-context:** Annotationen etc. für Spring Dependency Injection
 - **commons-lang3**
- Warum keine Plugins für JPA und Spring Context?
 - JPA, Spring und apache commons sind verbreitete Standards
 - In der Praxis häufig „sinnlos“, Standards als Plugins zu behandeln
 - Abhängigkeit gegen Library, nicht gegen Framework
 - Kann in voraussichtlich sehr, sehr langlebigen Projekten aber sinnvoll sein!

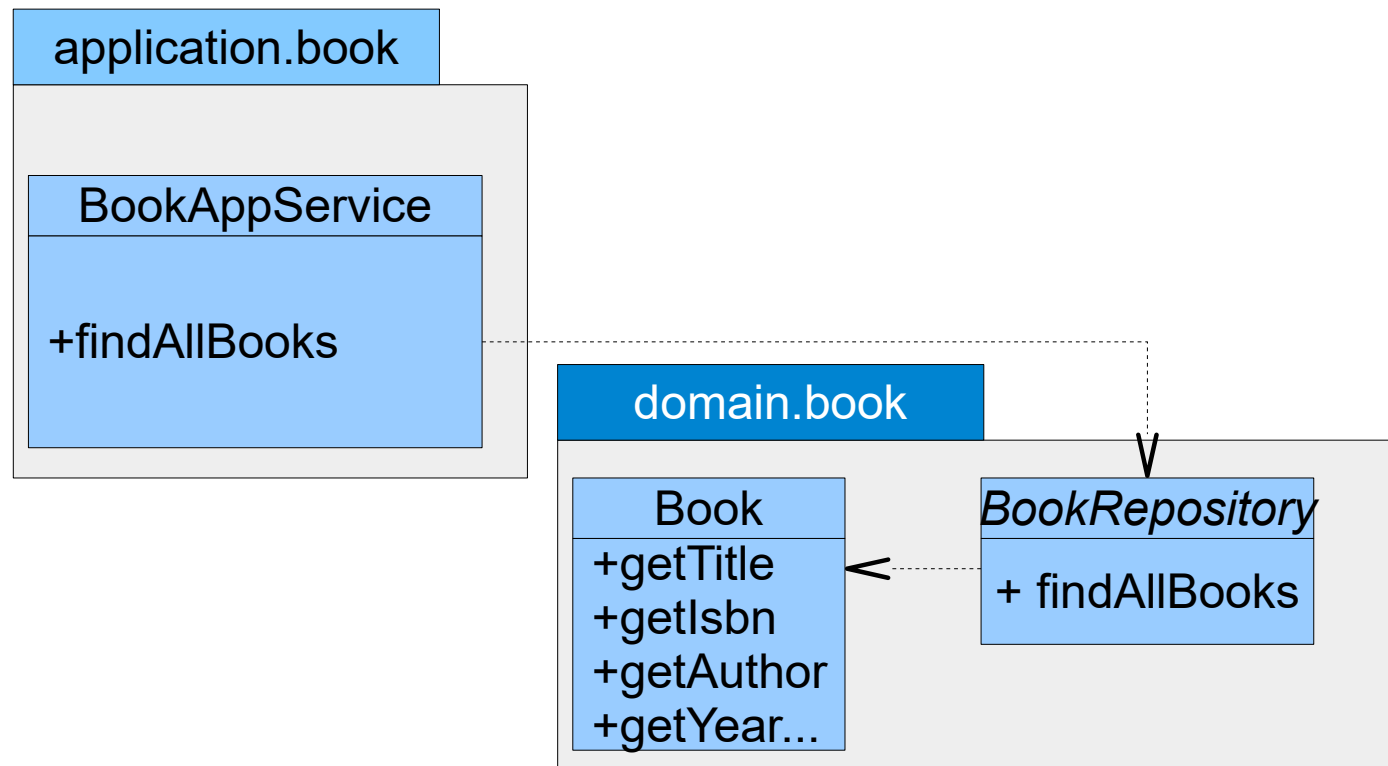
Beispiel: Domain Code

- Aggregat: Book
- Business Rule: Buch muss ISBN, Titel, Autor und Erscheinungsjahr haben (nicht null, nicht leer)
- BookRepository definiert mögliche Abfragen-unabhängig von konkretem JPA-Provider



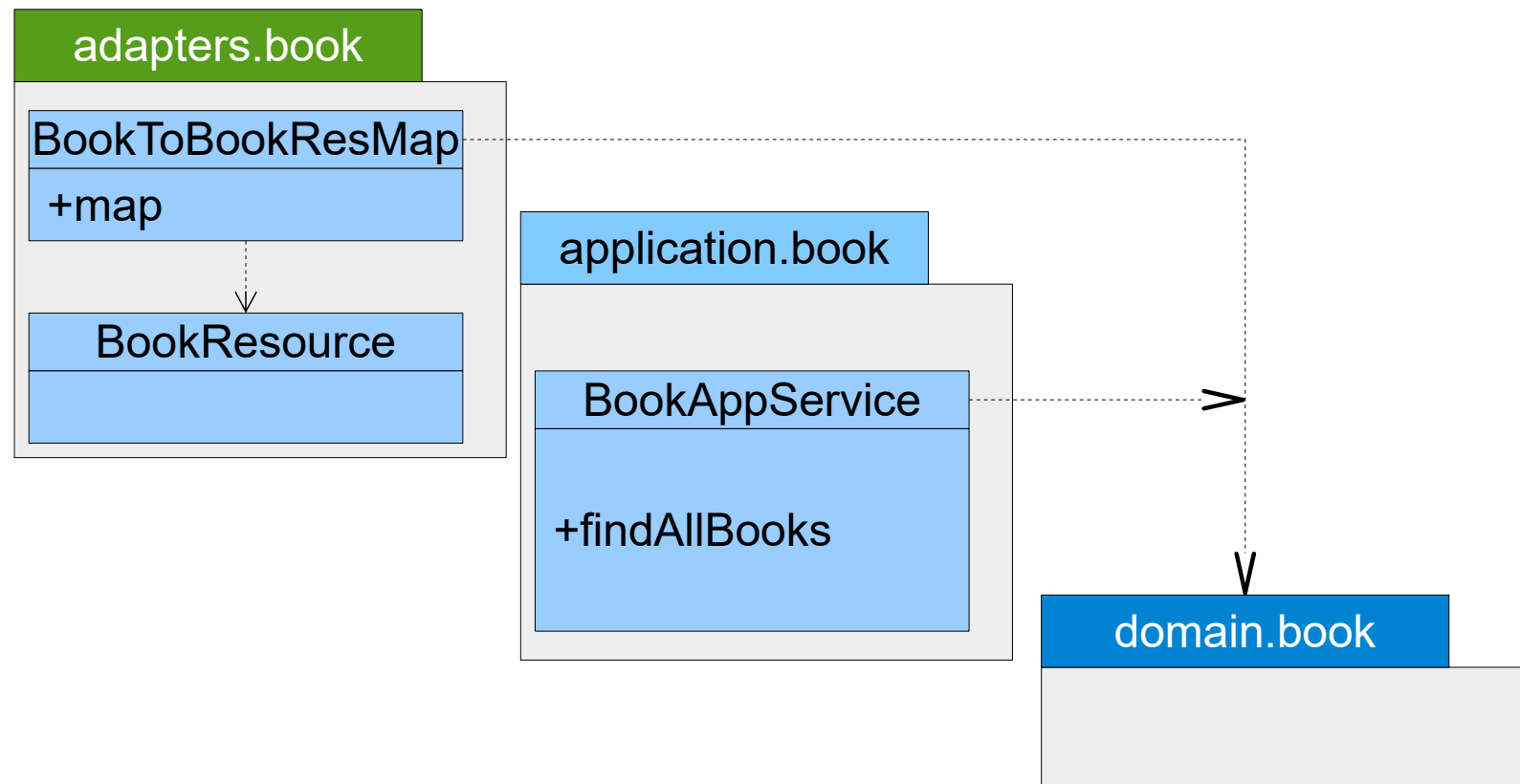
Beispiel: Application Code

- Use Case: „alle Bücher anzeigen“
- Implementierung in Klasse BookApplicationService



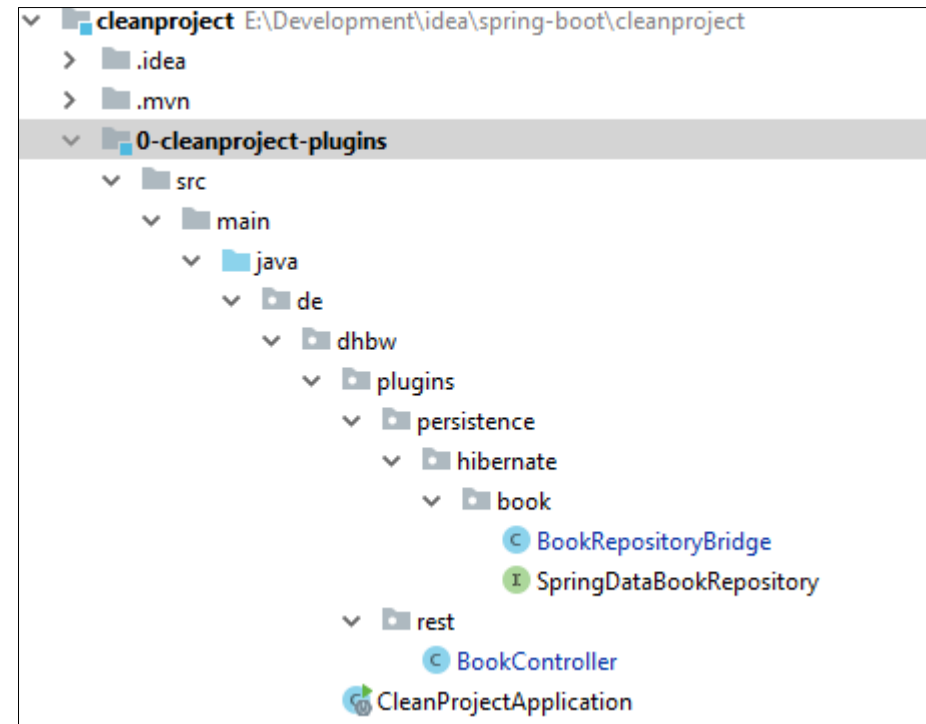
Beispiel: Adapters

- BookToBookResourceMapper wandelt Aggregat „Book“ passend für die Ausgabe (REST) um



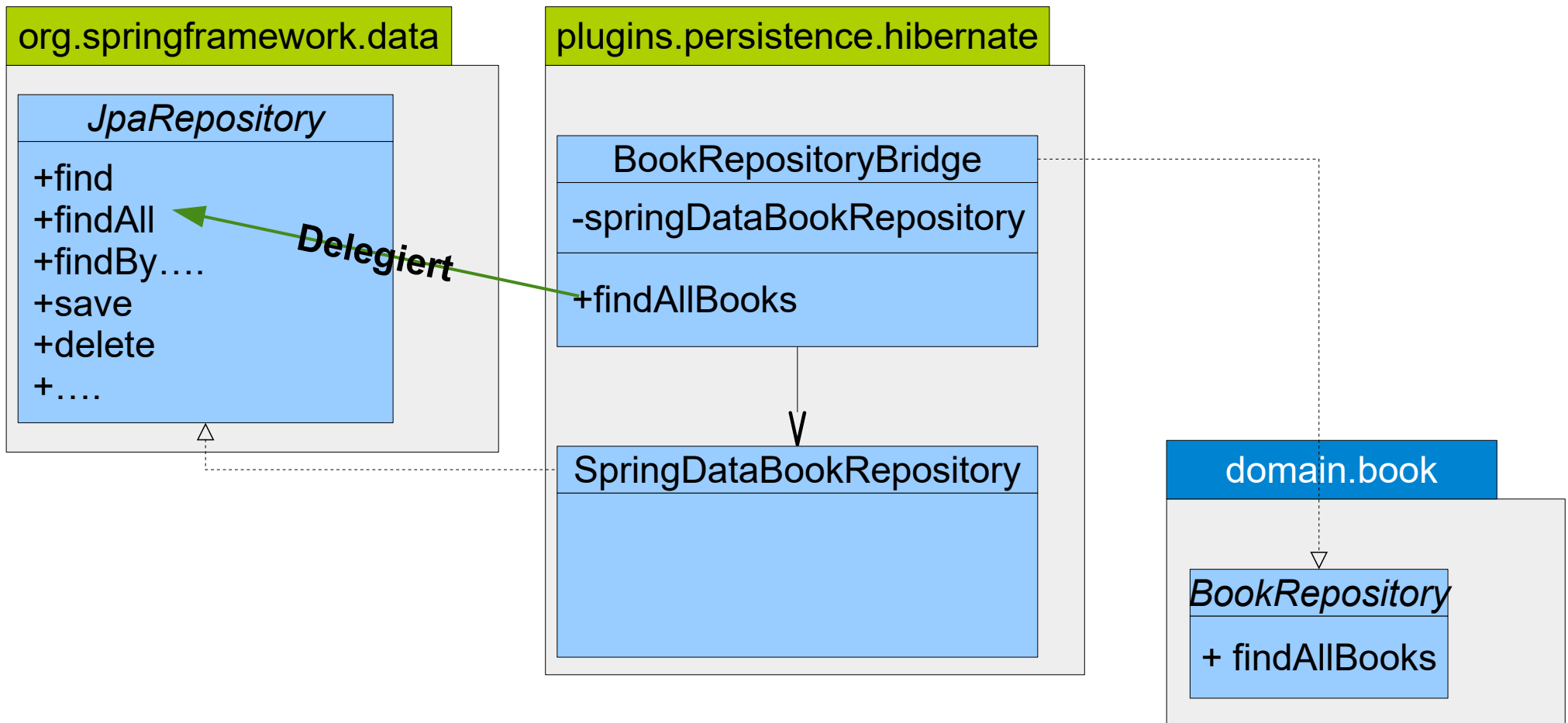
Beispiel: Plugins

- Enthält konkrete Persistenz-Implementierung, REST-Controller und SpringBoot-Application-Launcher
- Einziges Modul mit Abhängigkeit auf Framework



Beispiel: Plugins

Plugin „persistence.hibernate“:

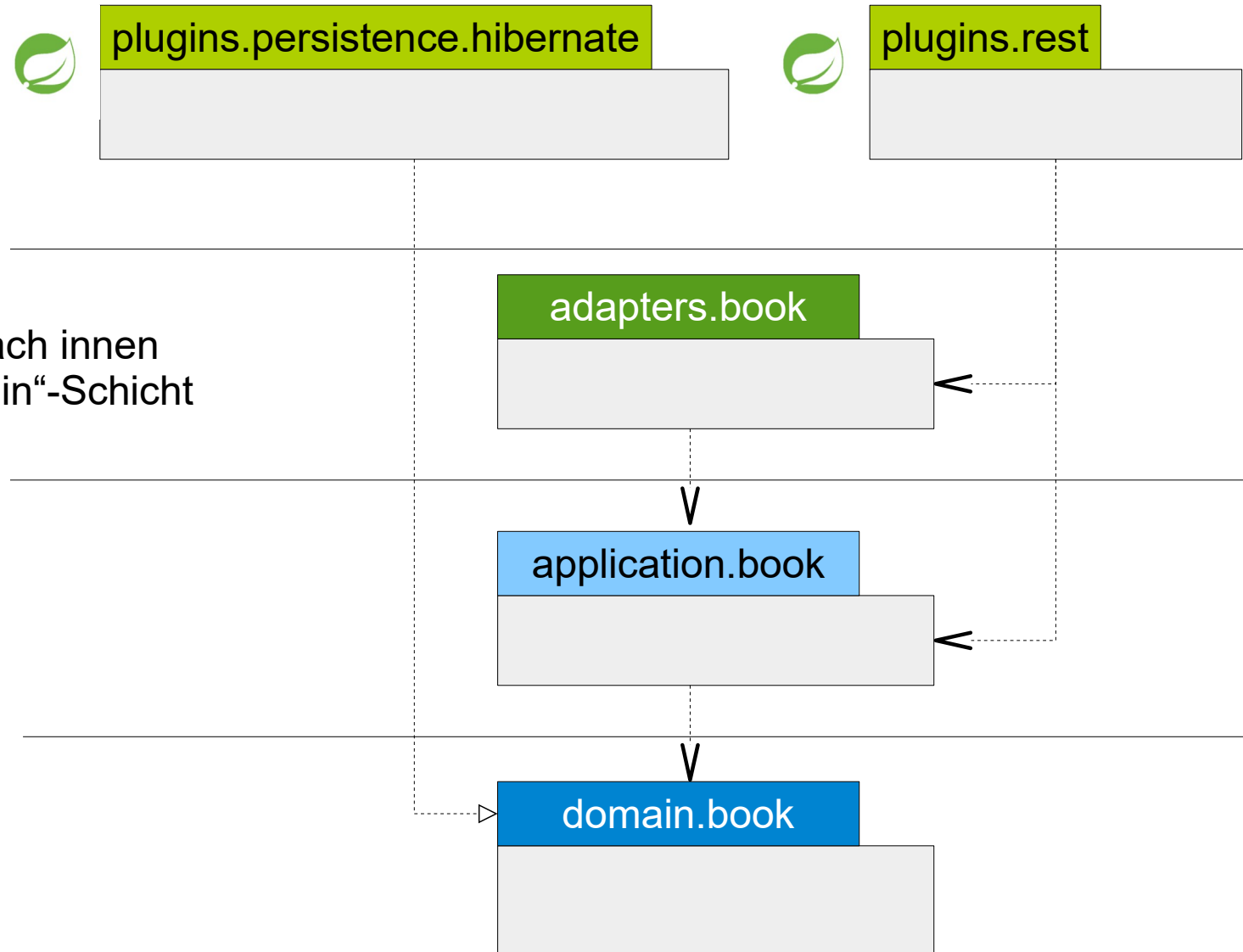


Beispiel: Plugins

Warum „BookRepositoryBridge“

- Das SpringData-Interface „JpaRepository“ liefert per „Spring-Magie“ automatisch diverse Methoden zum Abfragen/Speichern/Löschen von Entities
- Magie hat einen Preis:
 - Kopplung an vorgegebene Methodensignaturen
 - Automatische Implementierung evtl. unerwünschter Methoden (Verletzung des Interface Segregation Principle)
 - Kontrollverlust
- Die Bridge implementiert das Domain-Repository und delegiert Abfragen an das „magische“ JPA-Repository
 - Kontrollierte Magie ist OK

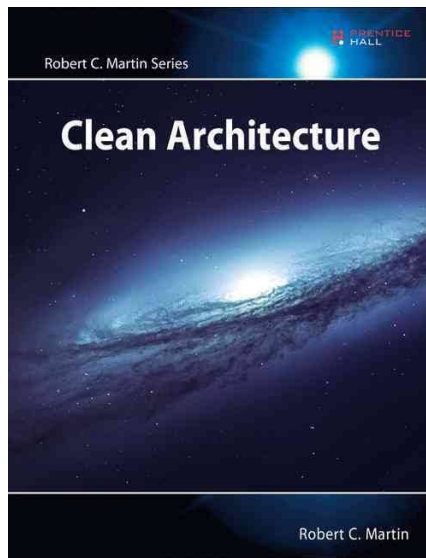
Beispiel: Clean Spring Boot



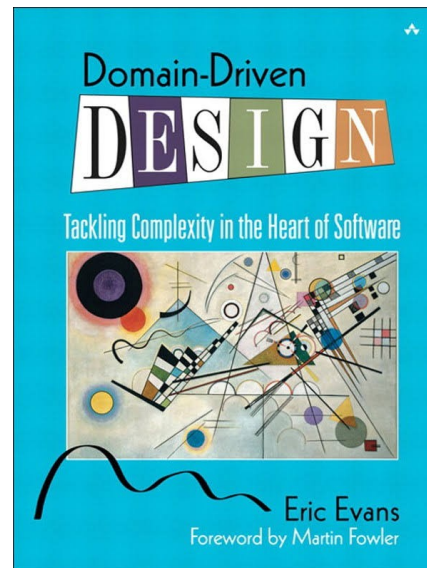
Ergebnis:

- Abhängigkeiten zeigen nach innen
- Framework liegt auf „Plugin“-Schicht
- Kein Verzicht auf „Magie“

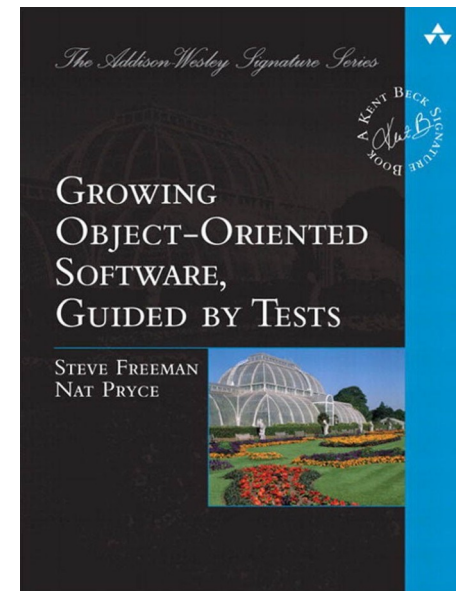
Clean Architecture: Weiterführende Literatur



2017



2004



2009

Weiterführende Web-Literatur

- Hexagonal Architecture

<http://alistair.cockburn.us/Hexagonal+architecture>

- The Onion Architecture

<http://jeffreypalermo.com/blog/the-onion-architecture-part-1>

- The Clean Architecture

<https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html>

- Layers, Onions, Ports, Adapters: it's all the same

<http://blog.ploeh.dk/2013/12/03/layers-onions-ports-adapters-its-all-the-same>

Bildnachweise

- Monolith: By Source, Fair use, <https://en.wikipedia.org/w/index.php?curid=31738209>
- Oval Baroque Gold Frame: Fotolia Datei #77261068 | Urheber: dmitrygolikov
- Ausmalbuch: <http://www.traum-salon.de/pages/buecher/uebersicht/herr-wolke-lese-raetsel-ausmalbuch.php>
- Trend für Stressabbau - Ausmalbuch für Erwachsene: Fotolia Datei: #102219361 | Urheber: moltaprop
- Bricklayer worker installing brick masonry on exterior wall: Fotolia Datei: #117356924 | Urheber: Hoda Bogdan
- Suspicious Looking Device: <http://art.junkfunnel.com/?p=83> by Junkfunnel Labs (Casey Smith)
- Two cogwheels configuration interface symbol: <div>Icons made by Freepik from www.flaticon.com is licensed by CC 3.0 BY</div>
- Moore neighborhood with cardinal directions: Von MorningLemon - Eigenes Werk, CC-BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=38746075>
- Pont du Gard, Benh LIEU SONG [CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0>)]