DHBW
Duale Hochschule
Baden-Württemberg

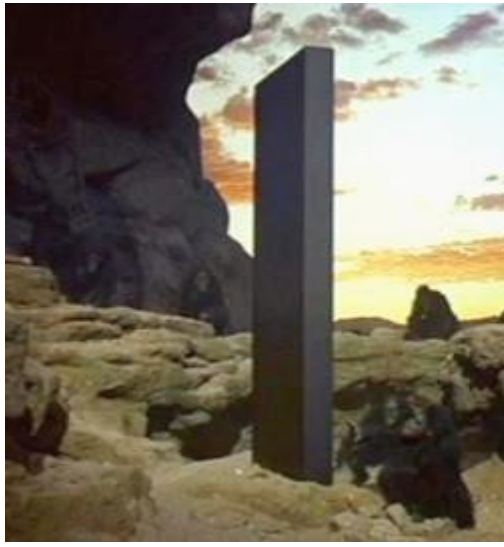
Clean Architecture

Systeme für die Ewigkeit

Softwareentwicklung

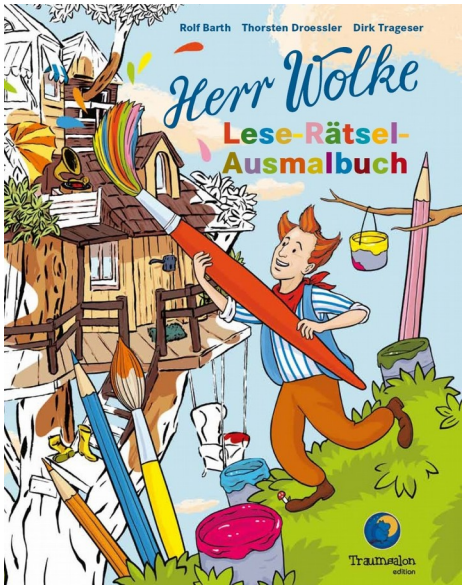
- Beständiger Wandel alle fünf Jahre
  - Kein zentraler Takt
  - Unterschiedliche Zykluslänge von Produkten

- 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



Framework vs. Library

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

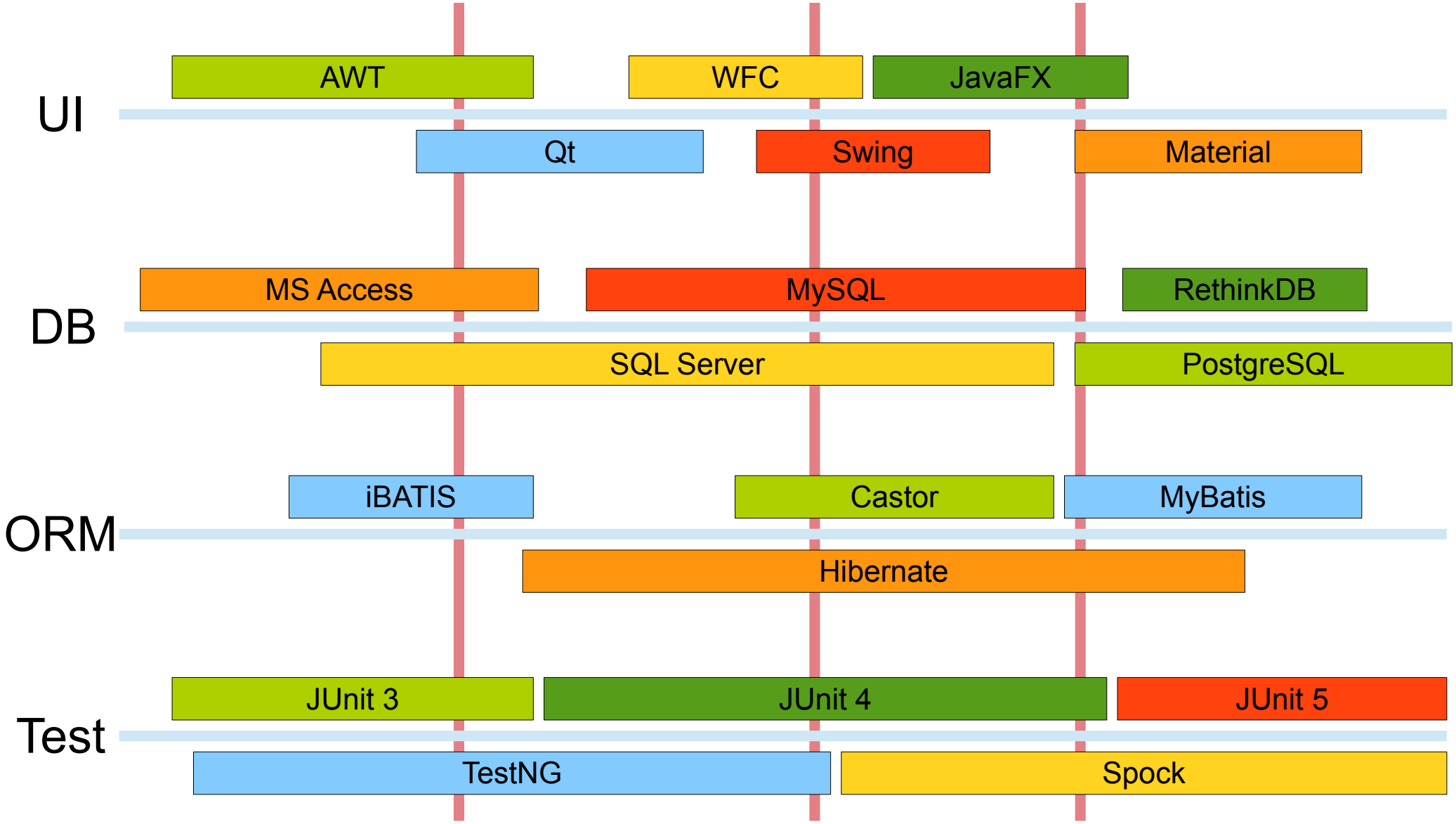
http://art.junkfunnel.com/?p=83

Framework vs. Library

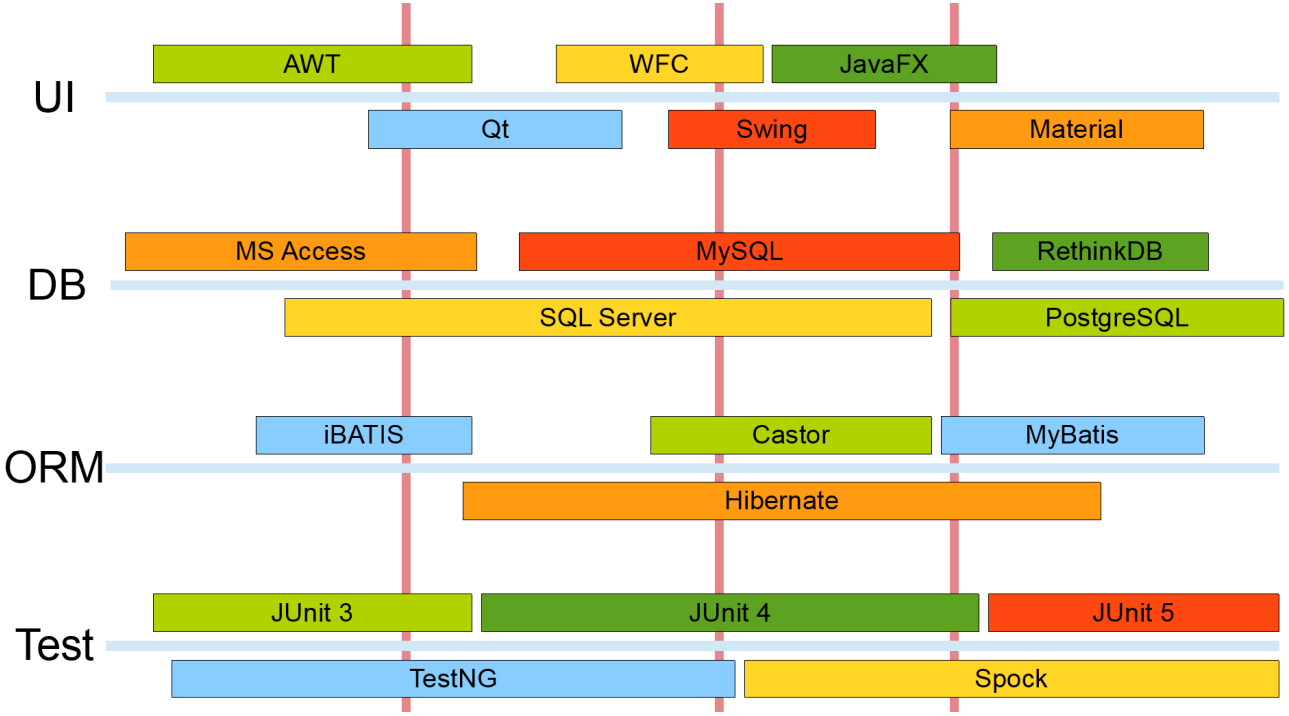
- Frameworks binden die Anwendung an sich
  - Starke Kopplung (Vendor lock-in)
  - Kopplung auch an den Lebenszyklus
- Libraries lassen mehr Freiheiten
  - Starke Kopplung vermeidbar

- Lieber Libraries als Frameworks verwenden
- Frameworks nicht „wie gedacht“ einsetzen

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

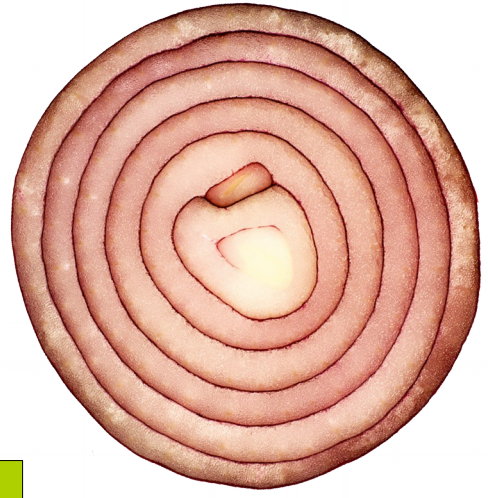
- Gute Entscheidungen werden spät getroffen

- Strukturen (Architektur) so wählen, dass Entscheidungen verzögert werden können
  - Ohne negative Folgen

- Minimalziel: Entscheidungen revidieren können
  - Mit möglichst geringen negativen Folgen

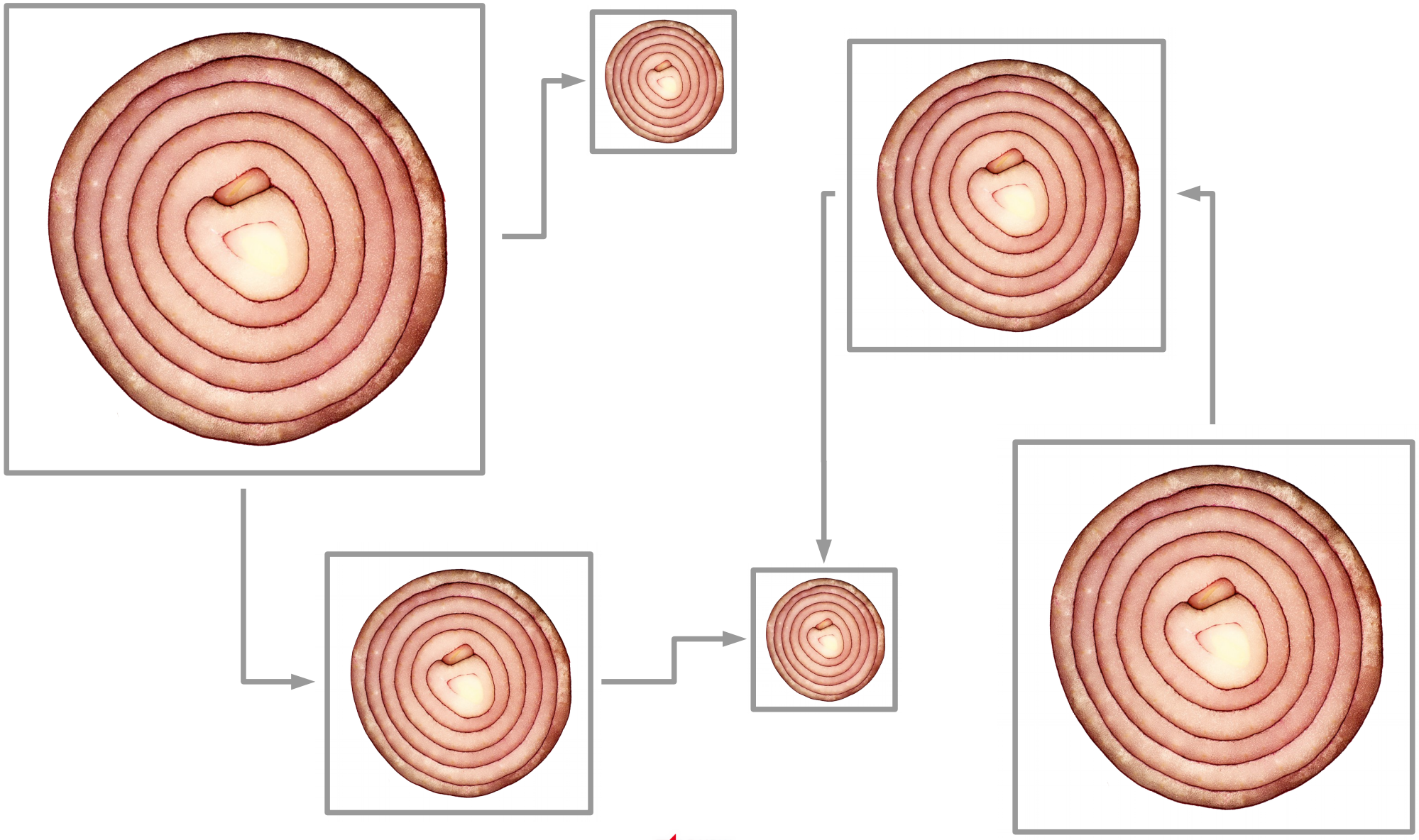
Kriterien für nachhaltige Architektur

- 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“

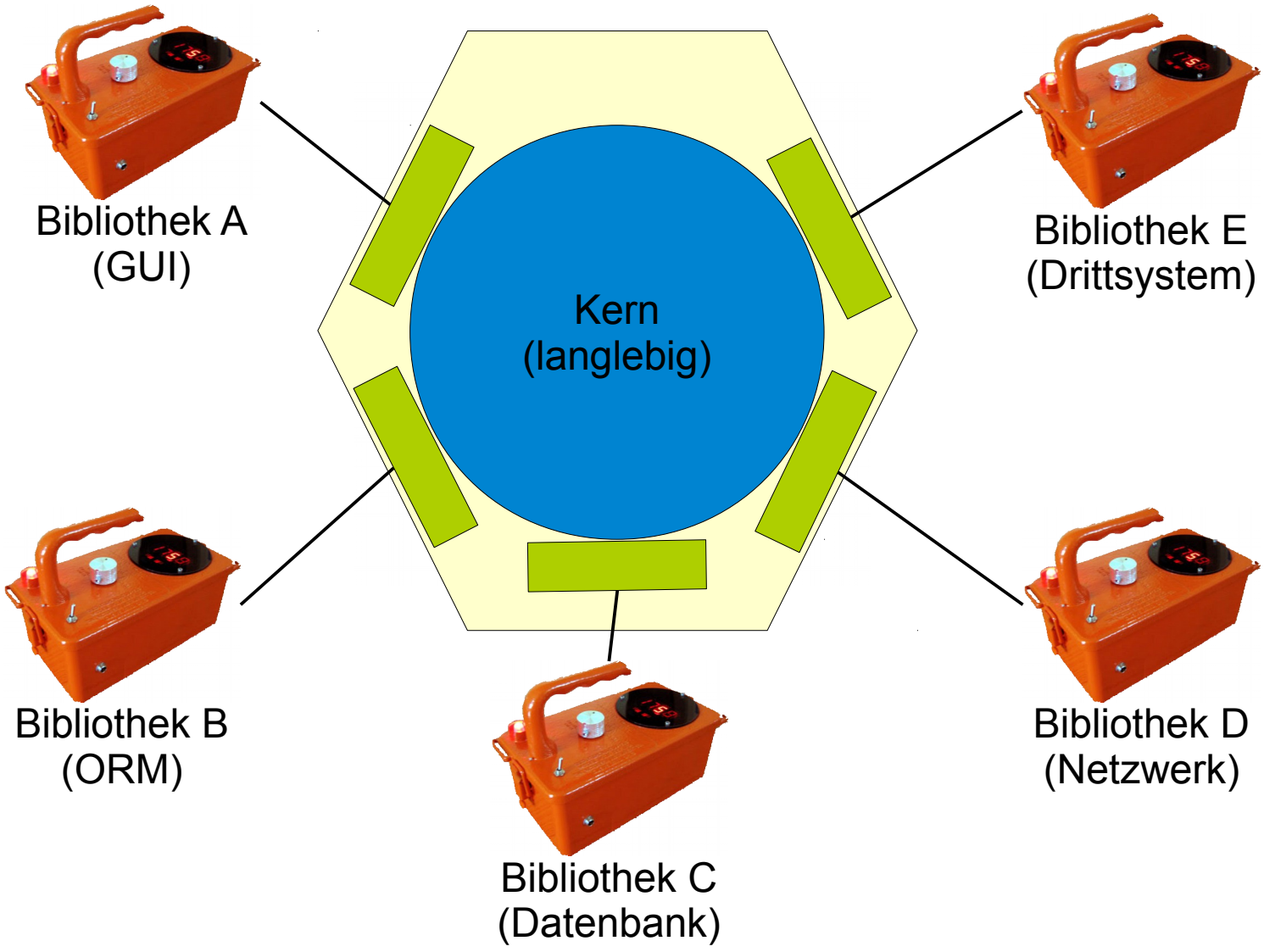


http://jeffreypalermo.com/blog/the-onion-architecture-part-1

Eine Architektur pro Anwendung

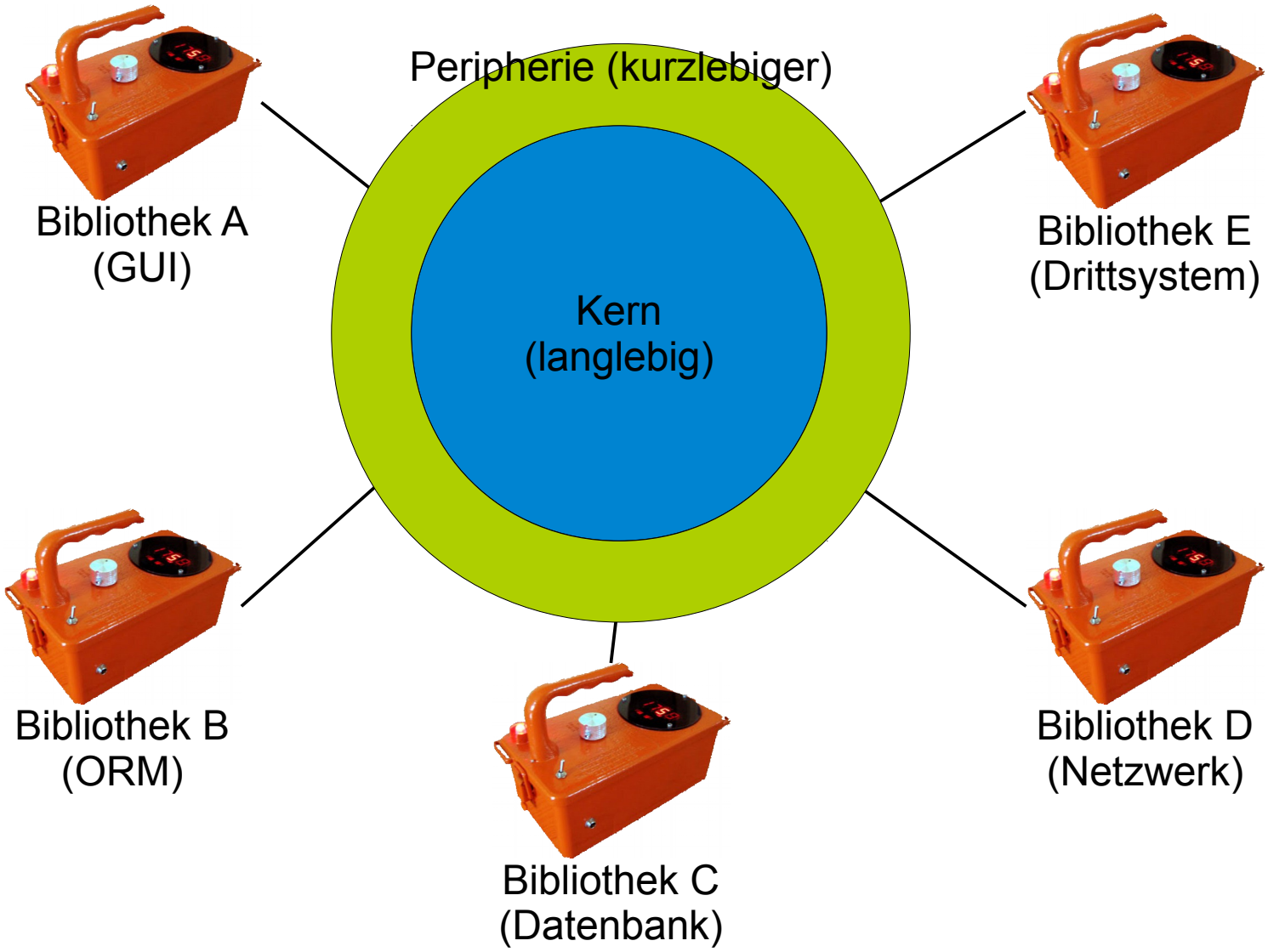


Struktur der Clean Architecture

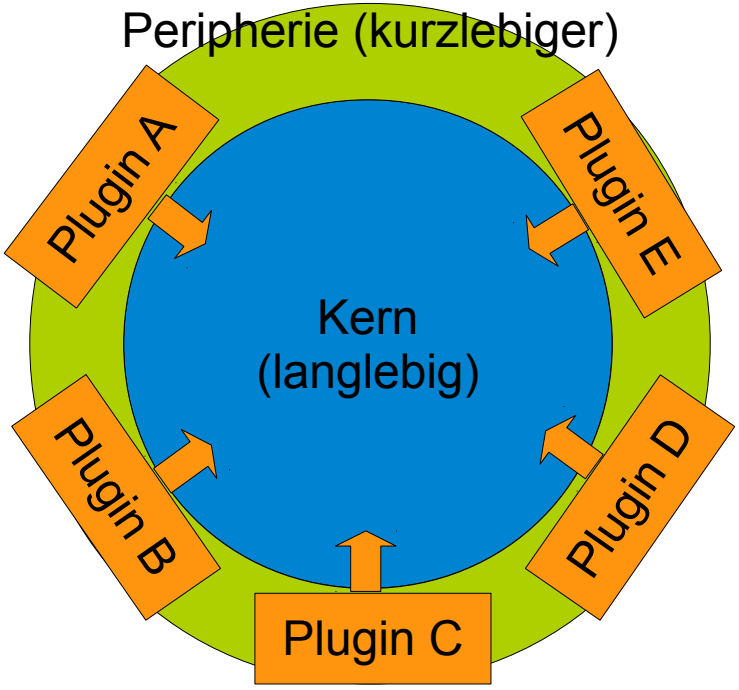


http://alistair.cockburn.us/Hexagonal+architecture

Struktur der Clean Architecture



Struktur der Clean Architecture



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

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

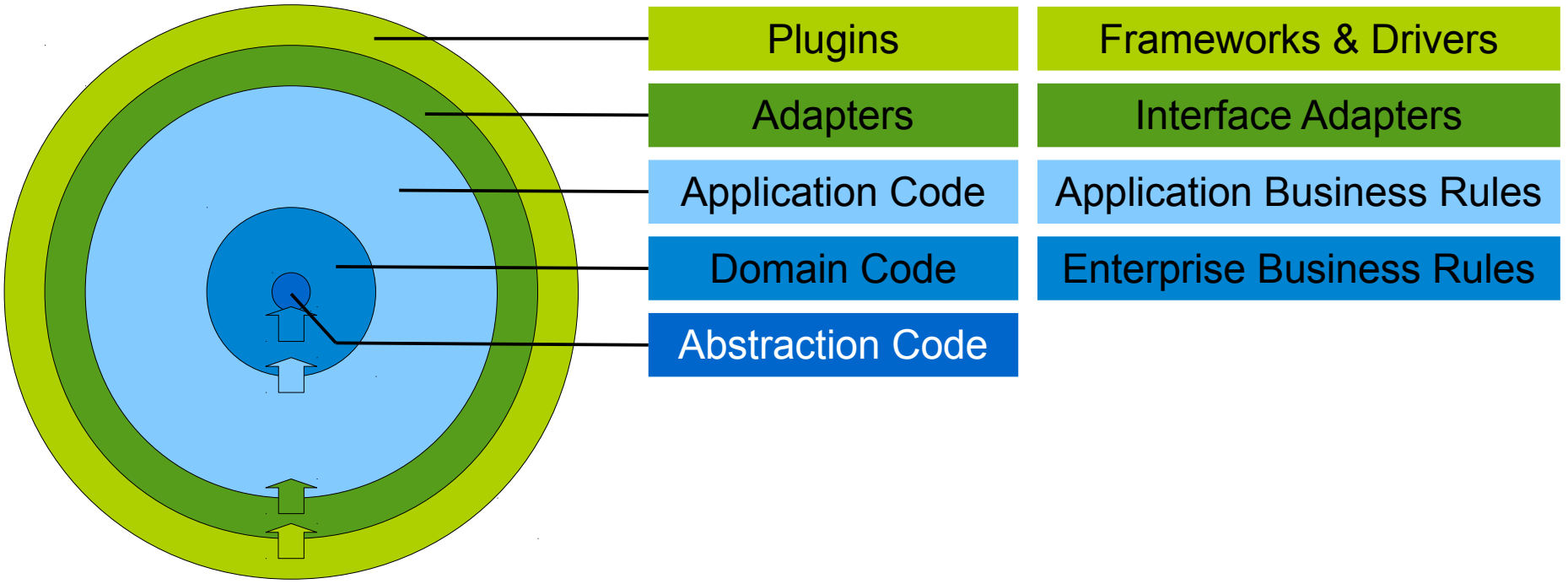
Die Dependency Rule

- Zentrale Regel für Abhängigkeiten

Abhängigkeiten immer von außen nach innen

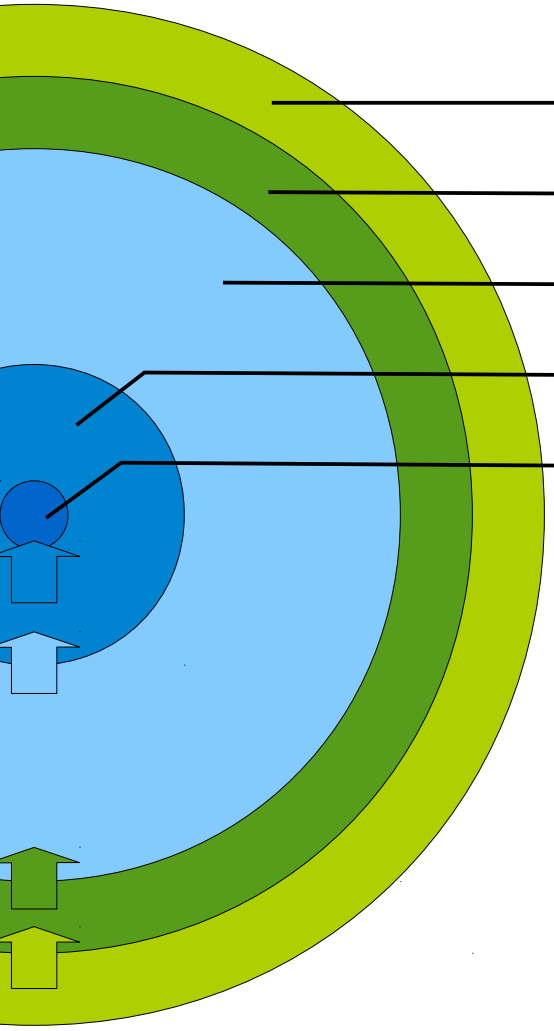
- Erfordert für jede Klasse eine klare Positionierung
- Abhängigkeitspfeile gehen immer von außen nach innen
  - Aufrufpfeile können in beide Richtungen gehen

Struktur der Clean Architecture



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

Position der Elemente



Plugins	Datenbank, GUI, Web, Drittsysteme, Geräte
Adapters	Presenters, Controllers, Gateways
Application Code	Use Cases
Domain Code	Entities
Abstraction Code	Generic Entities (mathematische Konzepte)

- Die Position eines Elements wird durch seinen Einsatzzweck bestimmt
  - Nicht durch technische Bequemlichkeit
- Je konkreter („low level“) der Code ist, desto weiter außen liegt er

Grundregeln der Clean Architecture

- Der Anwendungs- und Domaincode ist frei von Abhängigkeiten
  - 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)

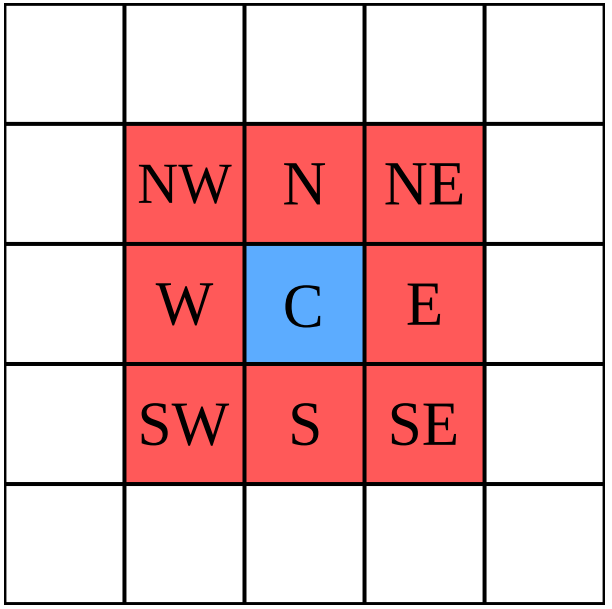
http://jeffreypalermo.com/blog/the-onion-architecture-part-3

Schicht 4: Abstraction Code

- Enthält domänenübergreifendes Wissen
  - Mathematische Konzepte (z.B. Matrizen)
  - Algorithmen und Datenstrukturen (z.B. Zelluläre Automaten)
  - Abstrahierte Muster (z.B. Quantitäten)
- Häufig nicht notwendig und/oder nicht vorhanden
- Wahrscheinlich bereits als Library verfügbar
- Kann nachträglich extrahiert werden
- Nicht aus Angeberei anlegen!

Beispiel für Abstraction Code

- Bei Minesweeper gibt es das Konzept der benachbarten Zellen
- Ist tatsächlich ein universelles Konstrukt, die „Moore-Nachbarschaft“
- Wird auch in vielen zellulären Automaten verwendet
- Es gibt auch die Von-Neumann-Nachbarschaft mit maximal vier Nachbarn



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
- Hohes emotionales Investment der Entwickler

Beispiel für Domain Code

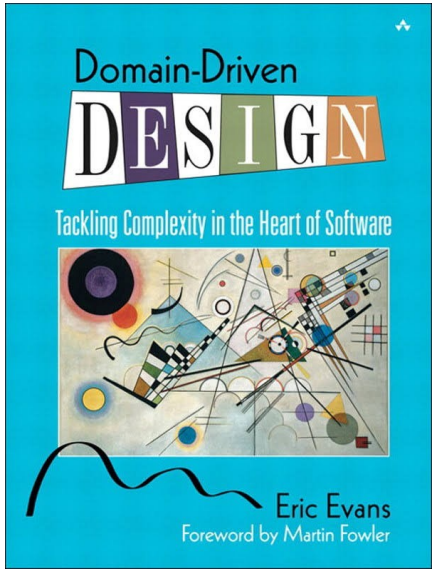
- Domäne: Bankkonto-Verwaltungssoftware
- Ein zentraler Begriff ist das „Konto“
- Jedes Konto muss der zentralen Regel genügen:

  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

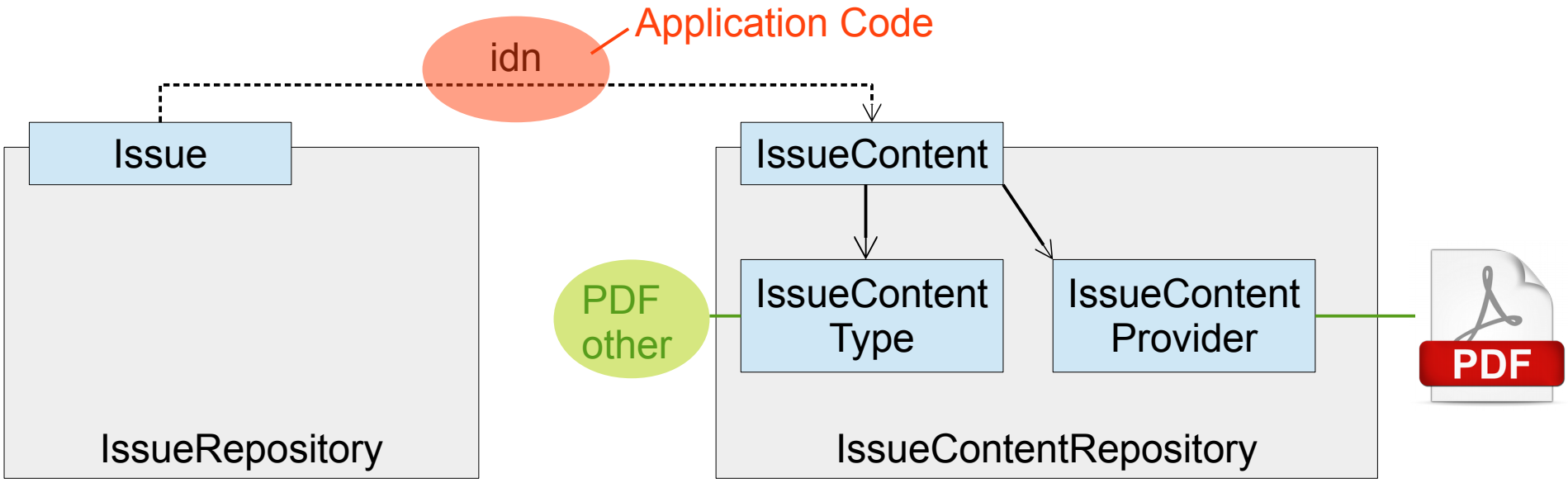
DDD und Domain Code

- Domain Driven Design (DDD) Aggregate sind typische Strukturen im Domain Code
  - Bestehen aus DDD Entities → Teil des Domain Code
  - Werden durch DDD Repositories verwaltet → Teil des Domain Code
- Weitere Bestandteile:
  - Verhalten der Entities
  - Verhalten im Aggregat
- Aggregat-übergreifendes Verhalten ist dann Bestandteil der Use Cases



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
- Implementiert die anwendungsspezifische Geschäftslogik
  - Application-specific Business Rules
- Steuert den Fluss der Daten und Aktionen von und zu den Entities
  - Verwendet die Geschäftslogik, um den jeweiligen Anwendungsfall umzusetzen

Schicht 2: Application Code

- Änderungen an dieser Schicht beeinflussen die Schicht 3 (v.a. die Entities) nicht
- Isoliert von Änderungen an der Datenbank, der graphischen Benutzeroberfläche, etc.
- Wenn sich Anforderungen ändern, hat das wahrscheinlich Auswirkungen auf diese Schicht
- Wenn sich der konkrete Betrieb der Anwendung ändert, kann das hier Auswirkungen haben
- Emotionale Bindung an den Code ist in Ordnung

Beispiel für Application Code

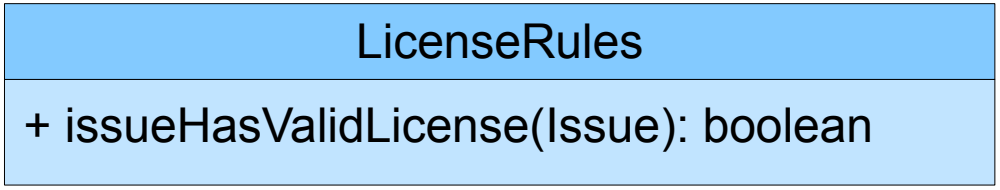
- Bankkonto-Verwaltungssoftware
- Zentraler Use Case: Überweisungen
  - Abbuchung von Konto 1, Zubuchung auf Konto 2
  - Auch hier muss eine wichtige Regel gelten:

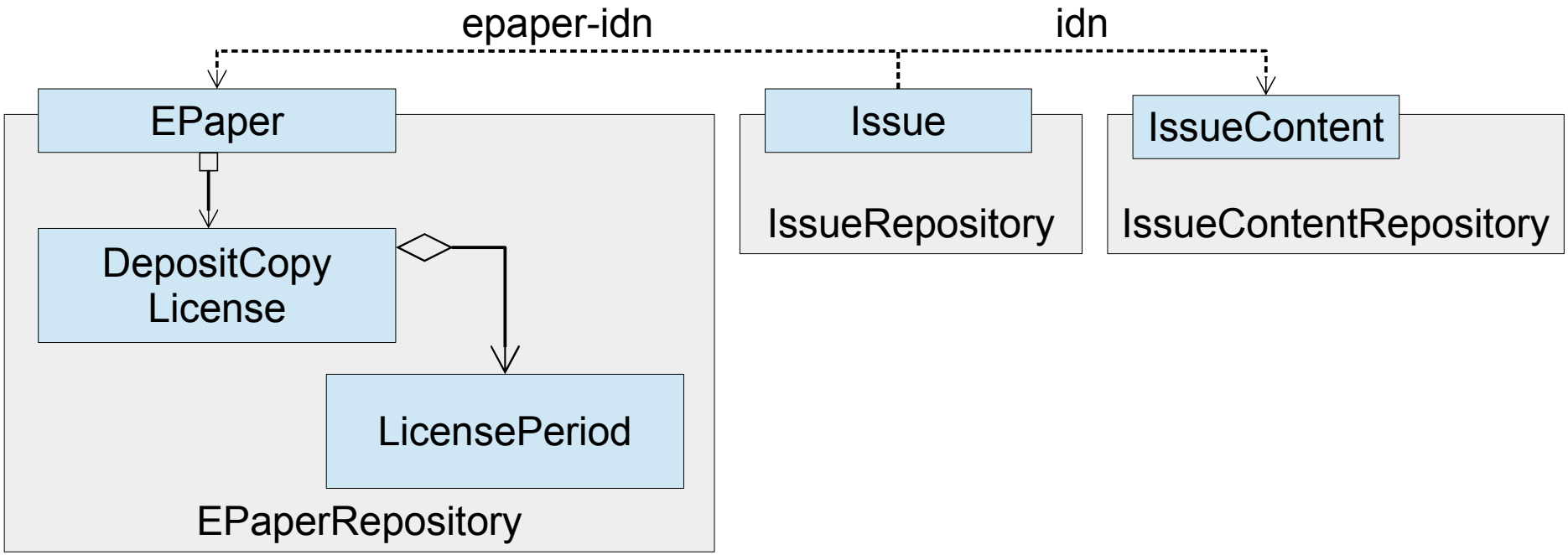
Die Summe aus Abbuchung und Zubuchung ergibt immer 0

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

Beispiel für DDD und Application Code

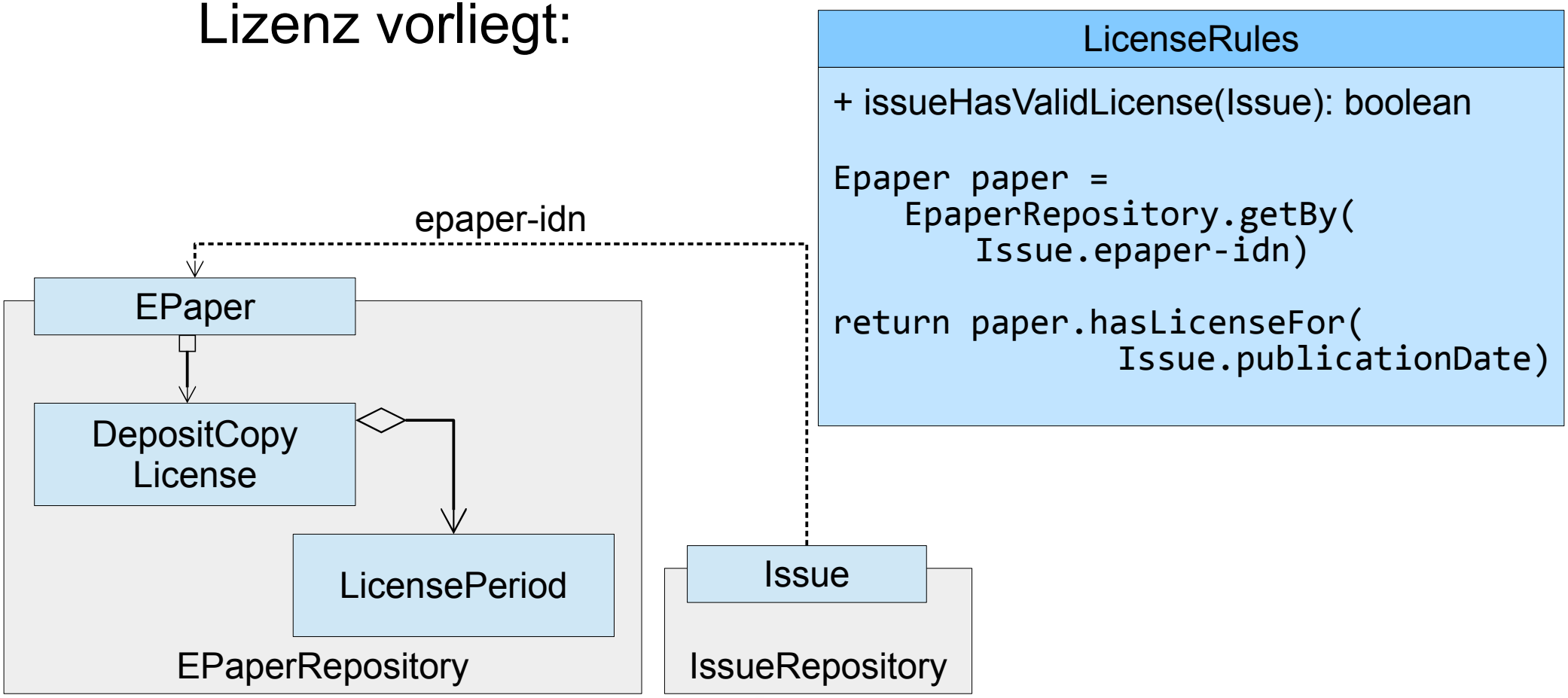
- Projekt zur Bereitstellung von E-Books
  - Prüfen, ob für eine konkrete Ausgabe eine gültige Lizenz vorliegt:





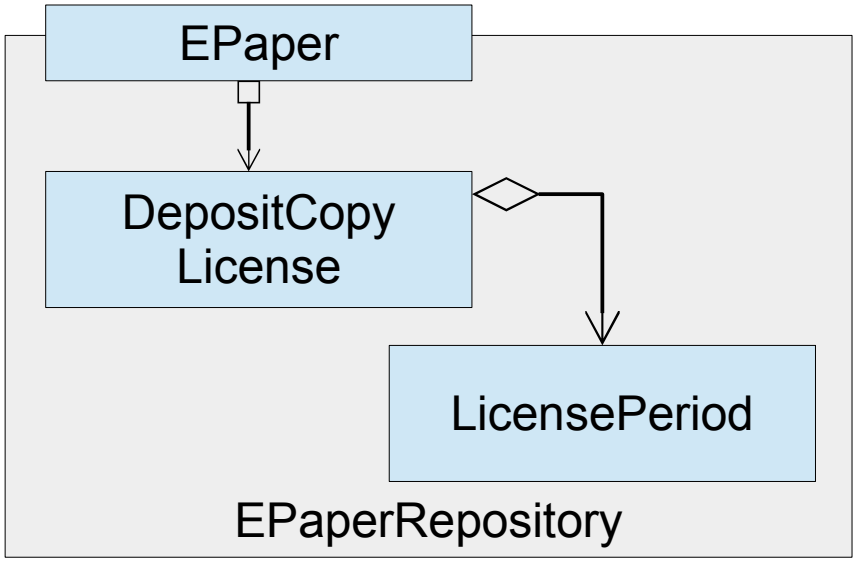
Beispiel für DDD und Application Code

- 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“

Schicht 1: Adapters

- Anti-Corruption Layer
- Beispiele:
  - GUI: Enthält alle Klassen einer MVC-Struktur
  - Datenbank: Wandelt Anfragen der Anwendung in SQL-Statements um
    - Kein SQL in der Anwendung selbst!
  - GUI: Direkt verwendbares Render-Model
    - Key-Value-Paket
- Diese Schicht hält die Applikation tauglich und die Plugins frisch

Beispiel für Adapters

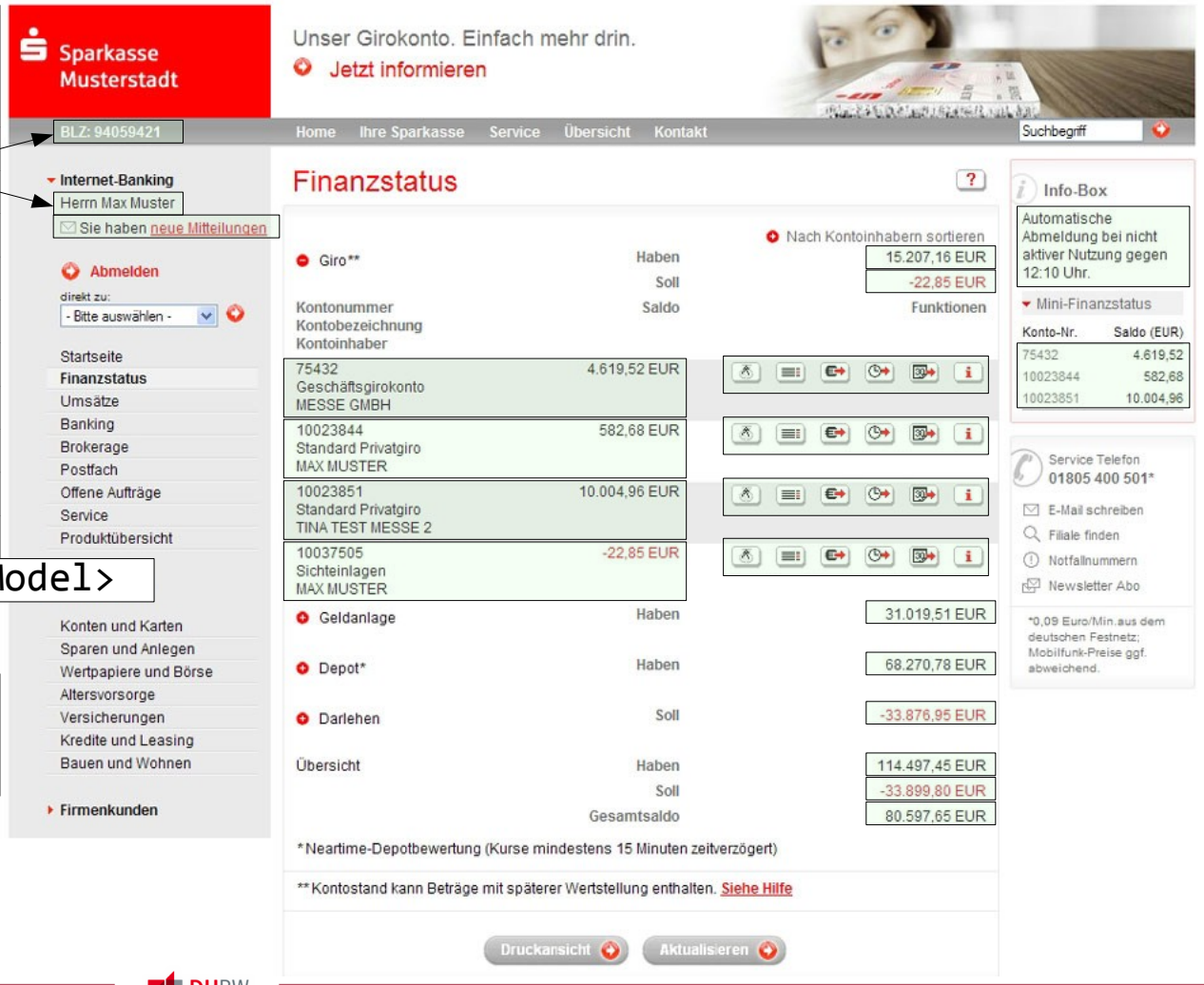
- Bankkonto-Verwaltungssoftware
- Anzeige auf Webseite (HTML) vorbereiten
- Alle veränderlichen Inhalte der Seite unzweideutig berechnen (RenderModel)
  - Geldbeträge als Zeichenketten im Format 1234,56 €
    - Die Anzeigeschicht benötigt keine numerischen Werte
  - Farben als HTML-Hexcodes
  - Attribute (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>)



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

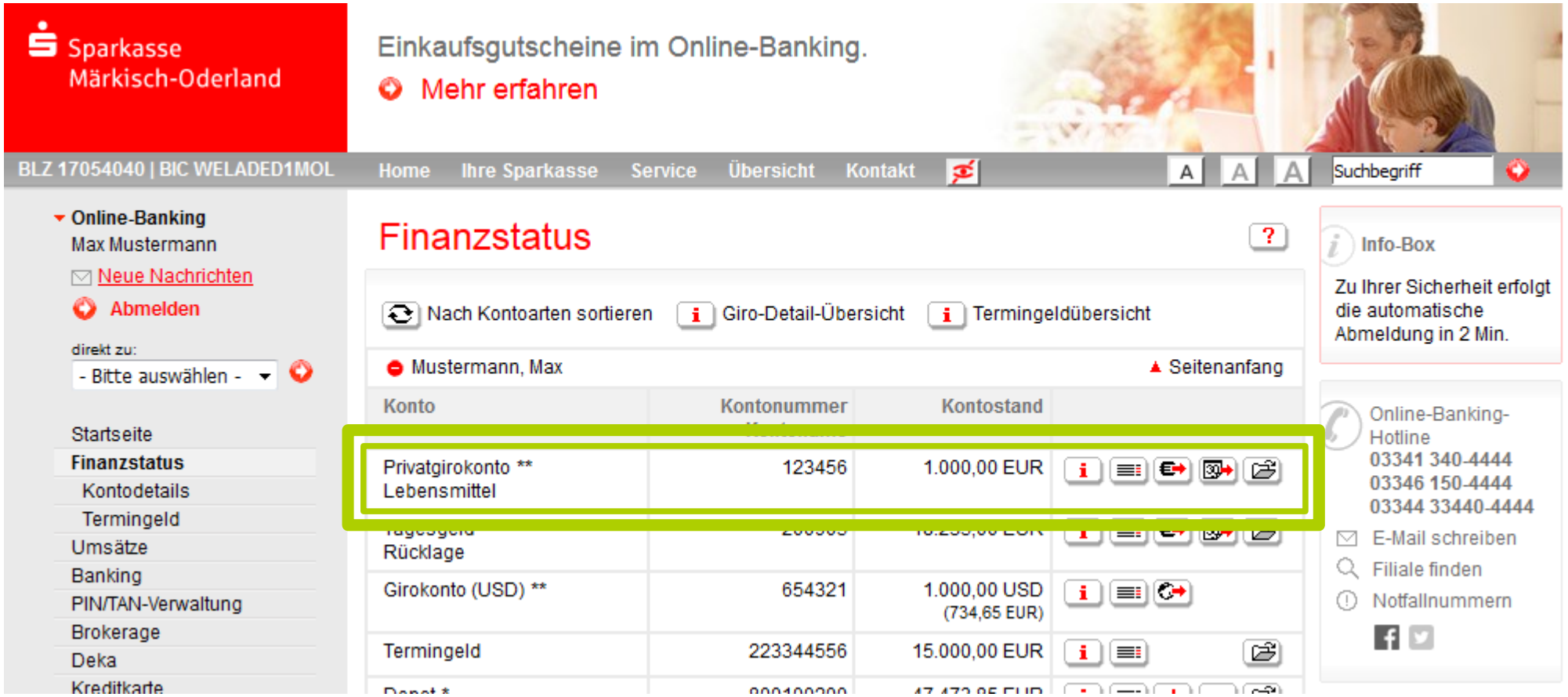
- Diese Schicht greift grundsätzlich nur auf die Adapter zu
- Enthält Frameworks, Datentransportmittel und andere Werkzeuge
  - v.a. Datenbank, Benutzeroberfläche, Web
  - Alle „Pure Fabrication“-Entscheidungen
- Wir versuchen, hier möglichst wenig Code zu schreiben
  - Hauptsächlich Delegationscode,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



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 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

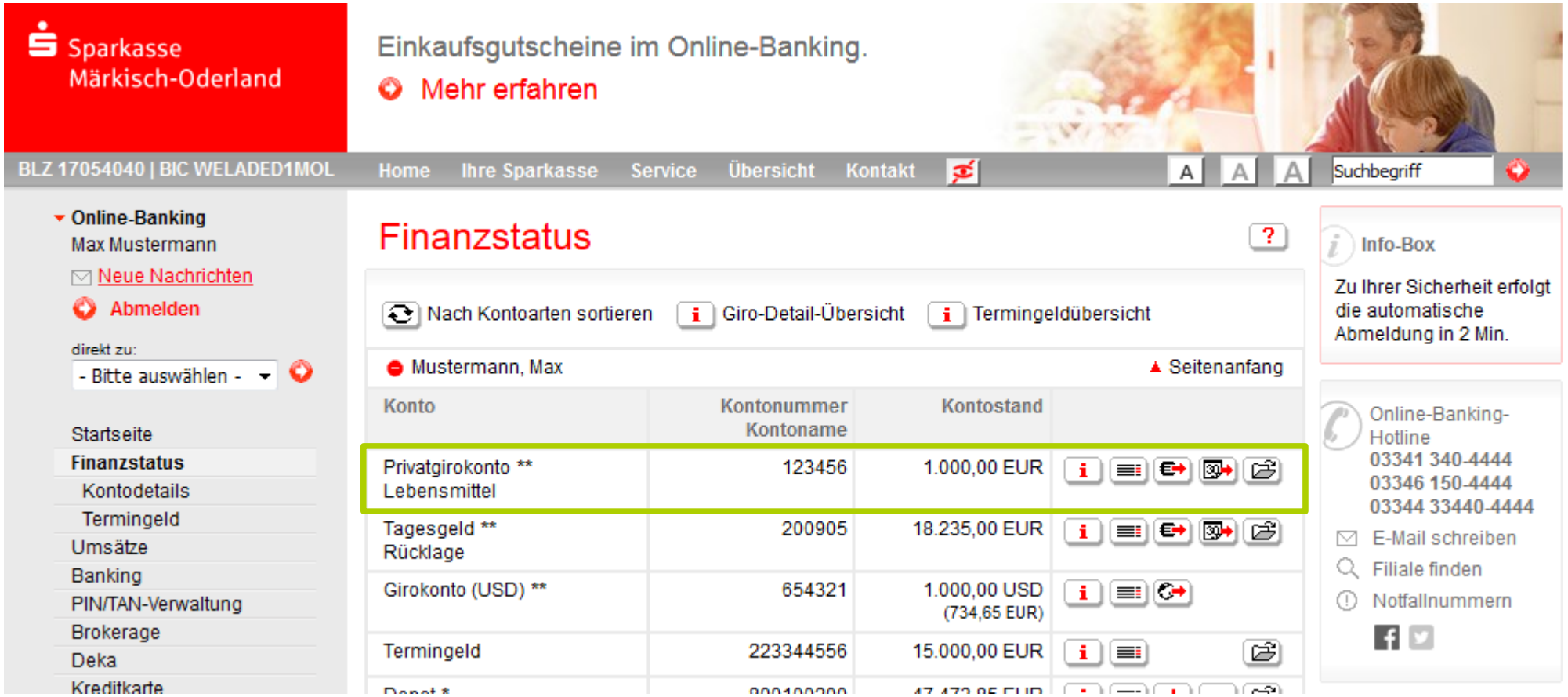
- Veränderliche Inhalte als benannte Variablen
- Velocity setzt die Werte des RenderModel ein

```
<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



Beispiel für DDD und Plugins

- Projekt zur Bereitstellung von E-Books
  - Anbindung ans Internet per JAX-RS

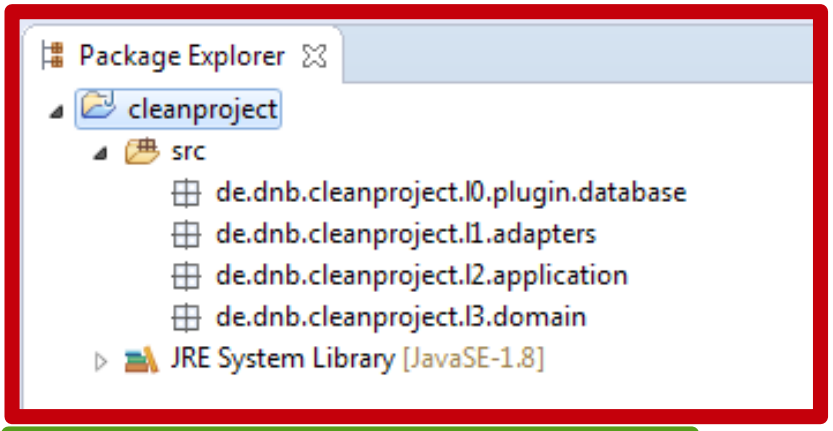
```
@Path("/epapers")
public class EPapersEndpoint {
 @Inject private EPaperAccess ePaperAccess;
 @Inject private EPaperToEPaperResourceMapper ePaperToEPaperResource;

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

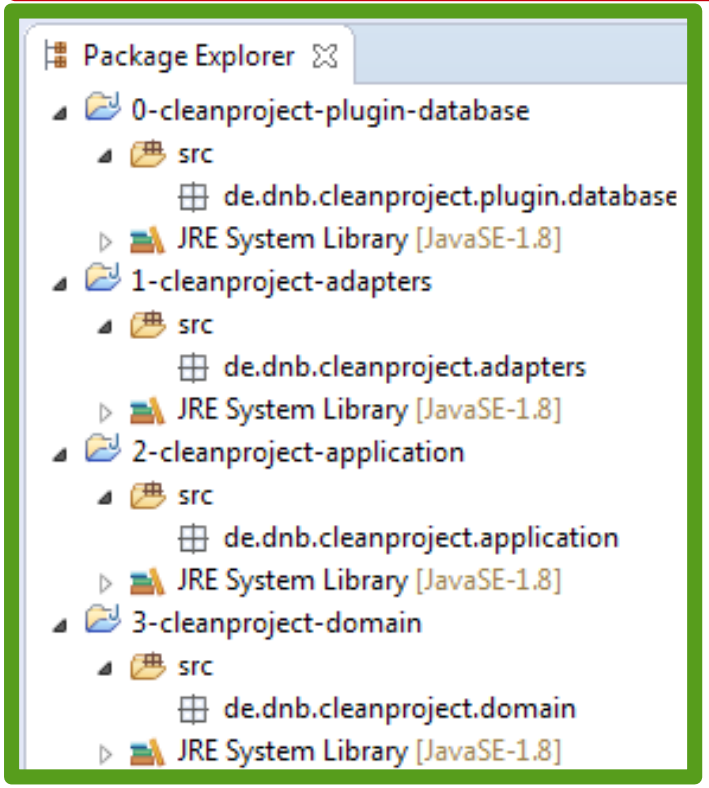
Application Code

Adapters

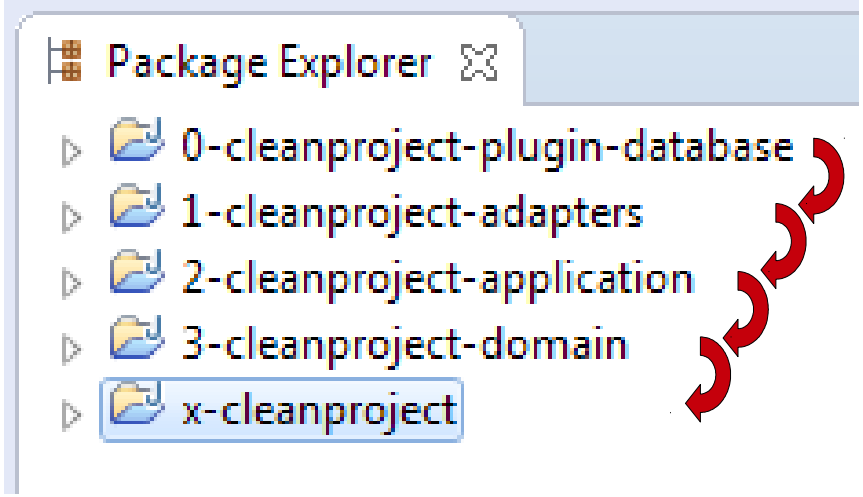
Konkrete Umsetzung

- 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





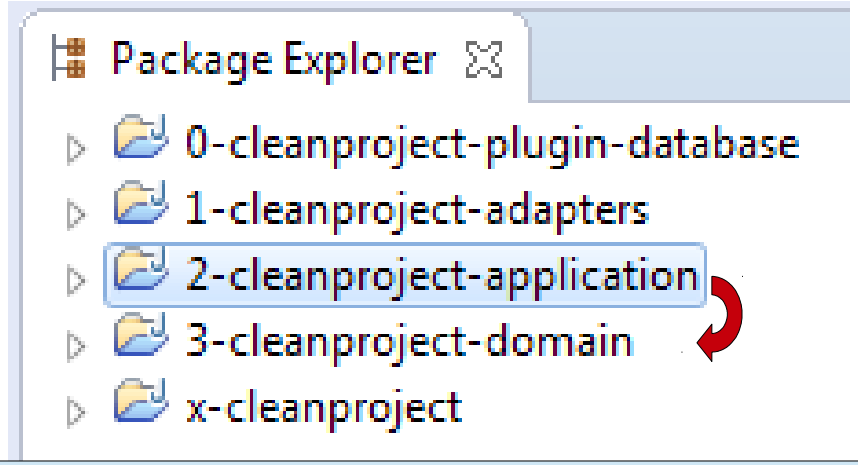
Konkrete Umsetzung: Maven



- 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>
```

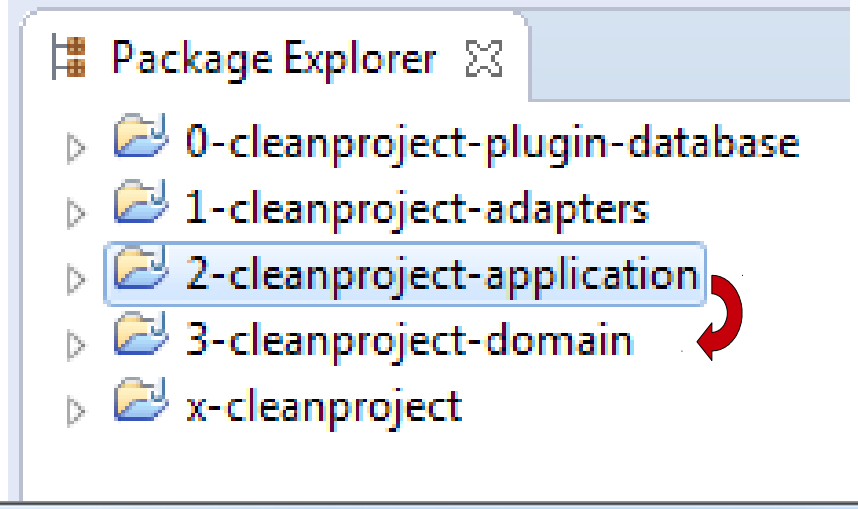
Konkrete Umsetzung: Maven



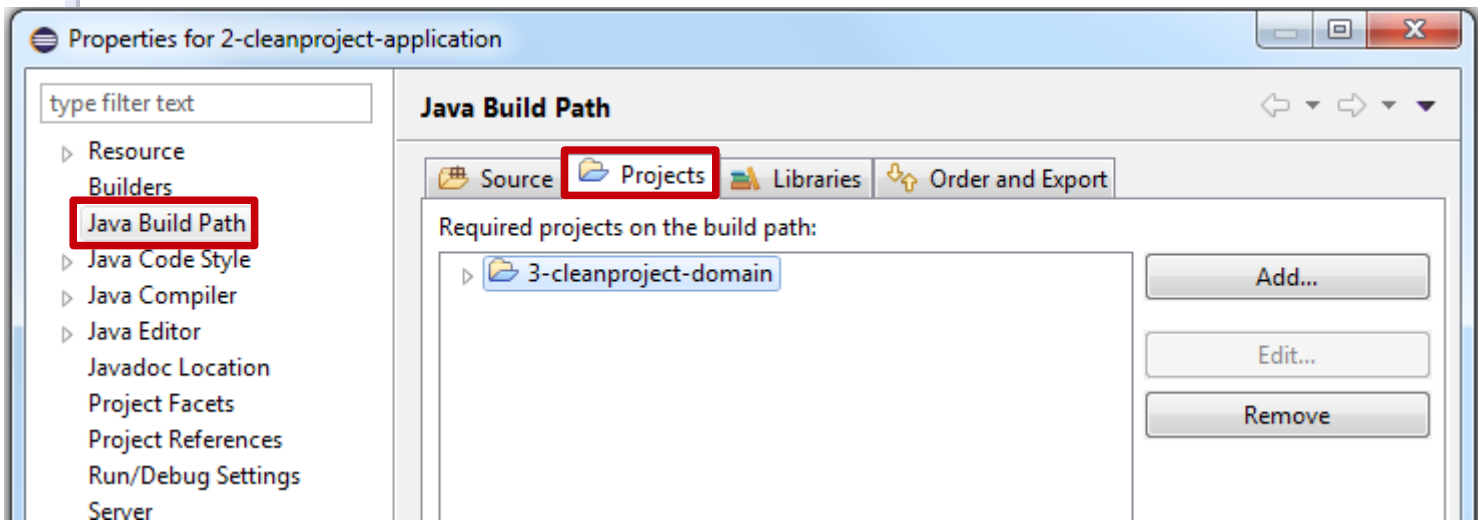
- Projekt 2 soll von Projekt 3 abhängen
  - Im pom.xml als Dependency eintragen

```
<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>
```

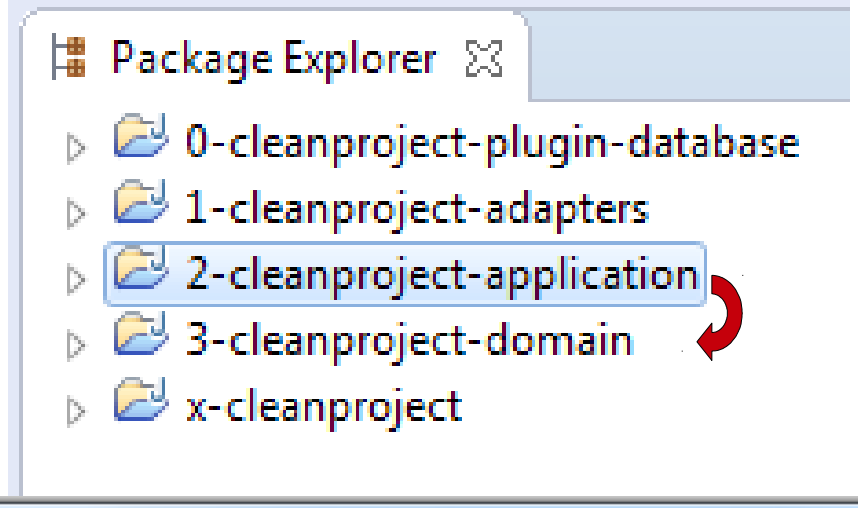
Konkrete Umsetzung: Manuell



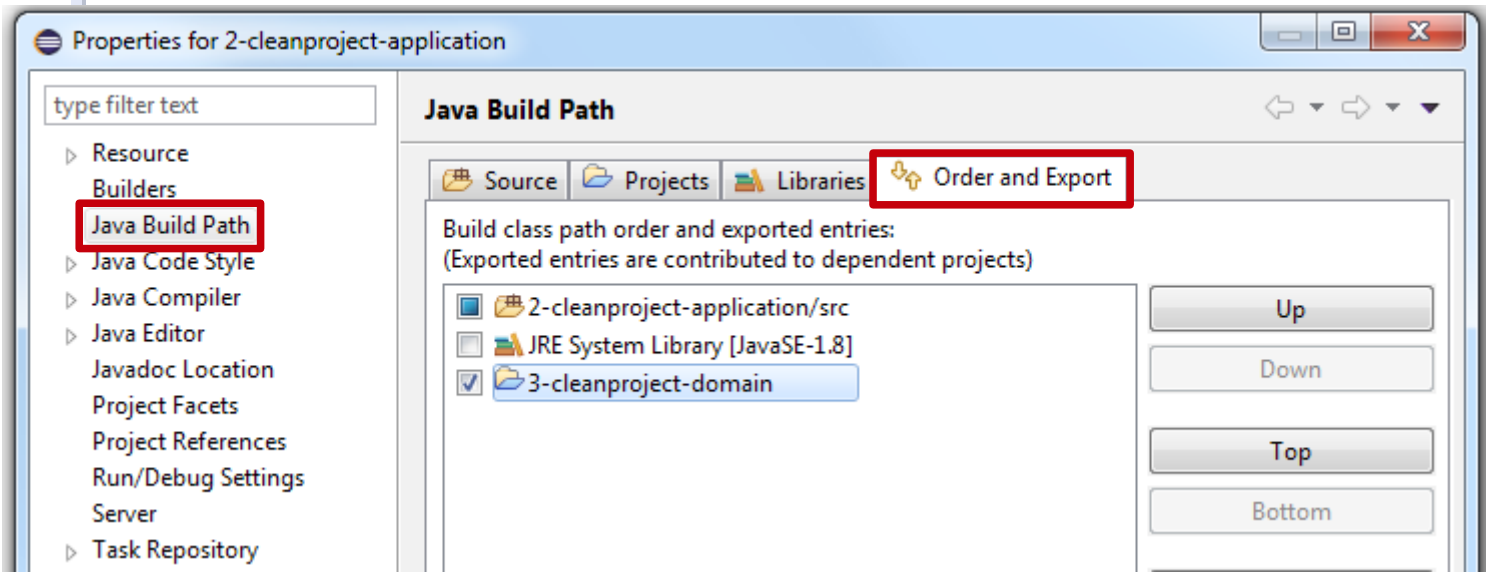
- Projekt 2 soll von Projekt 3 abhängen
  - In den Eclipse-Projekteinstellungen angeben



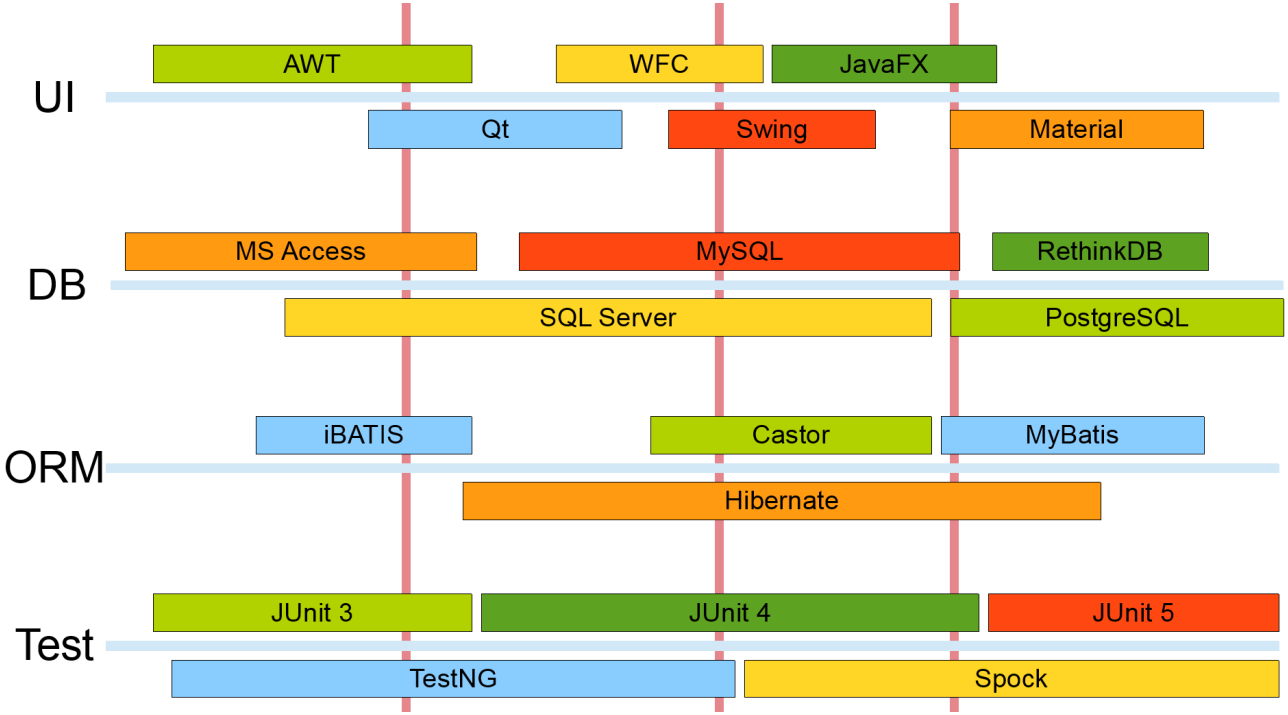
Konkrete Umsetzung: Manuell



- Transitive Abhängigkeiten freigeben
  - In den Eclipse-Projekteinstellungen



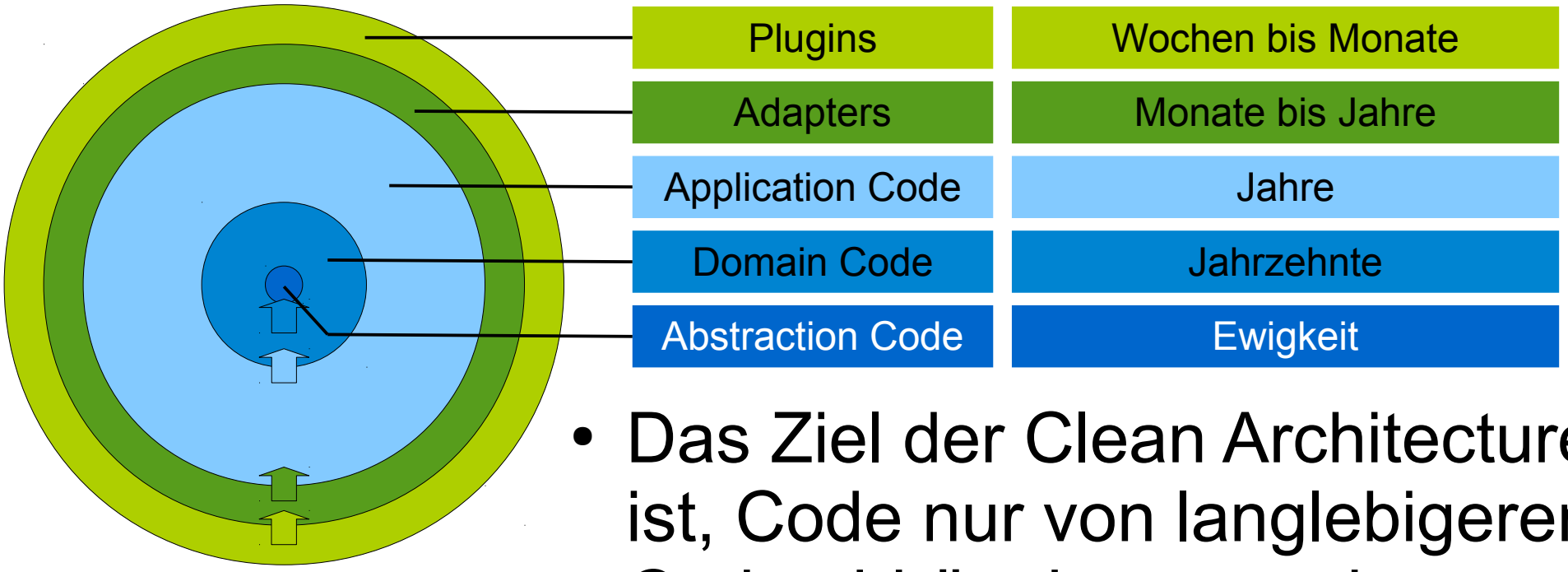
Review der Technologiewahl



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



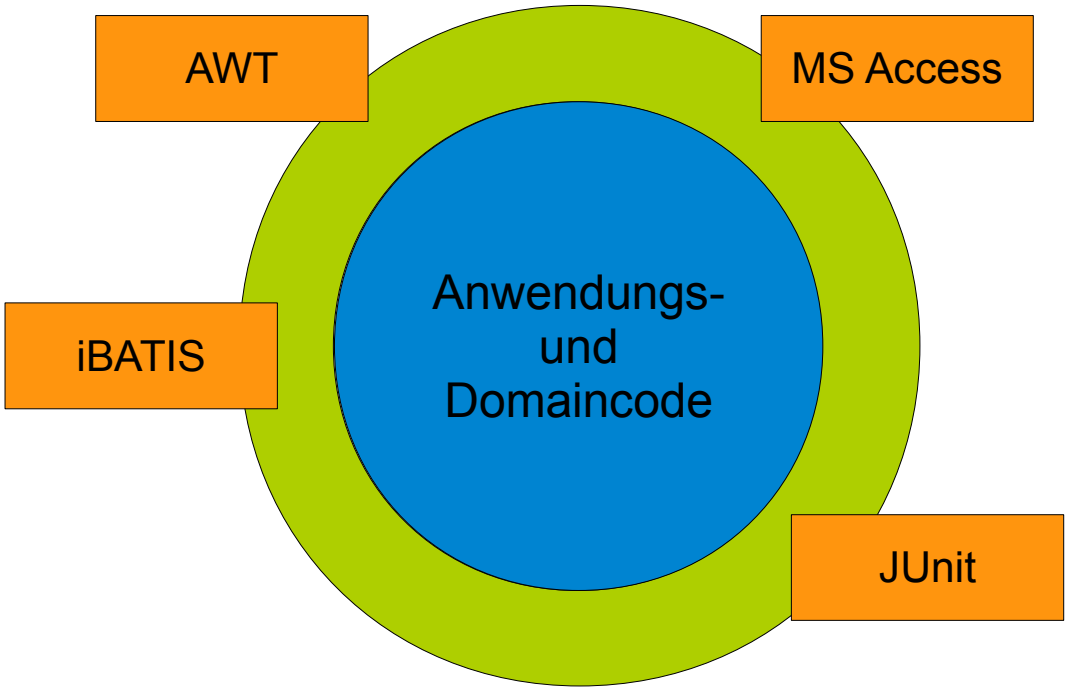
- Das Ziel der Clean Architecture ist, Code nur von langlebigerem Code abhängig zu machen
- Wenn sich Technologien ändern müssen, kann die Anwendung unverändert bleiben

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

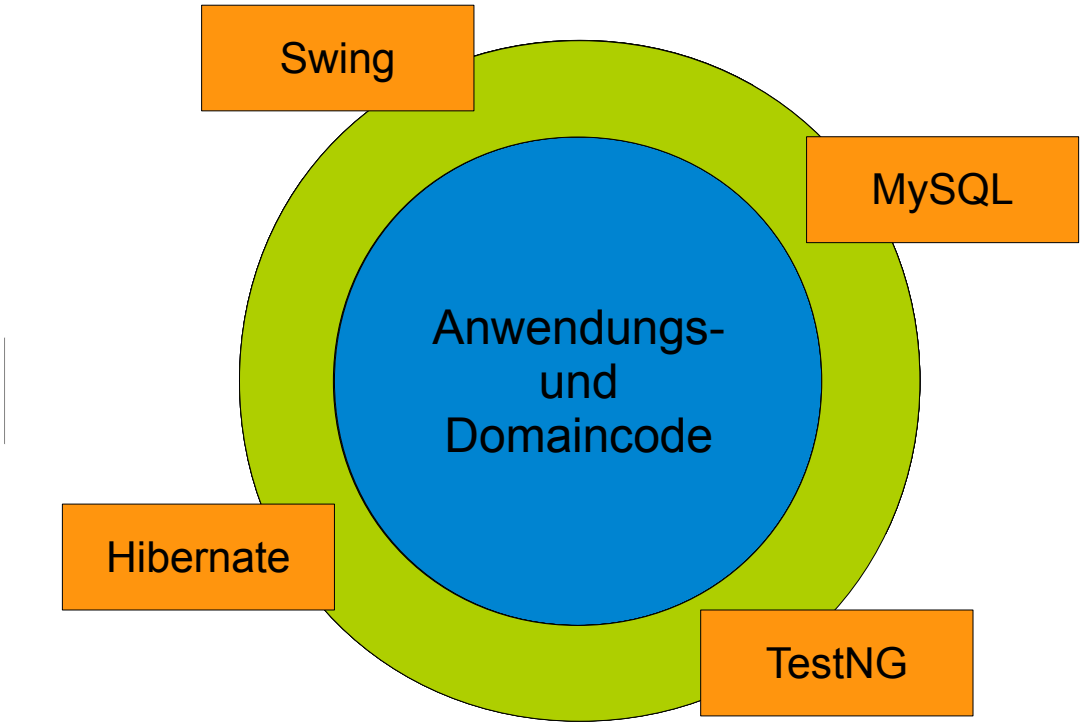
Clean Architecture Technologiewahl



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

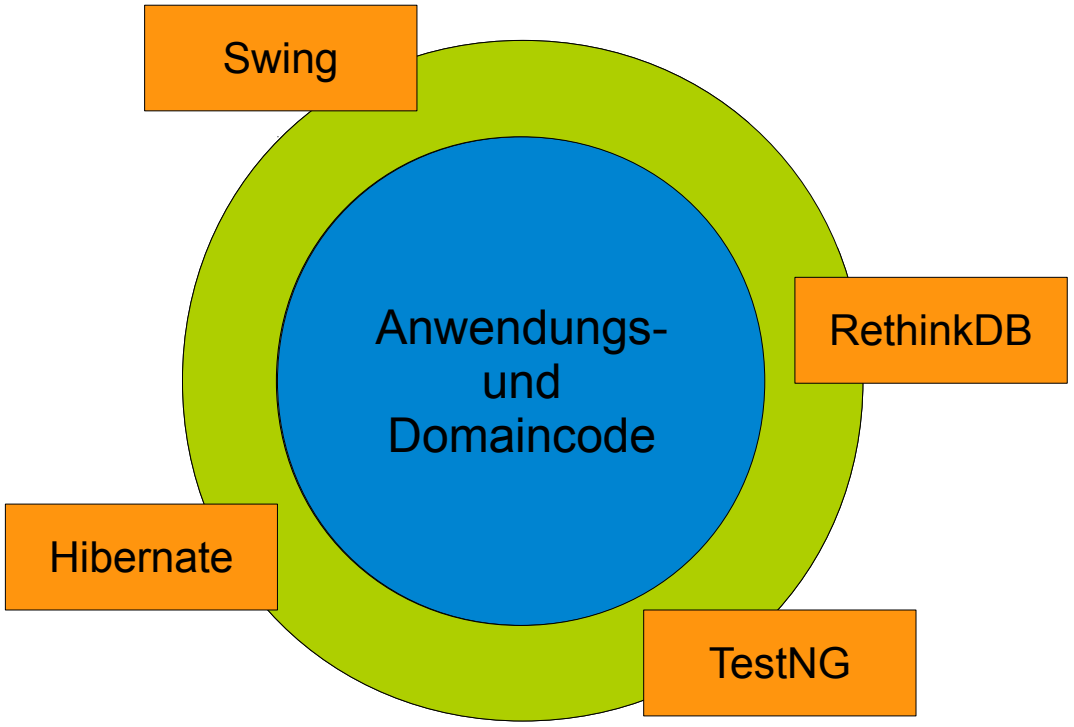
Clean Architecture Technologiewahl



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

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

Clean Architecture Technologiewahl

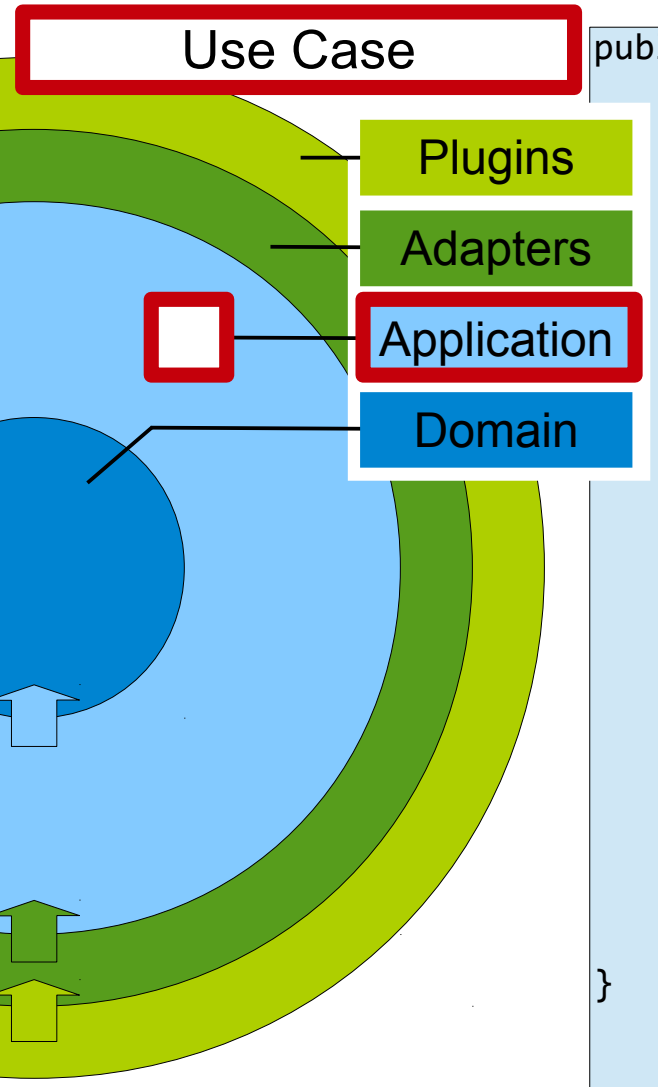


- Jedes Plugin kann einzeln ersetzt werden
- Keine oder nur minimale Abhängigkeiten zwischen Plugins

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

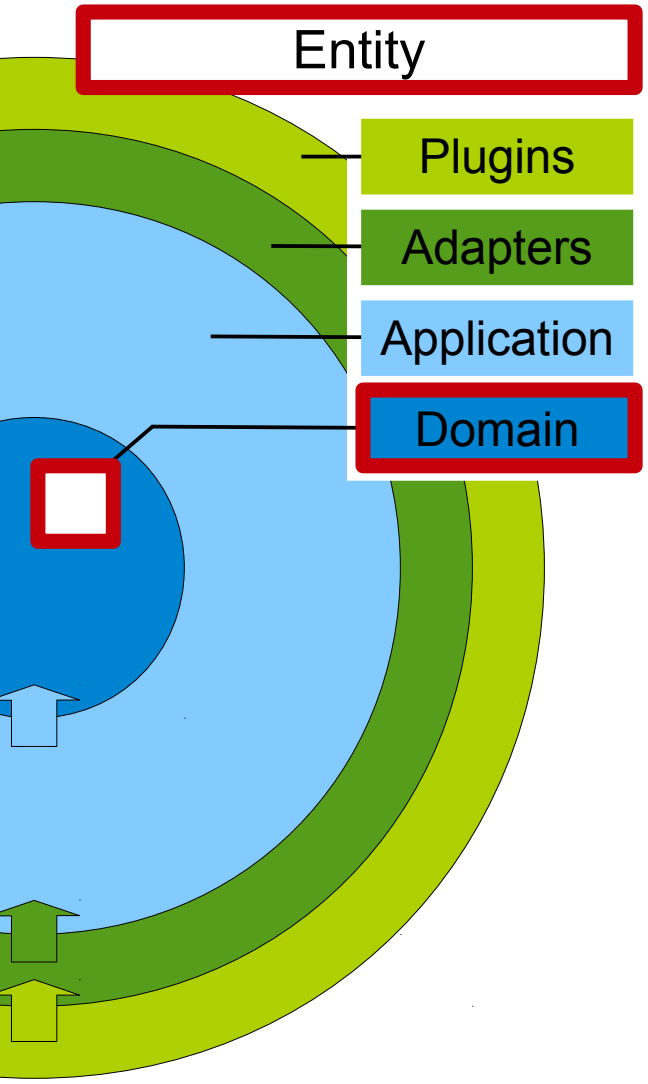
- Separation of Concerns

Positionierung: Beispiel 1



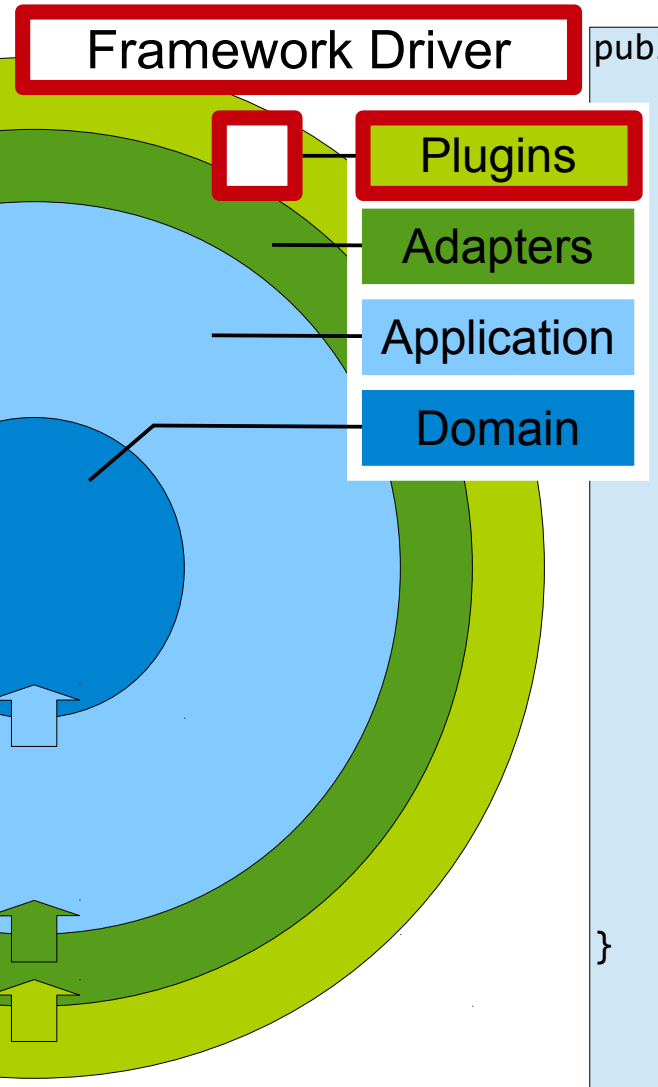

```
public class ChangeUserPassword {
     private final AuthenticationService authenticationService;

     @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



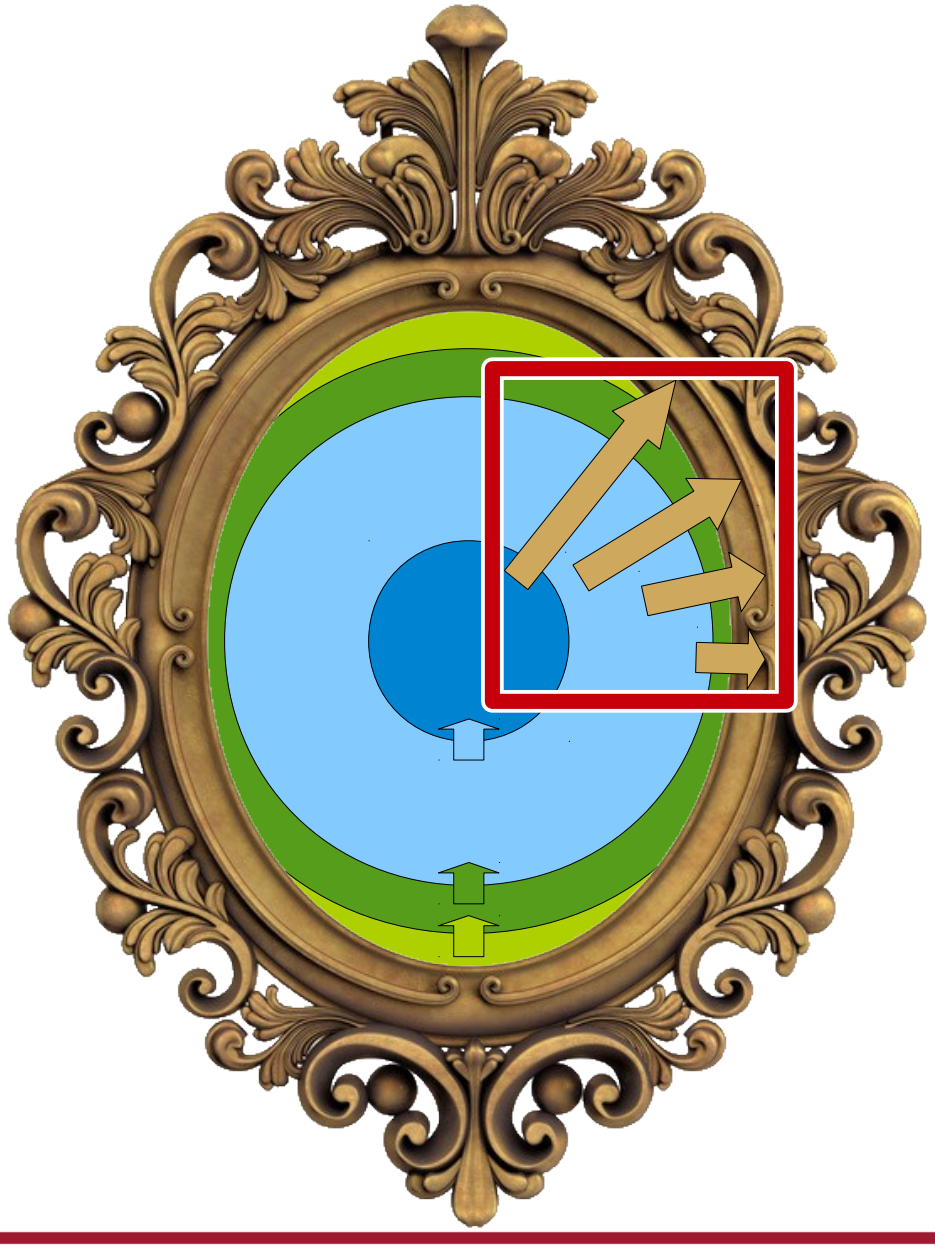

```
public class User {

     @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



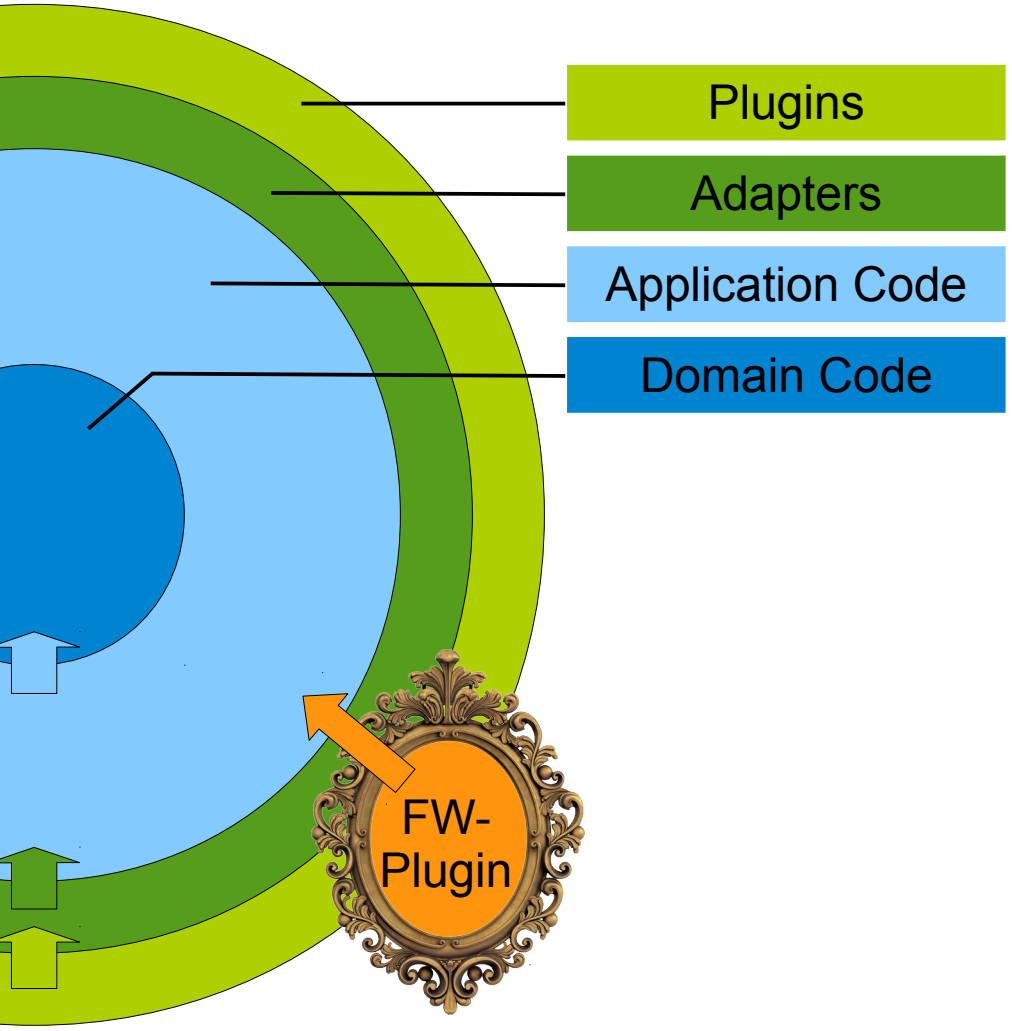

```
public class JPAAuthenticationService implements AuthenticationService {

    private JPAUserEntityRepository userEntityRepository;
    private JPAAuthenticationEntityRepository authenticationRepository;

    @Inject
    public JPAAuthenticationService(
                    final JPAUserEntityRepository userEntityRepository,
                    final JPAAuthenticationEntityRepository repository) {
        this.userEntityRepository = userEntityRepository;
        this.authenticationEntityRepository = 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



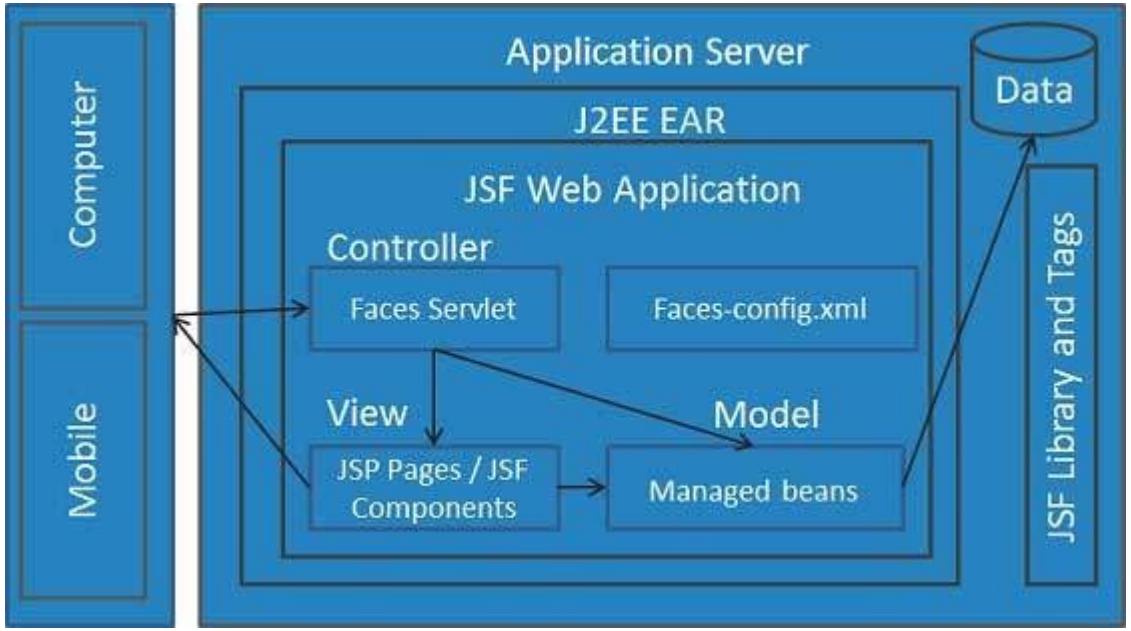
- Frameworks sind Details
- Details sind Plugins
- Plugins gehören „an den Rand“ der Anwendung
- Das Framework ausfüllen heißt, Aufrufe an die Anwendung zu delegieren
- Problematisch bei Frameworks mit Metaprogrammierung

Frameworks separieren

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

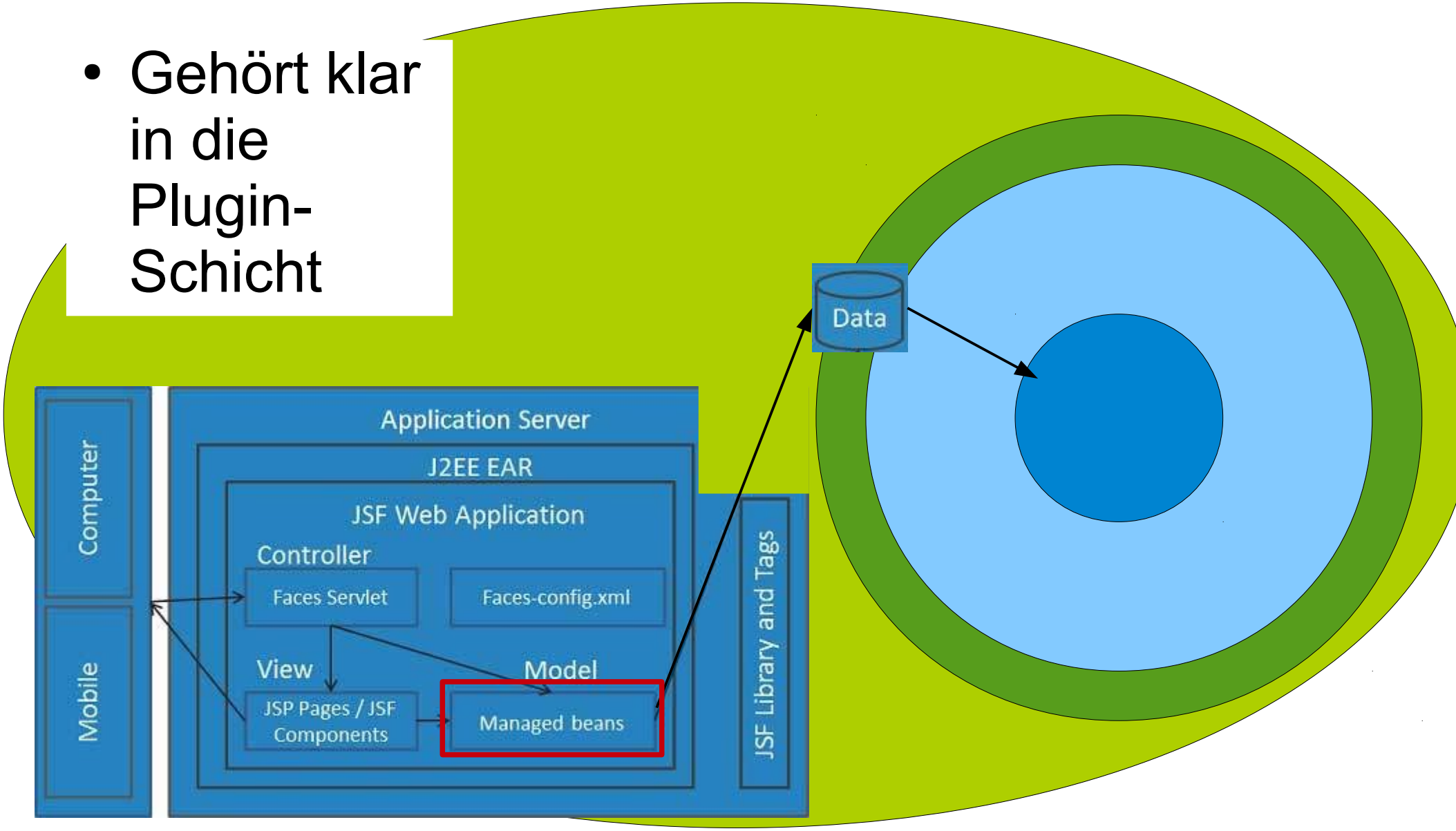
Clean Java Server Faces

- Java Server Faces (JSF):
  - „an application framework for creating web-based user interfaces“
- MVC-Framework (Model-View-Controller)



Clean Java Server Faces

- Gehört klar in die Plugin-Schicht



JSF Managed Beans – woher?

- Wenn irgend möglich, CDI verwenden
  - Seit JSF 2.2 gibt es Scoping auch in CDI
- @ManagedBean wird in JSF 2.3 deprecated
- Aus der JSF-Javadoc:

Package javax.faces.bean Description

These javadoc files constitute the "Faces Managed Bean Annotation Specification for Containers Conforming to Servlet 2.5 and Beyond"

At the time of this writing, a forthcoming JCP effort is being planned to extract the specification for managed beans from JSF and place it into its own specification. To account for this effort and to avoid introducing classes into JSF 2.0 that would have to be deprecated when this effort is complete, implementations of JSF 2.0 are not required to implement the "Faces Managed Bean Annotation Specification for Containers Conforming to Servlet 2.5". However, JSF implementations are strongly encouraged to implement this specification, as it provides significant improvements in ease of use.

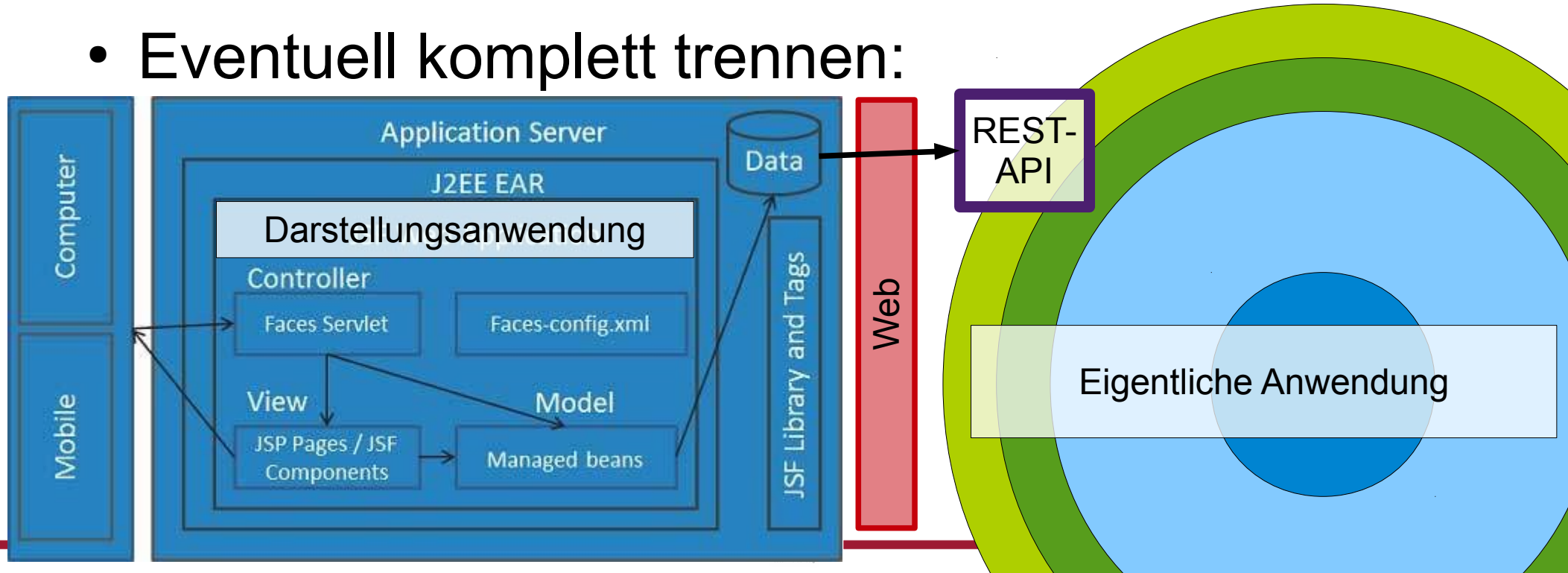
http://arjan-tijms.omnifaces.org/p/jsf-23.html#1417

http://stackoverflow.com/a/18388289

CDI (Contexts and Dependency Injection)

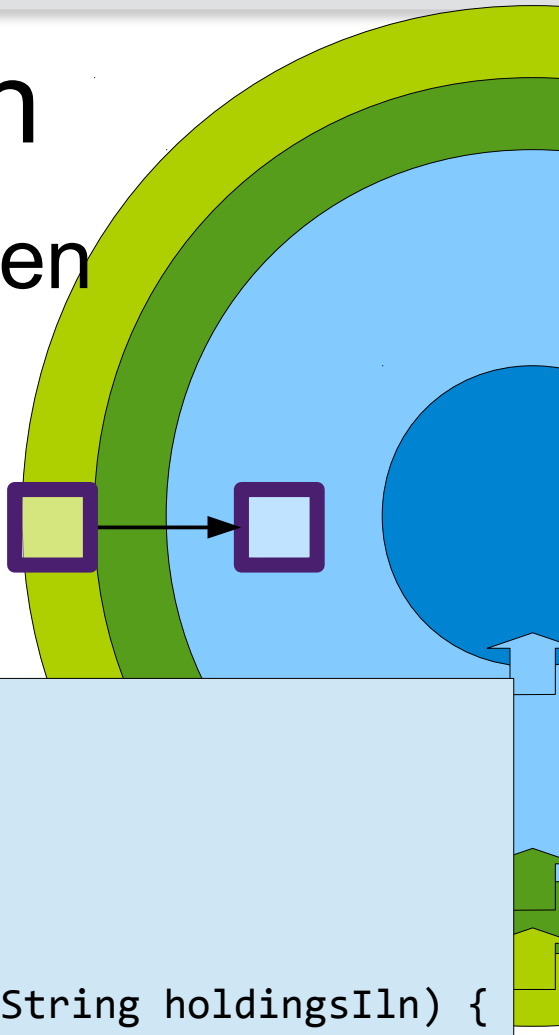
JSF in der Clean Architecture

- JSF ist ein Plugin für die UI
  - JSF-Beans sind durch Adapter kopiert
    - Entkoppelter Lebenszyklus von Entities
- Alle Besonderheiten von JSF im Plugin lassen
- Eventuell komplett trennen:



Übergabe von Daten

- An einer Schichtgrenze müssen Daten übergeben werden
- Von außen nach innen ist einfach
  - Parameter eines Methodenaufrufs

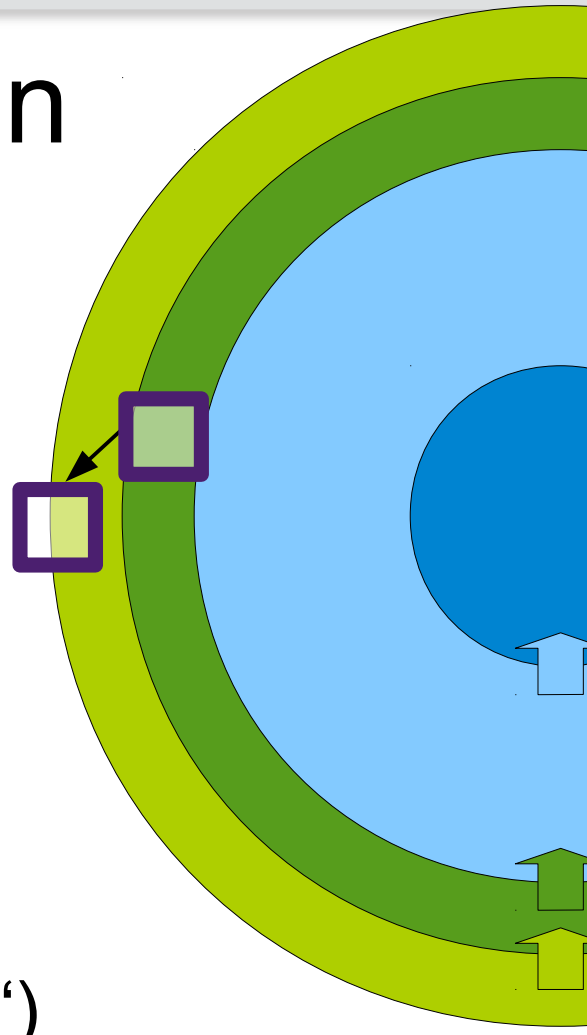


```
@Path("/epapers")
public class EPapersEndpoint {
 @GET
 @Path("")
 @Produces(MediaType.APPLICATION_JSON + "; charset=utf-8")
 public Response findEPapers(
         @QueryParam("iln")
         @NotNull(message = "add required query parameter iln") String holdingsIln) {
     return Response.ok(
  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

Application Code

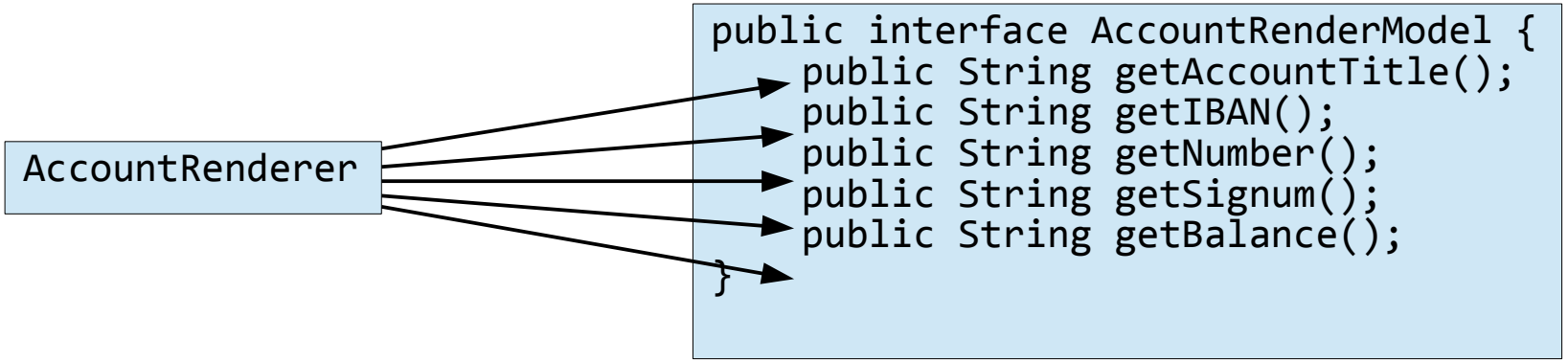
Übergabe von Daten

- Von innen nach außen gibt es zwei grundsätzliche Möglichkeiten
  - Reaktiv: Als Rückgabewert eines Methodenaufrufs von außen
  - Aktiv: Rückruf (Callback) einer äußeren Methode von innen
- Sicht von außen:
  - Reaktiv == Pull (Außen muss „ziehen“)
  - Aktiv == Push (Innen meldet sich von alleine)



Daten reaktiv herausgeben

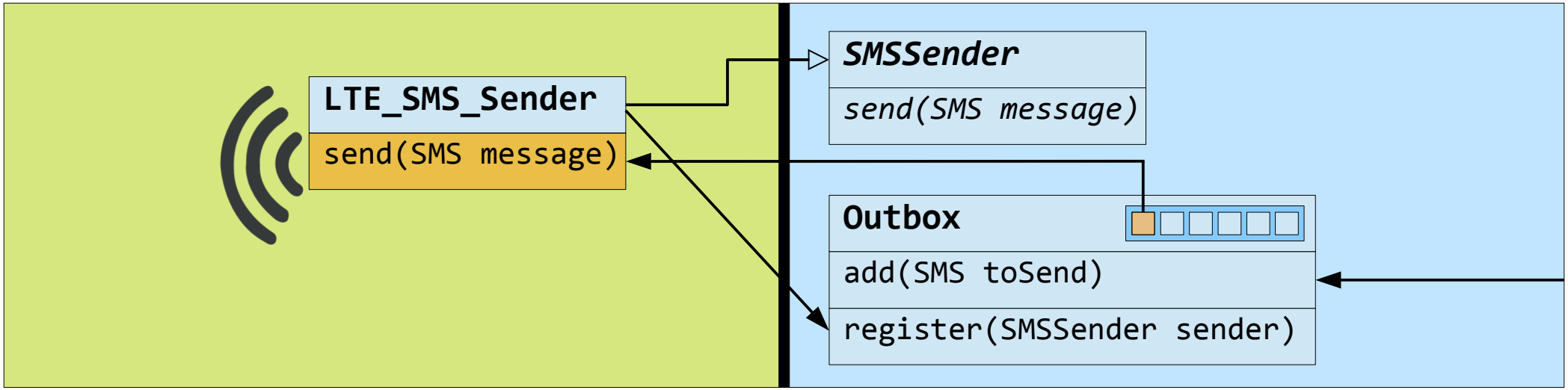
- Das äußere Plugin fragt zu einem ihm genehmen Zeitpunkt nach den Daten
- Die inneren Schichten antworten nur



AccountRenderModel (Map<String, String>)	
account_title	Privatgirokonto
iban	IBAN: DE89 1705 4040 0000 1234 56
number	123456
sgn	plus
balance	1.000,00 EUR

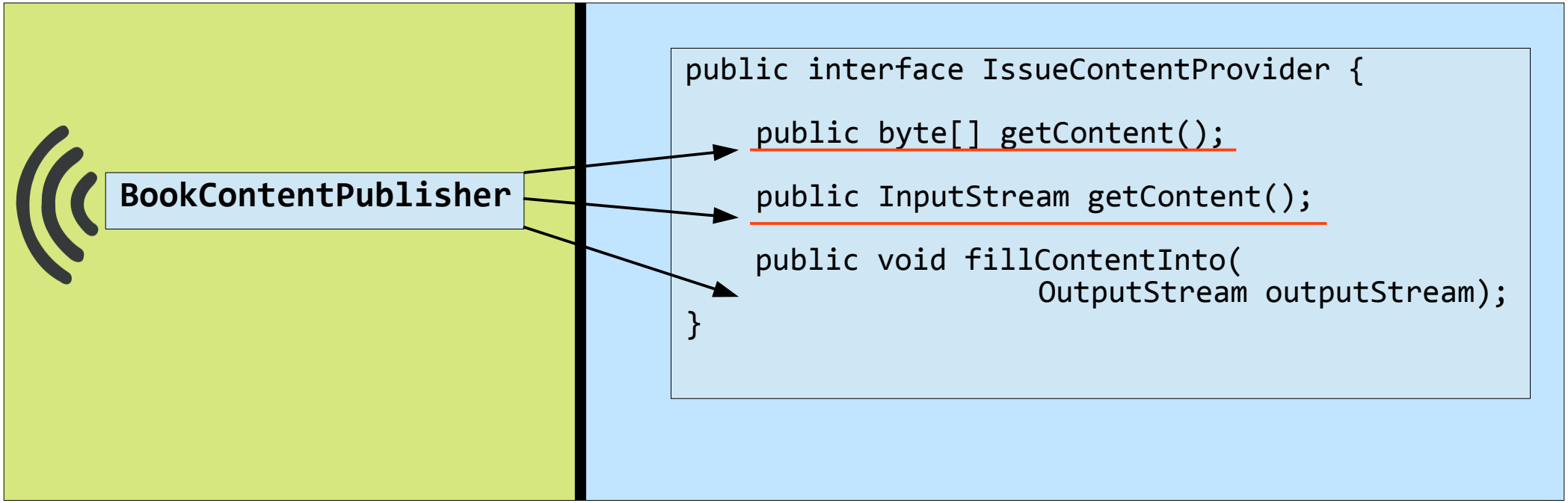
Daten aktiv herausgeben

- Das äußere Plugin meldet sich zu einem frühen Zeitpunkt als Befehlsempfänger an
- Die inneren Schichten geben die Befehle zum ihnen genehmen Zeitpunkt
- Oft als Beobachter-(Listener)-Muster realisiert



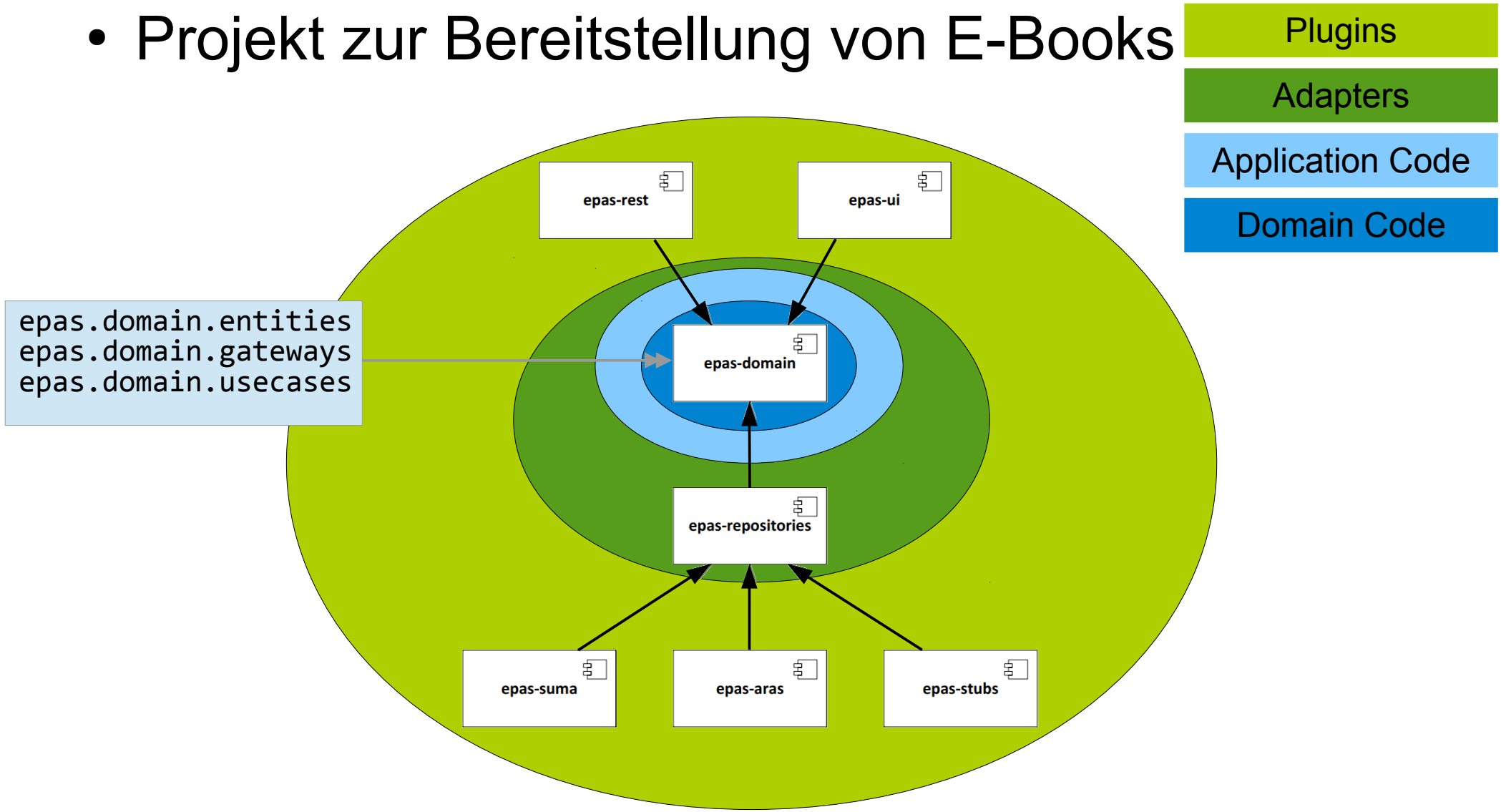
Daten aktiv herausgeben

- Variante: Die Datenverarbeitungsrichtung umdrehen
  - OutputStream geben statt InputStream geben lassen
- Beispielsweise bei Bereitstellung von E-Books



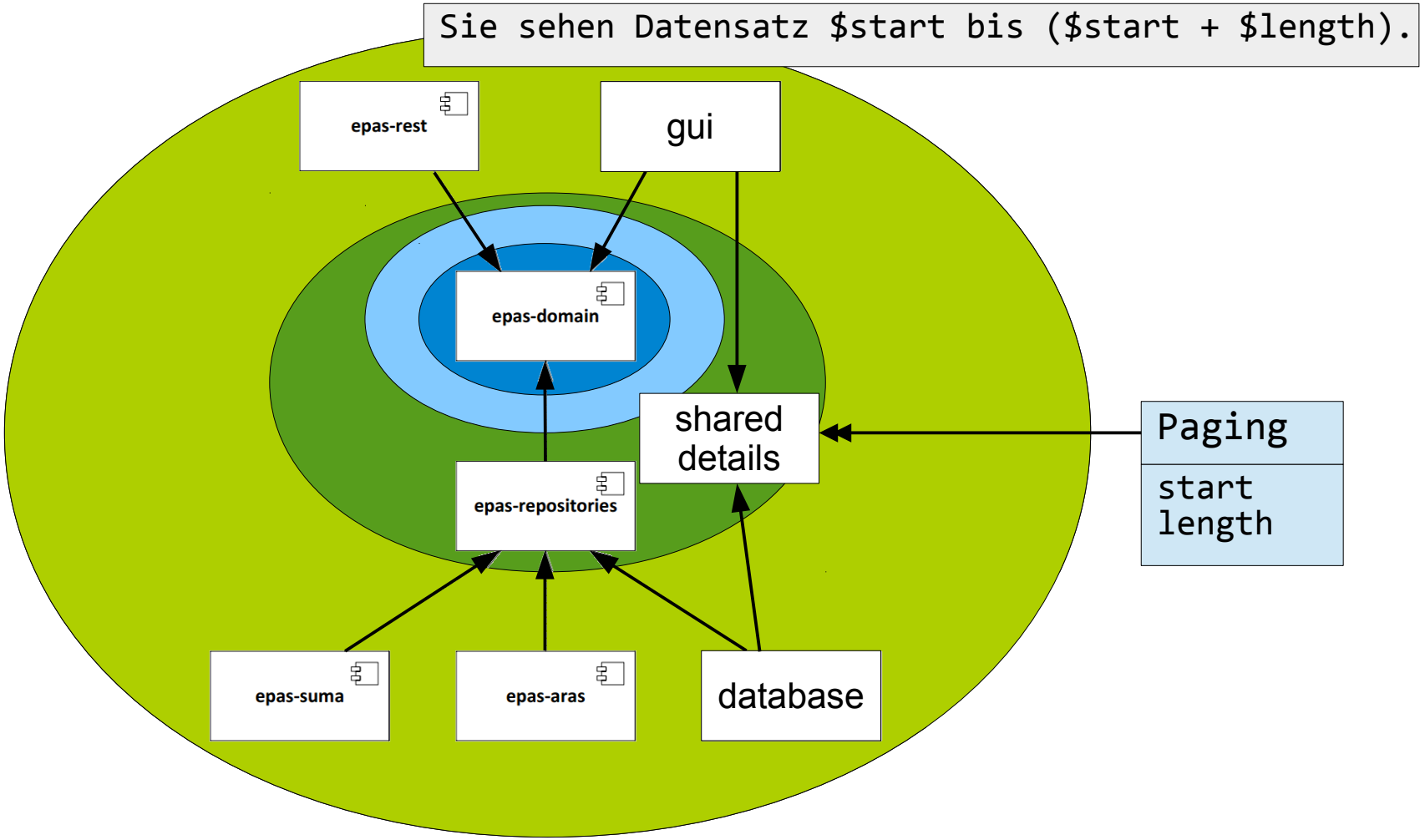
Beispielhafte Projektstruktur

- Projekt zur Bereitstellung von E-Books

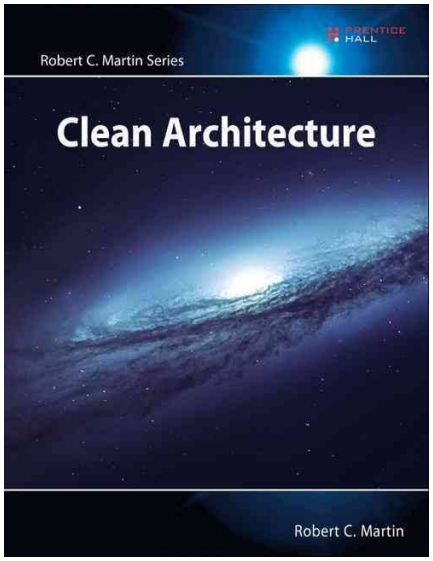


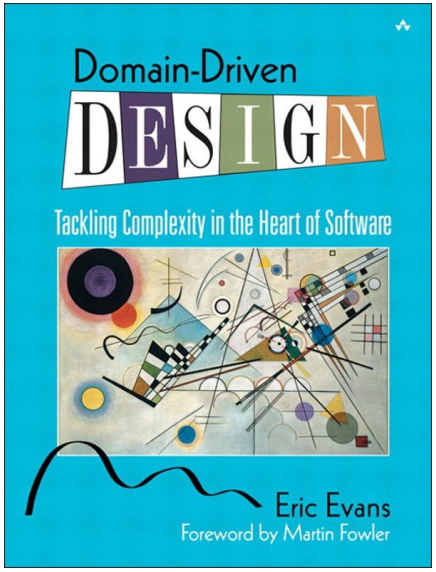
Details durchschleifen

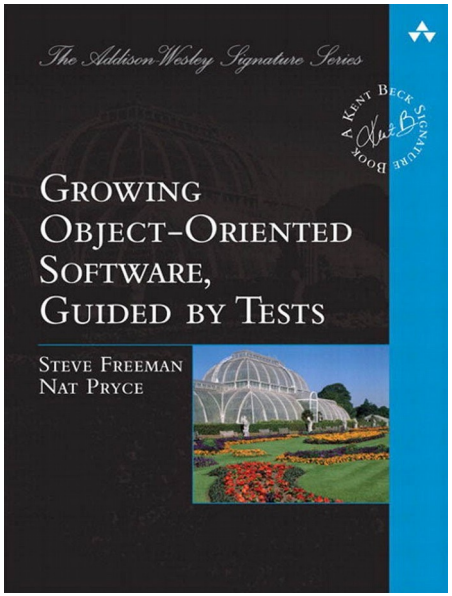
- Beispielsweise Paging in Datenbank und GUI

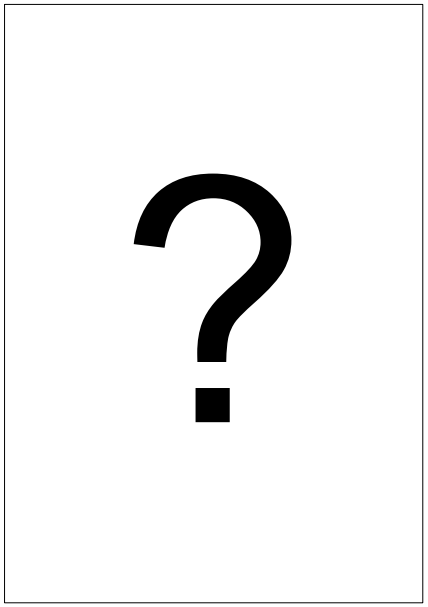


Clean Architecture: Weiterführende Literatur









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: Icons made by Freepik from www.flaticon.com is licensed by CC 3.0 BY
- Moore neighborhood with cardinal directions: Von MorningLemon - Eigenes Werk, CC-BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=38746075
- epas project dependencies - Von Thomas Seidel - Eigenes Werk