



Softwareschneiderei

Clean Architecture

Systeme für die Ewigkeit



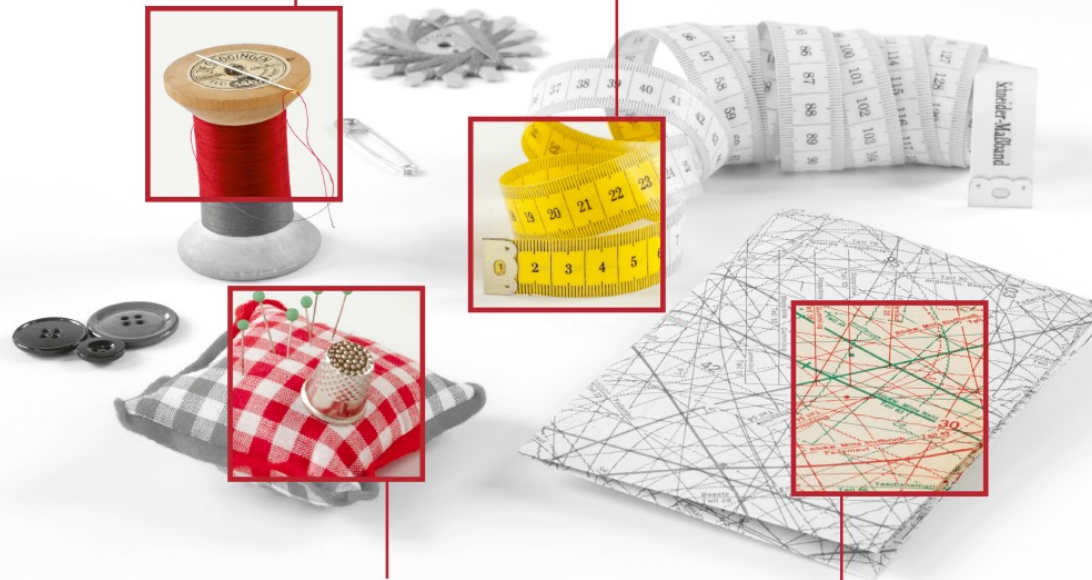


Softwareschneiderei

Softwareschneiderei GmbH

Softwareentwicklung

User Experience (UX)



Fortbildung

Strategische IT-Beratung





Inhaltsverzeichnis

- Launige Einführung
- Nervige Wiederholung
- Definition des Feindbildes
- Verkündung der frohen Botschaft
- Langatmige Predigt
- Überraschender Quiz
- Preisen der Verbesserungen
- Viel zu konkrete Details
- Originalquellen
- Bekannte Architekturen
- Notwendige Grundlagen
- Nachteile von Frameworks
- Grundidee der Clean Architecture
- Details zur Clean Architecture
- Leichte Verständnisfragen
- Folgen der Clean Architecture
- Umsetzungsbeispiele
- Literatur





Softwareentwicklung

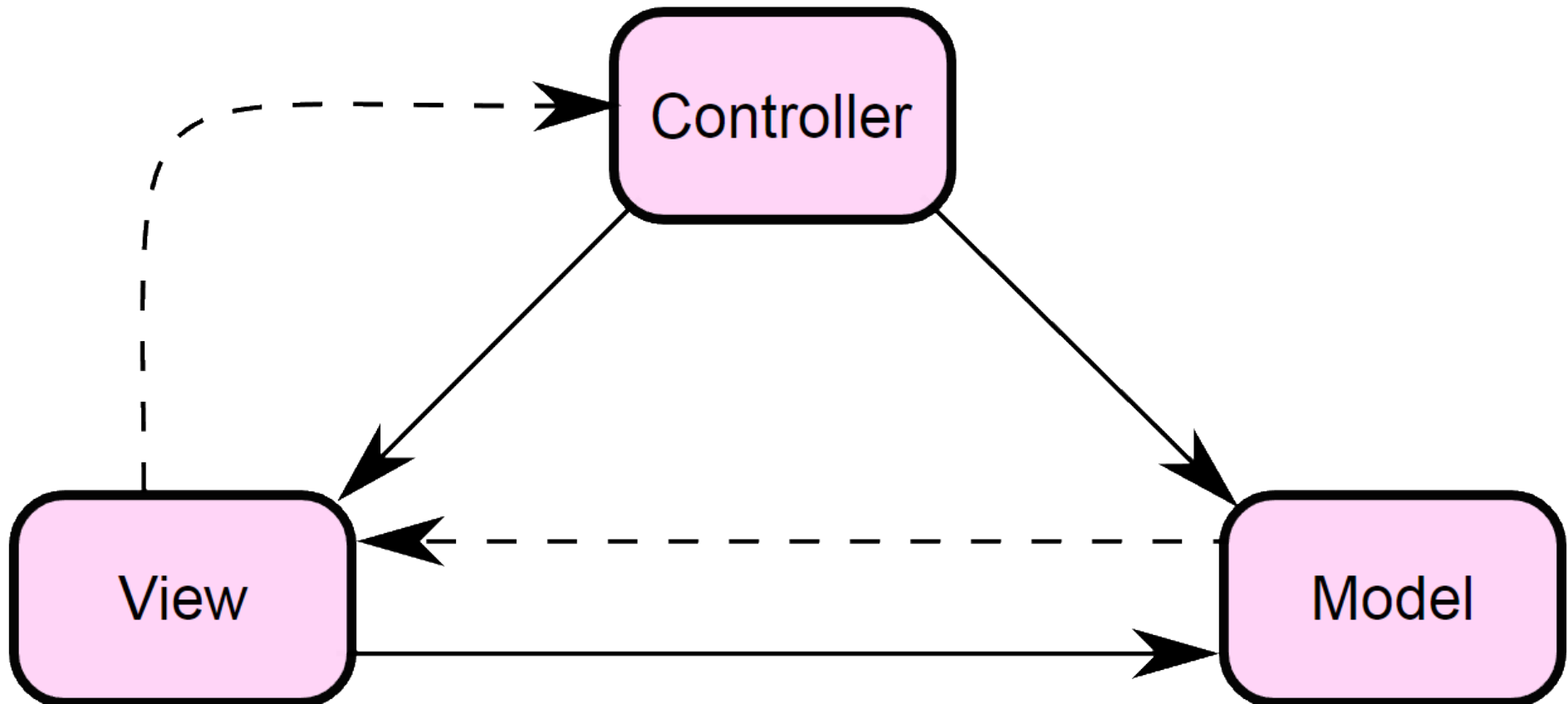
- Beständiger Wandel alle fünf Jahre
 - Kein zentraler Takt
 - Unterschiedliche Zykluslänge von Produkten
- Keine monolithischen Systeme mehr
 - Zusammenstöpseln von Bausteinen (Building Blocks)
 - Fundament bildet häufig ein Framework





MVC

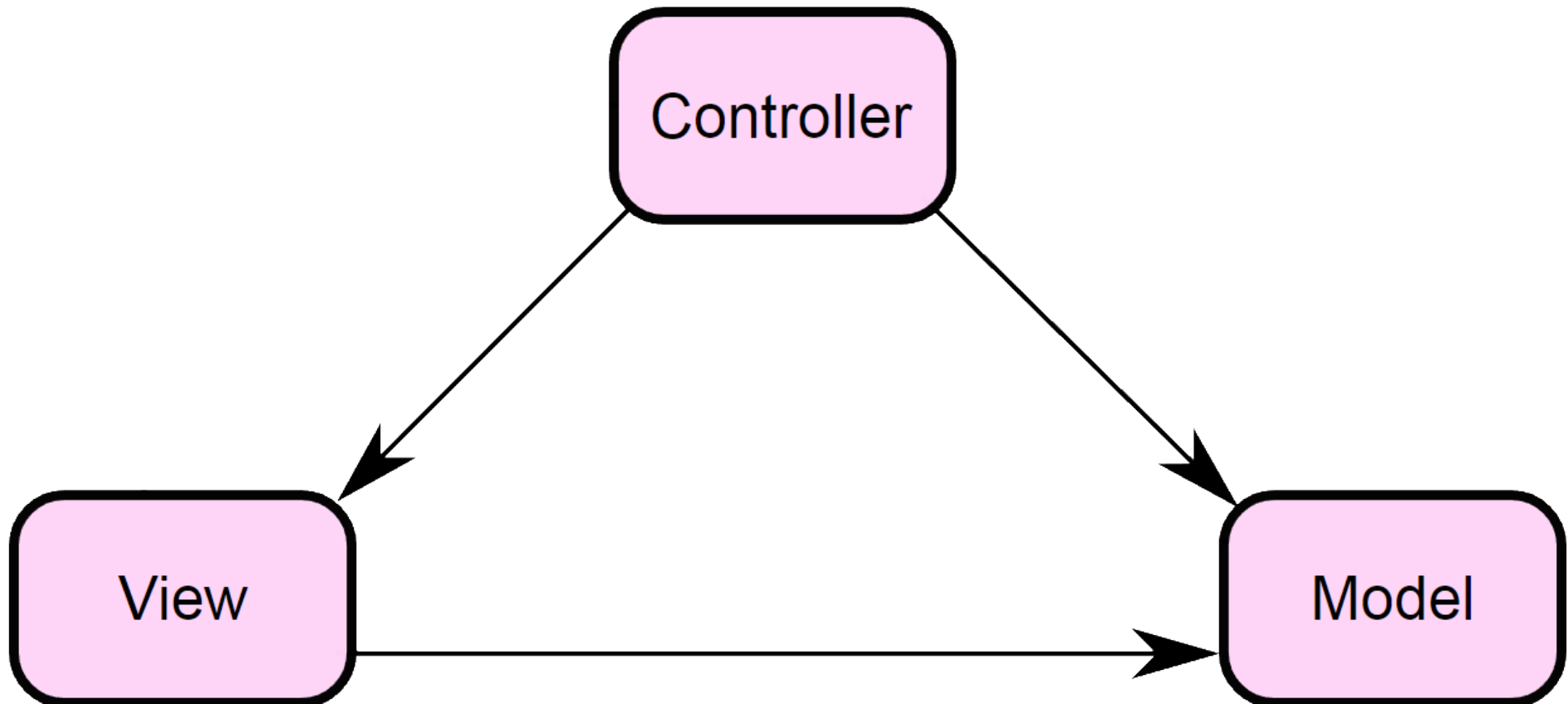
- Model, View, Controller





MVC

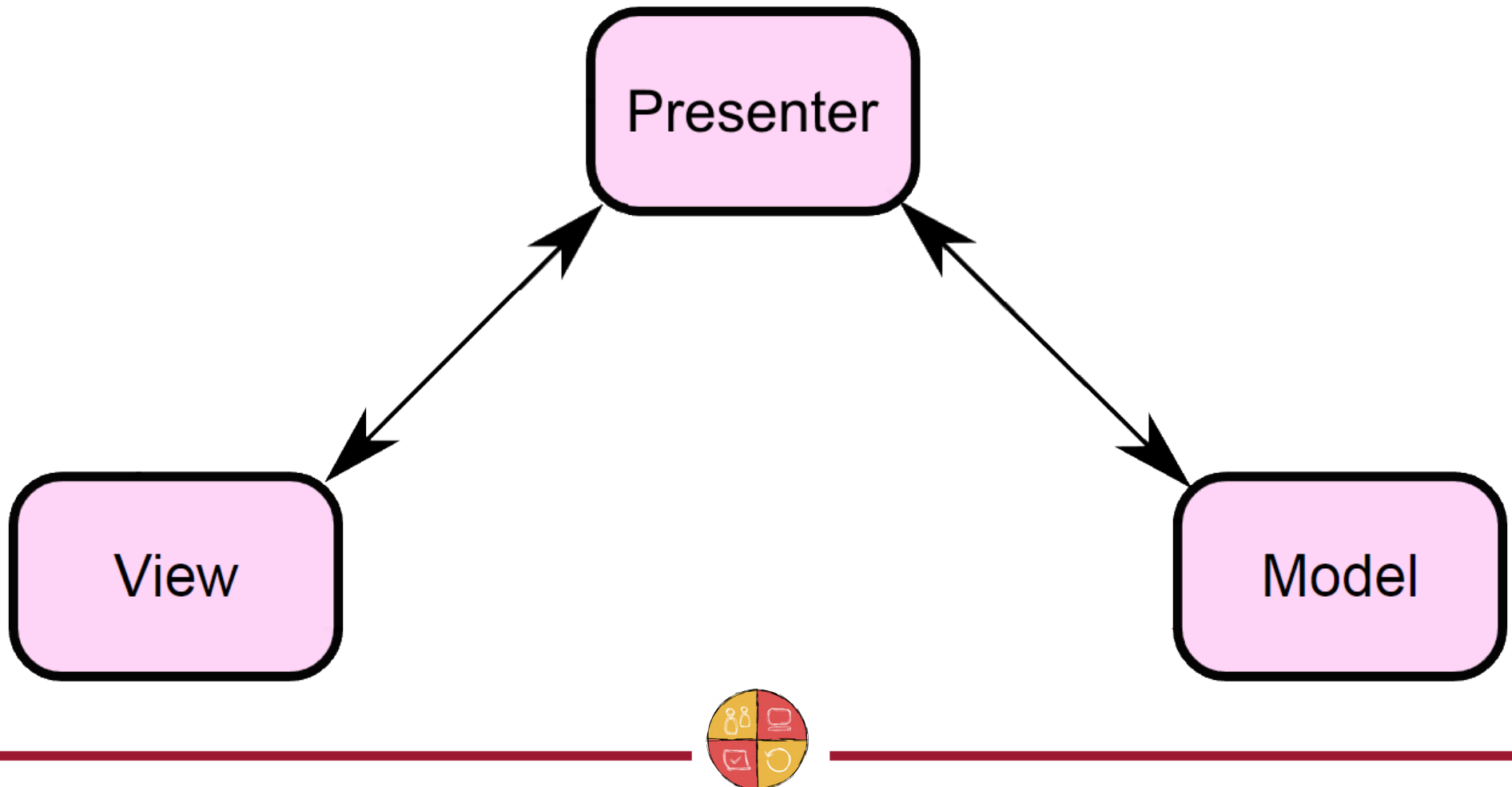
- Model, View, Controller





MVP

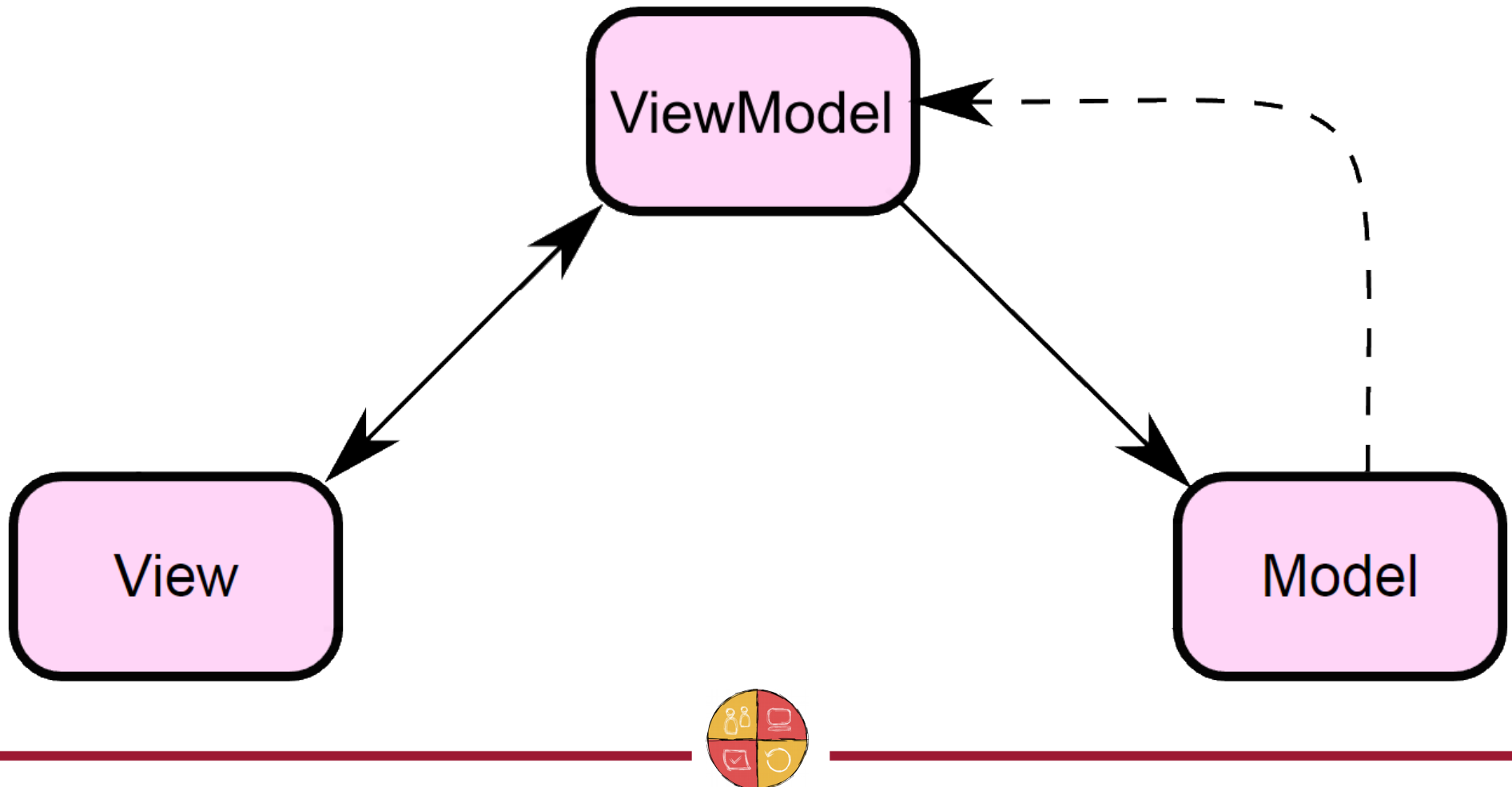
- Model, View, Presenter





MVVM

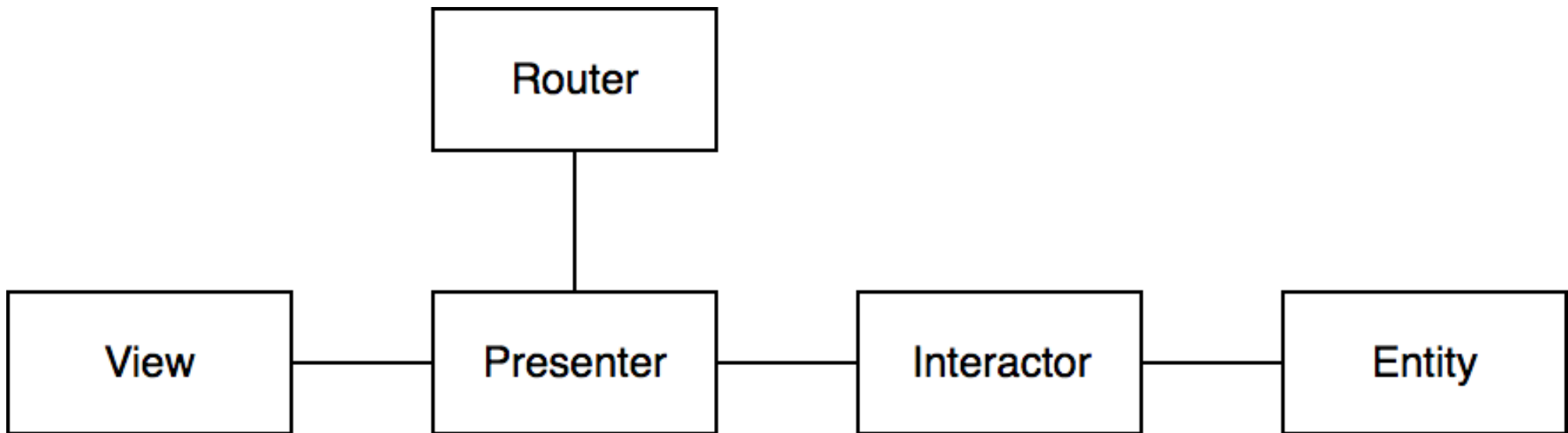
- Model, View, ViewModel





VIPER

- View, Interactor, Presenter, Entity, Router





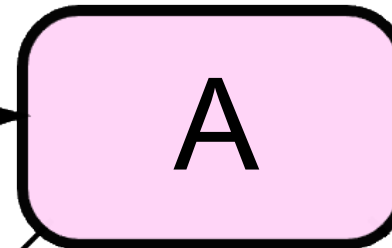
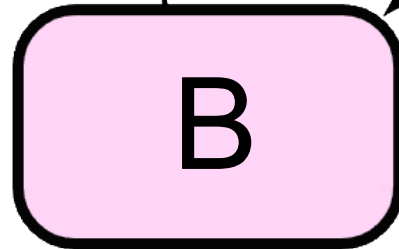
Die Bedeutung der Pfeile

Aufrufpfeil

Runtime Dependency

Systemteil B ruft während der Ausführung Elemente in Systemteil A auf

Code B bekommt erst zur Laufzeit eine Referenz auf Code A



Abhängigkeitspfeil

Compile-Time Dependency

Systemteil A benötigt Systemteil B, um überhaupt compilieren zu können

Code in A referenziert den Code in B direkt mit Namen





Abhängigkeiten gestalten

- Software-Architektur ist die Kunst, die Abhängigkeiten zwischen Systemteilen willentlich und zum Vorteil der Beteiligten zu gestalten
- Die Richtung und Art der Pfeile im Architektordiagramm festzulegen, ist die Aufgabe des Software-Architekten
- Die Richtung kann beliebig gewählt werden
- Wir können die Richtung jederzeit umdrehen!





Inversion of Control



A

```
public class Schalter {  
    private final Lampe lampe;  
    private boolean gedrueckt;  
  
    public Schalter(final Lampe lampe) {  
        this.lampe = lampe;  
    }  
  
    public void drueckeSchalter() {  
        if (this.gedrueckt) {  
            lampe.ausschalten();  
            this.gedrueckt = false;  
            return;  
        }  
        lampe.anschalten();  
        this.gedrueckt = true;  
    }  
}
```



B

```
public class Lampe {  
    private boolean leuchtet = false;  
  
    public void anschalten() {  
        this.leuchtet = true;  
    }  
  
    public void ausschalten() {  
        this.leuchtet = false;  
    }  
}
```





Inversion of Control



A

```
public class Schalter {  
    private final Schaltbar lampe;  
    private boolean gedrueckt;  
  
    public Schalter(Schaltbar lampe) {  
        this.lampe = lampe;  
    }  
  
    public void drueckeSchalter() {  
        if (this.gedrueckt) {  
            lampe.ausschalten();  
            this.gedrueckt = false;  
            return;  
        }  
        lampe.anschalten();  
        this.gedrueckt = true;  
    }  
}
```

```
public interface Schaltbar {  
    public void ausschalten();  
  
    public void anschalten();  
}
```

```
public class Lampe  
    implements Schaltbar {  
  
    private boolean leuchtet = false;  
  
    @Override  
    public void anschalten() {  
        this.leuchtet = true;  
    }  
  
    @Override  
    public void ausschalten() {  
        this.leuchtet = false;  
    }  
}
```





Inversion of Control



```
public interface Schaltbar {  
    public void ausschalten();  
  
    public void anschalten();  
}
```

```
public class Schalter {  
    private final Schaltbar lampe;  
    private boolean gedrueckt;  
  
    public Schalter(Schaltbar lampe) {  
        this.lampe = lampe;  
    }  
  
    public void drueckeSchalter() {  
        if (this.gedrueckt) {  
            lampe.ausschalten();  
            this.gedrueckt = false;  
            return;  
        }  
        lampe.anschalten();  
        this.gedrueckt = true;  
    }  
}
```

B

```
public class Lampe  
    implements Schaltbar {  
  
    private boolean leuchtet = false;  
  
    @Override  
    public void anschalten() {  
        this.leuchtet = true;  
    }  
  
    @Override  
    public void ausschalten() {  
        this.leuchtet = false;  
    }  
}
```



Eigenschaften von Frameworks

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





Framework verwenden

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.MediaType;

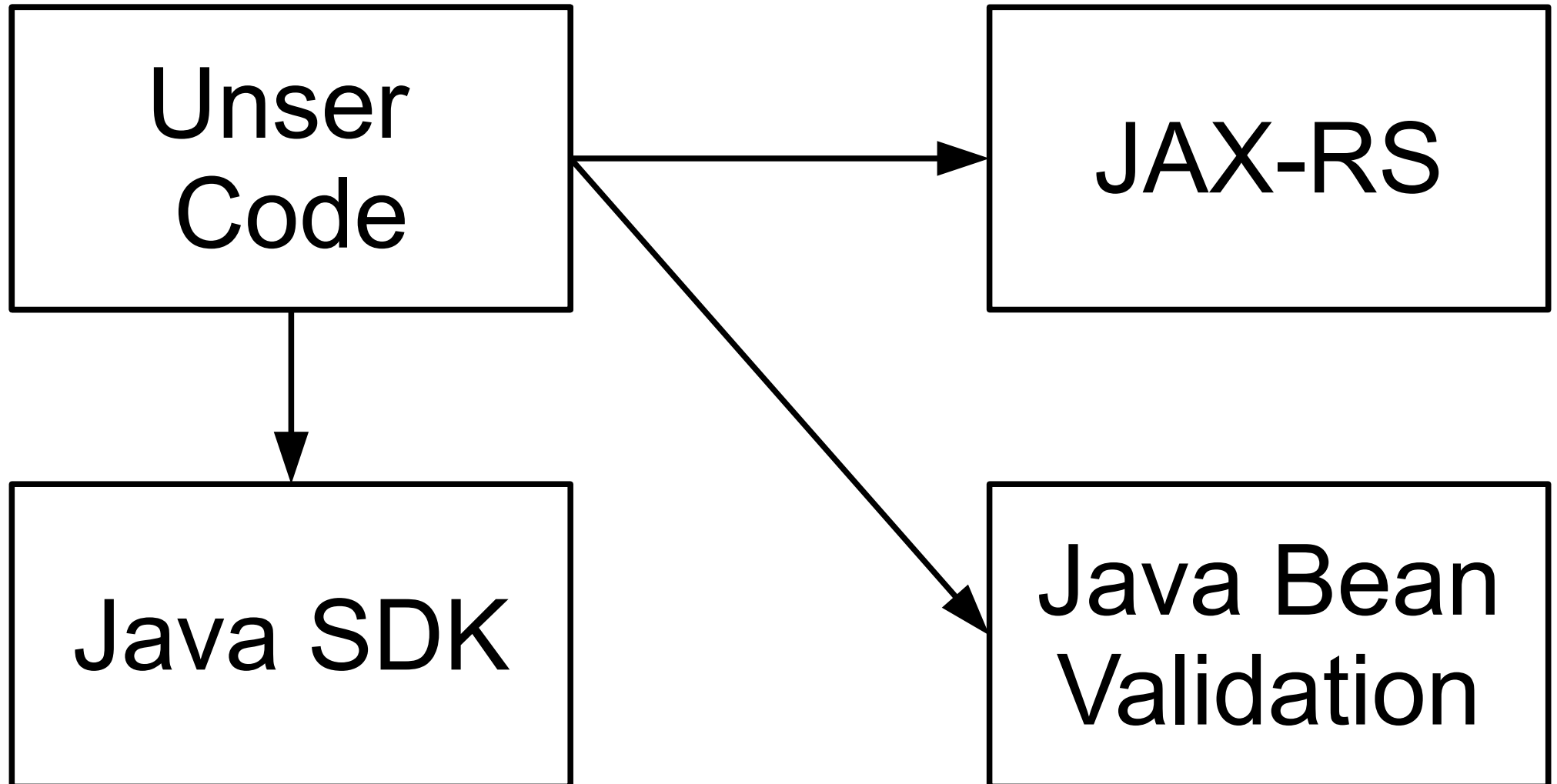
import javax.validation.constraints.NotNull;

@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();
    }
}
```



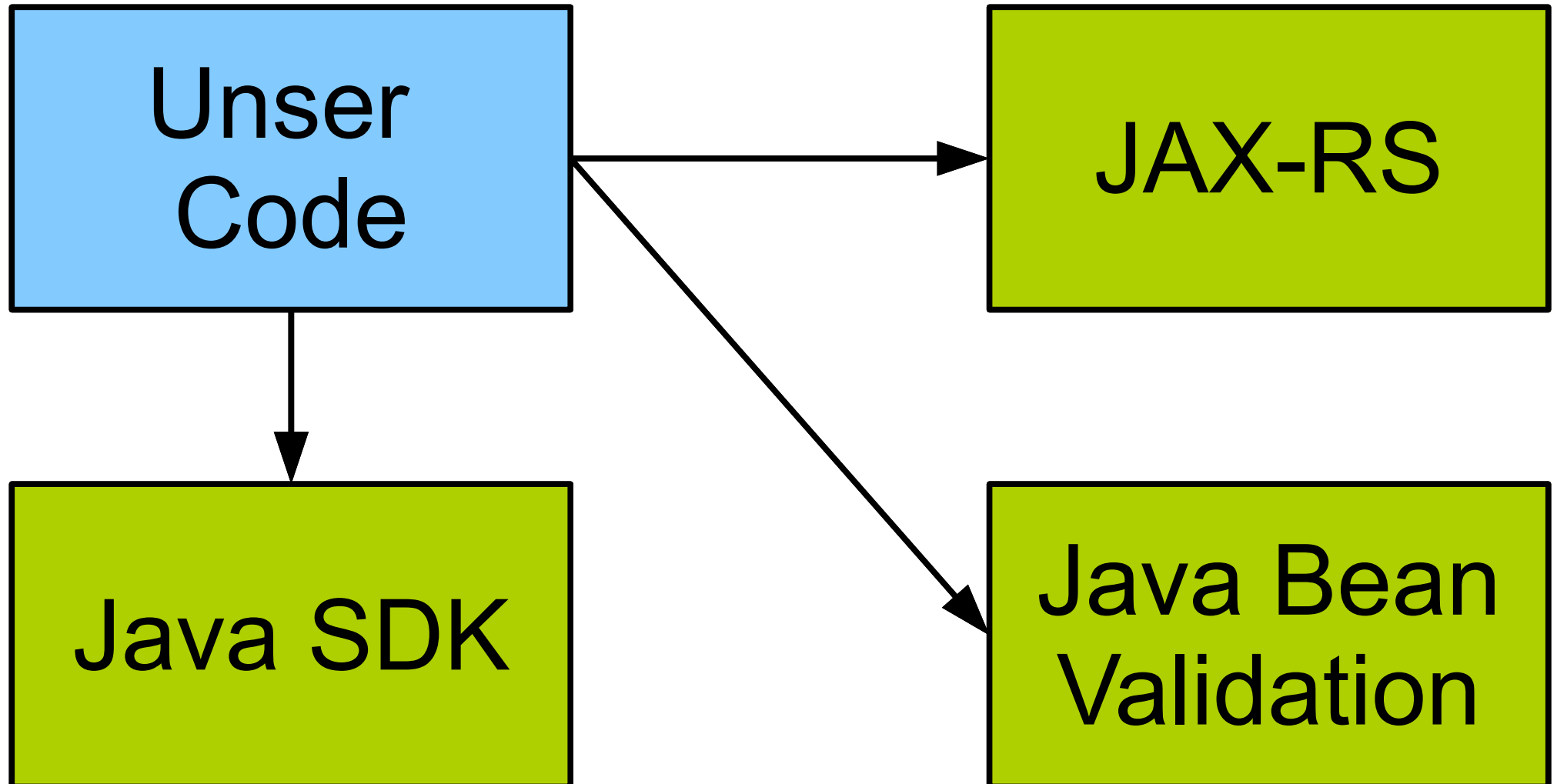


Abhängigkeitspfeile





Area of Control





Abhängigkeitspfeile umdrehen?

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.MediaType;

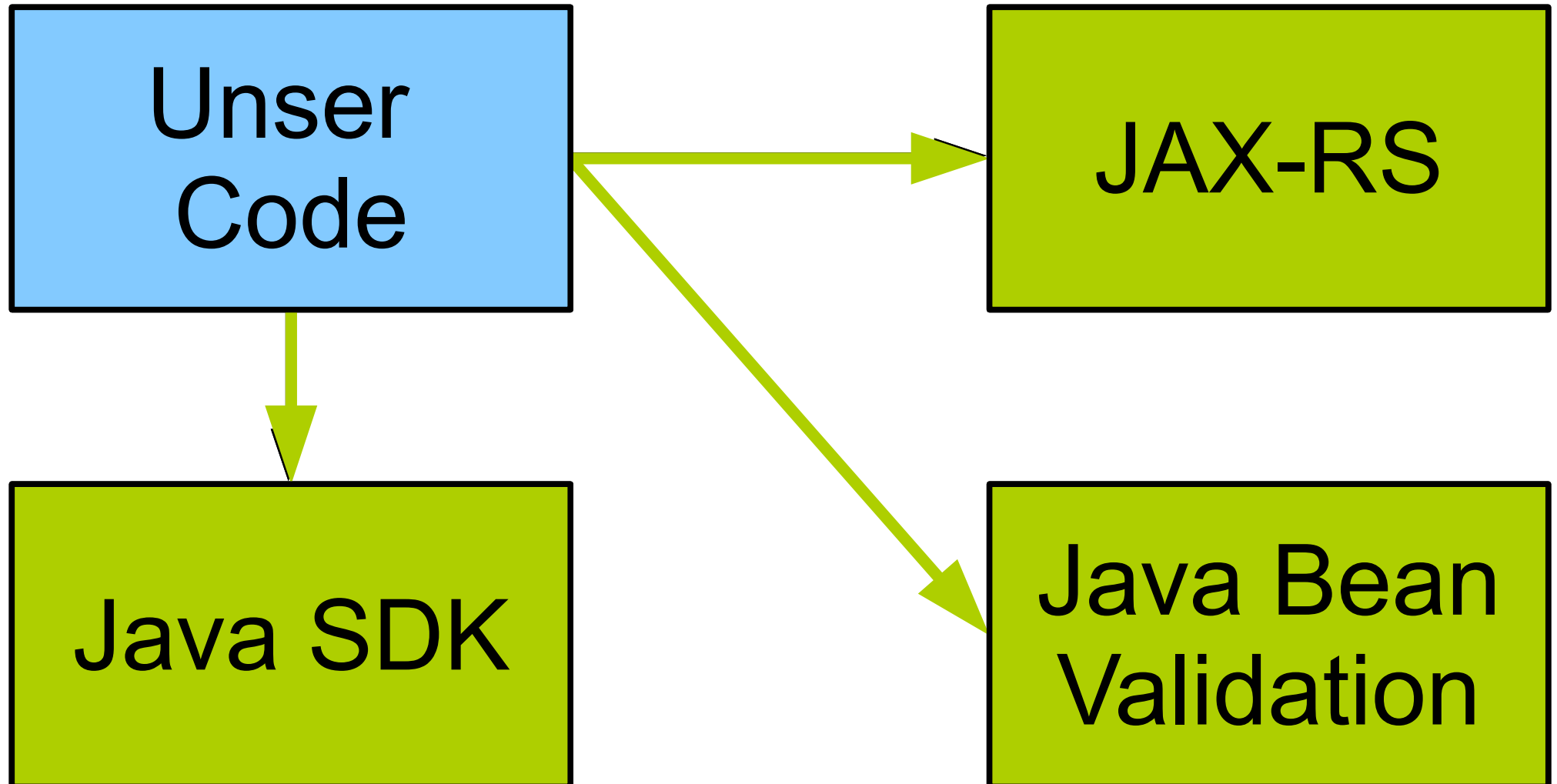
import javax.validation.constraints.NotNull;

@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();
    }
}
```



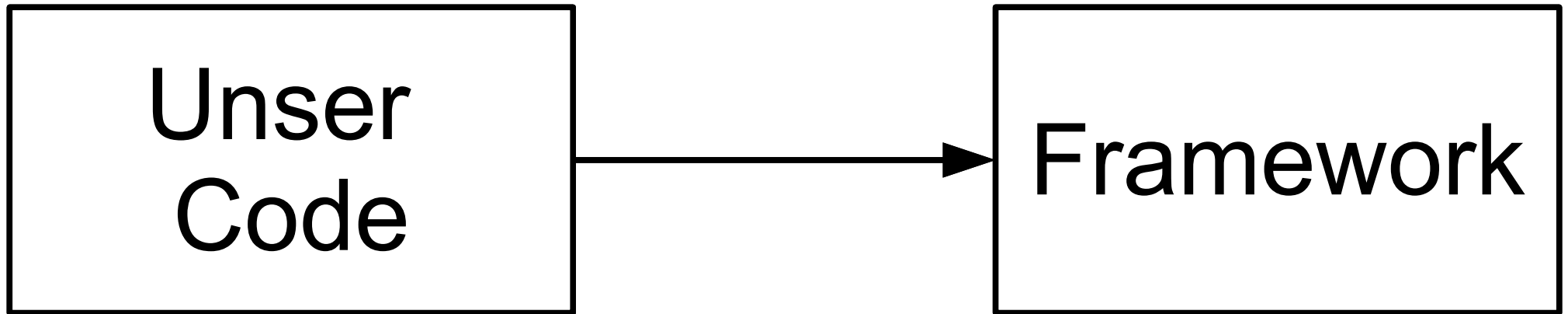


Area of Control (Realität)





Frameworks koppeln hart



- Frameworks zwingen uns normalerweise zu Abhängigkeitspfeilen in nur eine Richtung
- Damit sind wir an den Lebenszyklus des Frameworks gebunden
- Neues Framework? Neuer Code!





Eigenschaften von Libraries

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





Framework vs. Library

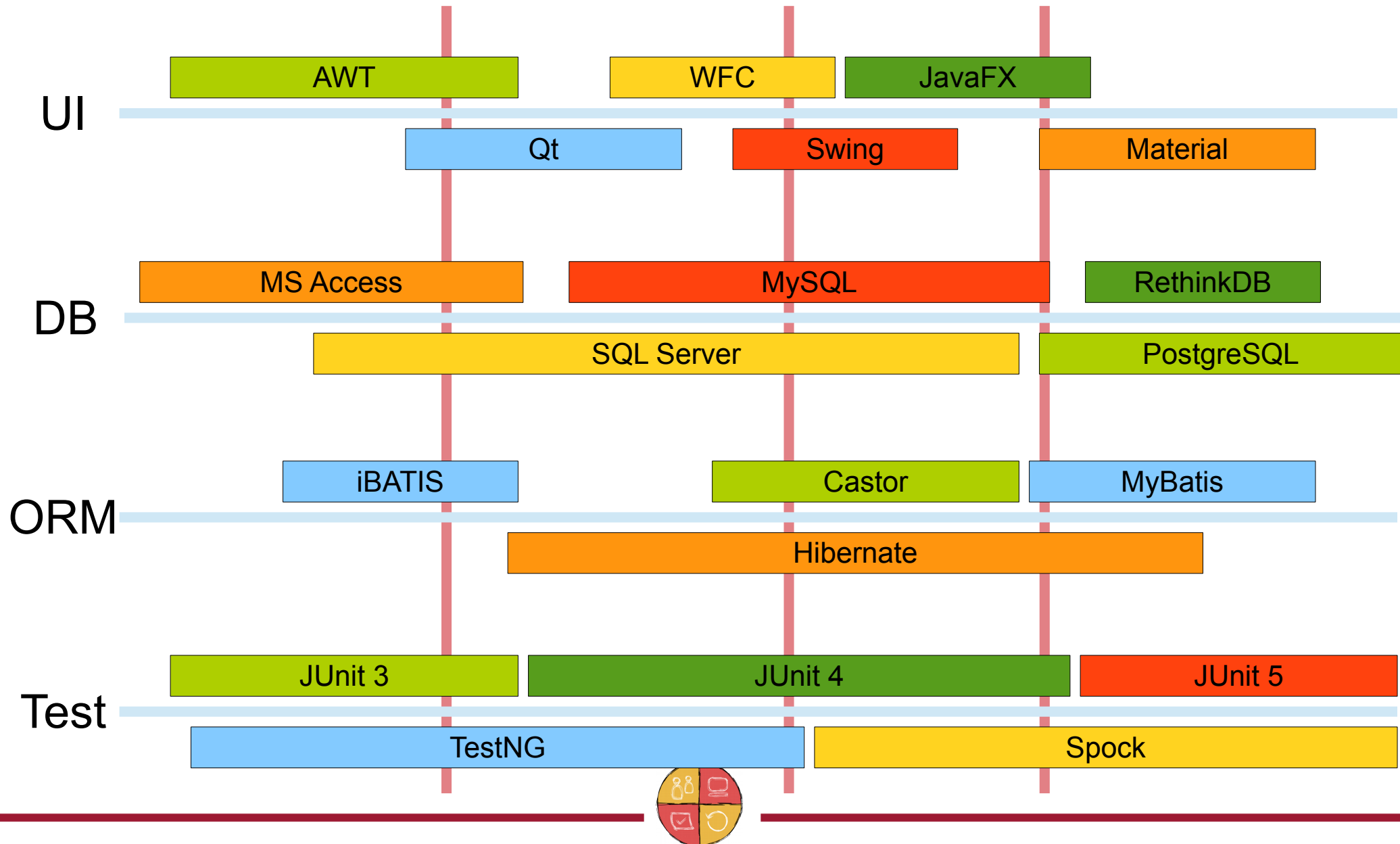
- Frameworks binden die Anwendung an sich
 - Starke Kopplung (Vendor lock-in)
 - Kopplung auch an den Lebenszyklus
- Libraries lassen mehr Freiheiten
 - Starke Kopplung vermeidbar

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



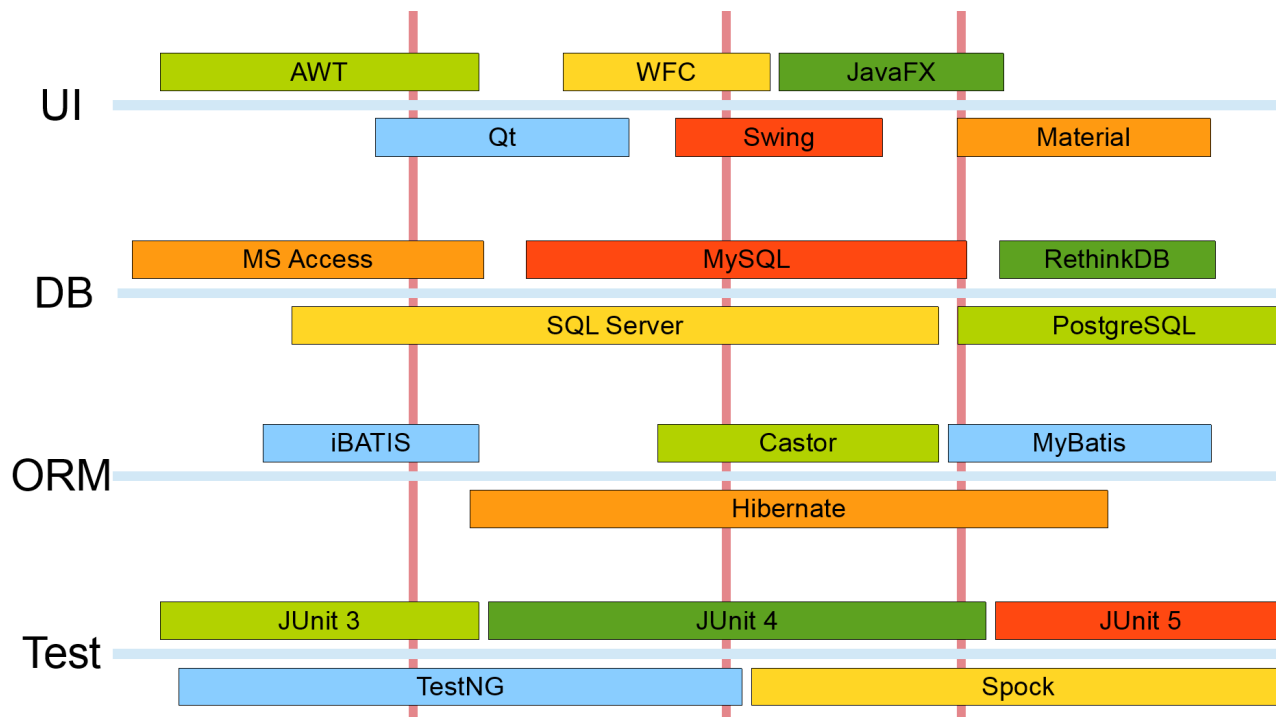


Technologiewahl für Projekte





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

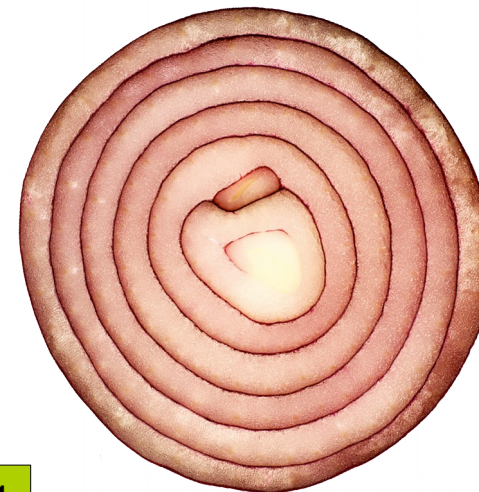
- Gute Entscheidungen werden spät getroffen
- Strukturen (Architektur) so wählen, dass Entscheidungen verzögert werden können
 - Ohne negative Folgen
- Minimalziel: Entscheidungen revidieren können
 - Mit möglichst geringen negativen Folgen





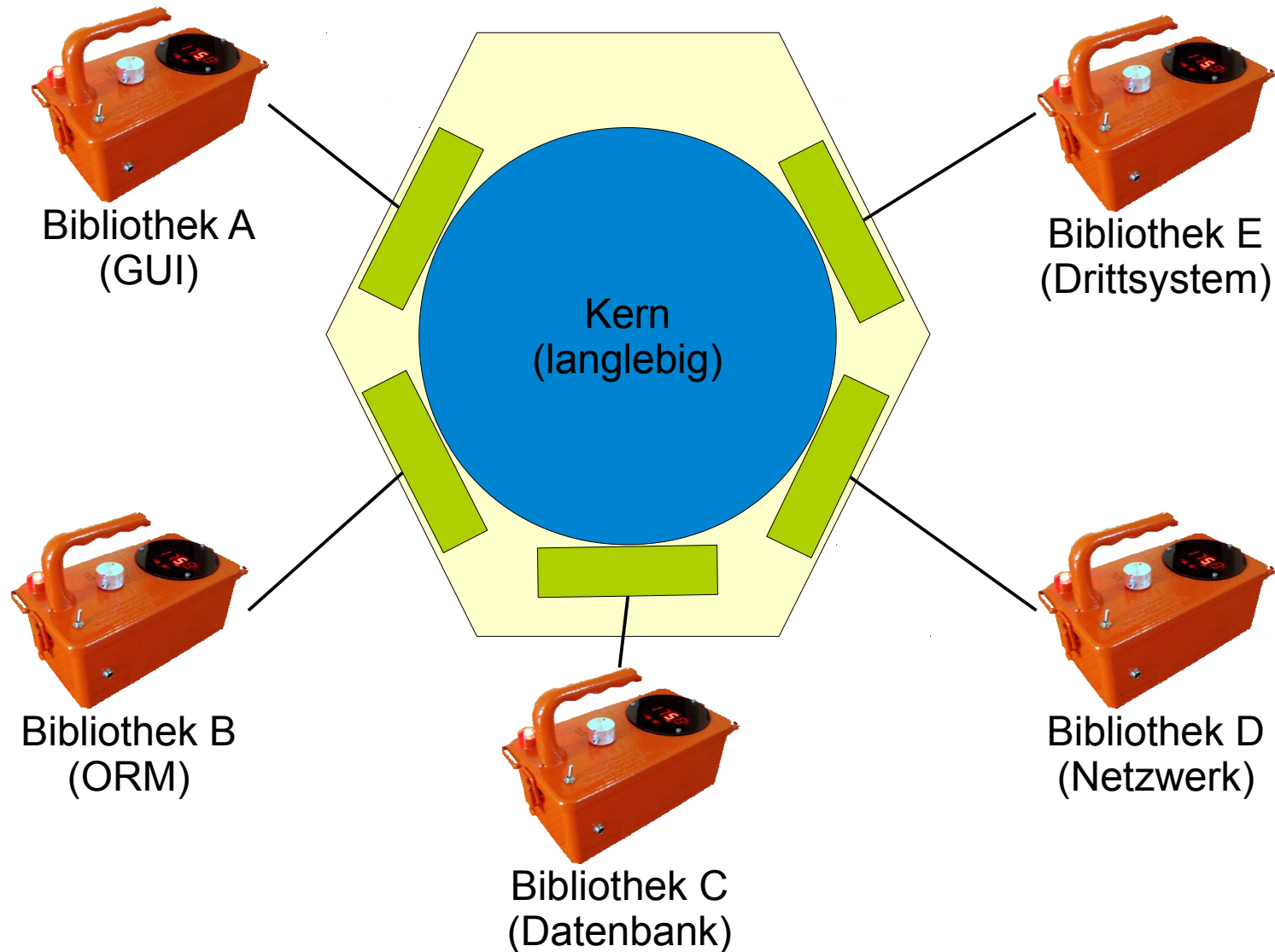
Kriterien für nachhaltige Architektur

- 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“



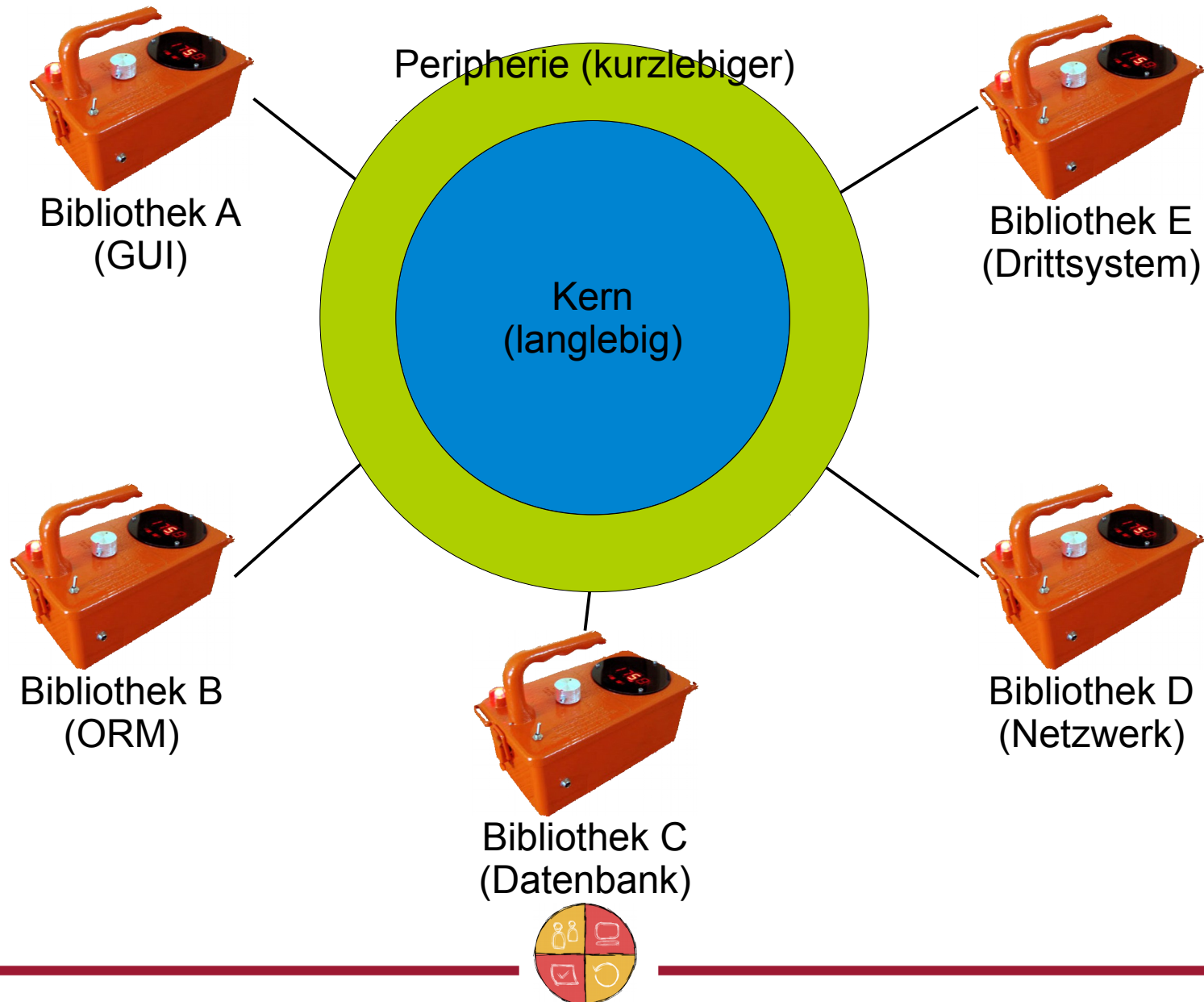


Struktur der Clean Architecture



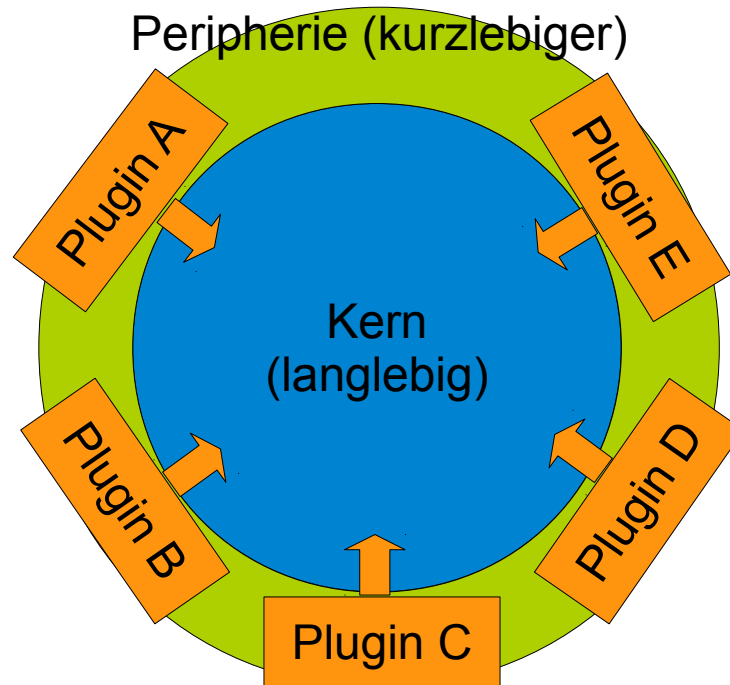


Struktur der Clean Architecture

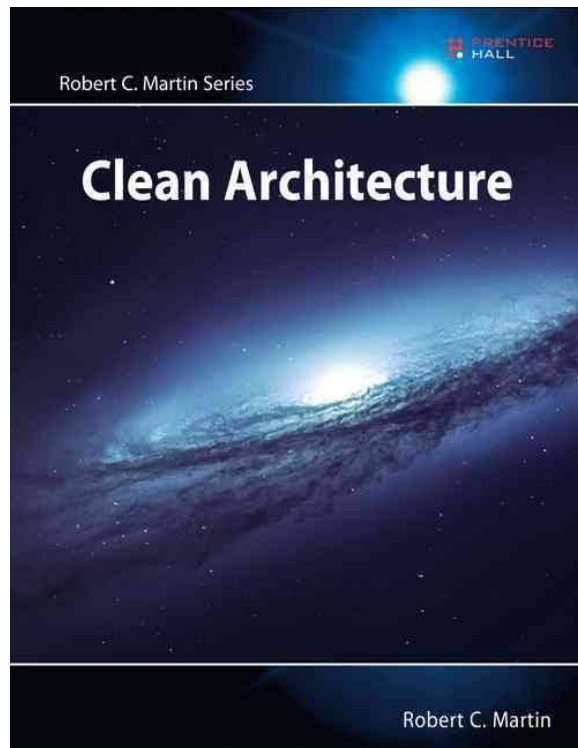




Struktur der Clean Architecture



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





Die Dependency Rule

- Zentrale Regel für Abhängigkeiten

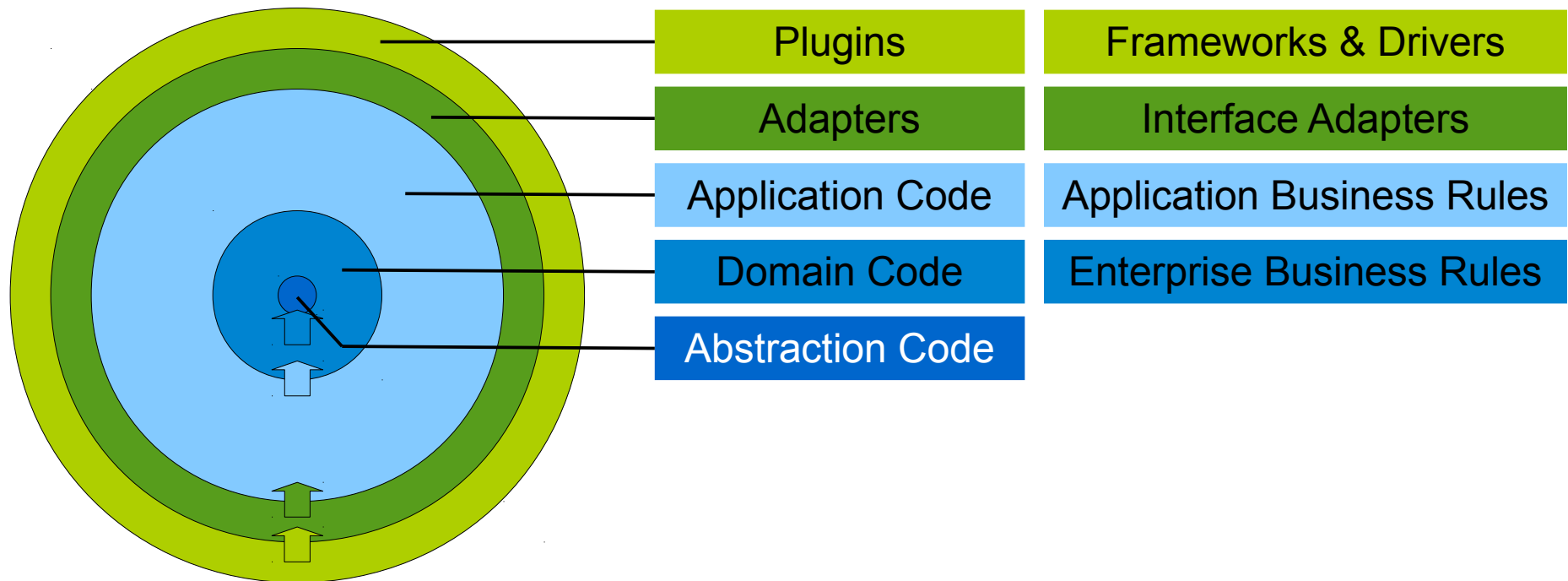
Abhängigkeiten immer von außen nach innen

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





Struktur der Clean Architecture

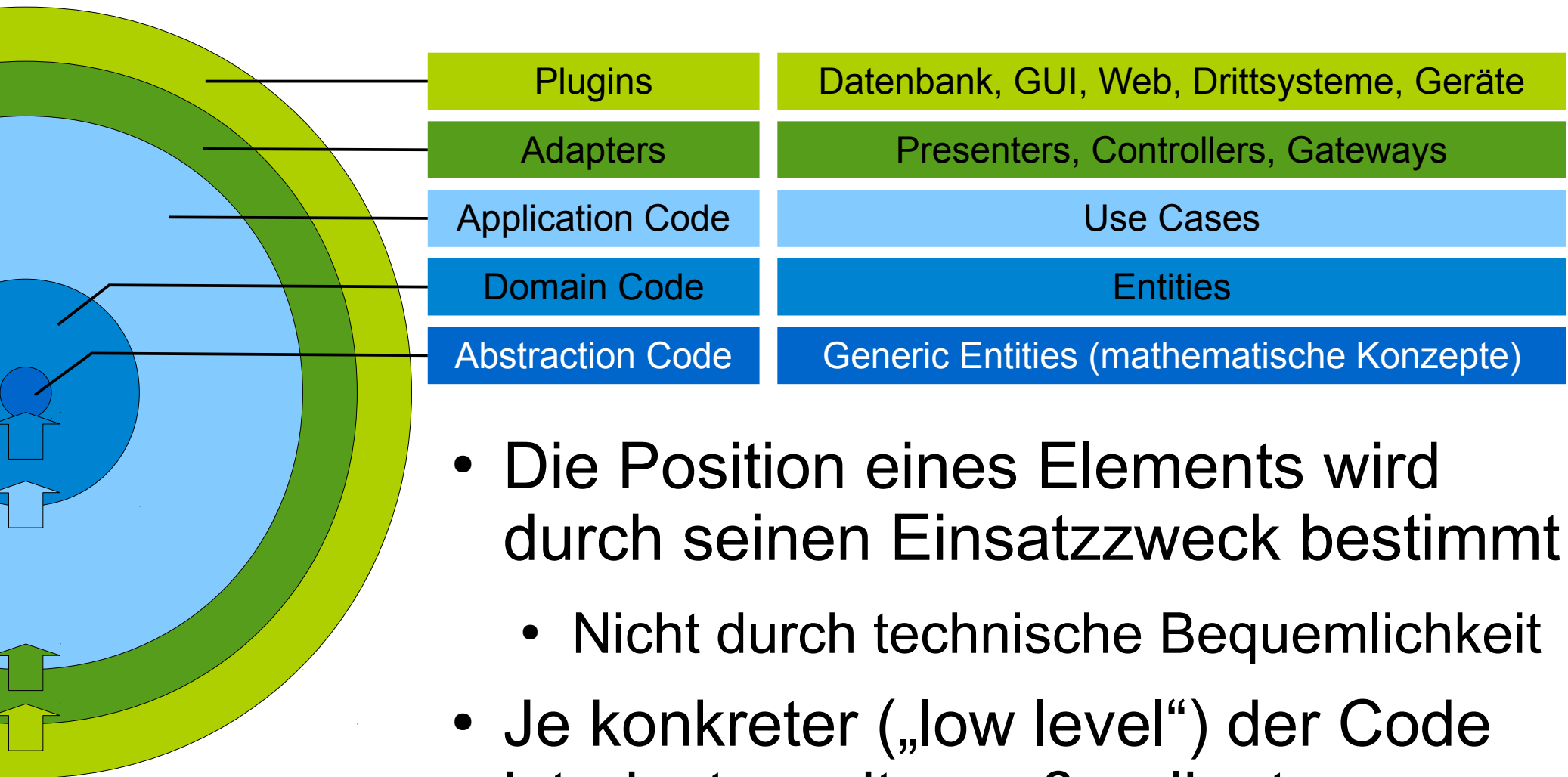


- Innere Schichten wissen nichts von den Äußeren
 - Abhängigkeiten immer von außen nach innen
- Beliebig viele innere Schichten (oft drei)





Position der Elemente





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)





Schicht 4: Abstraction Code

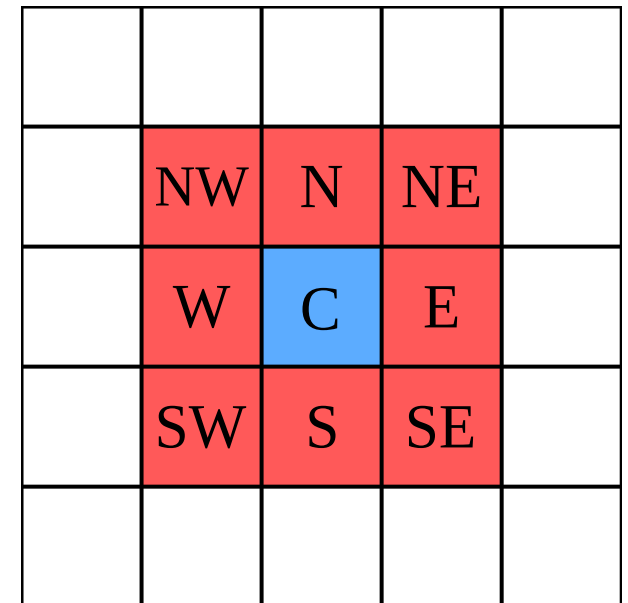
- Enthält domänenübergreifendes Wissen
 - Mathematische Konzepte (z.B. Matrizen)
 - Algorithmen und Datenstrukturen (z.B. Zelluläre Automaten)
 - Abstrahierte Muster (z.B. Quantitäten)
- Häufig nicht notwendig und/oder nicht vorhanden
- Wahrscheinlich bereits als Library verfügbar
- Kann nachträglich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





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!





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>)

Sparkasse Musterstadt

BLZ: 94059421

Home Ihre Sparkasse Service Übersicht Kontakt

Suchbegriff

Finanzstatus

Nach Kontoinhabern sortieren

Giro**

Kontonummer	Kontobezeichnung	Kontoinhaber	Haben	Soll	Saldo
75432	Geschäftsgirokonto	MESSE GMBH	4.619,52 EUR		
10023844	Standard Privatgiro	MAX MUSTER	582,68 EUR		
10023851	Standard Privatgiro	TINA TEST MESSE 2	10.004,96 EUR		
10037505	Sichteinlagen	MAX MUSTER		-22,85 EUR	

Geldanlage

Depot*

Darlehen

Übersicht

Haben	Soll	Gesamtsaldo
31.019,51 EUR		
68.270,78 EUR		
	-33.876,95 EUR	
114.497,45 EUR	-33.899,80 EUR	80.597,65 EUR

* NearTime-Depotbewertung (Kurse mindestens 15 Minuten zeitverzögert)

** Kontostand kann Beträge mit späterer Wertstellung enthalten. [Siehe Hilfe](#)

Druckansicht Aktualisieren

Info-Box

Automatische Abmeldung bei nicht aktiver Nutzung gegen 12:10 Uhr.

Mini-Finanzstatus

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

Service Telefon 01805 400 501*

E-Mail schreiben

Filiale finden

Notfallnummern

Newsletter Abo

* 0,09 Euro/Min. aus dem deutschen Festnetz; Mobilfunk-Preise ggf. abweichend.



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


 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	Kontostand	
Privatgirokonto ** Lebensmittel	123456	1.000,00 EUR	    
Tagesgeld Rücklage	200000	10.200,00 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
 





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<span>&nbsp;EUR</span></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	i ☰ ➔ 🔄 📄
Tagesgeld ** Rücklage	200905	18.235,00 EUR	i ☰ ➔ 🔄 📄
Girokonto (USD) **	654321	1.000,00 USD (734,65 EUR)	i ☰ 🔄
Termingeld	223344556	15.000,00 EUR	i ☰ 📄
Deka *	000100000	47.472,85 EUR	i ☰ ➔ 🔄 📄

[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](#)



Positionierung: Beispiel 1

Use Case

Plugins

Adapters

Application

Domain

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

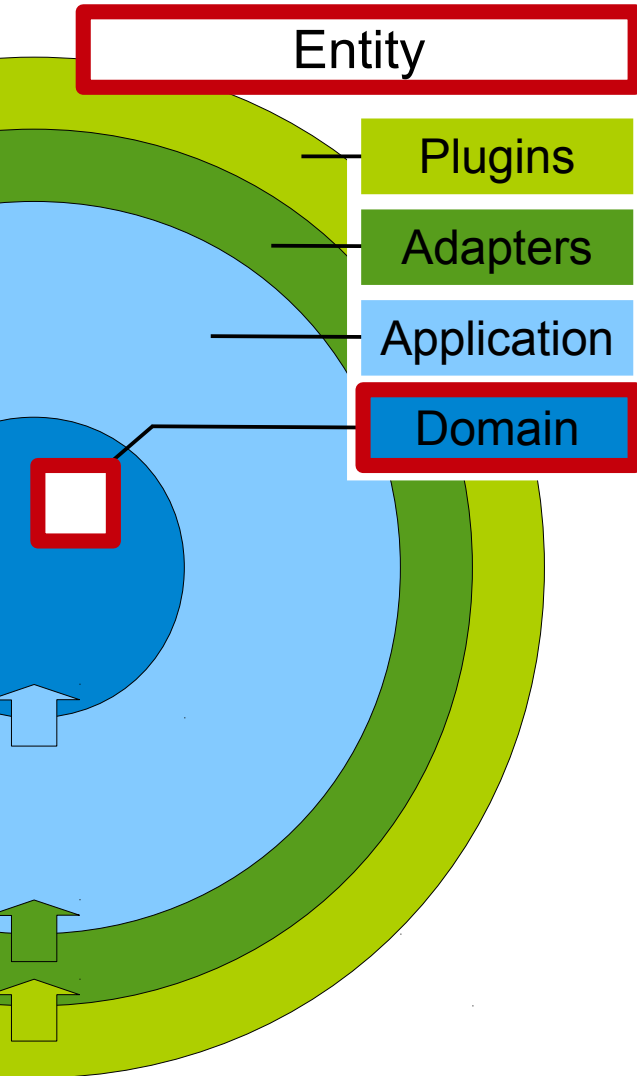
    @Inject
    public ChangeUserPassword(
        final AuthenticationService authenticationService) {
        this.authenticationService = authenticationService;
    }

    @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

Framework Driver

Plugins

Adapters

Application

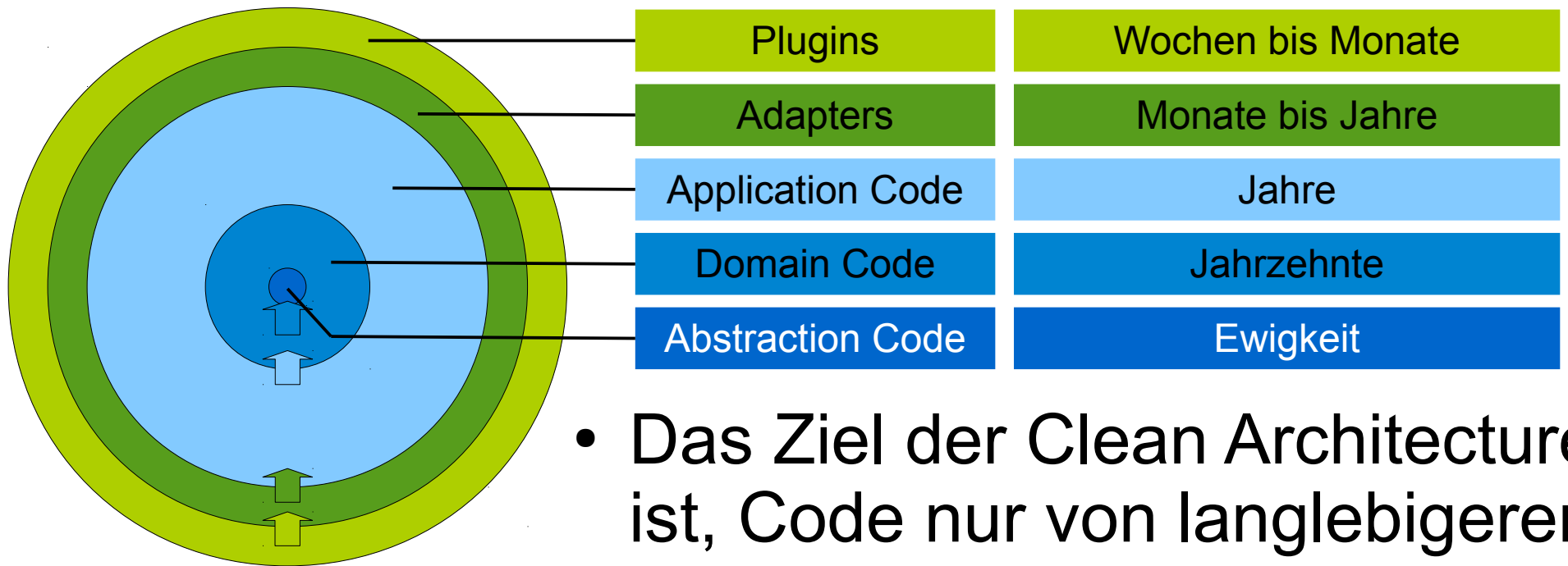
Domain

```
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();  
    }  
    [...]  
}
```





Ziel der Clean Architecture

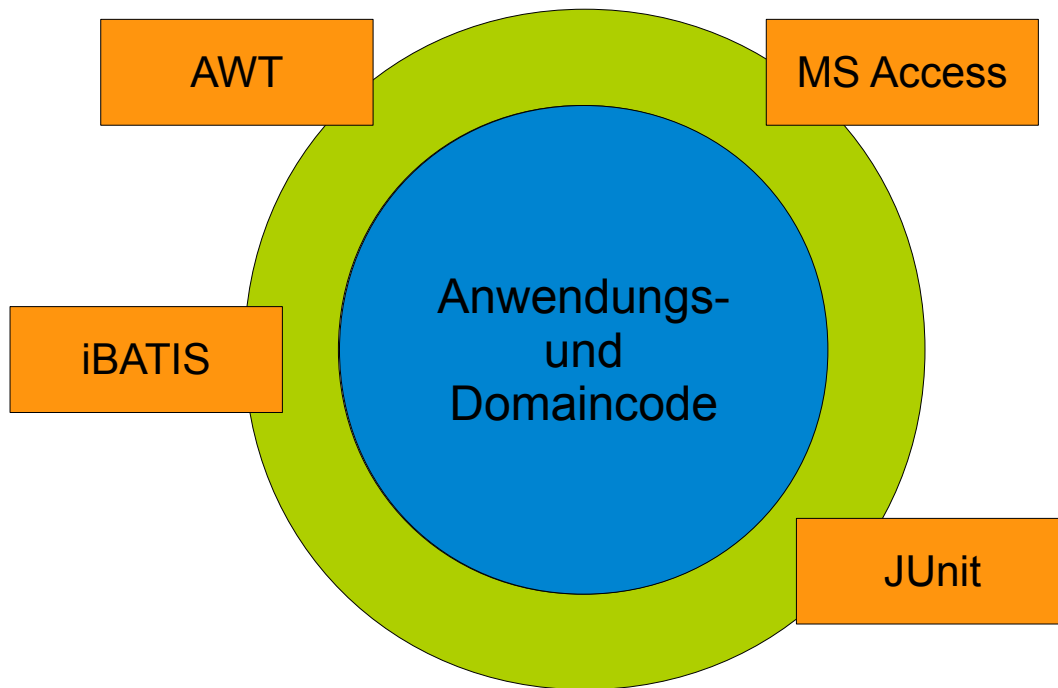


- Das Ziel der Clean Architecture ist, Code nur von langlebigerem Code abhängig zu machen
- Wenn sich Technologien ändern müssen, kann die Anwendung unverändert bleiben





Clean Architecture Technologiewahl



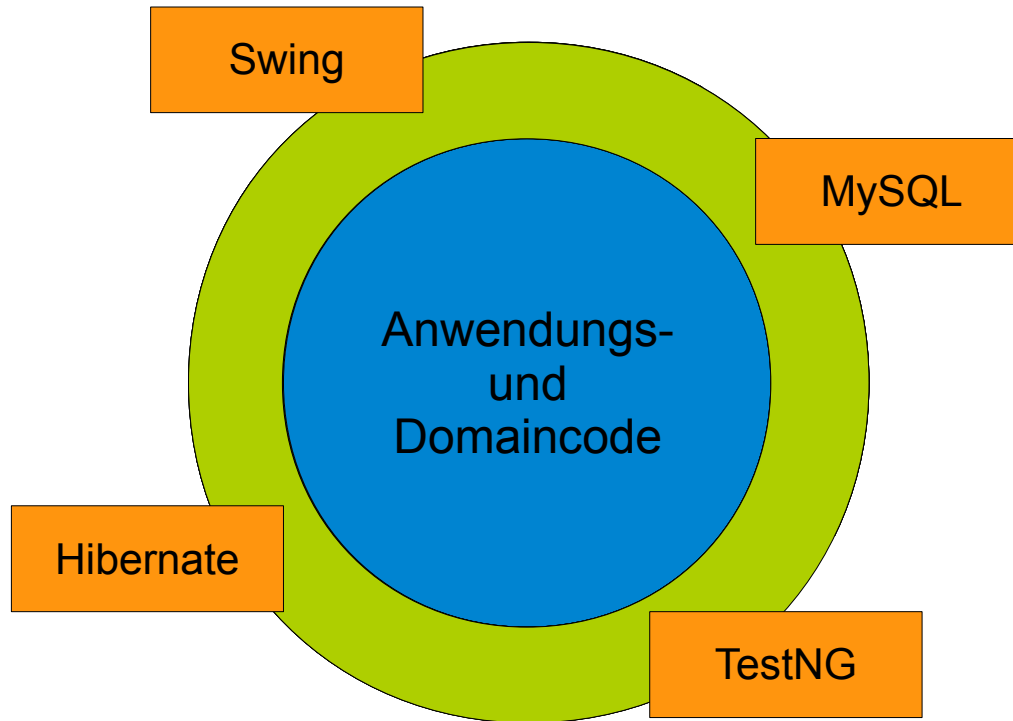
UI	AWT	Swing	JavaFX
DB	MS Access	MySQL	RethinkDB
ORM	iBATIS	Hibernate	MyBatis
Test	JUnit	TestNG	Spock

- 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

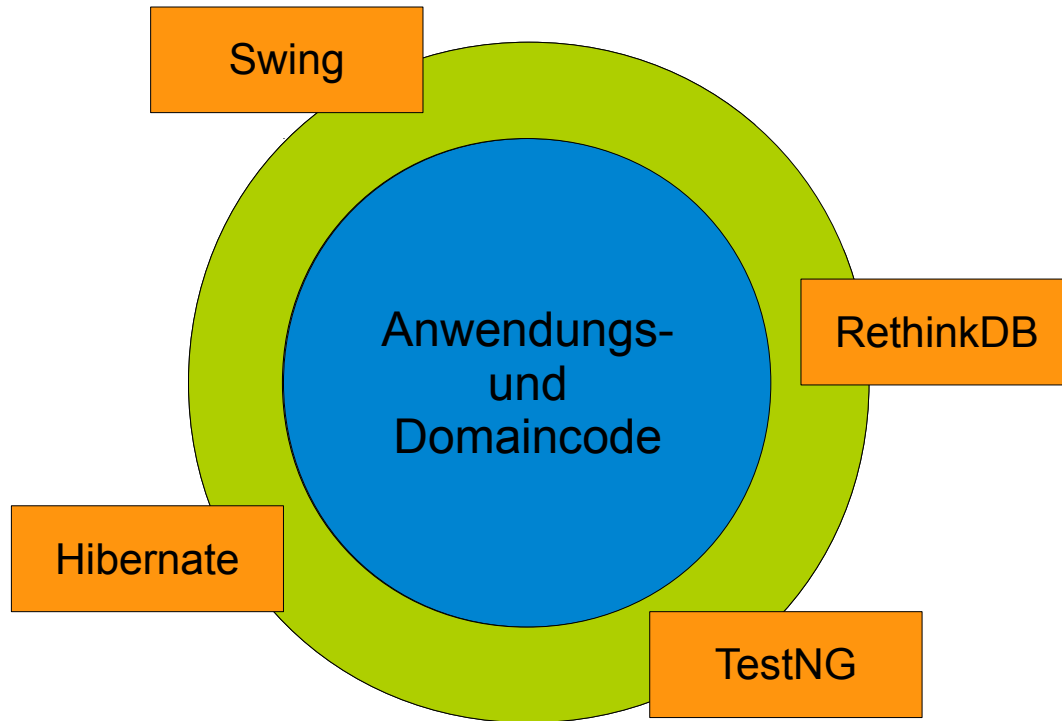
UI	AWT	Swing	JavaFX
DB	MS Access	MySQL	RethinkDB
ORM	iBATIS	Hibernate	MyBatis
Test	JUnit	TestNG	Spock

- 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





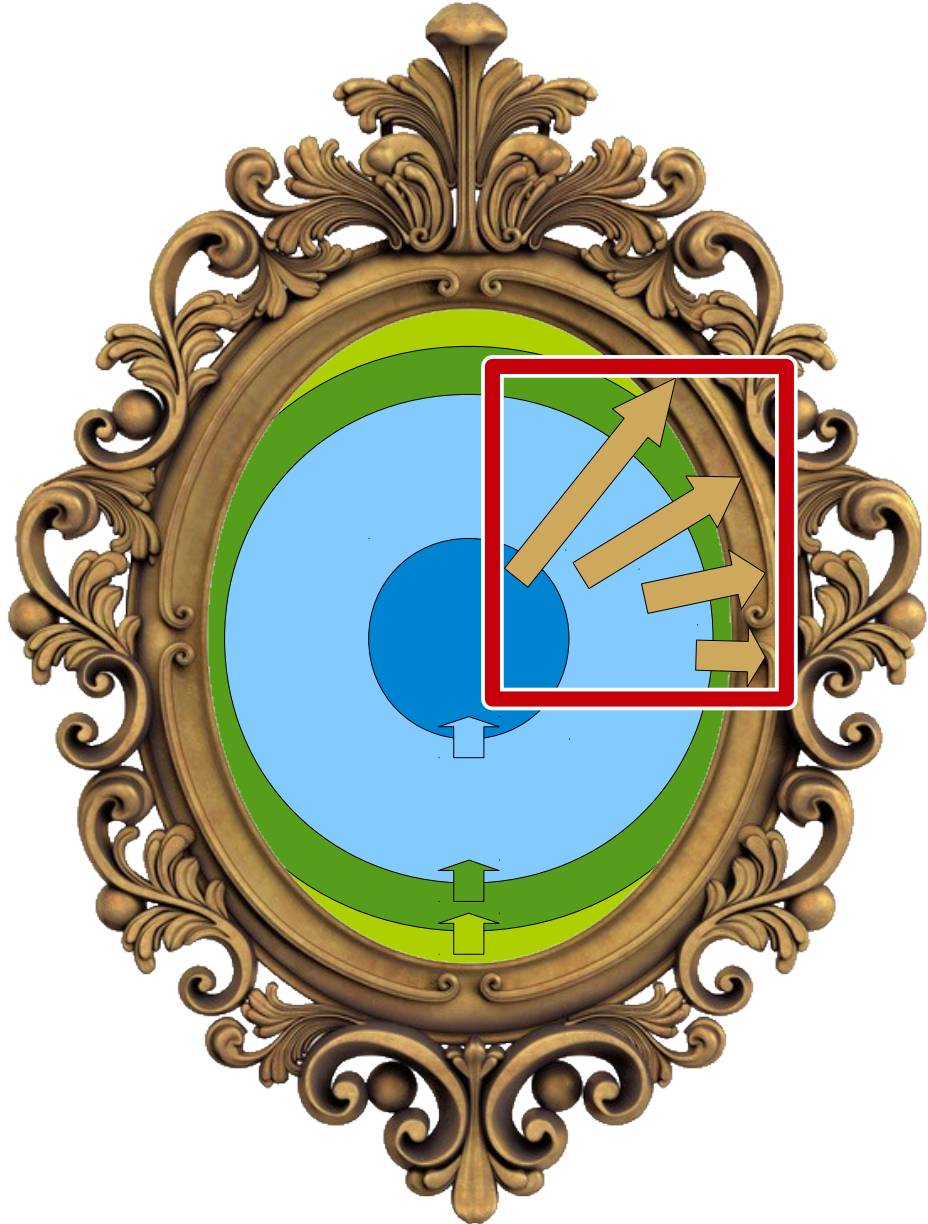
Grenzen der Clean Architecture

- 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

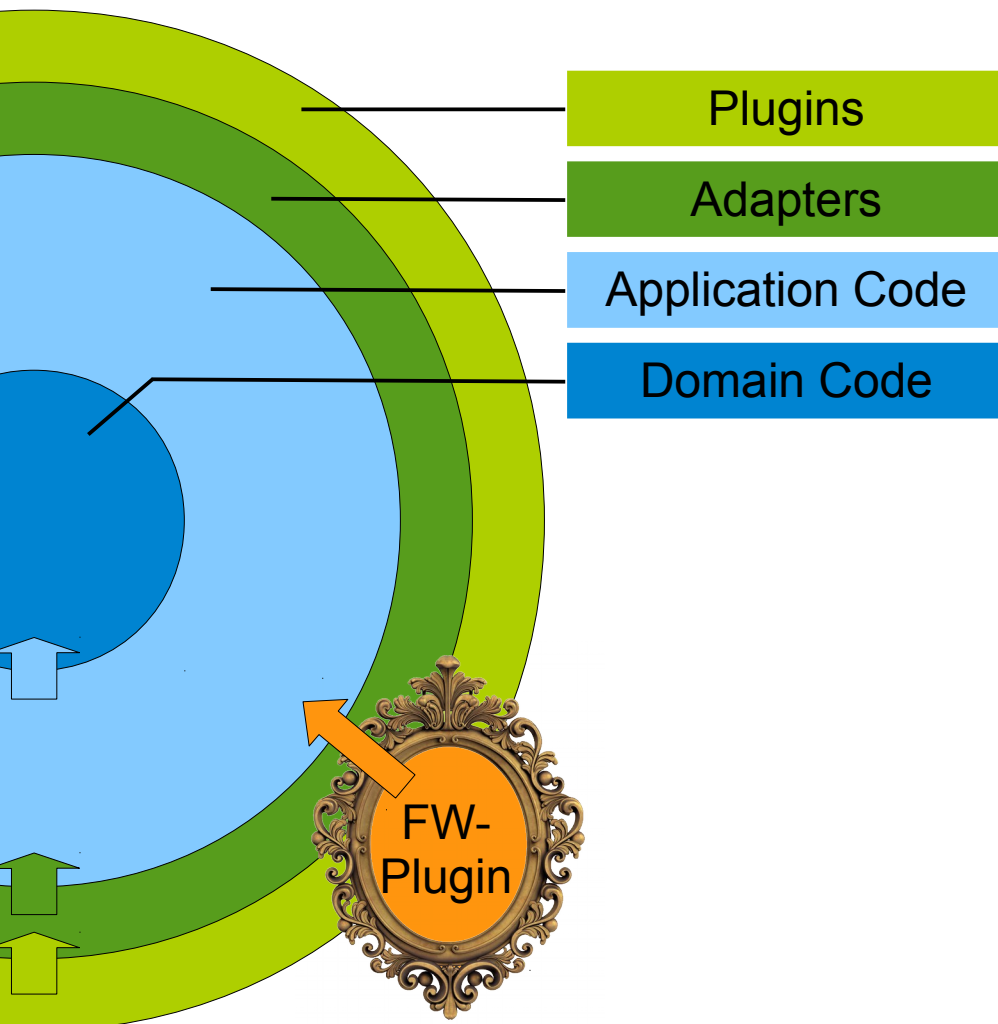
Clean Architecture und Frameworks



- 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

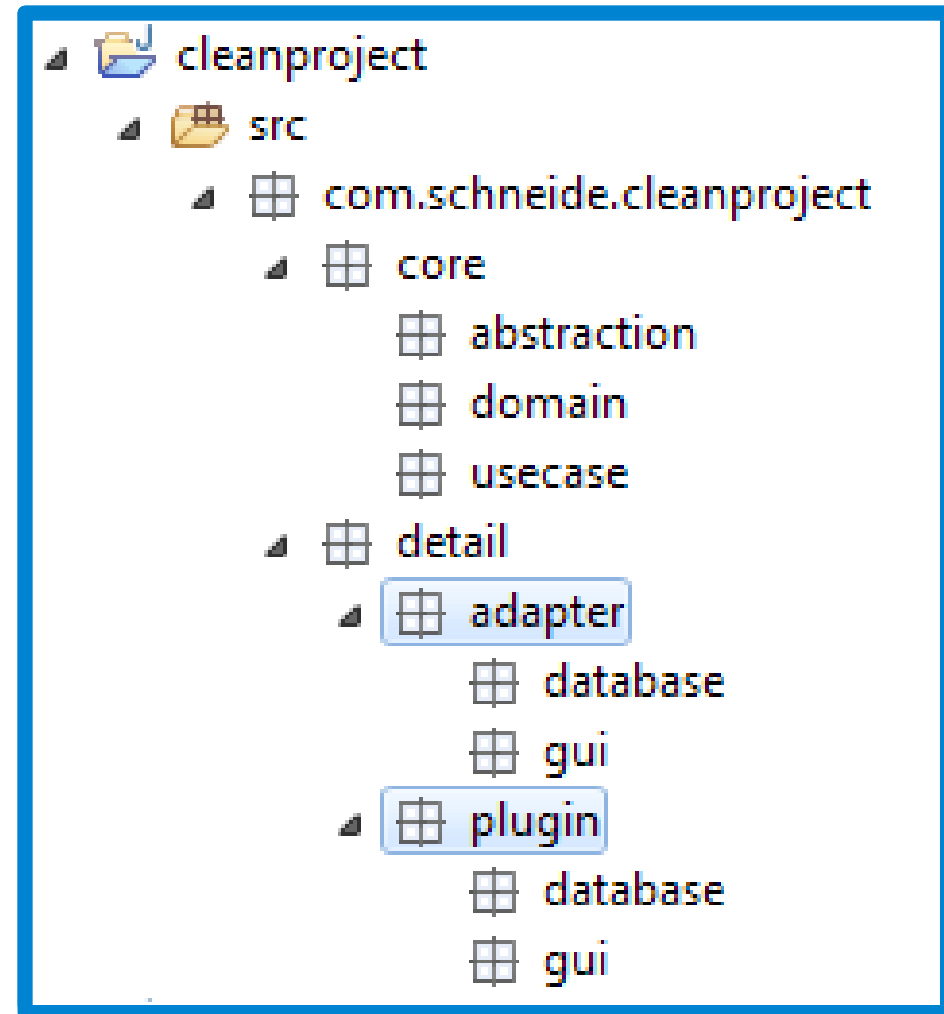
- Am besten den Code inkl. Framework in eigenes Projekt auslagern
 - Framework-Projekt referenziert Anwendungs-Projekt
- Anwendung muss unabhängig vom Framework bau- und betreibbar sein
- Die Schnittstelle für das Framework-Projekt wird eventuell sehr spezifisch ausfallen
 - Versuchung widerstehen, eine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





Konkrete Umsetzung: Einzelprojekt

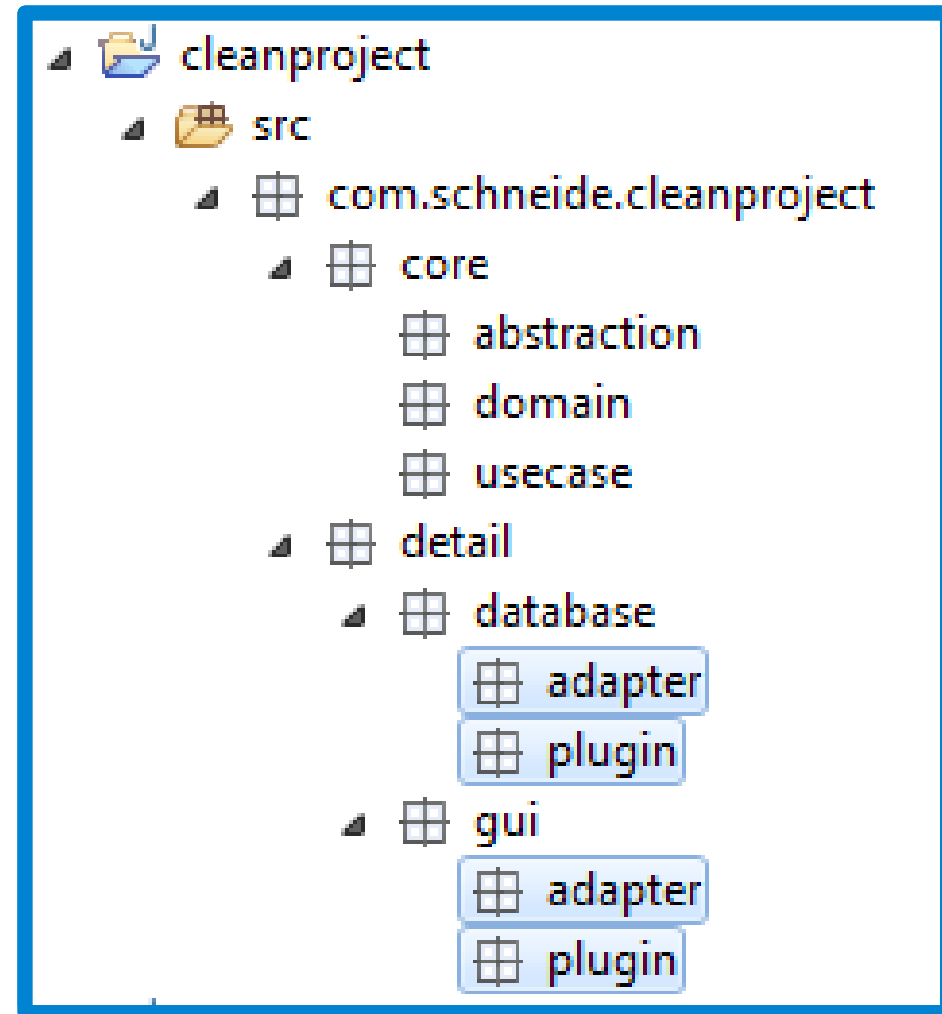
- Alle Klassen in einem Projekt
 - Packages bilden die Schichten ab
 - Namensgebung spiegelt die Reihenfolge wider
- Nachteile:
 - Keine Überprüfung durch den Compiler
 - Namen sind noch schwieriger zu finden





Konkrete Umsetzung: Einzelprojekt

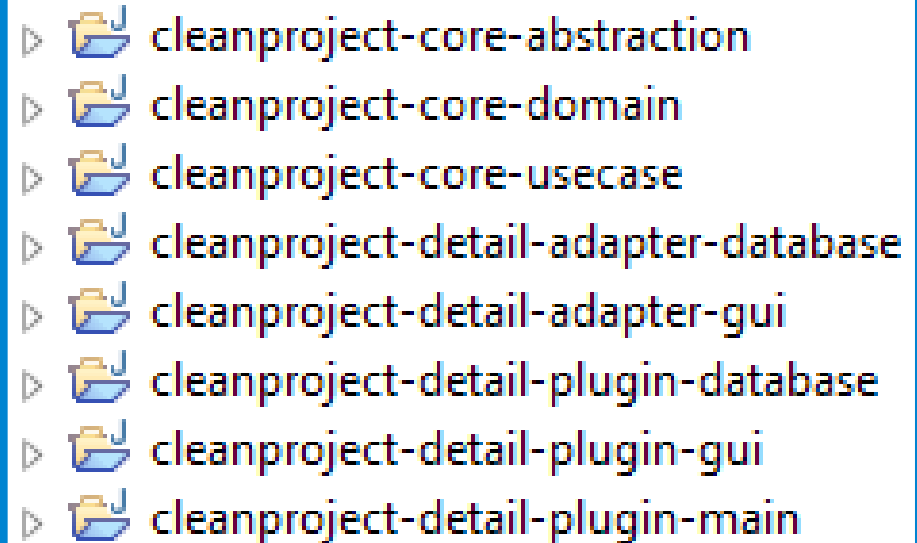
- Alle Klassen in einem Projekt
 - Packages bilden die Schichten ab
 - Namensgebung spiegelt die Reihenfolge wider
- Nachteile:
 - Keine Überprüfung durch den Compiler
 - Namen sind noch schwieriger zu finden





Konkrete Umsetzung: Mehrere Projekte

- Jede Schicht in einem eigenen Projekt
 - Projektabhängigkeiten spiegeln die Architekturabhängigkeiten wider
 - Reihenfolge der Projekte durch Projektnamen
- Vorteil:
 - Der Compiler zeigt nur noch erlaubte Typen
 - Typen im eigenen Projekt und Projekten, von denen das Projekt (transitiv) abhängig ist

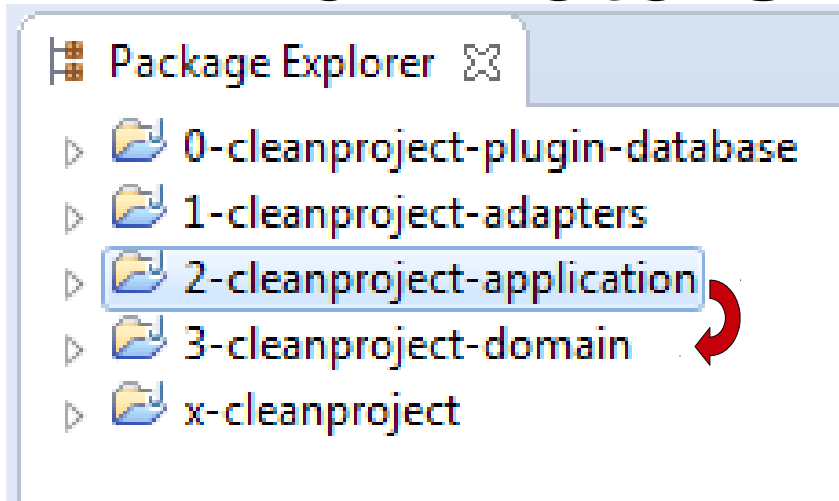


- ▷ cleanproject-core-abstraction
- ▷ cleanproject-core-domain
- ▷ cleanproject-core-usecase
- ▷ cleanproject-detail-adapter-database
- ▷ cleanproject-detail-adapter-gui
- ▷ cleanproject-detail-plugin-database
- ▷ cleanproject-detail-plugin-gui
- ▷ cleanproject-detail-plugin-main

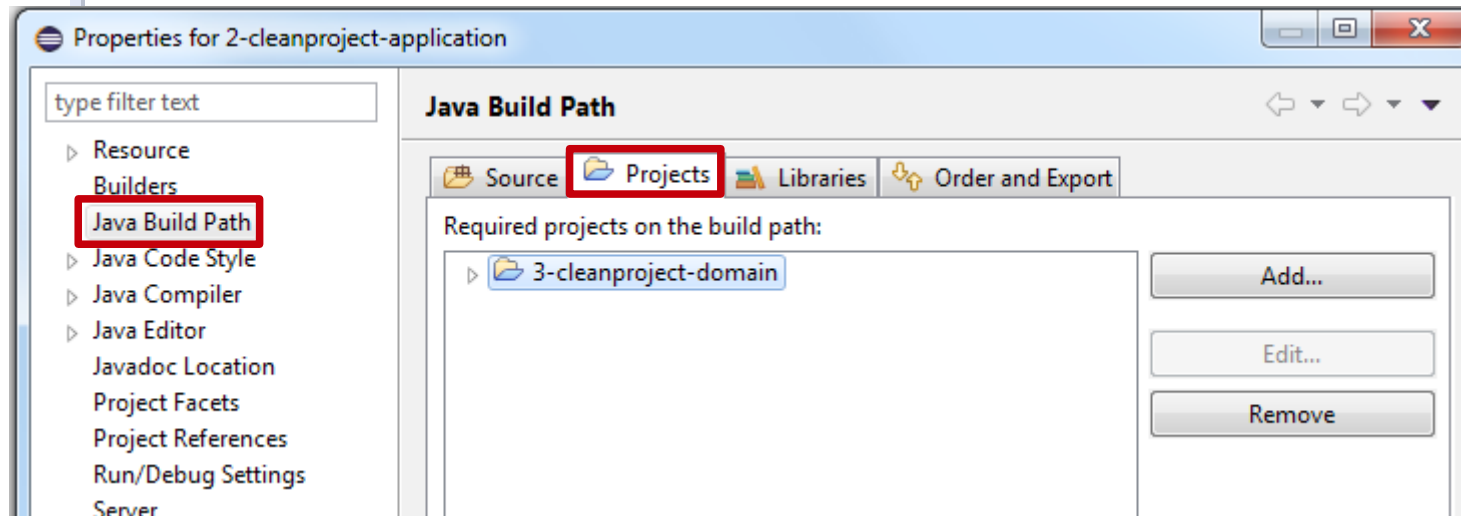




Konkrete Umsetzung: Eclipse

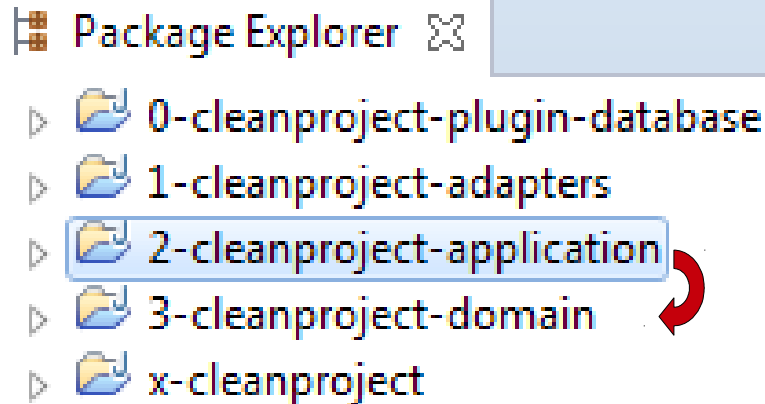


- Projekt 2 soll von Projekt 3 abhängen
- In den Eclipse-Projekteinstellungen angeben

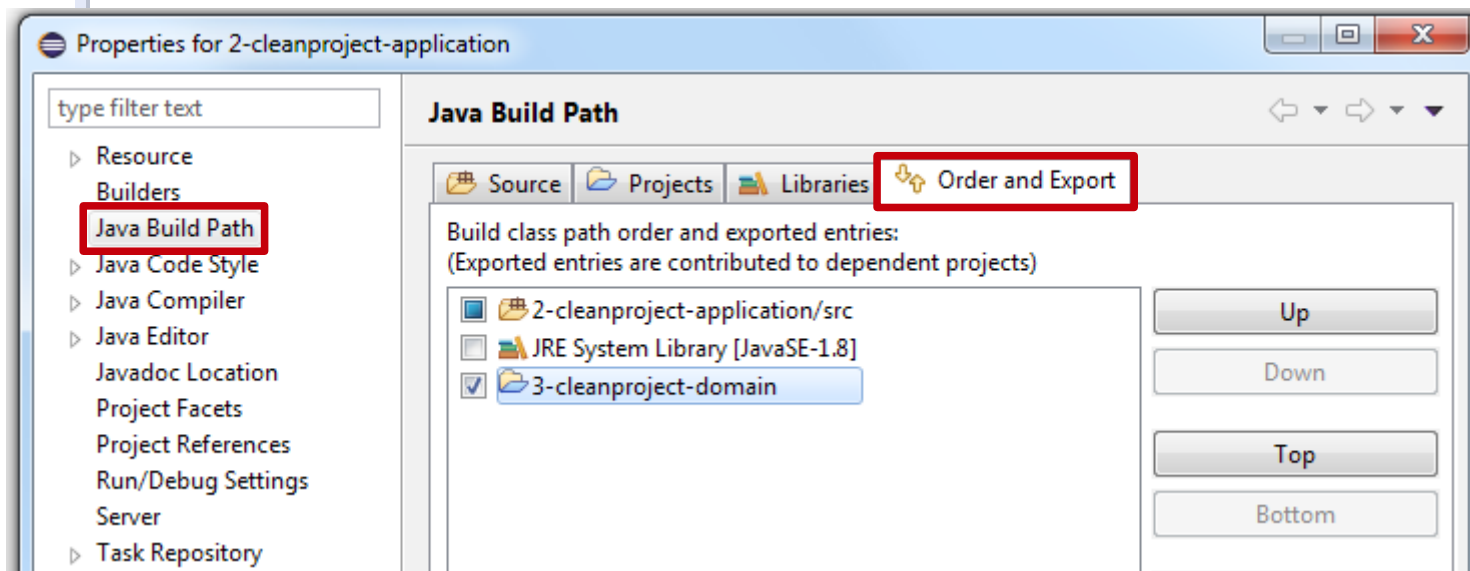




Konkrete Umsetzung: Eclipse



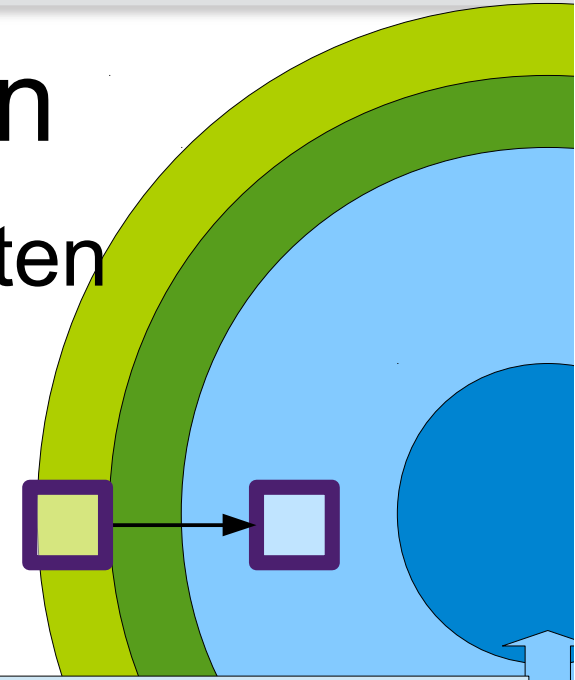
- Transitive Abhängigkeiten freigeben
- In den Eclipse-Projekteinstellungen





Übergabe von Daten

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



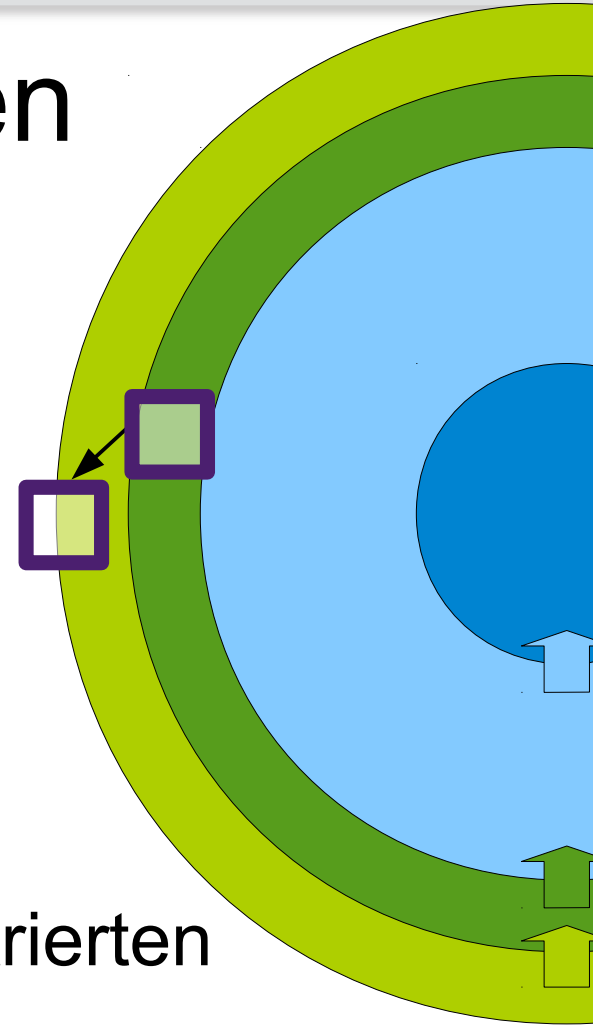
```
@Path("/epapers")
public class EPapersEndpoint {
    @GET
    @Path("")
    @Produces(MediaType.APPLICATION_JSON + "; charset=utf-8")
    public Response findEPapers(
        @QueryParam("iln")
        @NotNull(message = "add required query parameter iln") String holdingsIlN) {
        return Response.ok(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

← **Application Code**



Übergabe von Daten

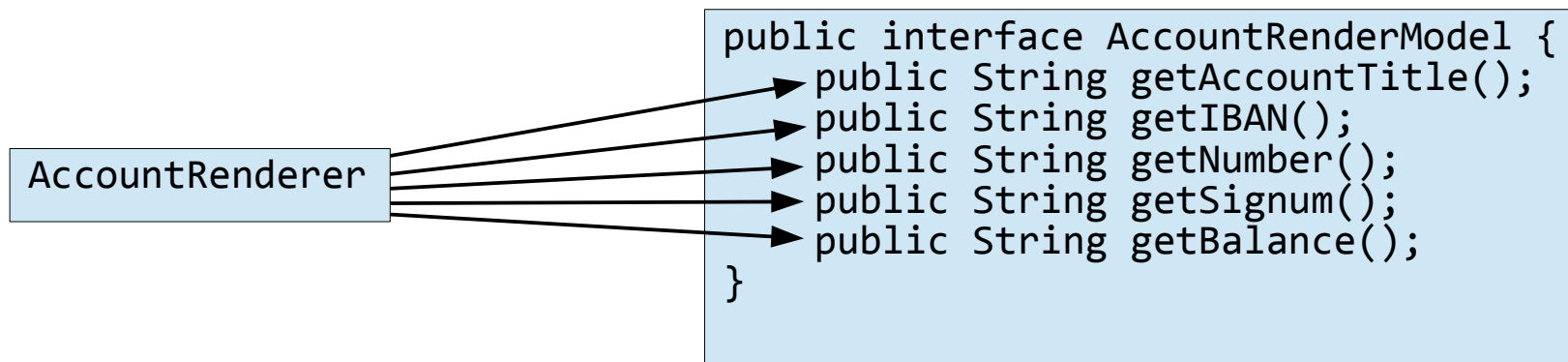
- Von innen nach außen gibt es zwei grundsätzliche Möglichkeiten
- Außen muss anfragen
 - Als Rückgabewert eines Methodenaufrufs von außen
- Innen meldet sich von alleine
 - Rückruf (Callback) einer vorher registrierten äußeren Methode
 - Aufrufpfeil verläuft dann entgegengesetzt zum Abhängigkeitspfeil





Übergabe von Daten: Außen fragt

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



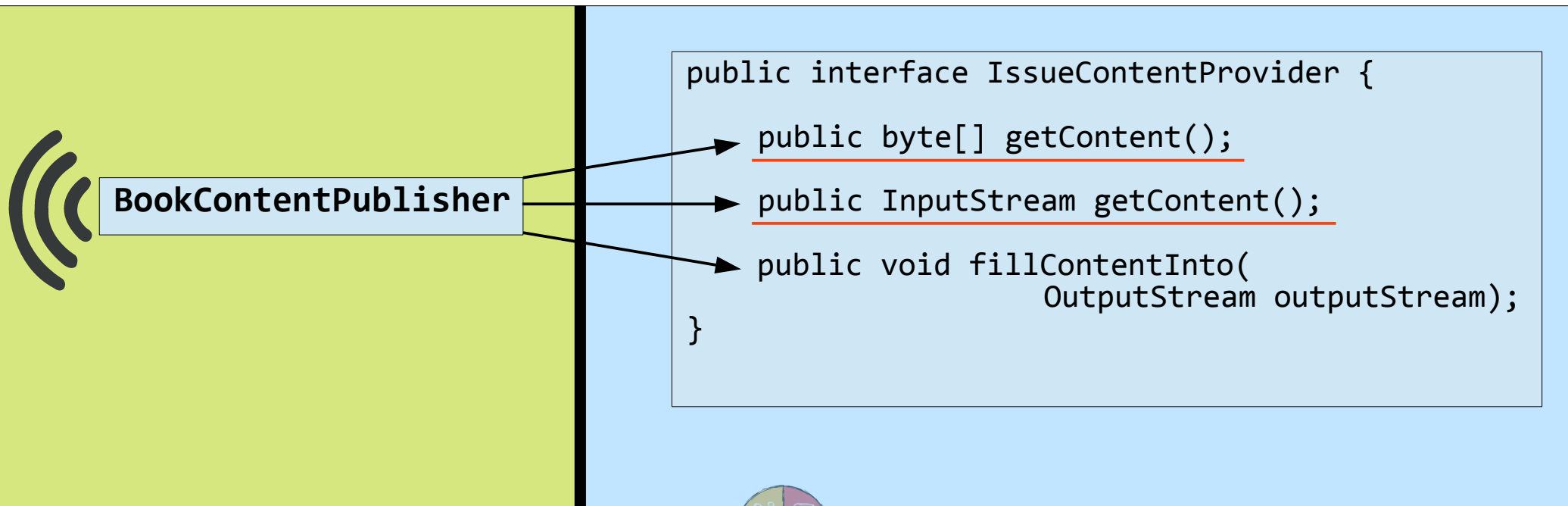
`AccountRenderModel`
(`Map<String, String>`)

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



Übergabe von Daten: Außen fragt

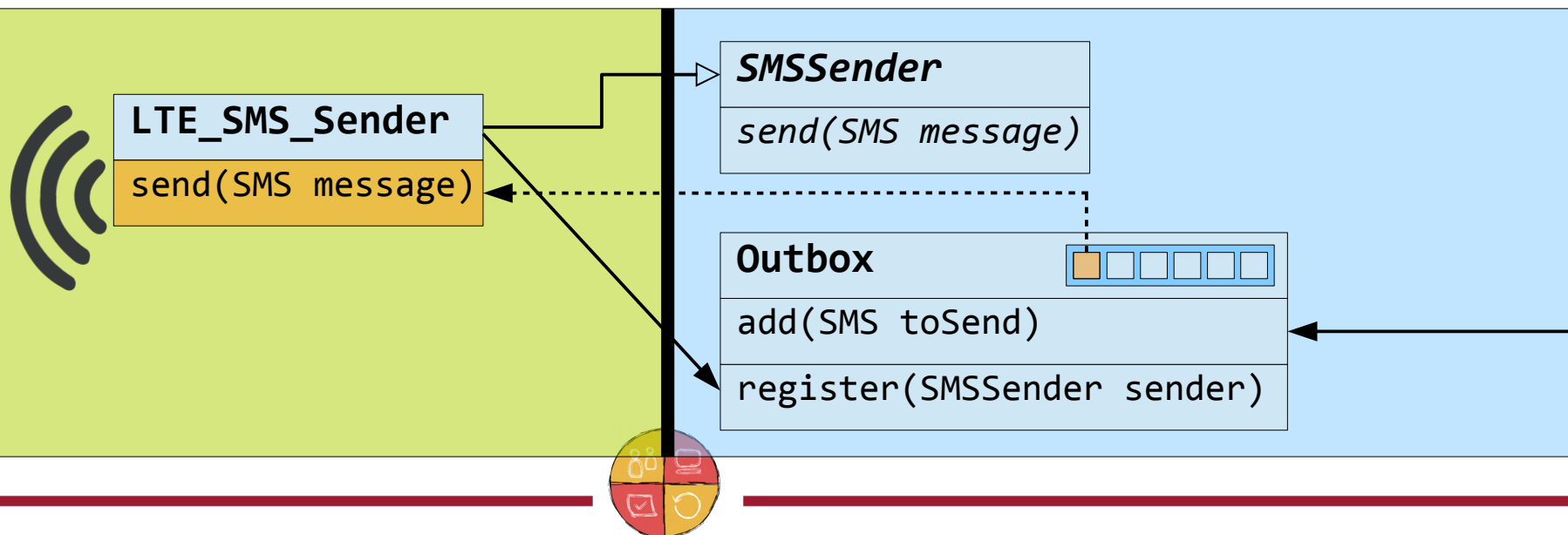
- Variante: Die Datenverarbeitungsrichtung umdrehen
 - OutputStream hineingeben statt InputStream erfragen
 - Consumer<T> bereitstellen statt auf T zu warten





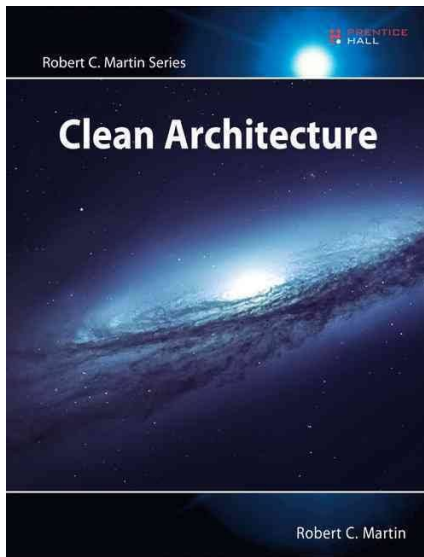
Übergabe von Daten: Innen meldet sich

- 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

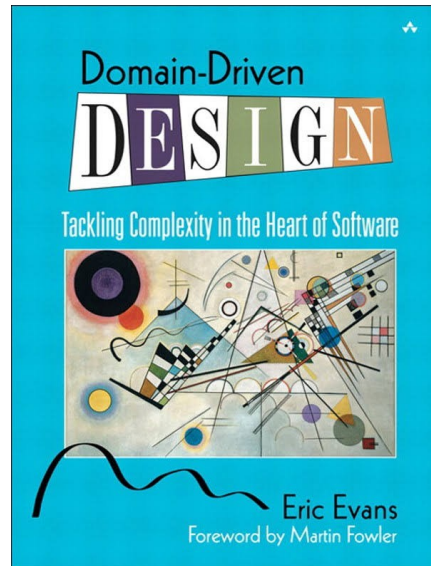




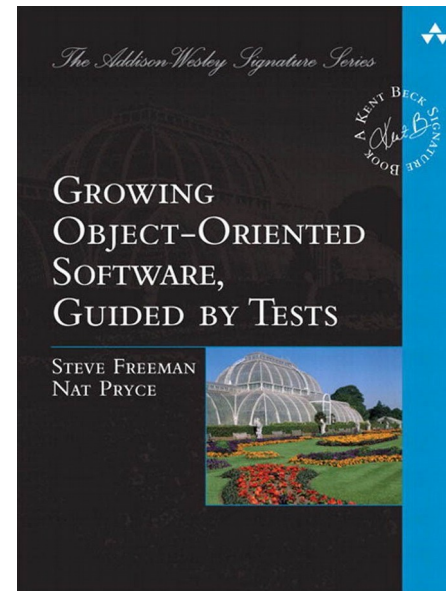
Clean Architecture: Weiterführende Literatur



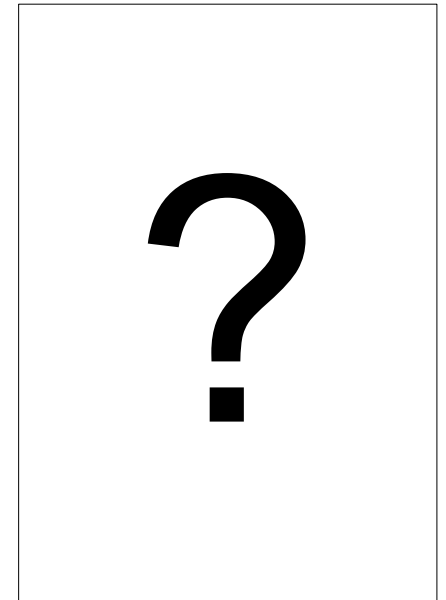
2017



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>



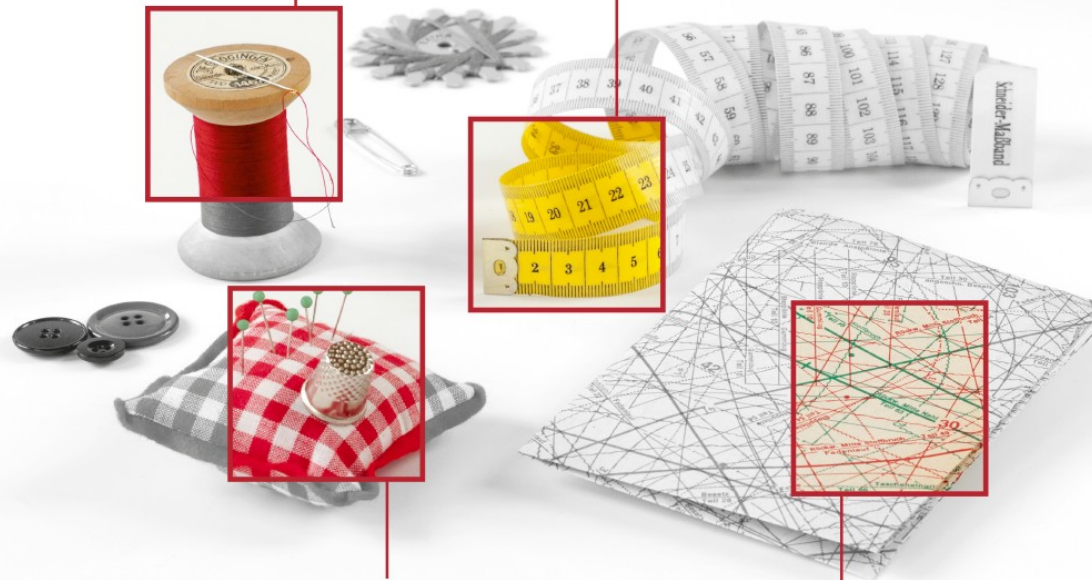


Softwareschneiderei

Softwareschneiderei GmbH

Softwareentwicklung

User Experience (UX)



Fortbildung

Strategische IT-Beratung





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>

