3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Schlankheitskur

Lean Web Architecture mit JSF 2.0, CDI & Co.

Andreas Hartmann

adesso AG

adesso business. people. technology.

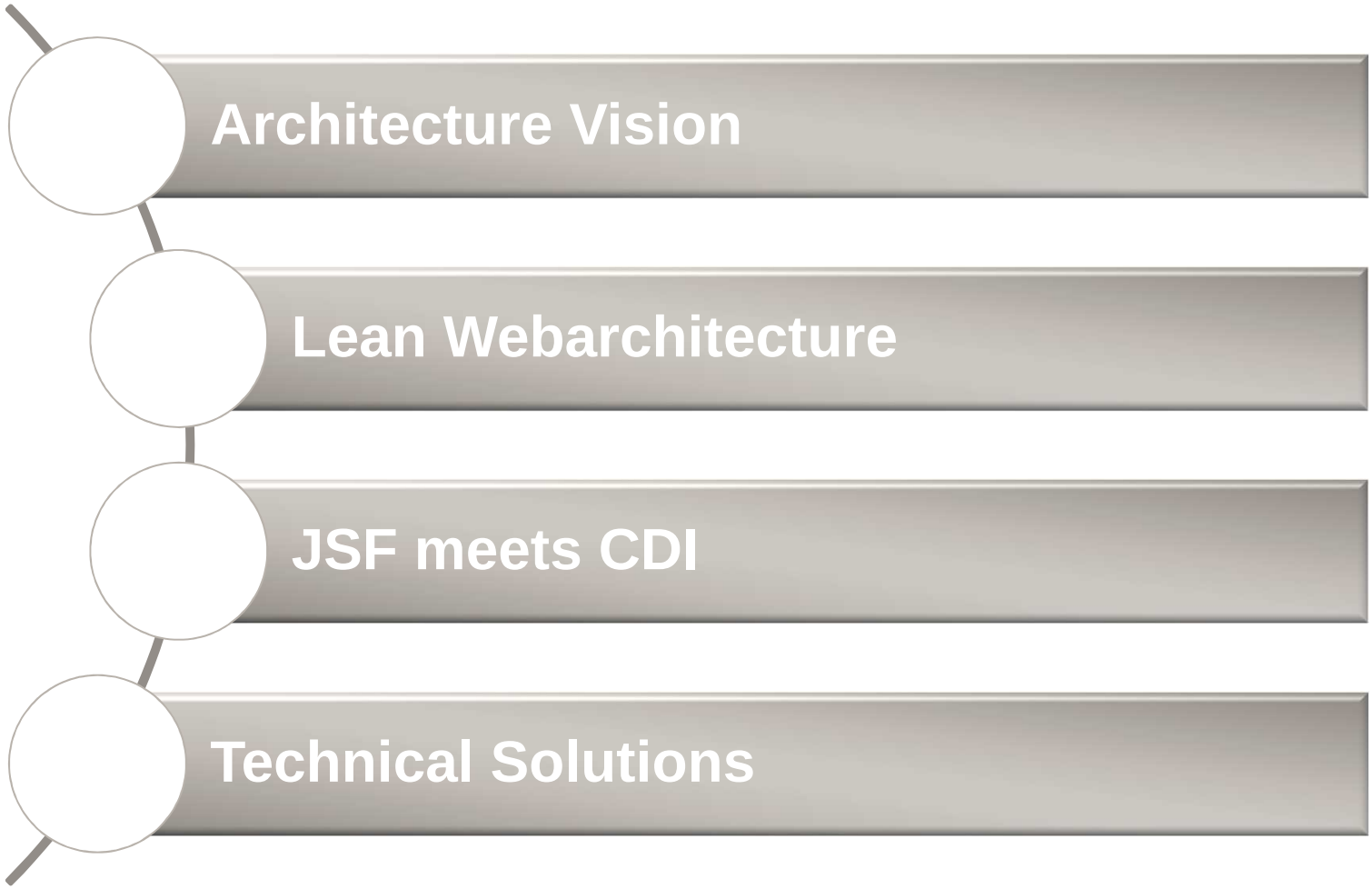
Schlankheitskur

Lean Web Architecture mit JSF 2.0, CDI & Co.

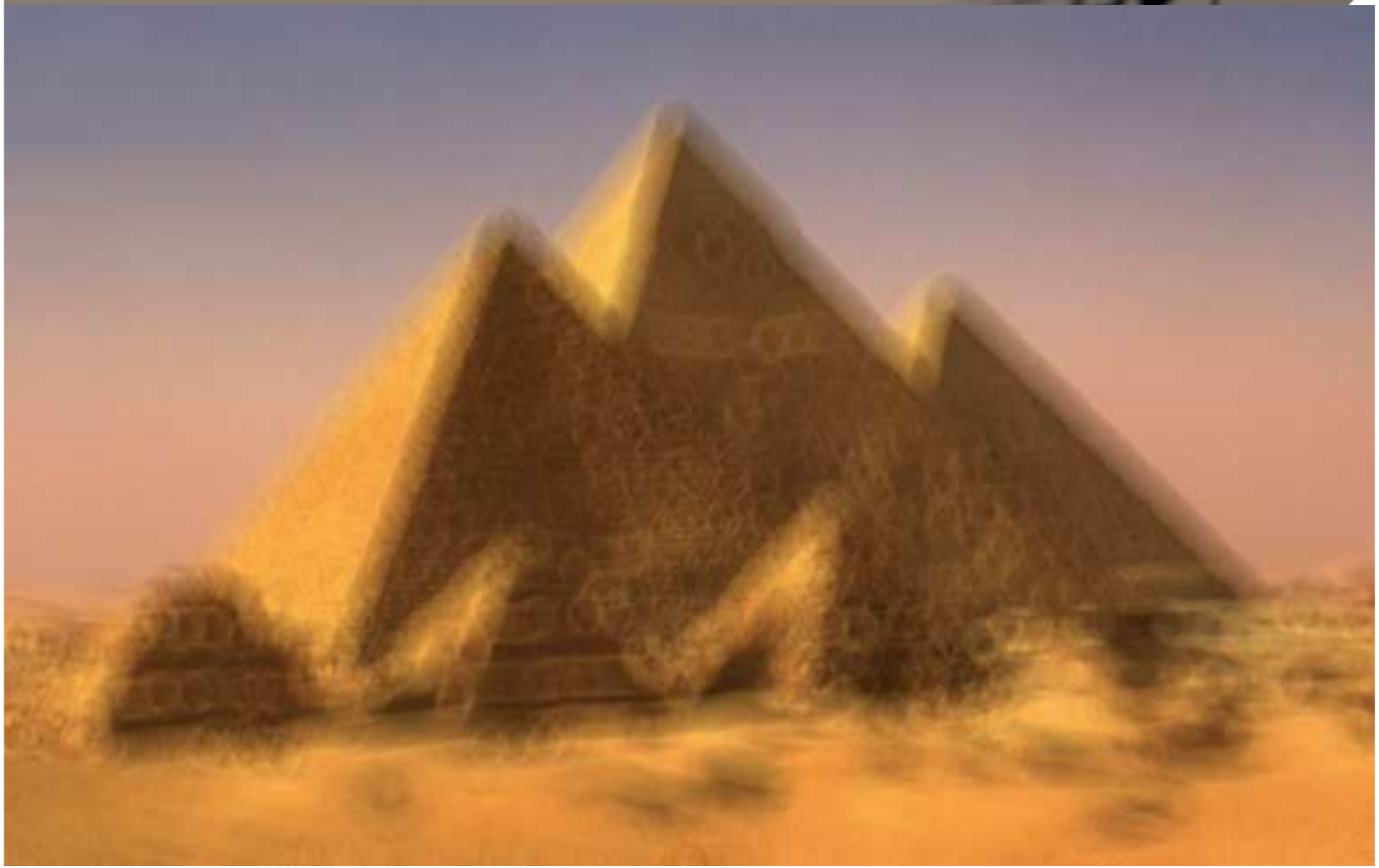
Andreas Hartmann

05.09.2012

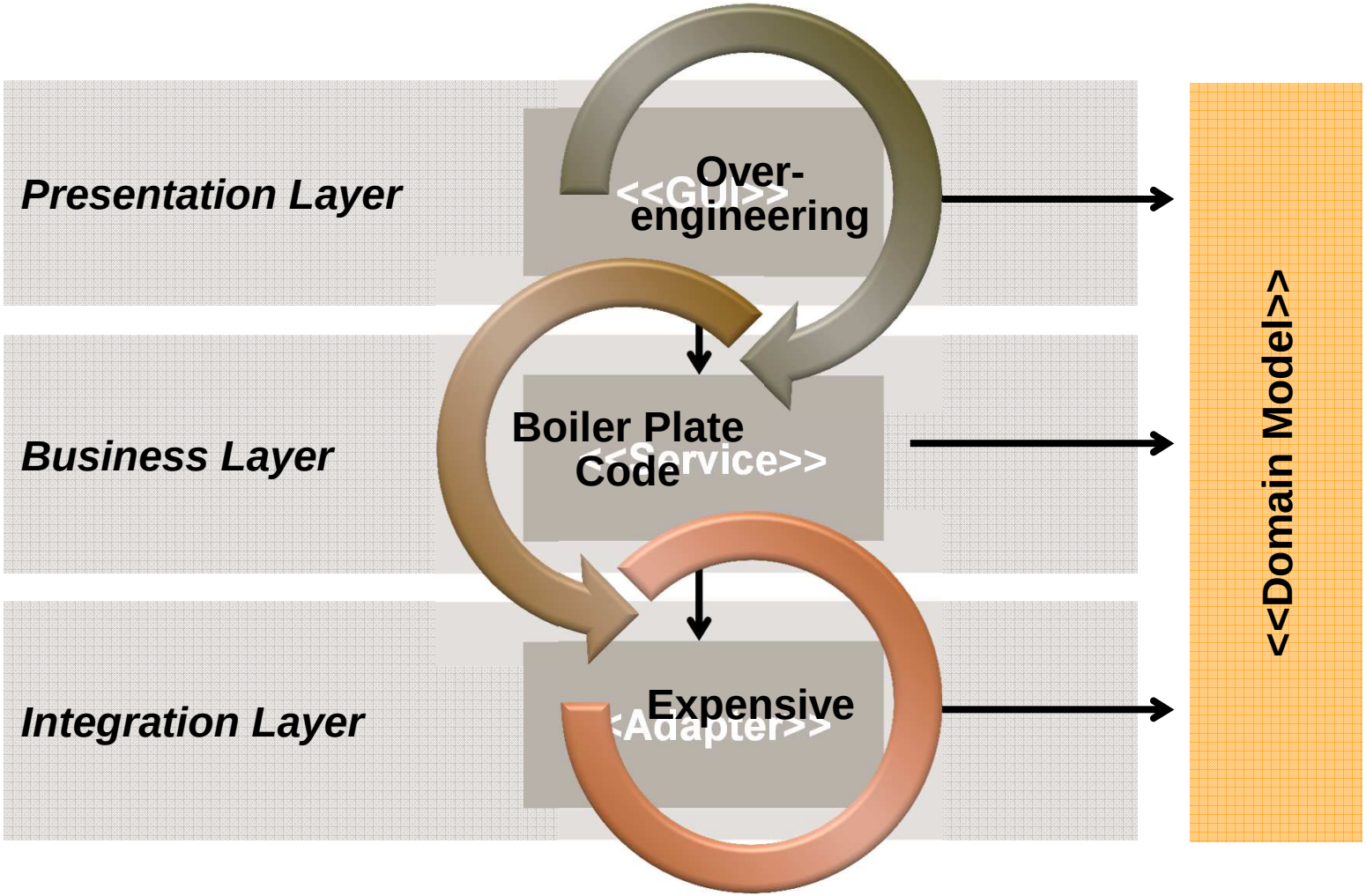
Agenda


Architecture Vision
Lean Webarchitecture
JSF meets CDI
Technical Solutions

Architecture Vision

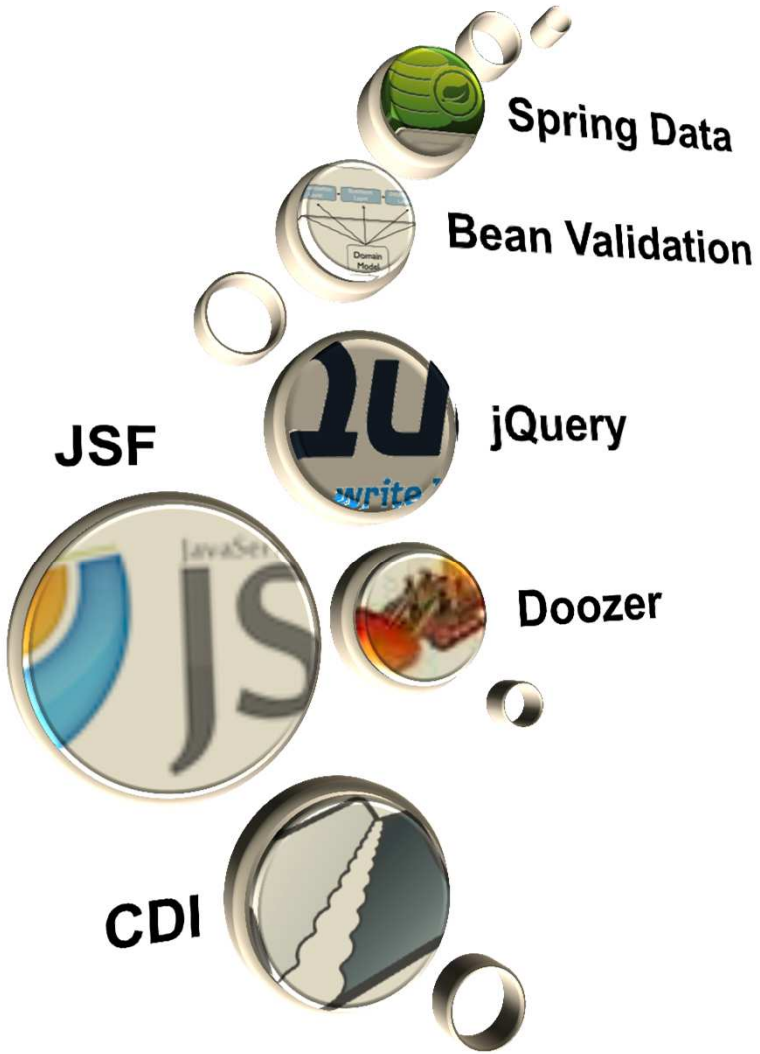


Architecture Vision – 3 Layer Reflex

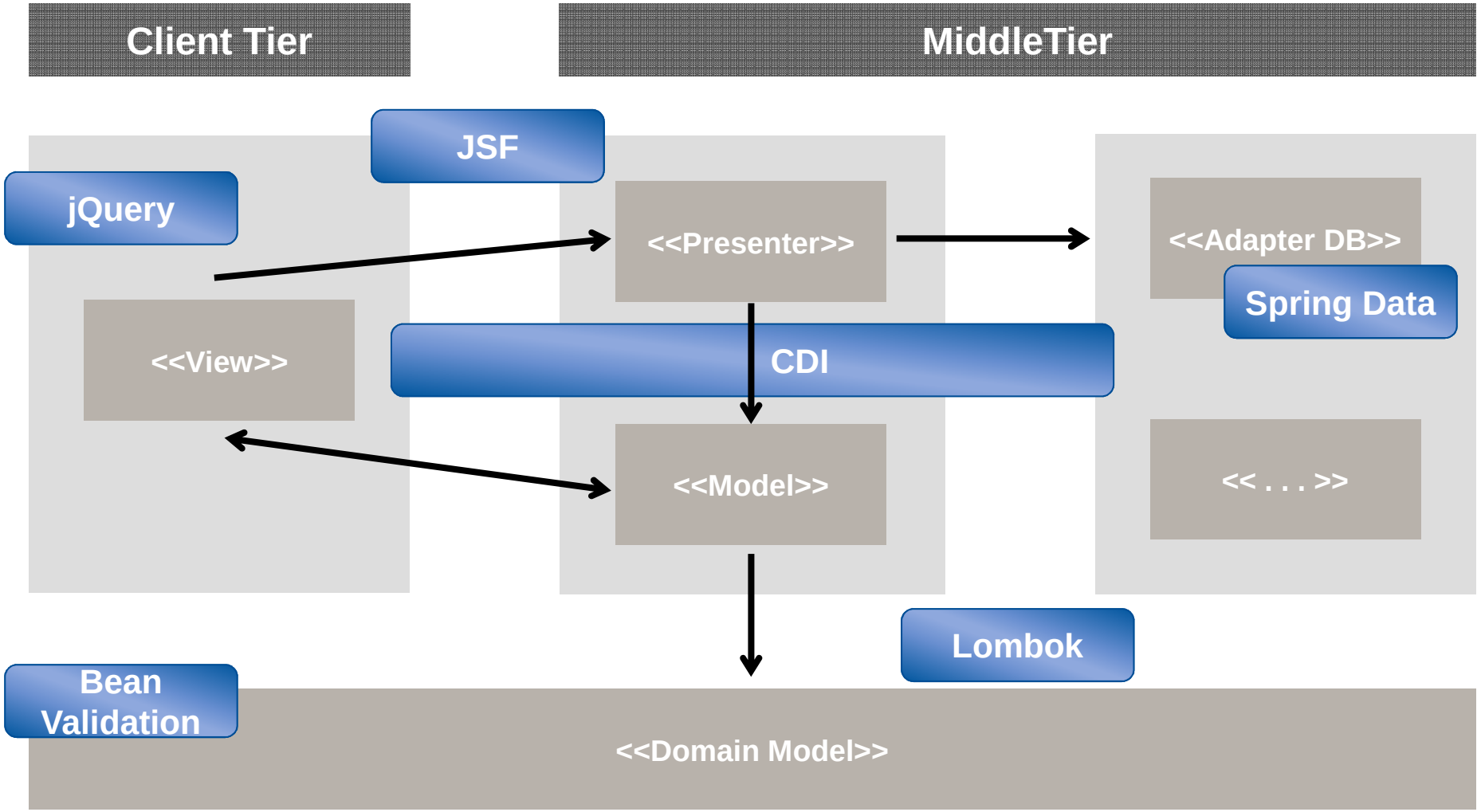


Architecture Vision – Cross-Sectional Tasks

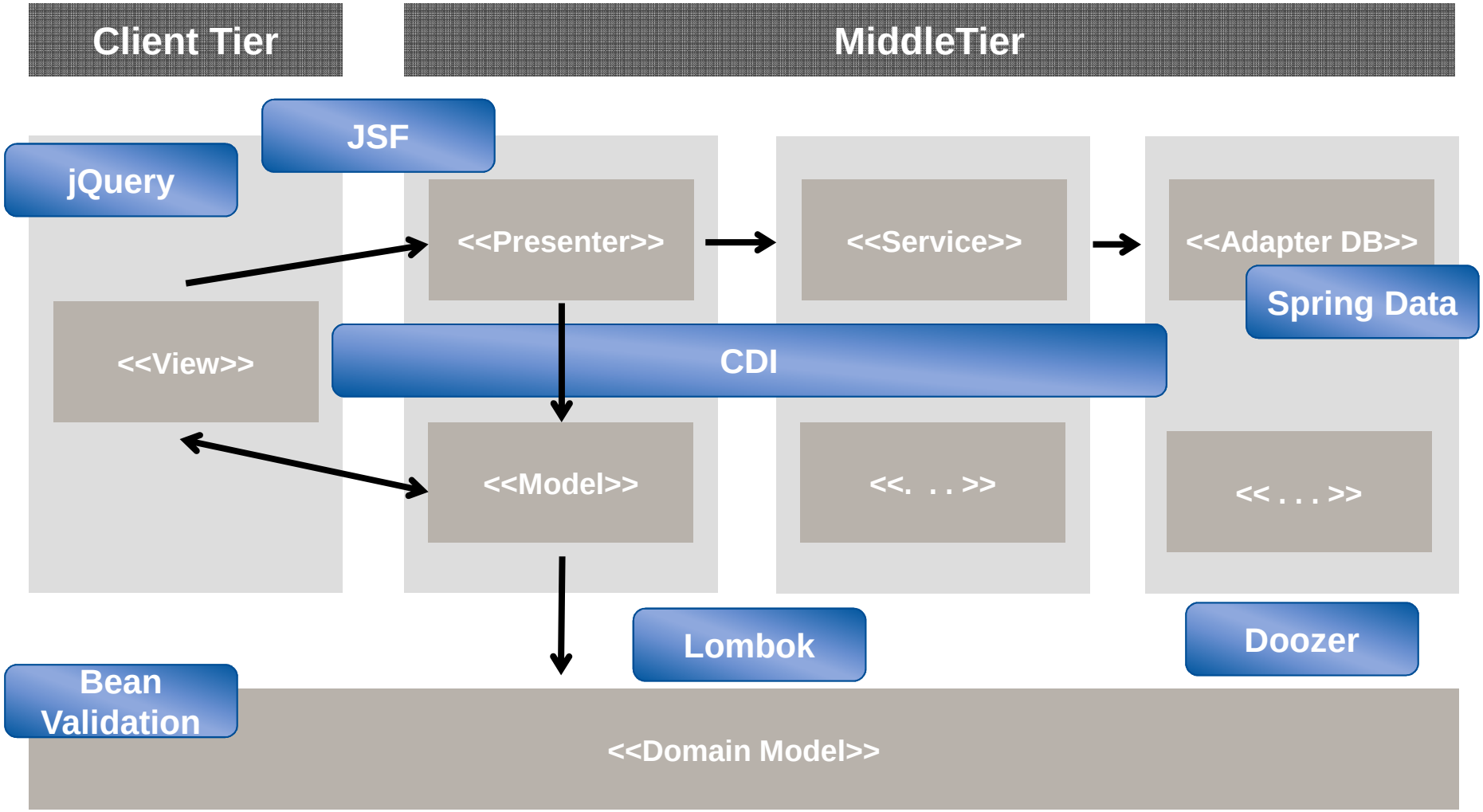
Architecture Vision – Technology Stack



Lean Webarchitecture – Tiny Applications



Lean Webarchitecture – Large Scale Applications



adesso

Warum CDI?

Dependency Injection

- Container steuert Objekte und Abhängigkeiten
  - Instanziiert Objekte
  - Injiziert Abhängigkeiten zur Laufzeit (lose Kopplung)
- Konfiguration über Annotations oder XML-Datei
- Populär geworden durch das Spring Framework
- Seit EJB3.0 auch in Java EE verfügbar
  - @EJB
  - @PersistenceUnit
  - @Resource

Warum CDI?

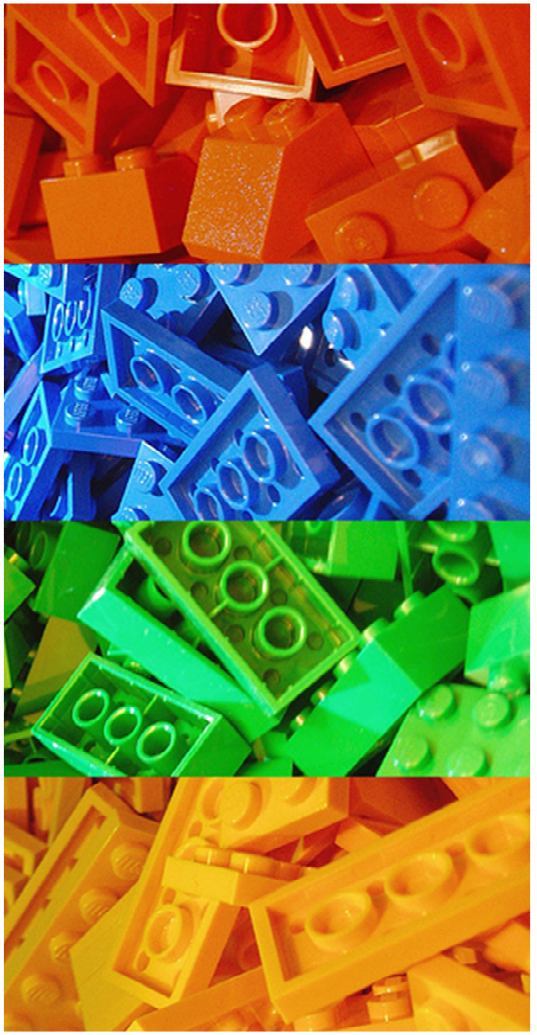
Dependency Injection mit CDI

- Standardisiertes DI für Java (SE/EE)
  - Typsicherheit
  - Wiederverwendbarkeit
  - Testbarkeit
  - Flexiblität
  - Lifecycle „Awareness“

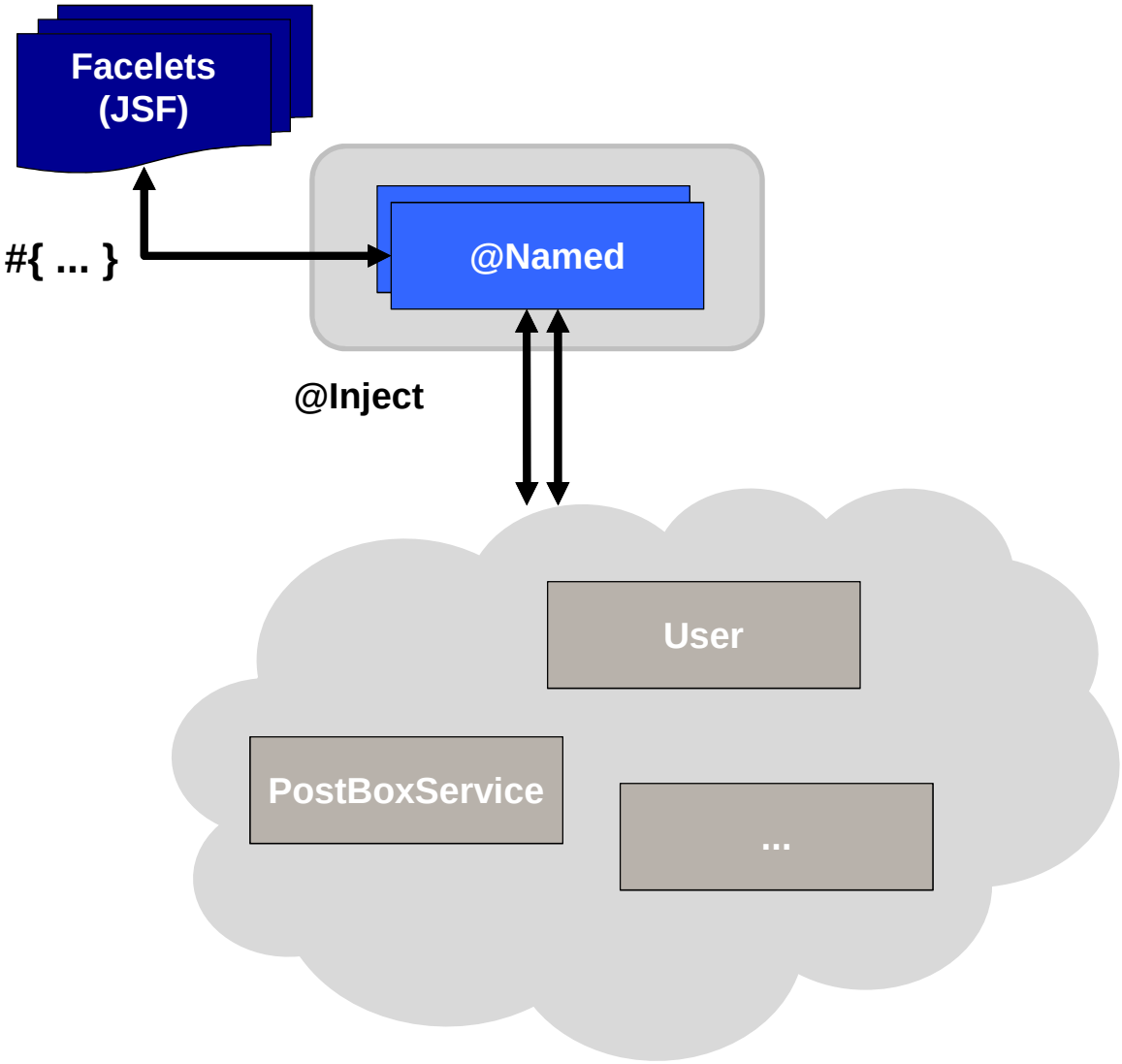
- User Story driven injizieren

CDI Bausteine

- Typsicherheit
  - Stereotypes
  - Qualifier
  - Alternatives
- Lose Kopplung
  - Interceptors
  - Decorator
  - Events
- Sichtbarkeiten
  - Scopes
  - Lifecycle Awareness



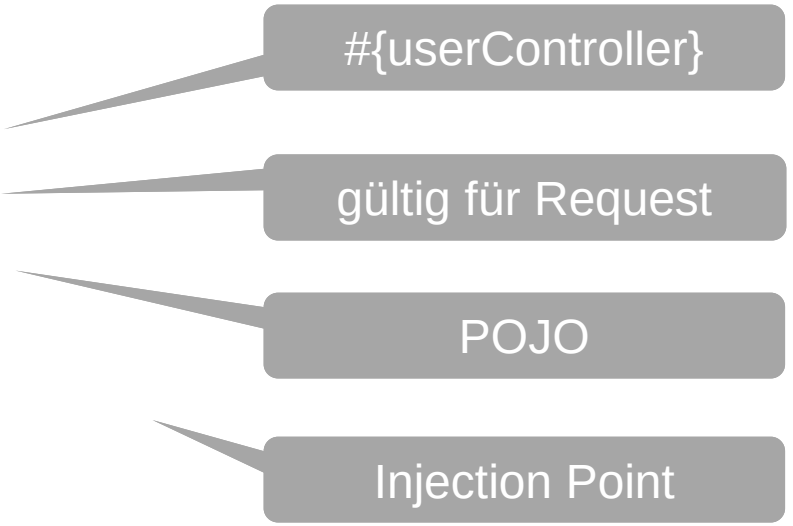
CDI im Detail



CDI im Detail

CDI Managed Bean

```
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;
import javax.inject.Inject;

@Named("userController")
@RequestScoped
public class UserController {

  @Inject
  private UserService userService;

  private List<User> users;

  @PostConstruct
  private void init() {
    this.users = userService.findAllUsers();
  }
```



CDI Injection Points

- Field Injection

```
@Inject
private UserService userService;
```

- Constructor Injection

```
@Inject
public UserController(UserService userService) {
    this.userService = userService;
}
```

- Setter Injection

```
@Inject
public void setUserService(UserService userService) {
    this.userService = userService;
}
```

User Story driven injizieren

Qualifier & Producer

- Aktuellen Benutzer injizieren

```
@Inject @Current private User user;
```

- Fachlichkeit injizieren

```
@Inject @Admin  private List<User> administrators;
@Inject @Editor private List<User> editors;
```

- Infrastruktur injizieren

```
@Inject @UserDB    private EntityManager userDB;
@Inject @ProductDB private EntityManager productDB;
```

User Story driven injizieren

Qualifier & Producer

```
import de.adesso.cdi.common.Current;

import javax.enterprise.inject.Produces;

@Named
@SessionScoped
public class Authenticationimplements Serializable {

  private User authenticatedUser;

  public String authenticate() {...}

  @Produces
  @Named("authenticatedUser")
  @Current
  public User getAuthenticatedUser() {
    return authenticatedUser;
  }
```

EL: #{authenticatedUser}

Fachlichkeit statt Infrastruktur injizieren

Qualifier & Producer

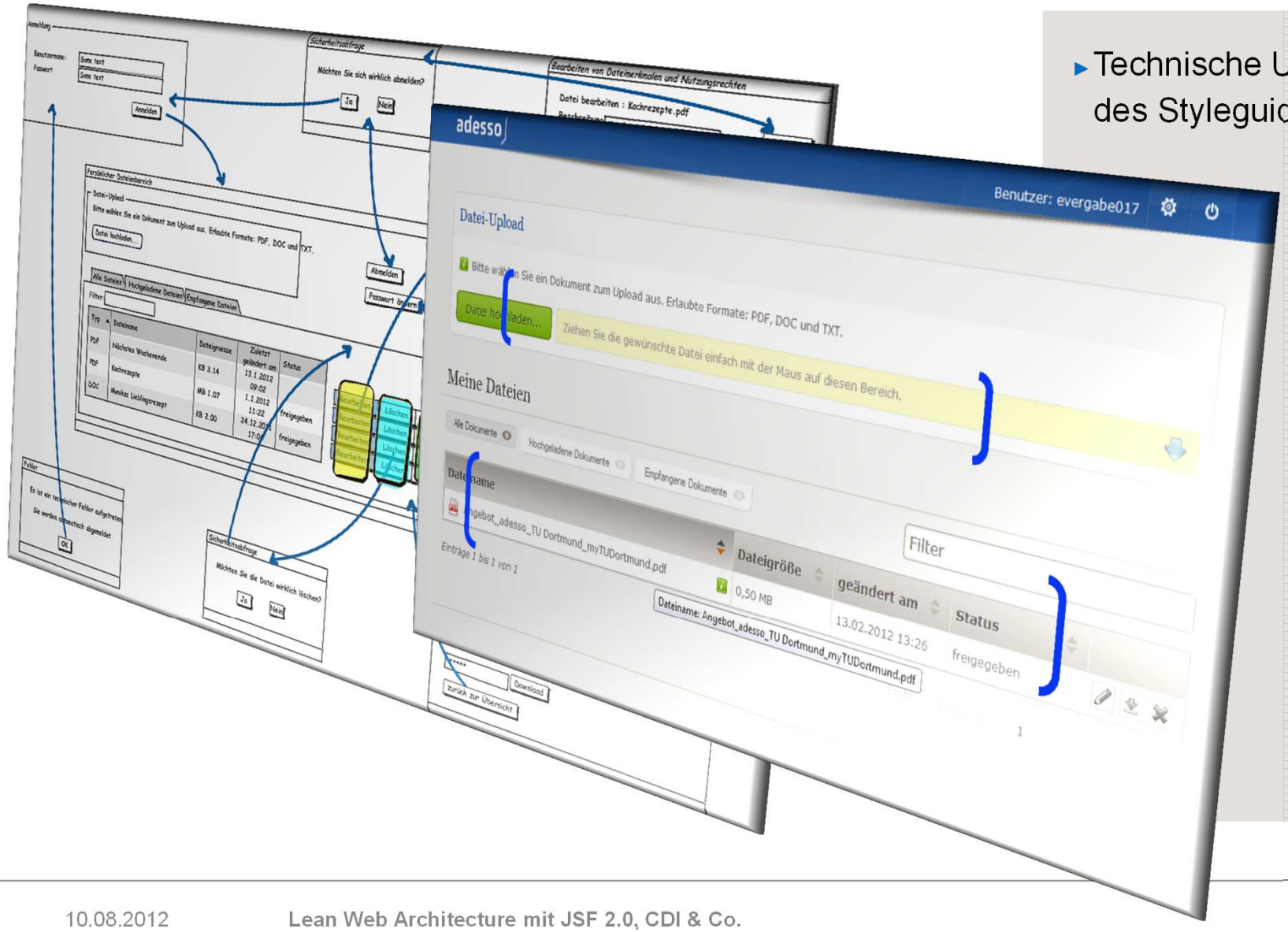
```
import de.adesso.cdi.common.Current;

public class CustomerService implements CustomerService {


  @Inject @Current
  private User currentUser;

  public void addCustomer(Customer customer) {
    customer.setCreator(currentUser);
    em.persist(customer);
  }
```

HTML 5

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

<h:head>
    <title><ui:insert name="pageTitle"></ui:insert></title>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />

    <h:outputScript name="js/jquery.js" />
    <h:outputScript name="js/jquery-ui.min.js" />
    <h:outputScript name="js/jquery.ui.datepicker-de.js" />
    <h:outputScript name="js/fileuploader.js" />
    <h:outputScript name="js/ui.multiselect.js" />
    <h:outputScript name="js/jquery.dataTables.min.js" />

    <h:outputStylesheet name="css/reset.css" />
    <h:outputStylesheet name="css/jquery-ui.css" />
    <h:outputStylesheet name="css/ui.multiselect.css" />
    <h:outputStylesheet name="css/style.css" />
</h:head>
```

- Mit JSF2.2 auch HTML5 Komponenten

Custom Components



- Technische Umsetzung des Styleguides

jQuery

- freie, umfangreiche JavaScript Bibliothek
- komfortable Möglichkeit zur DOM Manipulation
- Erweiterbarkeit durch Plugins (u.a. DataTable)

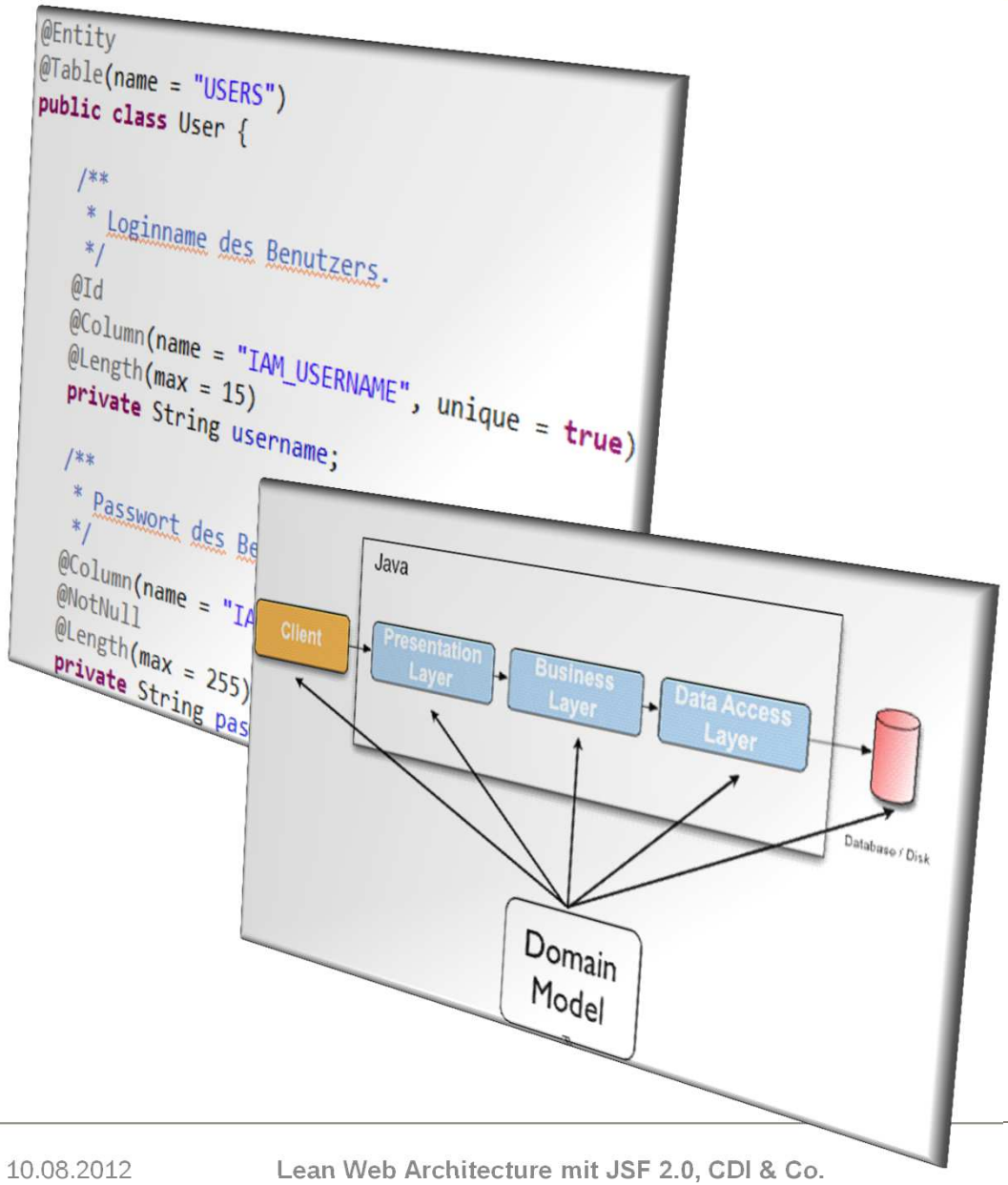
jQuery
write less, do more.

Lombok

```
public class Email implements Serializable {

    @Getter @Setter
    private Object id;

    @NotNull
    @Getter @Setter
    private Date incomingDate;

    @NotNull
    @Getter @Setter
    private String subject;

    @Getter @Setter
    private String text;
}
```

- Reduziert Boilerplate Code durch Annoations
  - > @Getter, @Setter
  - > @ToString
  - > @HashCodeAndEquals
  - > @Data
  - > . . .
- Delombok Funktionalität
- Eclipse Support

JSR 303: Bean Validation



- Standardisierung von Constraints in Form von Annotations
- Standardisierung des Metadata API
- Integration anderer Frameworks wie z.B. JSF und JPA

Doozer

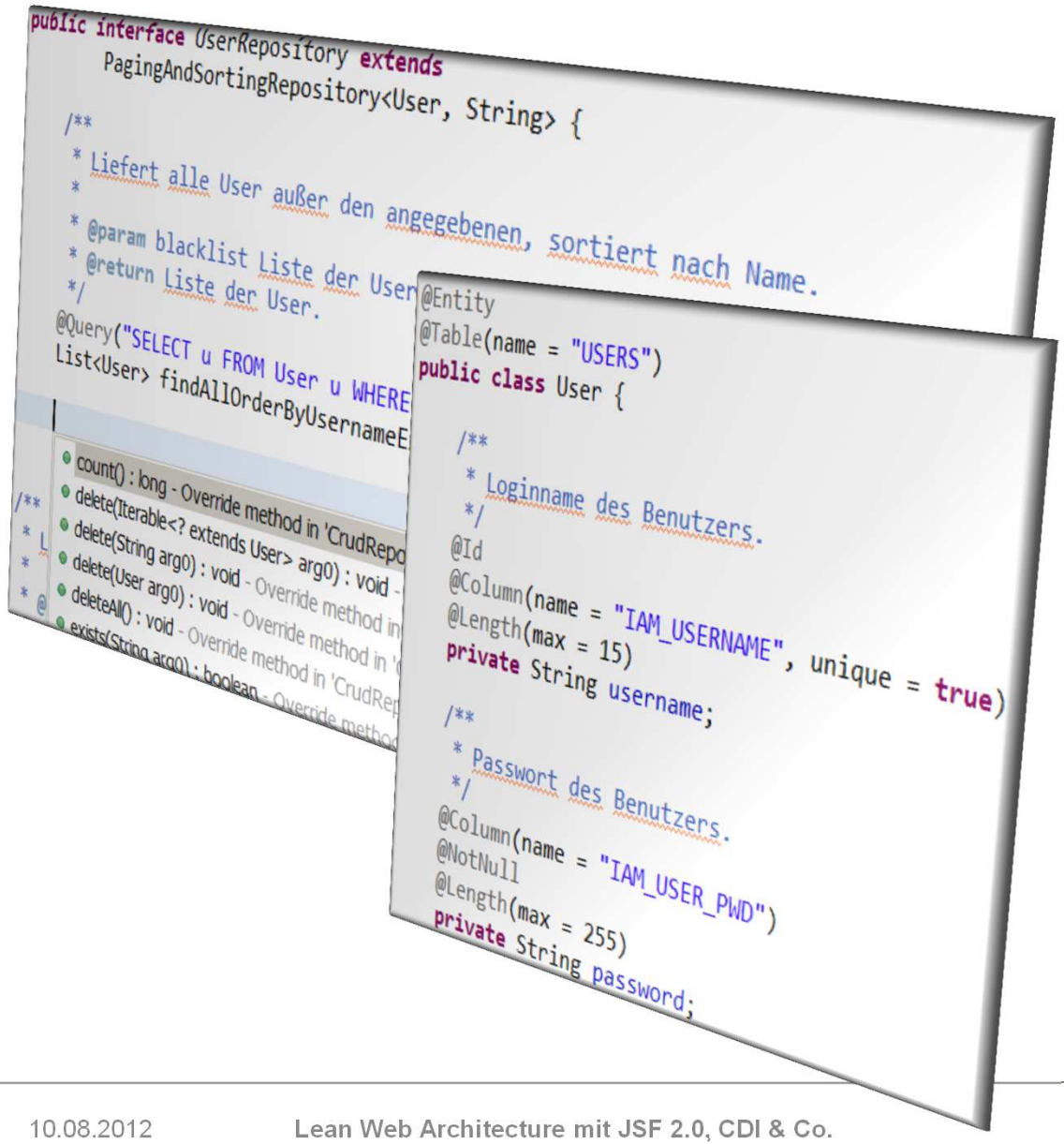
```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanm

 <mapping>
  <class-a>de.adesso.mvc4jsf.showcase.domain.Emai
  <class-b>de.adesso.mvc4jsf.showcase.adapter.enti
  <field custom-converter="de.adesso.mvc4jsf.sho
    <a>id</a>
    <b>id</b>
  </field>
  <field>
    <a>incomingDate</a>
    <b>incomingDate</b>
  </field>
  <field>
    <a>subject</a>
    <b>subject</b>
  </field>
  <field>
    <a>text</a>
```

```
public class Email impleme

    @Getter @Setter
    private Object id;

    @NotNull
    @Getter @Setter
    private Date incomingDa

    @NotNull
    @Getter @Setter
    private String subject;

    @Getter @Setter
    private String text;
}
```

```
@ApplicationScoped
@Named(value="userDAO")
public class UserDAO implements Serializable {

    @Inject DomainModelMapper modelMapper;

    @Inject DataSource dataSource;

    public boolean deleteEmail(Email email) {
        EmailEntity emailEntity = modelMapper.mapToMorphia(email);
        dataSource.getDataStore().delete(emailEntity);
        return true;
    }

    public Email saveEmail(Email email) {
        EmailEntity emailEntity = modelMapper.mapToMorphia(email);
        dataSource.getDataStore().save(emailEntity);
        return modelMapper.mapToDomainModel(emailEntity);
    }
```

- Java Bean to Java Bean Mapper

SpringData



- ermöglicht Zugriff auf relationale und NoSQL Datenbanken
- CDI Integration

Vielen Dank!

Andreas Hartmann

adesso AG