Einblicke in die Praxis –

Java EE Projekt einer Versicherung

Vorlesung P2, Universität Bern, 1. Juni 2012

Frank Buchli
Business Unit Manager
frank.buchli@zuehlke.com

Zu meiner Person

zühlke
empowering ideas

Frank Buchli

- MS in Computer Science, Uni Bern 2003
- Master Thesis: „Detecting Software Patterns using Formal Concept Analysis”
- 3 Jahre IT Projektleitung: AIESEC Careerdays
- 3 Jahre Software Entwicklung & Business Development im Bereich Handy (J2ME) und J2EE bei Glue Software Engineering AG
- Seit 2007 Software Engineer / Software Architect bei Zühlke
- Seit 2011 Leitung einer Business Unit bei Zühlke

Agenda

- **Zühlke?**
- **Software Entwicklung**
  - Vorstellung Projekt
  - Architektur, Technologie
  - Team, Rollen
  - Release Management
  - Testing
  - Design Patterns (inkl. Java Generics)
- **Fragen, Diskussion**


Zühlke. Empowering Ideas.
zühlke
empowering ideas

Drei vernetzte Bereiche

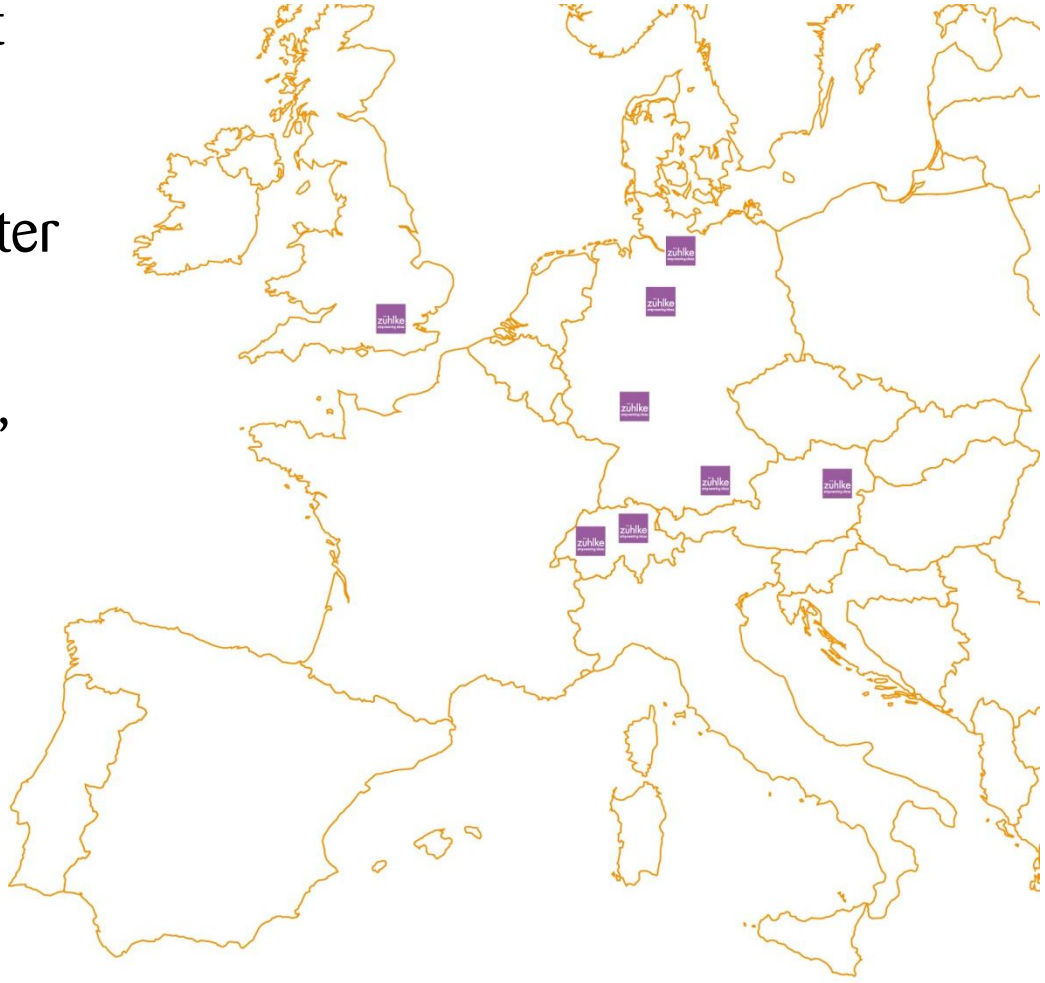
Für mehr Wachstum, Innovation und Produktivität

- Produkt- und Software-Engineering
- Managementberatung
- Startup-Finanzierung

Facts & Figures

zühlke
empowering ideas

- Mehr als 7000 Projekte realisiert
- 83 Mio. CHF Umsatz (2011)
- 500 Mitarbeiterinnen & Mitarbeiter (Ende 2011)
- In Deutschland, Grossbritannien, Österreich und in der Schweiz
  - Bern: Aarbergergasse 29
  - Schlieren bei Zürich, Hauptsitz
- Gegründet 1968, im Besitz von Partnern



GREAT PLACE TO WORK Best companies to work for 2011 Switzerland

Produktentwicklung

zühlke
empowering ideas

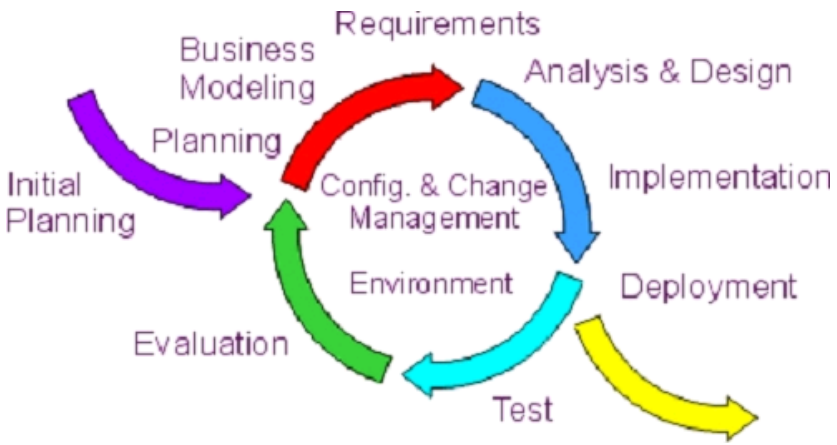
Beispiele Produktentwicklung

- Steuerungselektronik
- Firmware
- Realisierung vom Konzept bis zur Nullserie
- Verantwortung für die komplette Produktentwicklung

zühlke
empowering ideas

Software Entwicklung

Projektbeispiel Versicherungswesen



Frank Buchli

Das Projekt

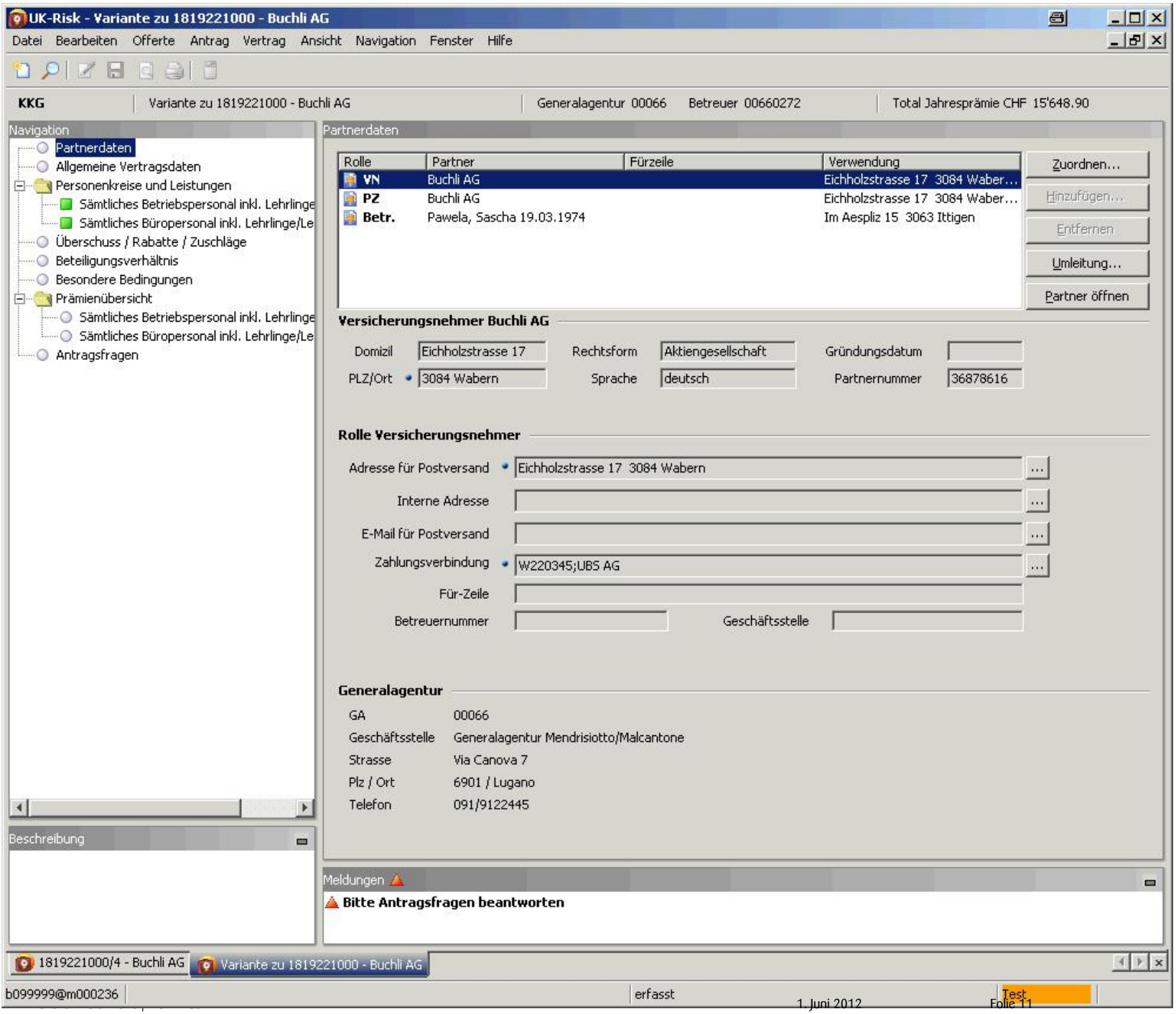
zühlke
empowering ideas

- Java EE Lösung für die Geschäftsabwicklung einer Versicherung
- Bereich Unfall/Kranken Versicherungen
- Ersetzt 10 – 15 jährige Host / Mainframe Lösung
- Realisierung durch eigene IT Abteilung
- Dauer: 2005 – 2011
- Involvierte Leute
- Kosten
- Unterstützung durch externe Spezialisten
- Meine Rolle: Chef Entwicklung TP Schaden

Abwicklung von Unfall/Kranken Versicherungen

zühlke
empowering ideas

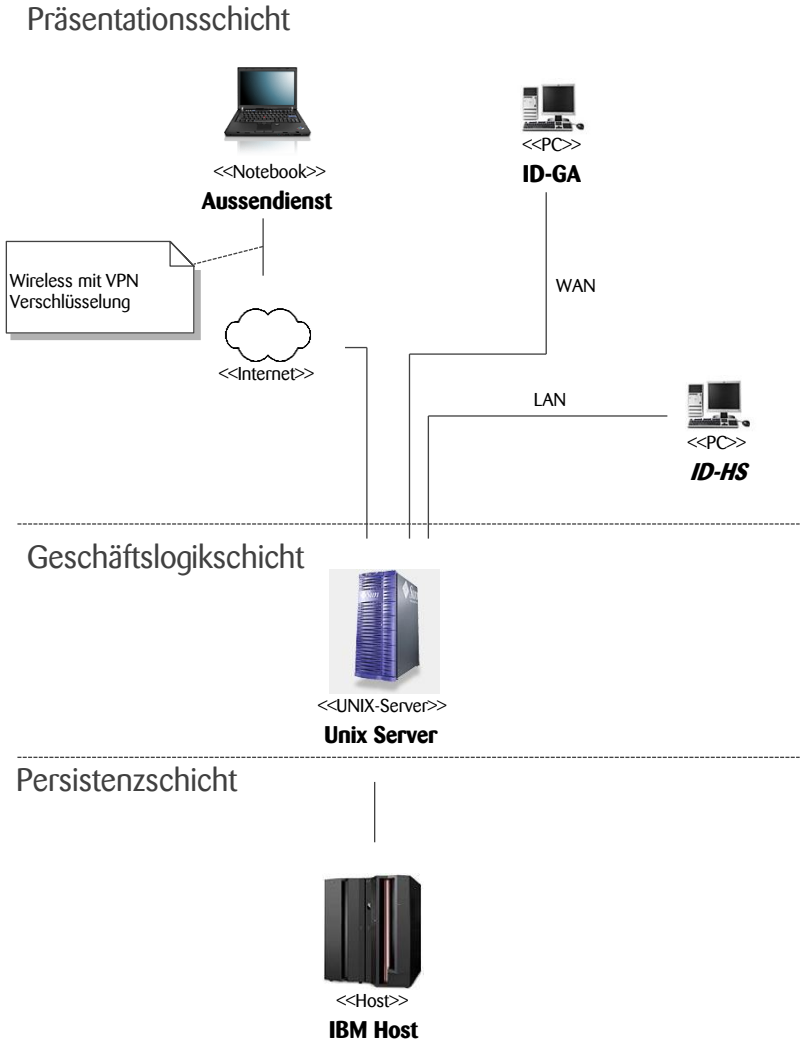
- Erstellen von Offerten
  - Darf offeriert werden?
  - Wie hoch ist die Prämie?
  - Geschäftsmodelle, Versicherungsmodelle
- Erfassen von Verträgen
  - Bsp: Besondere Bedingungen
  - Archivierung
  - Einfordern der Prämien
- Erfassen von „Schadensmeldungen“
  - Ist der Schaden gedeckt?
- Auszahlen von Leistungen
  - An wen geht das Geld?



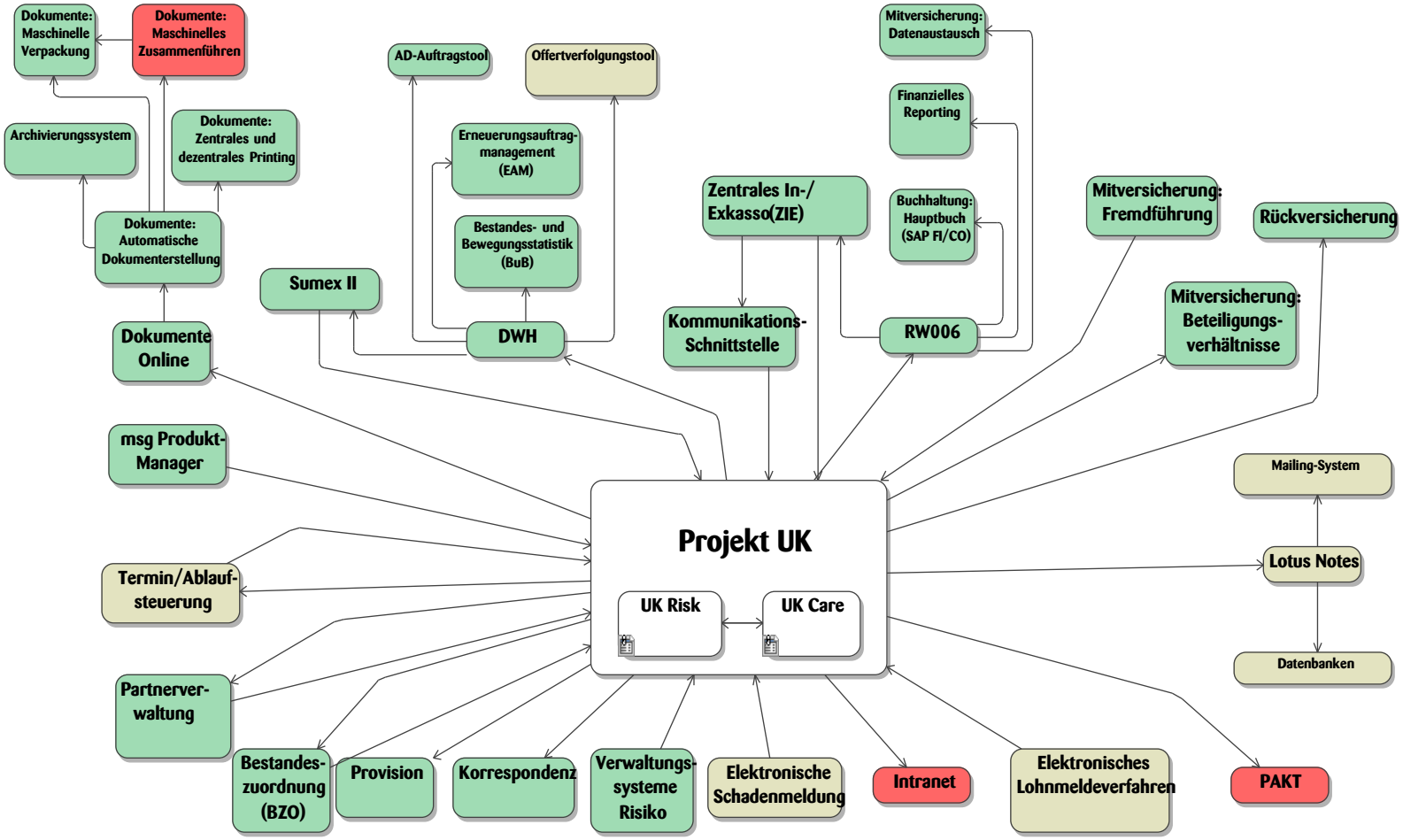
Screenshot

Architektur: Verteilungssicht

zühlke
empowering ideas



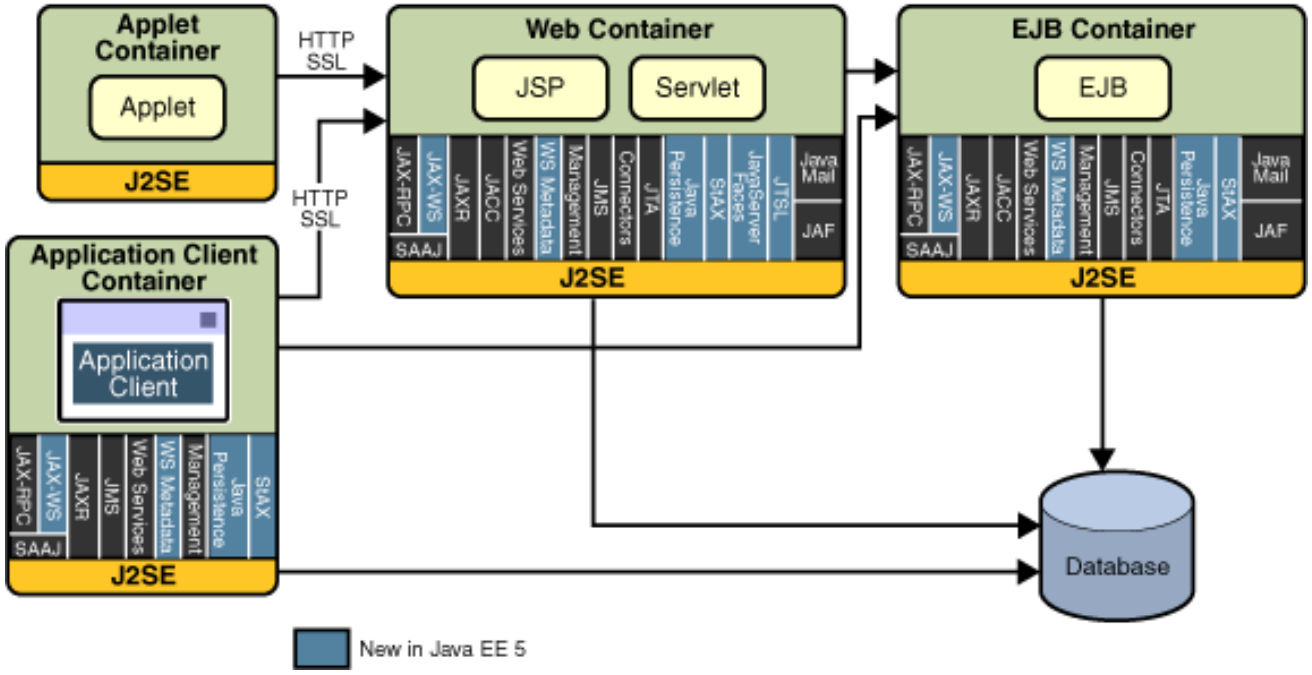
Umsysteme



Verwendete Technologien
Java Versionen

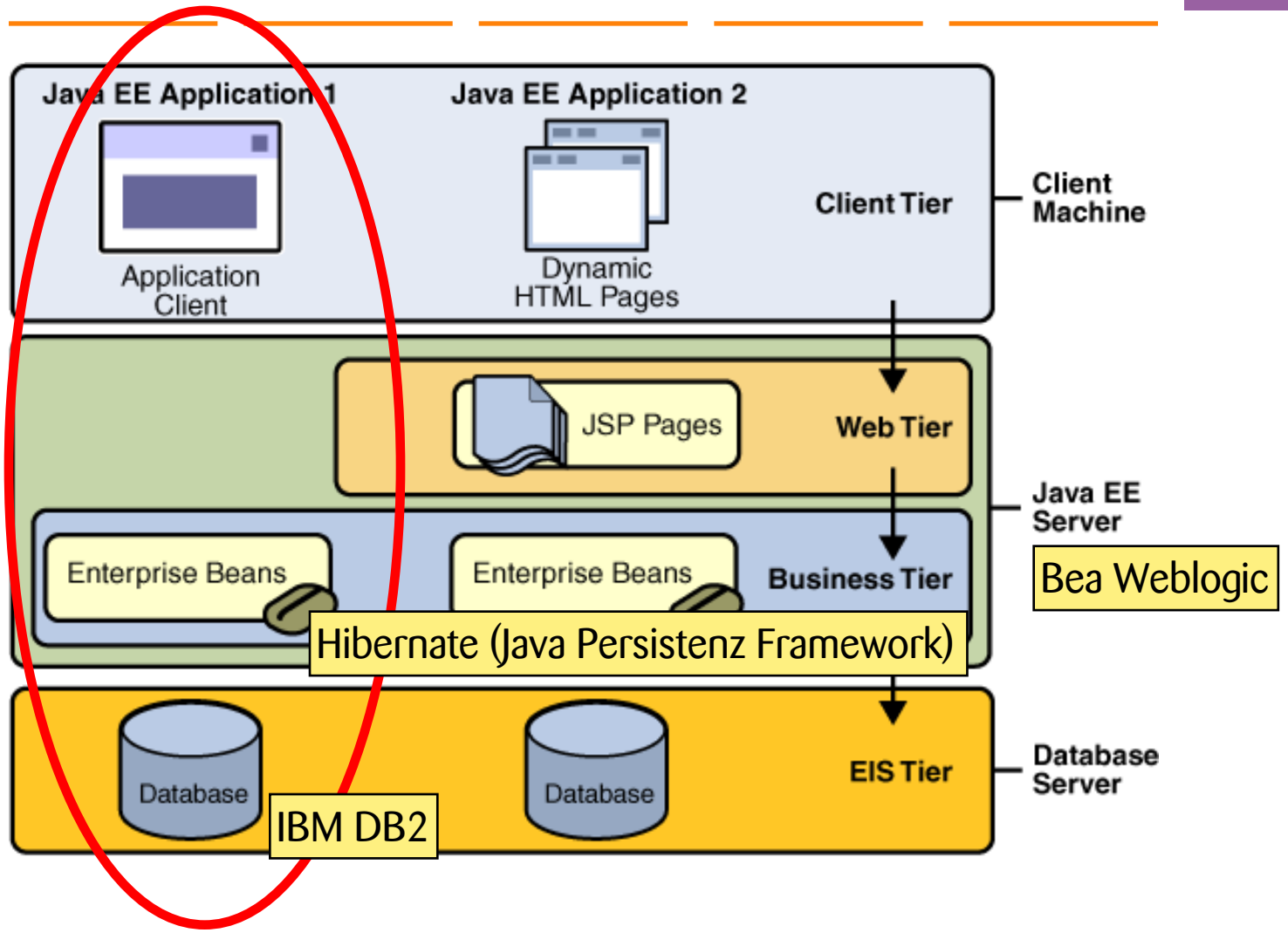
zühlke
empowering ideas

- Java SE 5
  - Generics & Annotation
- J2EE 1.4
  - EJB 2.0



Verwendete Technologien
Java EE

zühlke
empowering ideas



Java

Entwicklungsumgebung

- IDE: Eclipse
  - Diverse Plugins: MyEclipse, Format-On-Save
  - JFormDesigner
- Source Repositiory: CVS
- Build: Maven 2, Continuum
- MDSD: AndroMDA
- Design:
  - Magic Draw (UML)
  - Power Designer (DB)
- Requirement Verwaltung & Testen: HP Quality Center
- Viele eigene Frameworks, Tools

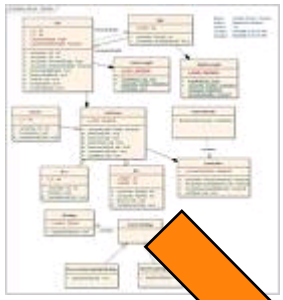
Entwicklungsvorgehen

zühlke
empowering ideas

- RUP Nahe
- Fachbereich von Anfang an integriert
- Early Prototyping: GUI Framework auf xml Basis
- Fach-Test Team
- Test-Konzept
- Konsolidierungsphasen
  - Refactoring
  - Aufarbeiten von Fehler-Tickets

Entwicklungsvorgehen
Model Driven Development (MDD)

zühlke
empowering ideas



Template 2
pu
…
}

Template 1
```
public class XXX {
…
}
```

MDSD Engine

SQL Code
cr
…
)

SQL Code
```
create table T_Vertrag (
…
)
```

Source Code
pu
…
}

Source Code
```
public class Vertrag {
…
}
```

System Complexity

zühlke
empowering ideas

- **Kennzahlen**
  - Anzahl Klassen: ~1800
  - Lines of Code: ~100‘000
  - Anzahl Methoden: ~10‘000
  - Anzahl Attribute: ~3‘000
  - Anzahl Packages: ~360

Stand nach 1 Jahr Entwicklung

- **Davon ca. 50 % Code für Tests**
- **Weitere generierte Klassen (~500)**
- **Weitere interne Frameworks**

Rollen

- *„Kunde“ (Sponsor), Leitungsausschuss*
- *Gesamtprojektleiter, Teilprojektleiter*
- *Verantwortliche Fachbereich*
- Architekten
- Integratoren
- Chef-Entwickler
- Applikations-Entwickler
- Test-Engineers
- DB-Spezialisten (DB, Batch Verarbeitung)

Usabality Engineering

zühlke
empowering ideas

- Story Board mit dem Kunden erarbeiten
- Beobachtungen mit dem Kunden, möglichst realitätsnah

«Peter klebt den Avisierungszettel auf das Paket und übernimmt dabei das vom Scanner angezeigte Datum.»

Paper Prototyping

zühlke
empowering ideas

«Peter hält Werner den Scanner zur Unterschrift hin (zusammen mit dem dazu notwendigen Stift).

Werner kontrolliert kurz die Anzahl der unterschriftspflichtigen Pakete (er zählt immer mit, sonst bekommt er Ärger mit seinem Chef wenn mal was nicht stimmt).

10 Stück, das hat er auch gezählt. Er unterschreibt.»

Rollen in agilen Vorgehensweisen

Bei Scrum gibt es nur drei Rollen

- Product Owner
- Scrum Master
- Entwicklungs-Team

http://www.scrum.org/storage/scrumguides/

Release Management

zühlke
empowering ideas

Definition Plattform / Umgebung

- Verbundenes System von
  - App-Server, DB, Umsysteme, …

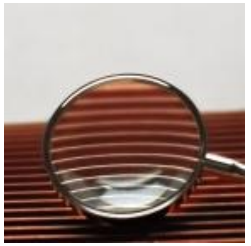
Umgebungen

- Lokal (Entwickler)
- Test-Umgebung
  - 1-2 wöchentlich durch Integrator bereitgestellt
  - Release Notes, Versionsnummer
  - CVS Tag
- Integrations-Umgebung
  - Fachbereich: Tests (alle 6 Wochen bei Iterationsende)
  - Schulungen
- Produktions-Umgebung

Testing

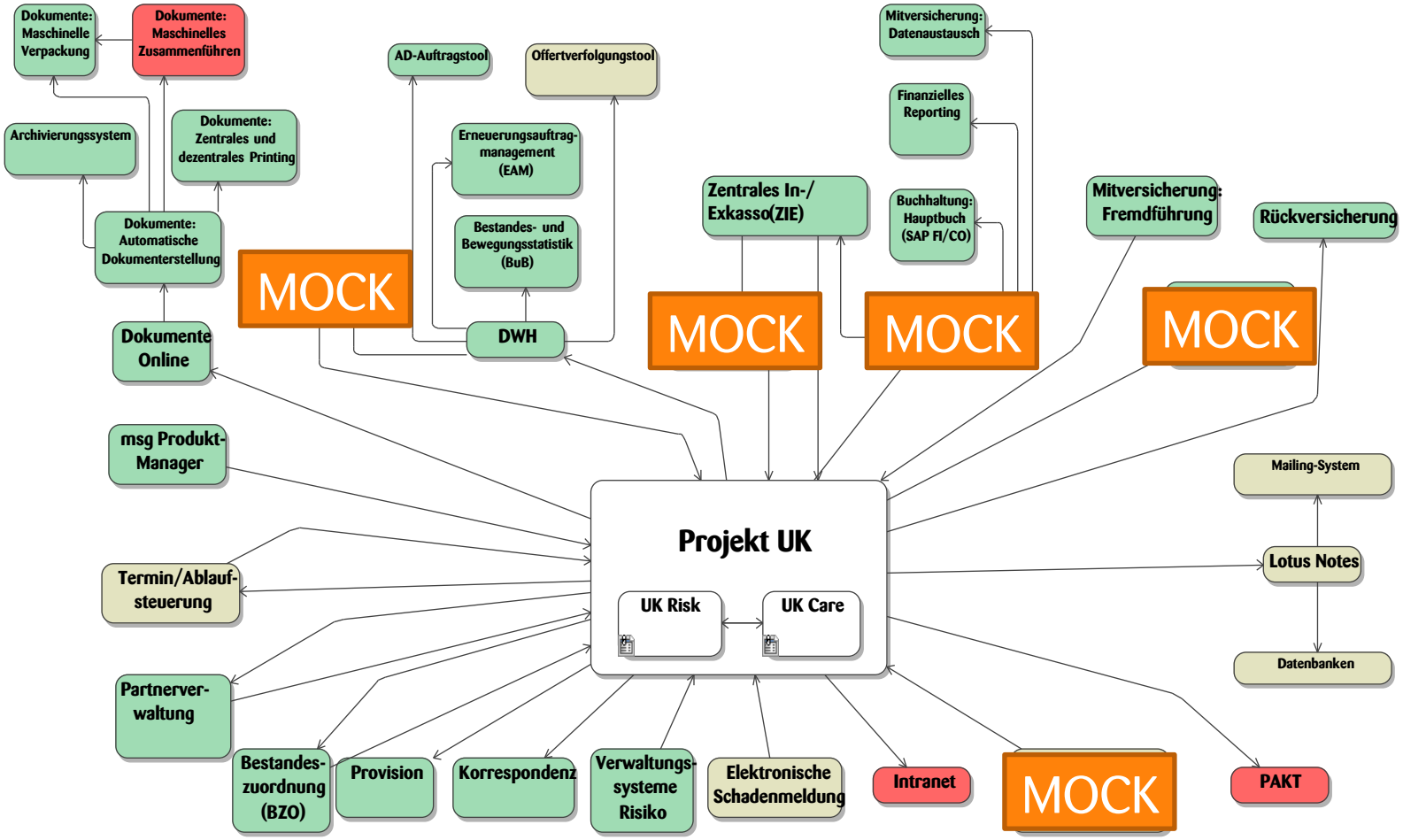
zühlke
empowering ideas

- Vorgaben
  - Unit Testing, 80% Abdeckung
  - Regressions Tests, möglichst automatisiert
- Tools
  - Coverage
  - „GUI-Player“
- Fehler Datenbank
  - Erfassen von Fehlern
  - Verfolgen von Fehlern
- Herausforderungen:
  - Umsysteme
  - Fremdsysteme
  - Realistische Daten
  - Datenschutz

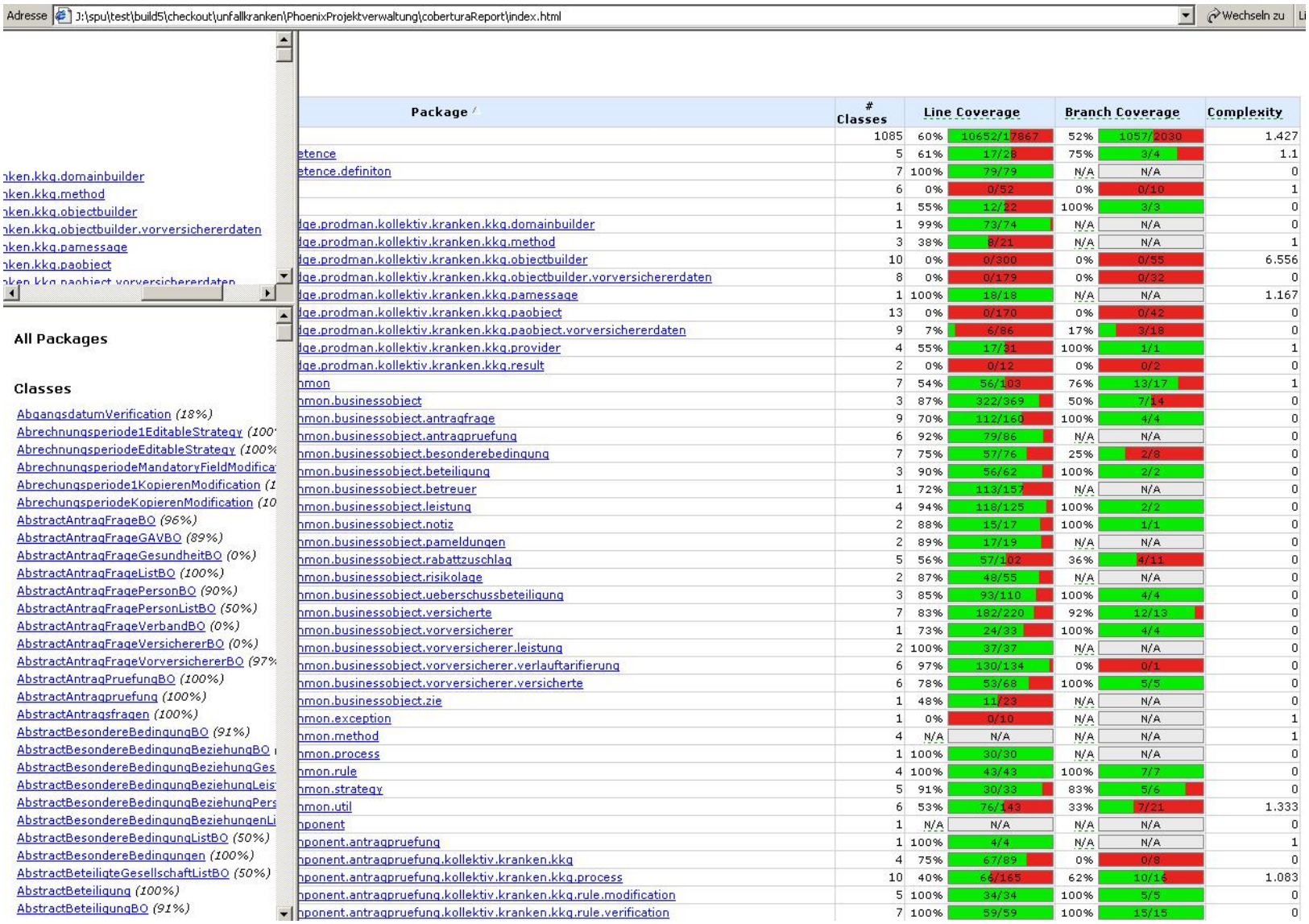


Umsysteme: Mocken!

zühlke
empowering ideas



Testing: Coverage Tool



Package	# Classes	Line Coverage		Branch Coverage		Complexity
	1085	60%	10652/17867	52%	1057/2030	1.427
etence	5	61%	17/28	75%	3/4	1.1
etence.definiton	7	100%	79/79	N/A	N/A	0
	6	0%	0/52	0%	0/10	1
	1	55%	12/22	100%	3/3	0
dge.prodman.kollektiv.kranken.kkq.domainbuilder	1	99%	73/74	N/A	N/A	0
dge.prodman.kollektiv.kranken.kkq.method	3	38%	8/21	N/A	N/A	1
dge.prodman.kollektiv.kranken.kkq.objectbuilder	10	0%	0/300	0%	0/55	6.556
dge.prodman.kollektiv.kranken.kkq.objectbuilder.vorversichererdaten	8	0%	0/179	0%	0/32	0
dge.prodman.kollektiv.kranken.kkq.pamessage	1	100%	18/18	N/A	N/A	1.167
dge.prodman.kollektiv.kranken.kkq.paobject	13	0%	0/170	0%	0/42	0
dge.prodman.kollektiv.kranken.kkq.paobject.vorversichererdaten	9	7%	6/86	17%	3/18	0
dge.prodman.kollektiv.kranken.kkq.provider	4	55%	17/31	100%	1/1	1
dge.prodman.kollektiv.kranken.kkq.result	2	0%	0/12	0%	0/2	0
hmon	7	54%	56/103	76%	13/17	1
hmon.businessobject	3	87%	322/369	50%	7/14	0
hmon.businessobject.antragfrage	9	70%	112/160	100%	4/4	0
hmon.businessobject.antragpruefung	6	92%	79/86	N/A	N/A	0
hmon.businessobject.besonderebedingung	7	75%	57/76	25%	2/8	0
hmon.businessobject.beteiligung	3	90%	56/62	100%	2/2	0
hmon.businessobject.betreuer	1	72%	113/157	N/A	N/A	0
hmon.businessobject.leistung	4	94%	118/125	100%	2/2	0
hmon.businessobject.notiz	2	88%	15/17	100%	1/1	0
hmon.businessobject.pameldungen	2	89%	17/19	N/A	N/A	0
hmon.businessobject.rabattzuschlag	5	56%	57/102	36%	4/11	0
hmon.businessobject.risikolage	2	87%	48/55	N/A	N/A	0
hmon.businessobject.ueberschussbeteiligung	3	85%	93/110	100%	4/4	0
hmon.businessobject.versicherte	7	83%	182/220	92%	12/13	0
hmon.businessobject.vorversicherer	1	73%	24/33	100%	4/4	0
hmon.businessobject.vorversicherer.leistung	2	100%	37/37	N/A	N/A	0
hmon.businessobject.vorversicherer.verlauftarifierung	6	97%	130/134	0%	0/1	0
hmon.businessobject.vorversicherer.versicherte	6	78%	53/68	100%	5/5	0
hmon.businessobject.zie	1	48%	11/23	N/A	N/A	0
hmon.exception	1	0%	0/10	N/A	N/A	1
hmon.method	4	N/A	N/A	N/A	N/A	1
hmon.process	1	100%	30/30	N/A	N/A	0
hmon.rule	4	100%	43/43	100%	7/7	0
hmon.strategy	5	91%	30/33	83%	5/6	0
hmon.util	6	53%	76/143	33%	7/21	1.333
hponent	1	N/A	N/A	N/A	N/A	0
hponent.antragpruefung	1	100%	4/4	N/A	N/A	1
hponent.antragpruefung.kollektiv.kranken.kkq	4	75%	67/89	0%	0/8	0
hponent.antragpruefung.kollektiv.kranken.kkq.process	10	40%	66/165	62%	10/16	1.083
hponent.antragpruefung.kollektiv.kranken.kkq.rule.modification	5	100%	34/34	100%	5/5	0
hponent.antragpruefung.kollektiv.kranken.kkq.rule.verification	7	100%	59/59	100%	15/15	0

zühlke
empowering ideas

Testing: Coverage Tool

zühlke
empowering ideas

```
48      /**
49       * Mappt das DO in das BO.
50       *
51       * @param aAbstractVertragBO
52       *
53       * @param aVertragDO
54       *           das zu mappende Vertrag-DO
55       *
56       *
57       */
58      public void fillDOAttributesInBO(VertragDO aVertragDO, AbstractVertragBO aAbstractVertragBO) {
59  4     super.fillDOAttributesInBO(aVertragDO, aAbstractVertragBO);
60  4     KKGVertragBO kkgvertragBO = (KKGVertragBO)aAbstractVertragBO;
61        // Wirtschaftsbranche wird in KKGPersonGruppeObjectMapper.fillDOinBO nachtr?glich gesetzt.
62  4     kkgvertragBO.setWirtschaftsbranche((short)0);
63        // Tarifausgabe wird in KKGLeistungVersichertObjectMapper.fillDOinBO nachtr?glich gesetzt.
64  4     kkgvertragBO.setTarifausgabe(null);
65
66        // Feld "LohnSummeKategorieAbwAnzeige" ist nur f?r Anzeige in GUI.
67  4     if (kkgvertragBO.getLohnSummeKategorieErmittelt().equals(kkgvertragBO.getLohnSummeKategorieWirksam())) {
68  4       kkgvertragBO.setLohnSummeKategorieAbwAnzeige(CodeLohnsummenKategorie.CODE_NULL);
69  4     }
70        else {
71  0       kkgvertragBO.setLohnSummeKategorieAbwAnzeige(kkgvertragBO.getLohnSummeKategorieWirksam());
72        }
73
74  4     Calendar cal = Calendar.getInstance();
75  4     cal.set(Calendar.MONTH, aVertragDO.getHauptfaelligmm() - 1);
76  4     cal.set(Calendar.DAY_OF_MONTH, aVertragDO.getHauptfaelligtt());
77
78  4     kkgvertragBO.setHauptFaelligkeit(new       Date(cal.getTimeInMillis()));
79  4     kkgvertragBO.setVertragsdauerCode(CodeVertragsdauer.getCodeObject(aVertragDO.getVertragdauer()));
80
81  4     setGavbez(aAbstractVertragBO, aVertragDO.getGavbez());
82  4   }
```

Testing & Annotation

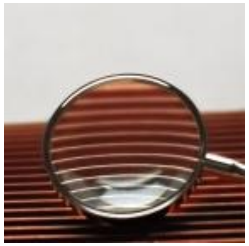
zühlke
empowering ideas

Gewisse Klassen können nur unter speziellen Umständen getestet werden.

Beispiel: Abhängigkeit zu Resourcen, wie DB, Umsysteme oder Fremdsysteme

Damit dies nicht auf die erforderlichen 80% Testabdeckung gefährdert, kann dies dem Coverage Tool mit Annotation mitgeteilt werden:

```
@Testcoverage(linecoverage = 0,
  comment = "Tested by VertragHibernateDaoCRDRdTest")
public class VertragHibernateDAO implements VertragDAO {

    ...

}
```

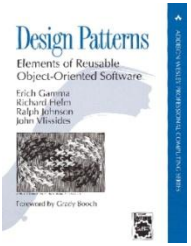


Design Patterns

zühlke
empowering ideas

- **Detailliertere Architektur / Design**

- **Patterns:**
  - Composite
  - Factory
  - Visitor

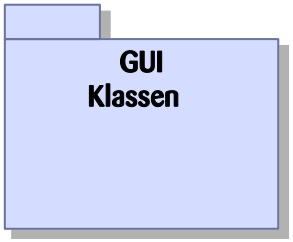


Detaillierte Architektur / Design

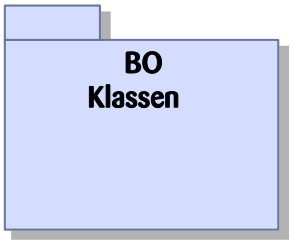
zühlke
empowering ideas

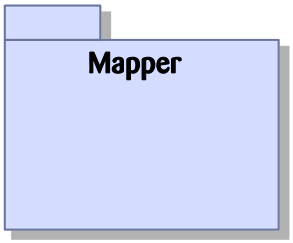
Layer

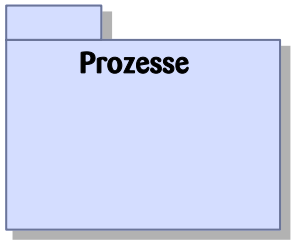
Präsentation



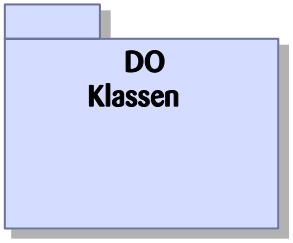
Business

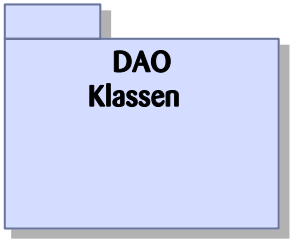






Persistenz

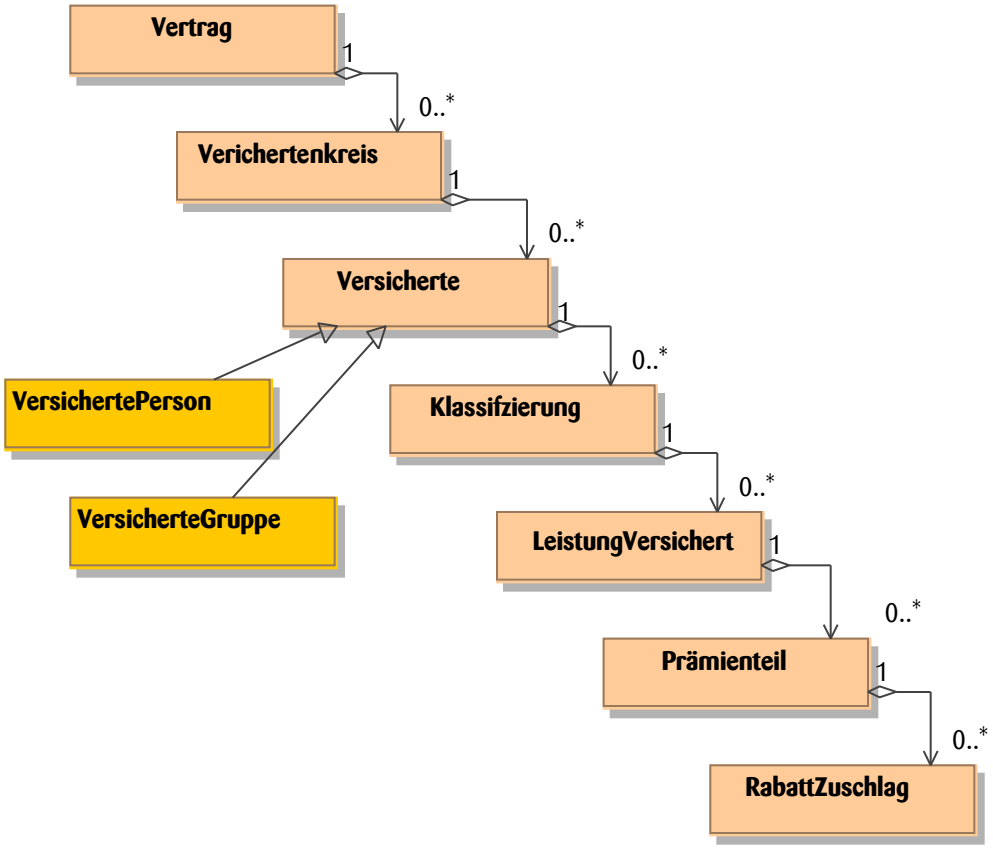




Business Model (BOs)

zühlke
empowering ideas

Versicherung Vertrag (vereinfacht)



Design Pattern
Composite

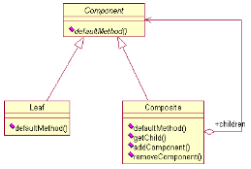
zühlke
empowering ideas

Problematik:

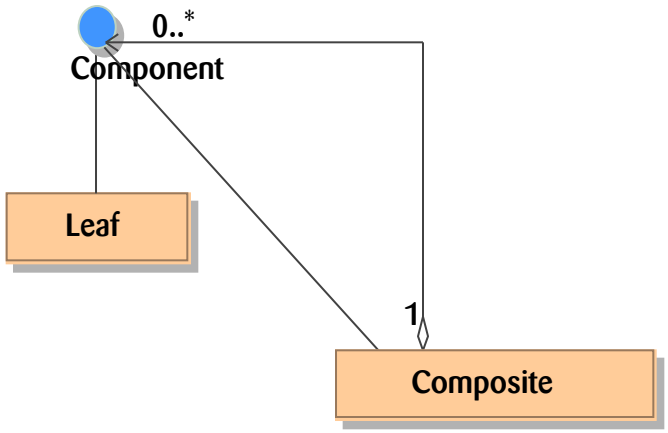
- Oft will man nun die einzelne BO-Elemente gleich behandeln
  - Bsp: Speichern, Kopieren

Lösung:

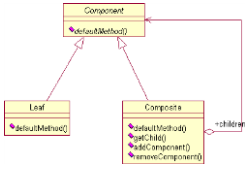
- Composite Pattern ermöglicht eine Gruppe von Objekten zu behandeln wie wenn es eines wäre.



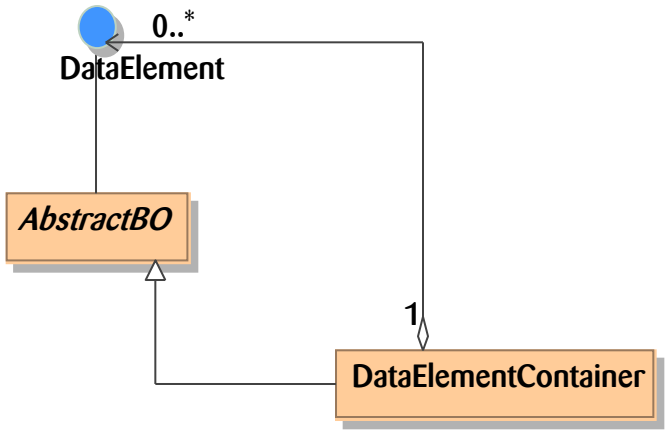
Design Pattern Composite



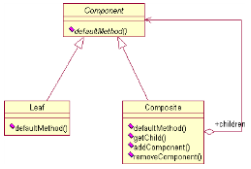
Übliche Bezeichnungen und Strukturen



Design Pattern Composite

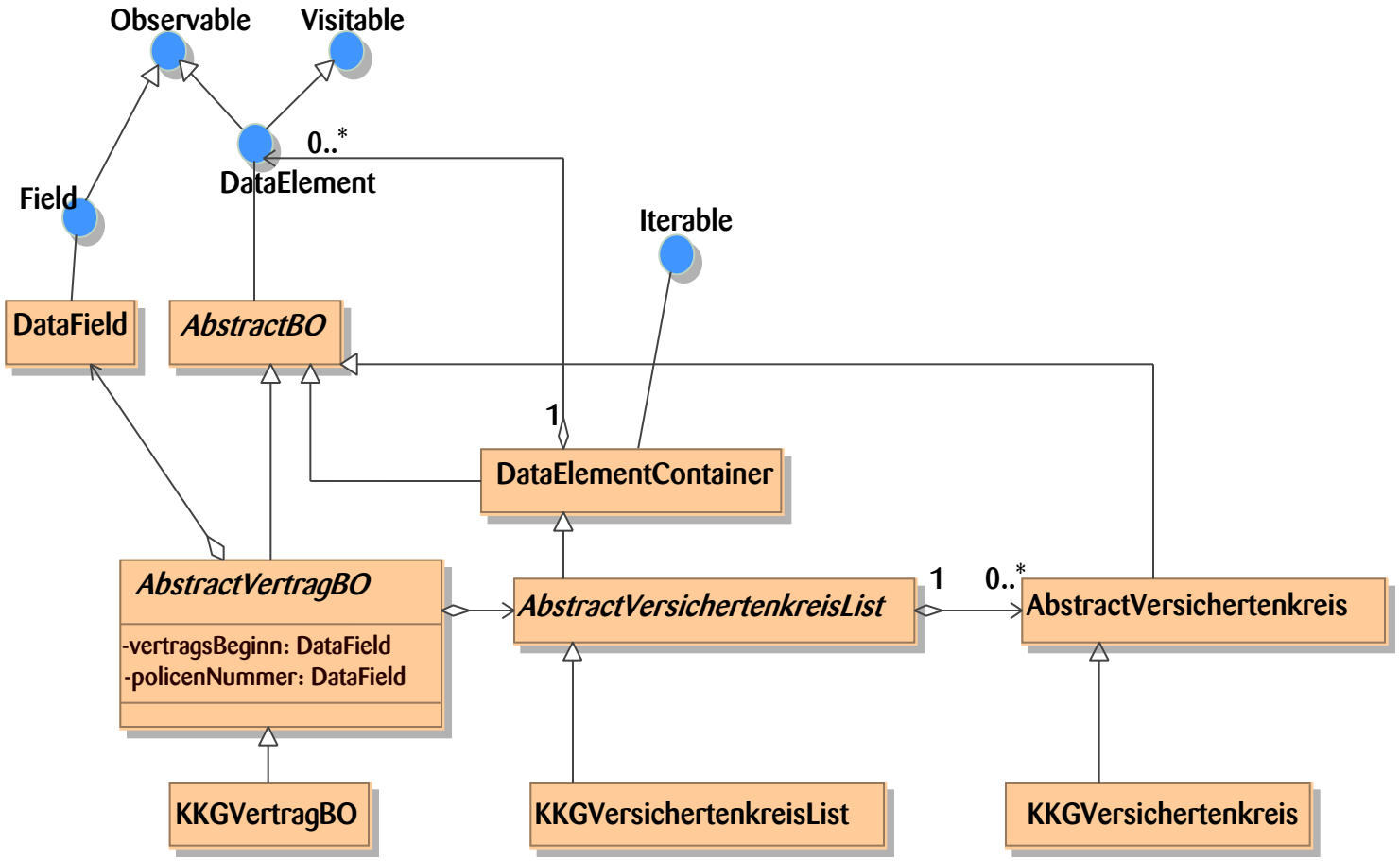


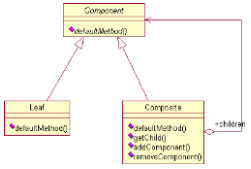
Bezeichnungen und Strukturen in unserem Projekt



Design Pattern Composite

zühlke
empowering ideas

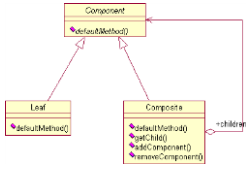




Design Pattern: Composite
Das Component Interface

zühlke
empowering ideas

```
public interface DataElement extends Observable, Visitable {


  public void setParent(DataElement aDataElement);

  public DataElement getParent();


  // Beispiel einer weiteren Operation auf dem Component:

  public void setChanged();

  …

}
```



Design Pattern: Composite
Das Leaf

zühlke
empowering ideas

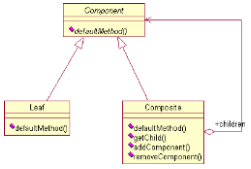
```
public abstract class AbstractBO implements DataElement {
  …

  private DataElement parent;
  …

  public DataElement getParent() { return parent; }
  public void setParent(DataElement aDataElement) { parent = aDataElement; }
  …

}
```

- Davon gibt es dann die Subklassen pro Business Objekt:

```
public abstract class AbstractVertragBO extends AbstractBO { … }

public abstract class AbstactVersichertenkreisBO
                                  extends AbstractBO { … }
```

- Davon die konkreten Klassen:

```
public class KKGVertragBO extends AbstractVetragBO { … }

public class KKGVersichertenkreisBO extends
                           AbstractVersichertenkreisBO { … }
```



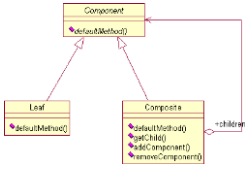
Design Pattern: Composite
Das Composite

zühlke
empowering ideas

```
public abstract class DataElementContainer<T extends DataElement>
 extends AbstractBO implements Iterable<T> {

  …

  private List<T> felder;

  public int size() { return felder.size(); }

  public T getDataElement(int aIndex) { return felder.get(aIndex); }
  public void removeDataElement(int aIndex) { felder.remove(aIndex); }

  public Iterator<T> iterator() { return felder.iterator(); }

  …

}
```

- Subklassen:

```
public abstract class
  AbstractVersichertenkreisList<K extends AbstractVerischertenkreisBO>
          extends DataElementContainer<K> { … }

public class KKGVerischertenkreisList extends
          AbstractVersichertenkreisList<KKGVersichertenkreisBO> { … }
```



Design Pattern
Factory

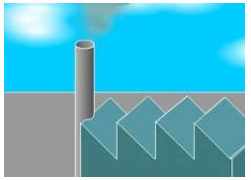
zühlke
empowering ideas

Problematik:

- Das ganze Persistenz Layer soll eventuell zu einem späteren Zeitpunkt ausgewechselt werden.

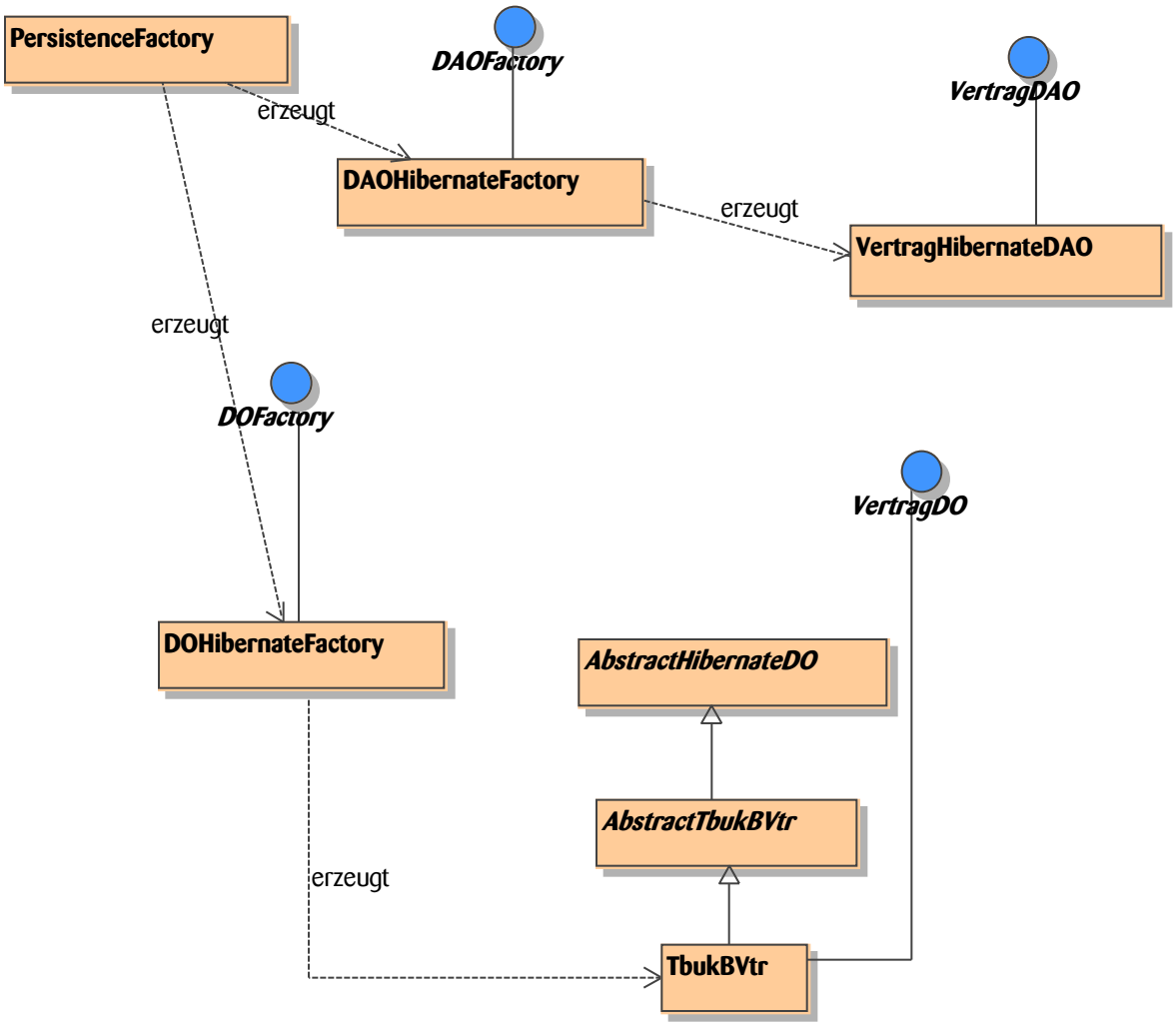
Lösung:

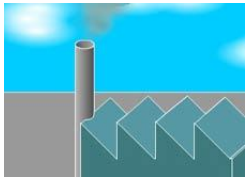
- „Oberes Layer“, das Business Layer, kommuniziert nur mit Interfaces.
- Es gibt eine **Persistenz-Factory** welche die konkreten Objekte des Persistenz-Layers (DAO & DO) kreiert.



Design Patterns: Factory UML

zühlke
empowering ideas



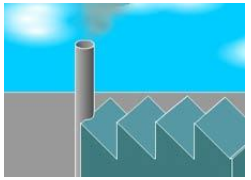


Design Pattern: Factory
Das Interface

zühlke
empowering ideas

- Das Interface für die Factory welche die Persistenz-Objekte (DO) erzeugt

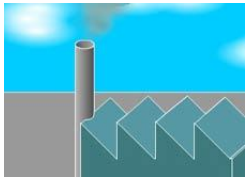
```
public interface DOFactory {

  public VertragDO createVertragDO();
  public VersichertenkreisDO createVersichertenkreisDO();
  public VersichertenkreisDO createVersicheDO();
  public KlassifzierungDO createKlassifzierungDO();
  public VersichertenkreisDO createVersichertenkreisDO();
  public LeistungVersichertDO createLeistungVersichertDO();
  public PraemienteilDO createPraemienteilDO();
  public RabattZuschlagDO createRabattZuschlagDO();
}
```



Design Pattern: Factory
Konkrete Implementation für Hibernate

zühlke
empowering ideas

```
public class DOHibernateFactory implements DOFactory {


    public VertragDO createVertragDO() {
       return new TbukBVtr();
    }


    public VersichertenkreisDO createVersichertenkreisDO() {
       return new TbukBVerskreis();
    }


    public VericherteDO createVerischerteDO() {
       return new TbukBVersichert();
    }


    public KlassifizierungDO createKlassifizierungDO() {
       return new TbukBKlassifiz();
    }

    ...

}
```



Alternative zu Factory Pattern? Dependency Injection!

zühlke
empowering ideas

```
public class PersistenceFassade {

        @Inject
        DAOFactory daoFactory;

        @Inject
        DOFactory doFactory;

}
```

http://agoncal.wordpress.com/2011/01/12/bootstrapping-cdi-in-several-environments/

http://docs.jboss.org/weld/reference/latest/en-US/html/

In Kombination mit:

zühlke
empowering ideas

```
public class TestDaoFactory {

   private PersistenceFassade persistenceFassade;

   @Test
   public void test() {
      persistenceFassade = new PersistenceFassade();
      DaoFactory mockDaoFactory = mock(DaoFactory.class);
     persistenceFassade.daoFactory = mockDaoFactory;
     when(mockDaoFactory.getVertragDao()).
        thenReturn(mock(VertragDao.class));
      persistenceFassade.daoFactory.getVertragDao().readVertrag(22);
   }
}
```

http://mockito.org/

Design Pattern
Visitor

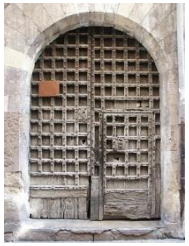
zühlke
empowering ideas

Problematik:

- Innerhalb unseres BO-Trees wollen nach gewissen Elemente suchen, zum Beispiel nach gewissen Leistungen.

Lösung:

- Ein BO kann einen **Visitor** empfangen. Dieser Visitor ist eine konkrete Klasse und kann auf das „besuchte“ BO zugreifen.
  Nach dem Besuch werden rekursiv alle Kinder das BOs besucht.



Design Pattern: Visitor Interfaces

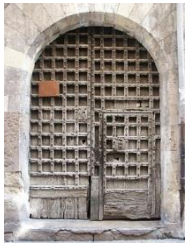
zühlke
empowering ideas

Visitor:

```
public interface Visitor {
    public void visit(Visitable aVisitable);
}
```

Visitable:

```
public interface Visitable {
    public void accept(Visitor aVisitor)
}
```

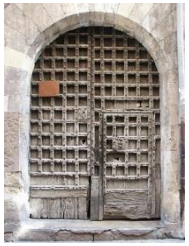


Design Pattern: Visitor Visitable

zühlke
empowering ideas

```
public abstract class AbstractBO implements DataElement {

    …

    public void accept(Visitor aVisitor) {
        aVisitor.visit(this);
    }

    …

}
```

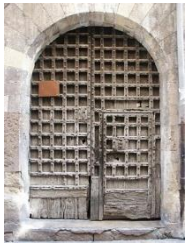
```
public abstract class DataElementContainer<T extends DataElement>
  extends AbstractBO implements Iterable<T> {

    …

    public void accept(Visitor aVisitor) {

        super.accept(aVisitor);

        Iterator<T> iter = iterator();
        while (iter.hasNext()) {
            T child = iter.next();
            child.accept(aVisitor);
        }
    }
```



Design Pattern: Visitor
Konkreter Visitor

zühlke
empowering ideas

```
class ElementCollector<T> implements Visitor {

  /** Hier werden die vom Visitor gefunden Objekte gespeichert. */
  private Collection<T> result = new LinkedHashSet<T>();

  /** Die zu suchende Klasse. */
  private Class<T> classToFind;


  public ElementCollector(Class<T> aClass) {
    classToFind = aClass;
  }


  public void visit(Visitable aVisitable) {
    if (classToFind.isAssignableFrom(aVisitable.getClass())) {
      result.add((T)aVisitable);
    }
  }

  public Collection<T> getResult() {
    return result;
  }}
```



Design Pattern: Visitor Anwendung

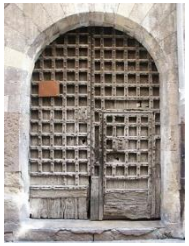
zühlke
empowering ideas

Anwendung des ElementCollectors:

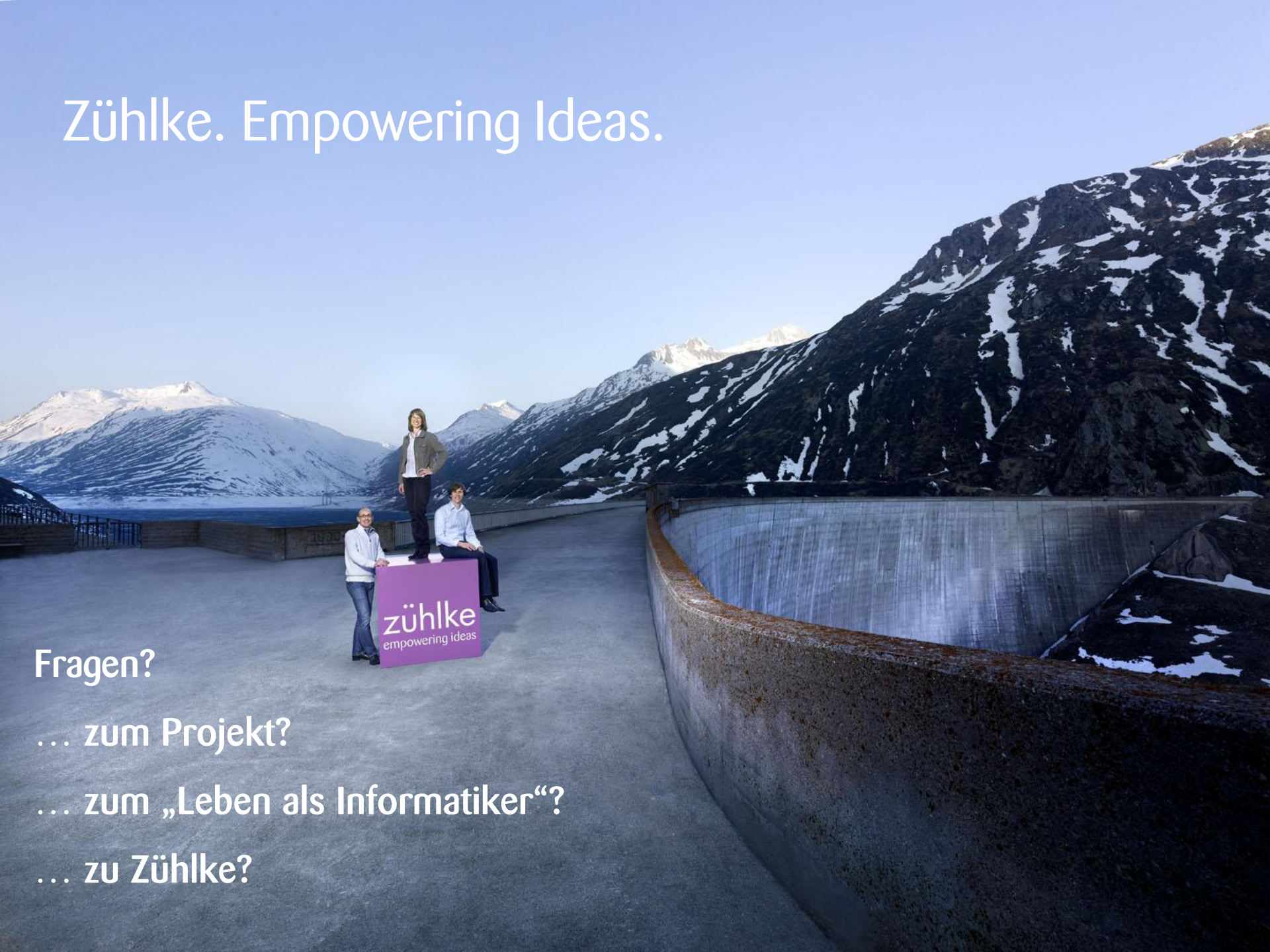
```
public abstract class AbstractBO implements Visitable {

    …

  public <T extends AbstractBO> Collection<T>
         getAllElementsMatchingClass(Class<T> aClass) {

    ElementCollector visitor = new ElementCollector<T>(aClass);

    this.accept(visitor);

    return visitor.getResult();

  }

    …

}
```

Aufruf für alle versicherten Leistungen:

```
Collection<AbstractLeistungVersichertBO>
    allLeistungVersichertBOs =
     getAllElementsMatchingClass(AbstractLeistungVersichertBO.class)
```



Zühlke. Empowering Ideas.



Fragen?

… zum Projekt?

… zum „Leben als Informatiker“?

… zu Zühlke?