Einblicke in die Praxis –

Java EE Projekt einer Versicherung

Vorlesung P2, Universität Bern, 28. Mai 2010

Frank Buchli
Software Architect
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 Januar 2007 Software Engineer bei Zühlke

Agenda

zühlke
empowering ideas

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

Frank Buchli

Zühlke

Frank Buchli

Consulting Development Integration

zühlke
empowering ideas

- Über 5000 Projekte in Europa realisiert
- 65 Mio. CHF Umsatz (2009)
- 350 Mitarbeitende (Ende 2009)
- In Bern, Zürich, Frankfurt, Hannover, München, Wien und London
- Bern: Aarbergergasse 29
  ~30 Mitarbeiter
- Gründung 1968,
  im Besitz von Partnern

Wir beraten, entwickeln und integrieren aufgabengerecht – mit überzeugender Qualität und Wirtschaftlichkeit.

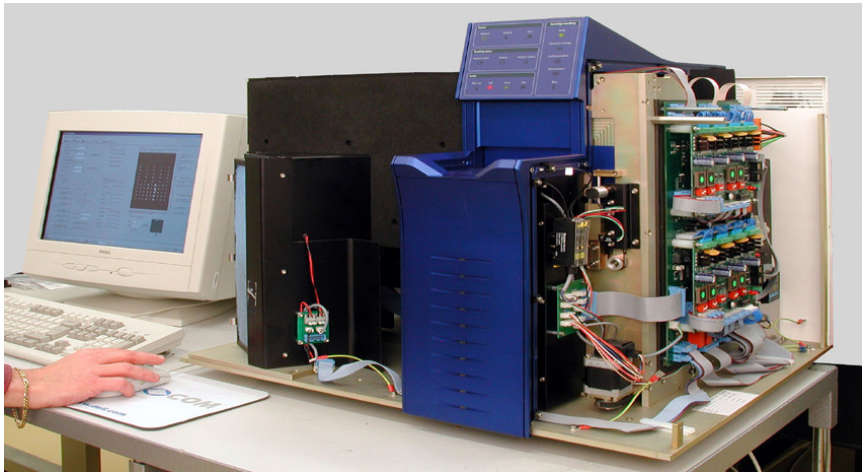
Frank Buchli

Produktentwicklung

zühlke
empowering ideas

Beispiele Produktentwicklung

- Steuerungselektronik
- Firmware
- Realisierung vom Konzept bis zur Nullserie

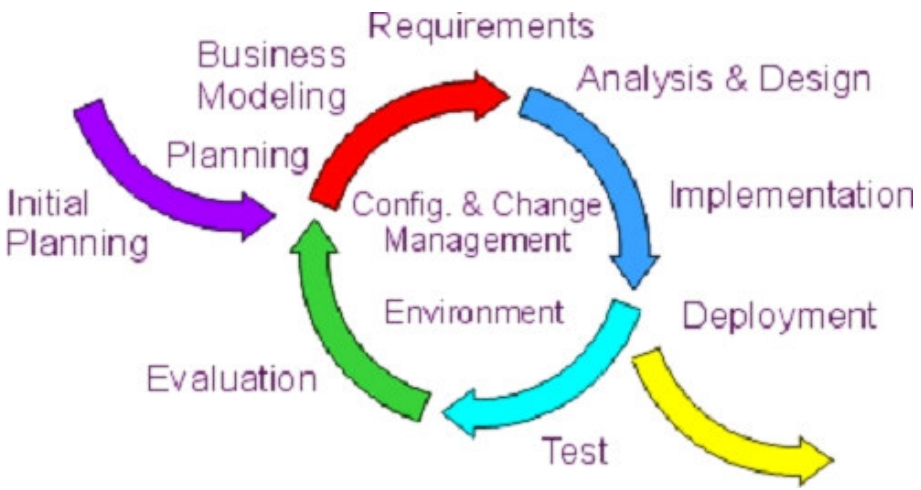


Frank Buchli

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 – 2012
- Involvierte Leute
- Kosten
- Unterstützung durch externe Spezialisten
- Meine Rolle: Chef Entwicklung TP Schaden

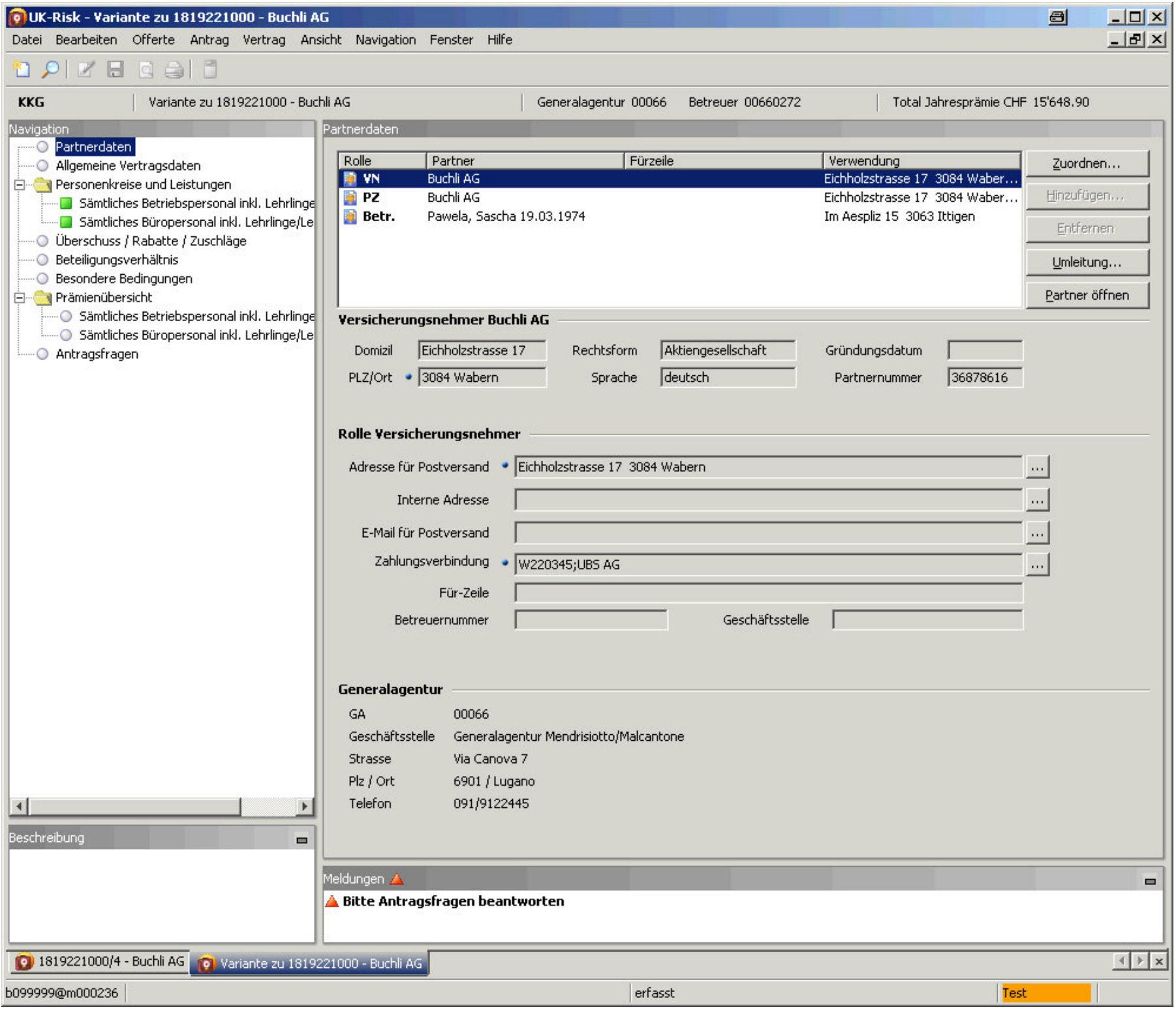
Frank Buchli

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?

Frank Buchli



Screenshot

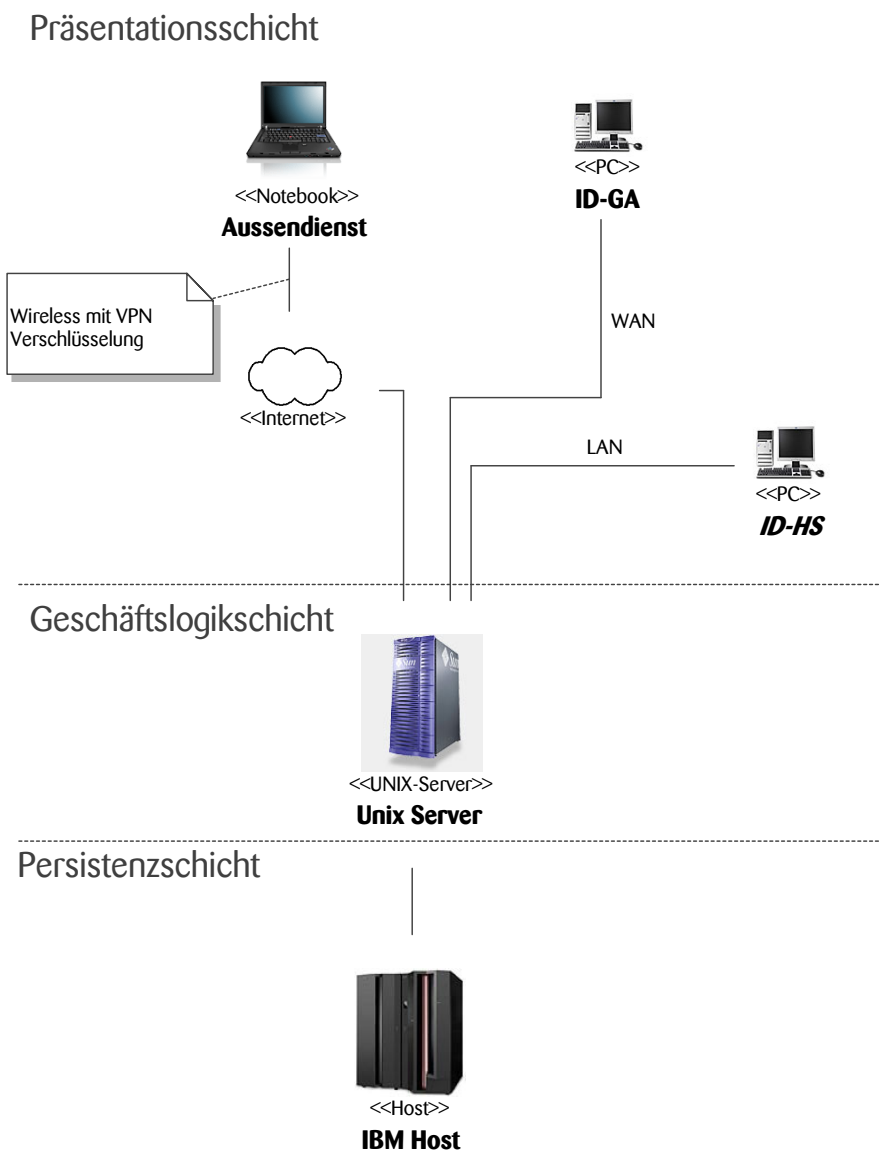
zühlke
empowering ideas

Einblicke in die Praxis
Folie 10
28. Mai 2010

Frank Buchli

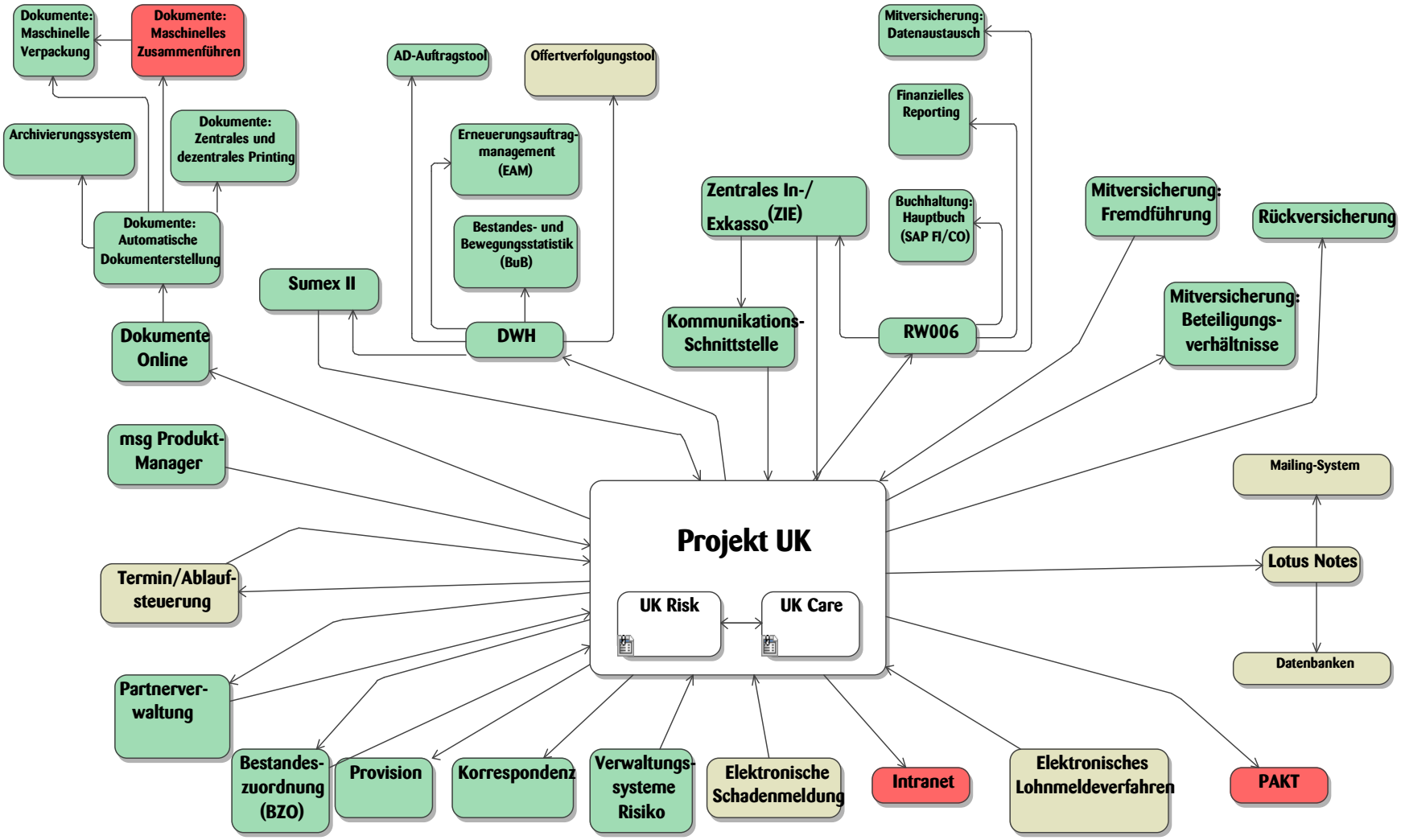
Architektur: Verteilungssicht

zühlke
empowering ideas



Frank Buchli

Umsysteme

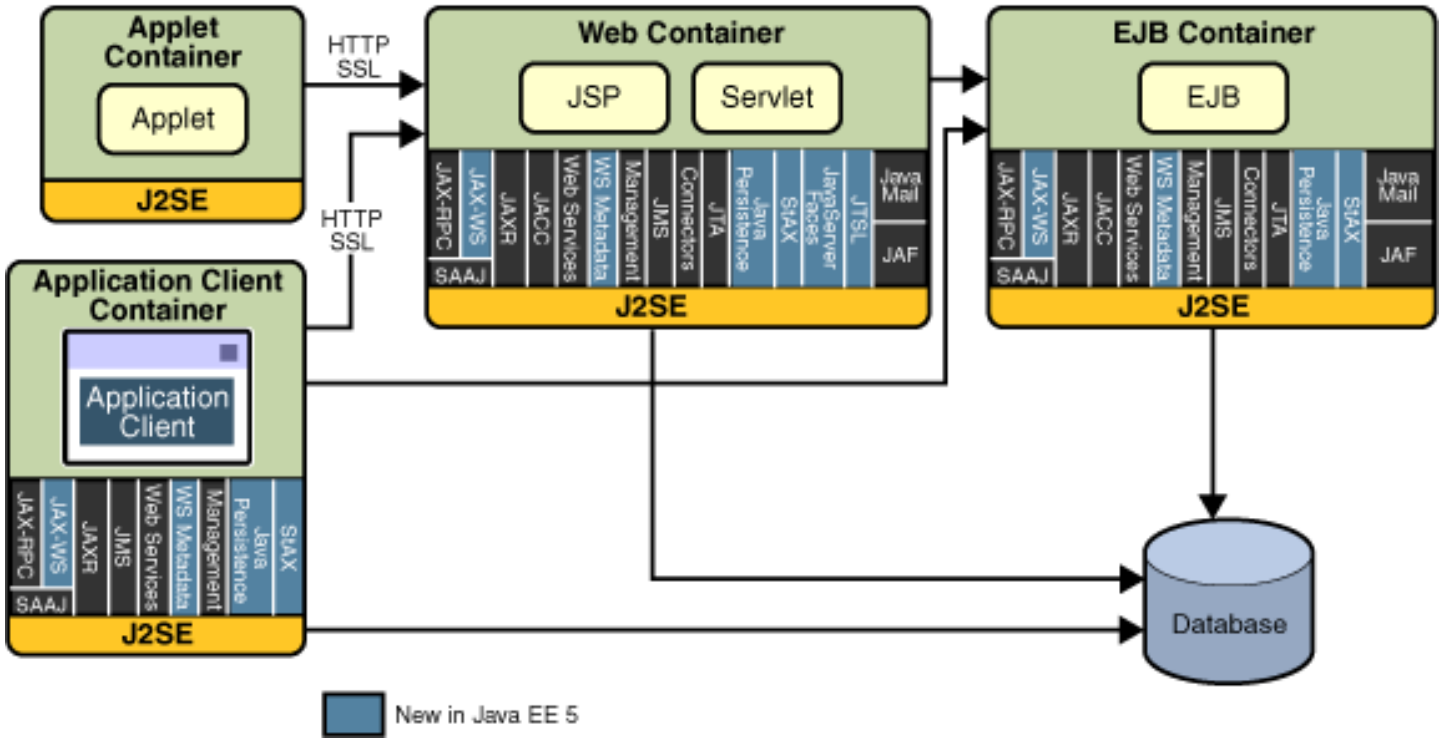


Frank Buchli

Verwendete Technologien
Java Versionen

zühlke
empowering ideas

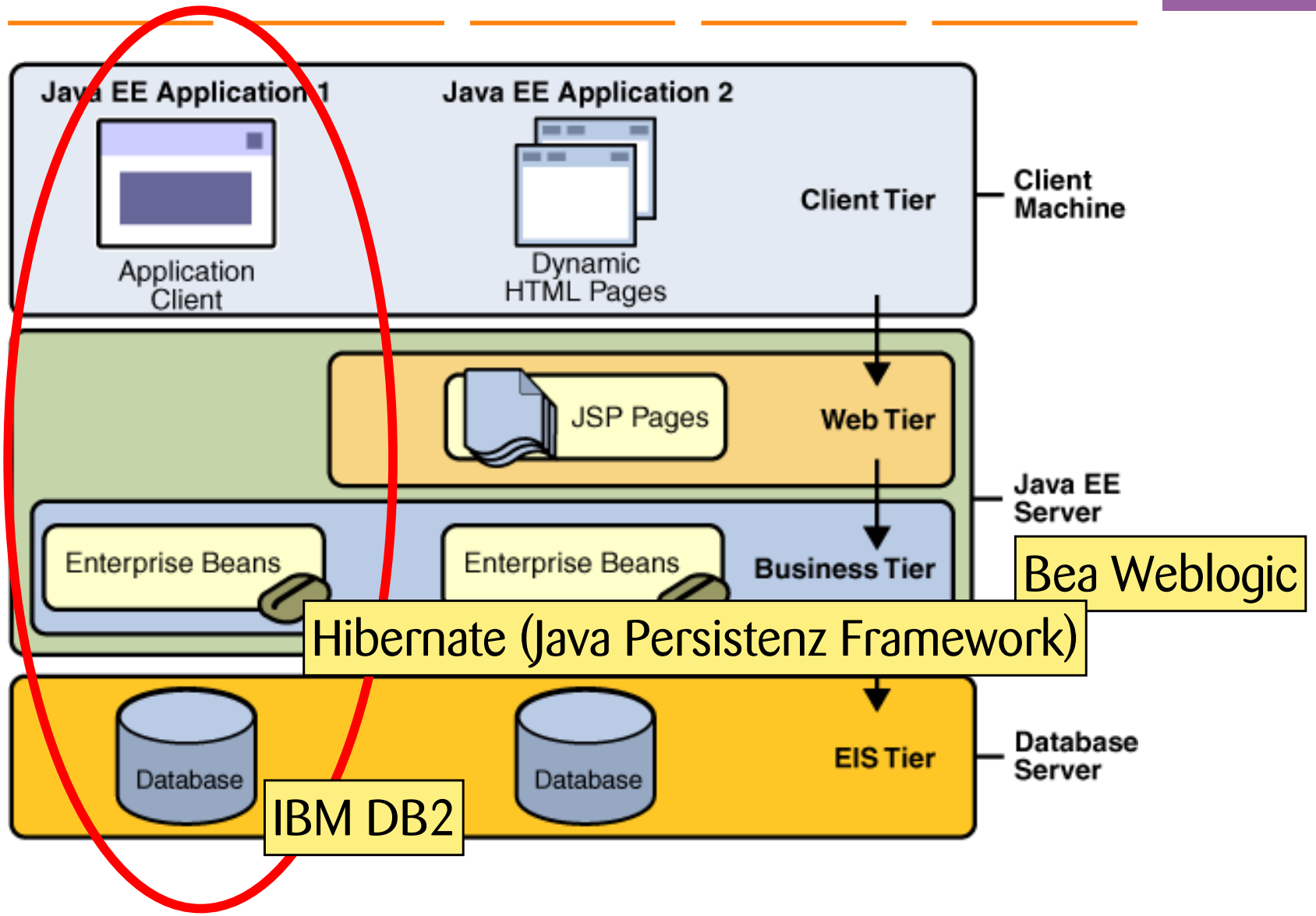
- Java SE 5
  - Generics & Annotation
- J2EE 1.4
  - EJB 2.0



Frank Buchli

Verwendete Technologien
Java EE

zühlke
empowering ideas



Frank Buchli

Entwicklungsumgebung

zühlke
empowering ideas

- IDE: Eclipse 3.2.2
  - Diverse Plugins: MyEclipse, Format-On-Save
  - JFormDesigner
- Source Repositiory: CVS
- Build: Maven 2, Continuum
- MDD: AndroMDA
- Design:
  - Magic Draw 12.1 (UML)
  - Power Designer (DB)
- Requirement Verwaltung & Testen: HP Quality Center
- Viele eigene Frameworks, Tools

Frank Buchli

Entwicklungsvorgehen

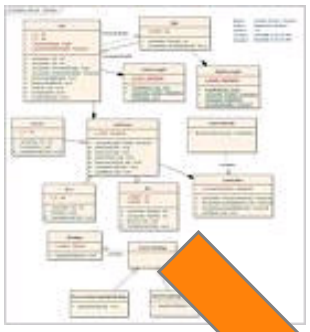
zühlke
empowering ideas

- 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

Frank Buchli

Entwicklungsvorgehen
Model Driven Development (MDD)

zühlke
empowering ideas



Template 2

Template 1

```
public class XXX {
...
}
```

MDD Engine

SQL Code

```
create table T_Vertrag (
...
)
```

Source Code

```
public class Vertrag {
...
}
```

Frank Buchli

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**

Frank Buchli

Rollen

zühlke
empowering ideas

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

Frank Buchli

Fachgremien

zühlke
empowering ideas

- Prüfen der Lösungsentwürfe
  - Korrektheit
  - Firmenkonform
  - Skalierbarkeit
- Formelles Review durch Fachgremium
- Beispiele von Fachgremien:
  - Business / Fachklassenmodell
  - Datenbank – Modell
  - GUI

Frank Buchli

Release Management

zühlke
empowering ideas

Definition Plattform / Umgebung

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

Umgebungen

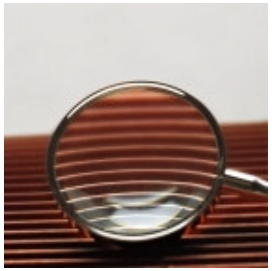
- Lokal (Entwickler)
- Test-Umgebung
  - 1-2 wöchentlich durch Integrator bereitgestellt
  - Release Notes, Versionsnummer
  - CVS Tag
- Integrations-Umgebung
  - Fachbereich: Tests (alle 6 Wochen bei Iterationsende)
  - Schulungen
- Produktions-Umgebung

Frank Buchli

Testing

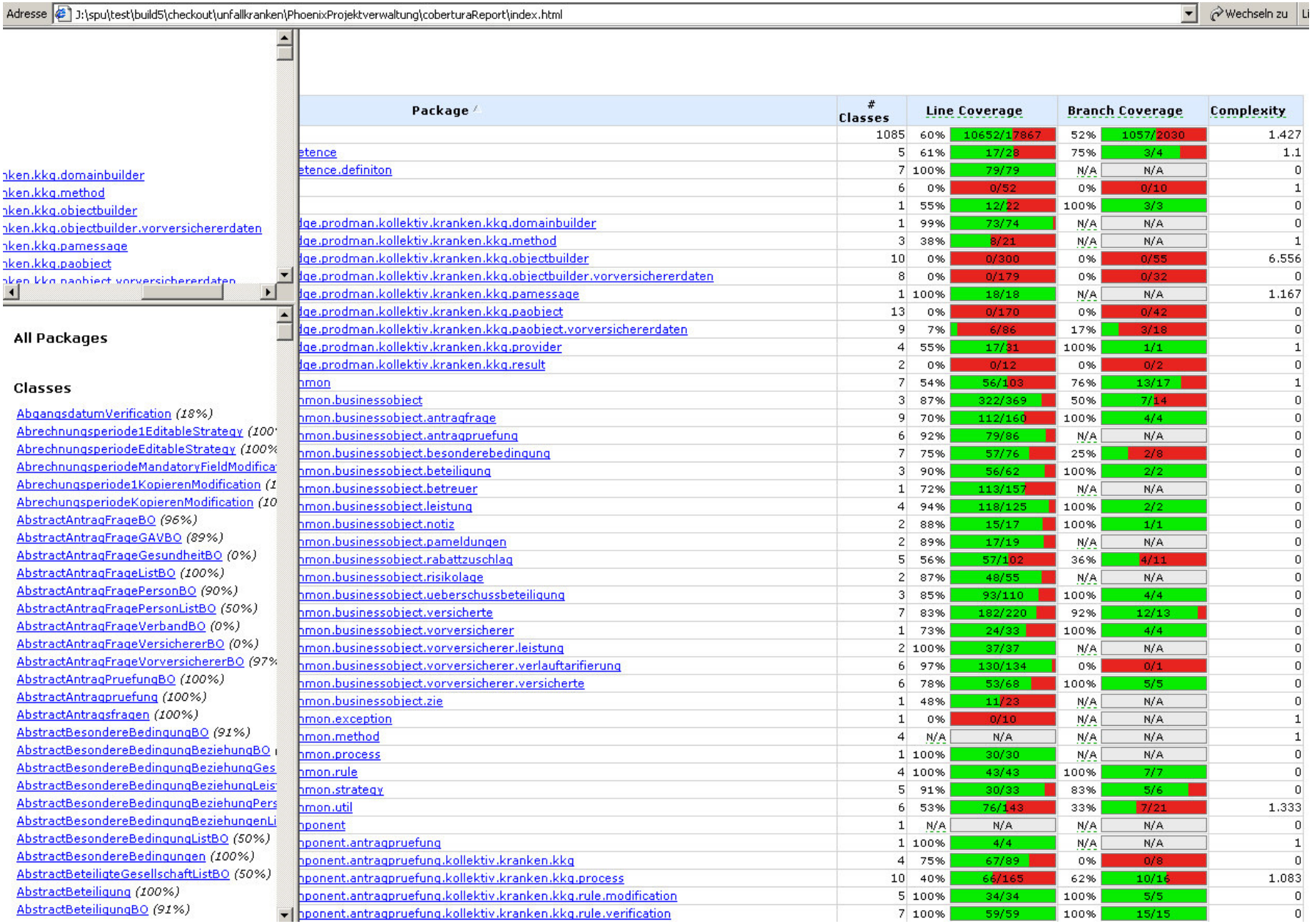
zühlke
empowering ideas

- 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



Frank Buchli

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

zühlke
empowering ideas

Frank Buchli

Testing: Coverage Tool

zühlke
empowering ideas

```
48      /**
49       * Mappt das DO im 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 im 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   }
```


Frank Buchli

Testing & Annotation

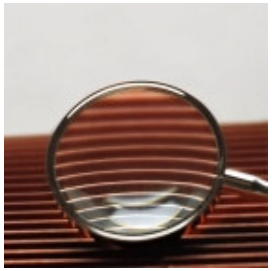
zühlke
empowering ideas

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 {

    ...

}
```



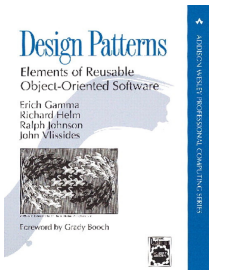
Frank Buchli

Design Patterns

zühlke
empowering ideas

- **Detailliertere Architektur / Design**

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



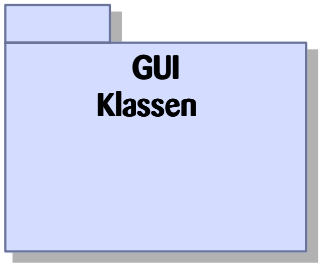
Frank Buchli

Detaillierte Architektur / Design

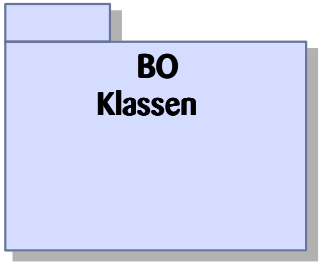
zühlke
empowering ideas

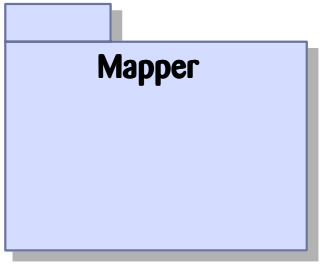
Layer

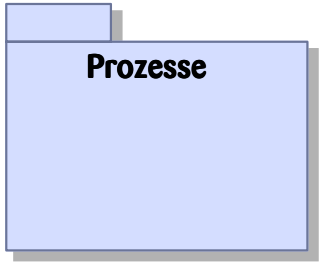
Präsentation



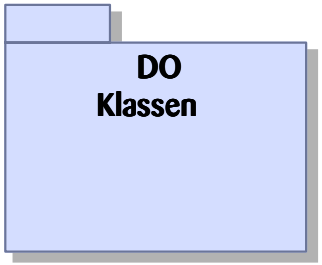
Business

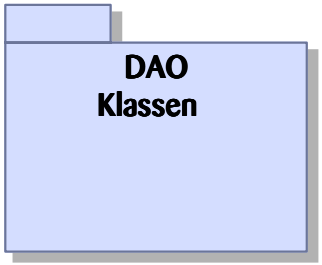






Persistenz



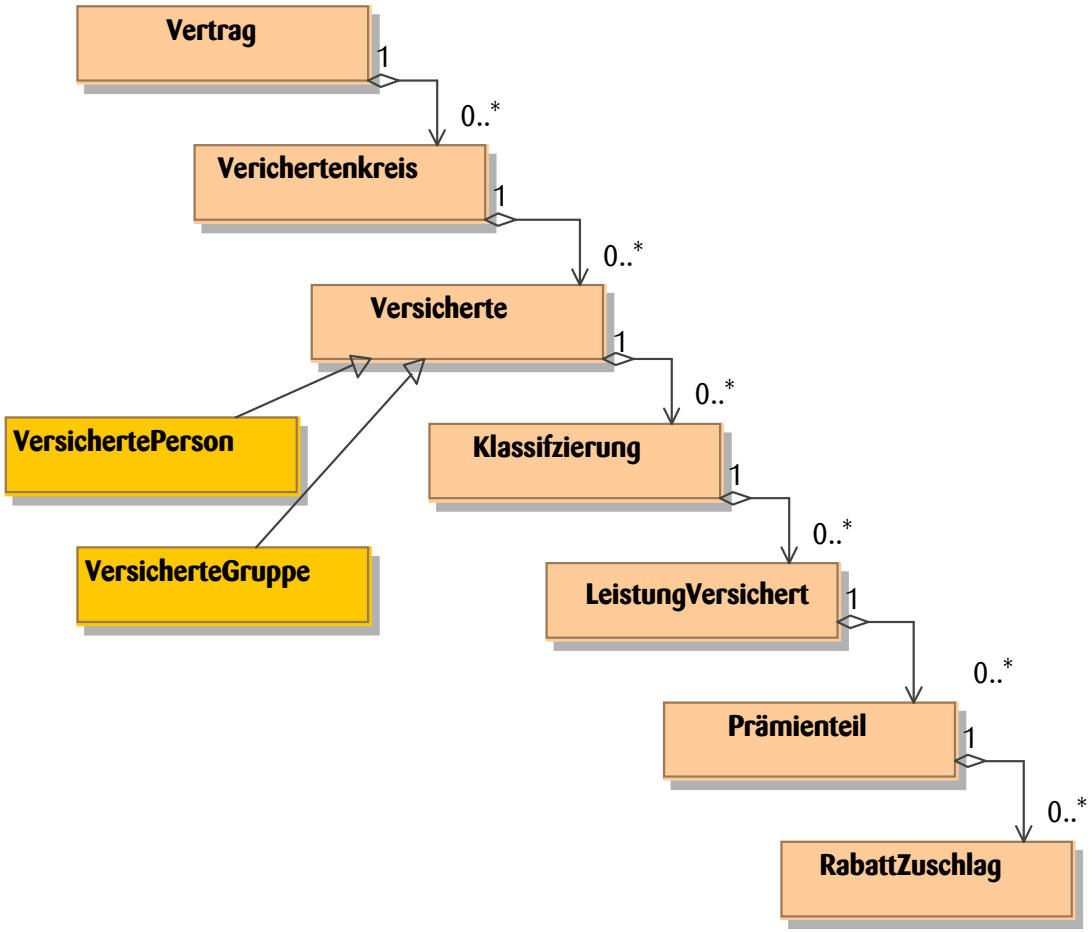


Frank Buchli

Business Model (BOs)

zühlke
empowering ideas

Versicherung Vertrag (vereinfacht)



Frank Buchli

Design Pattern
Composite

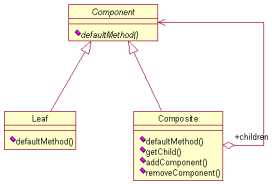
zühlke
empowering ideas

Problematik:

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

Lösung:

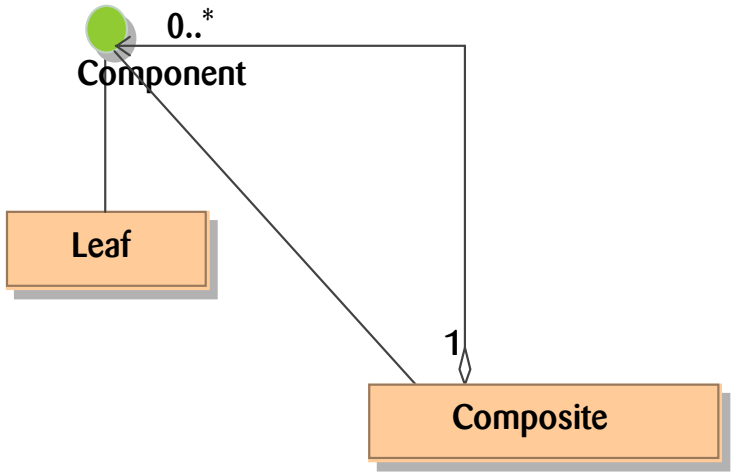
- Composite Pattern ermöglicht eine Gruppe von Objekten zu behandeln wie wenn es eines wäre.



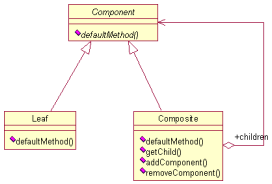
Frank Buchli

Design Pattern
Composite

zühlke
empowering ideas

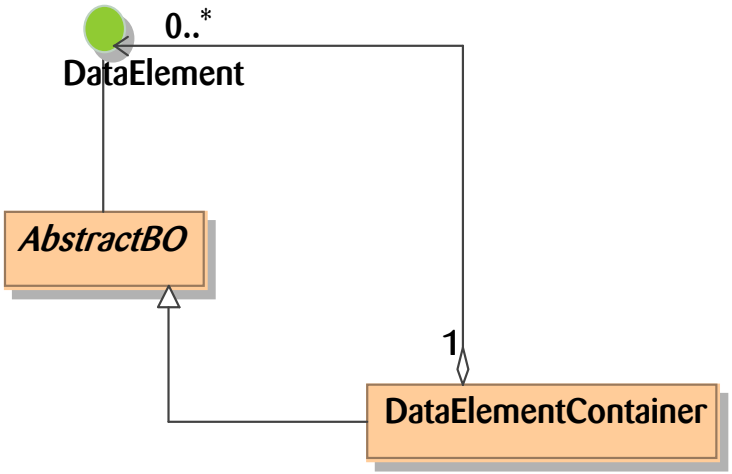


Übliche Bezeichnungen und Strukturen

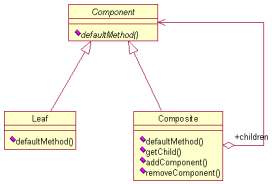


Frank Buchli

Design Pattern
Composite

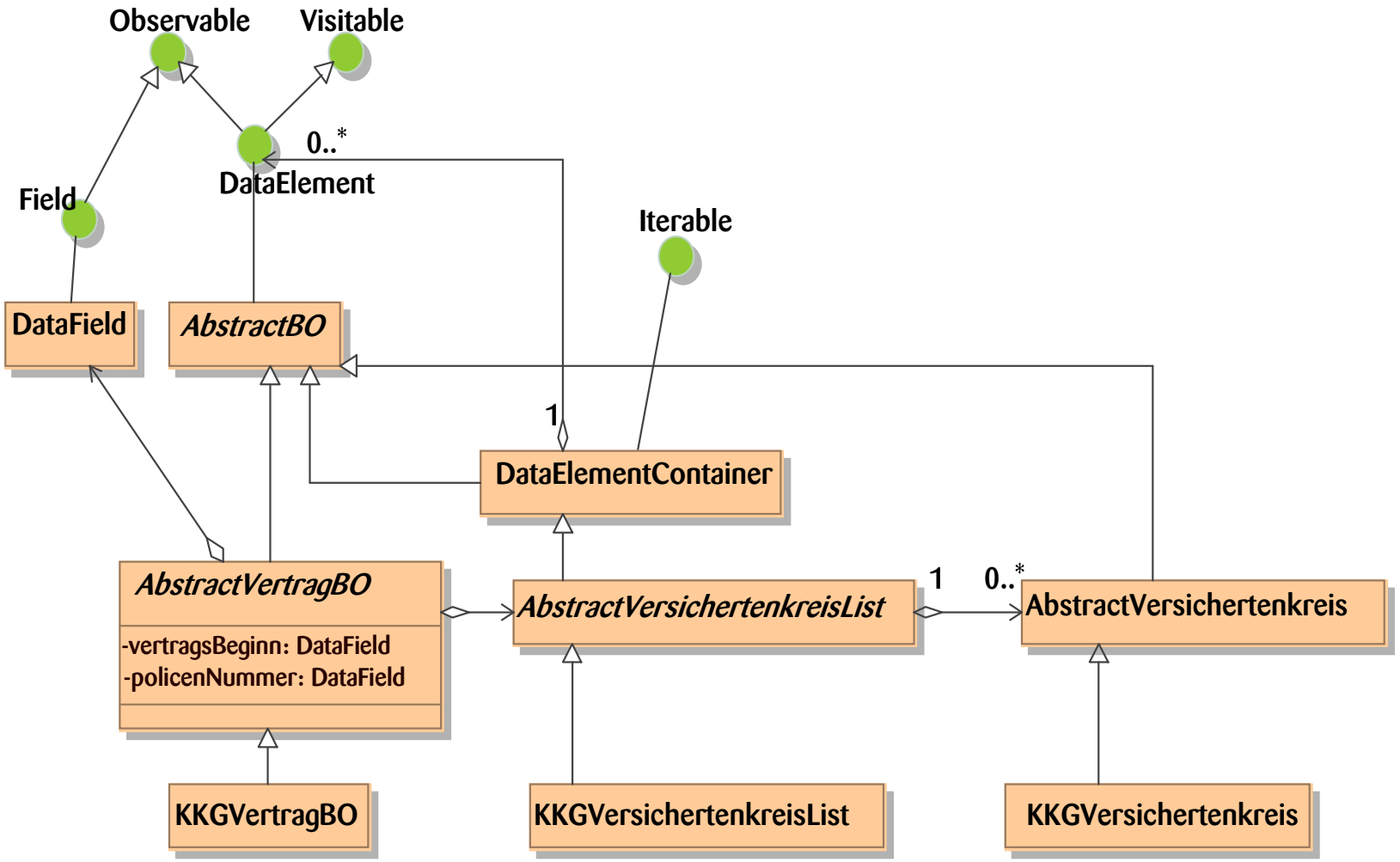


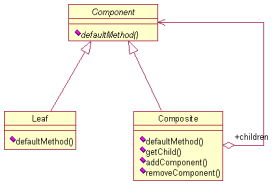
Bezeichnungen und Strukturen in unserem Projekt



Frank Buchli

Design Pattern Composite



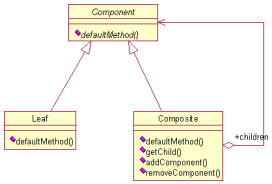


Frank Buchli

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();
  …
}
```



Frank Buchli

Design Pattern: Composite
Das Leaf

zühlke
empowering ideas

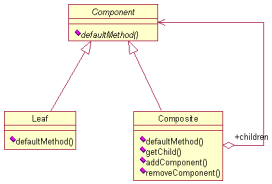
```
public abstract class AbstractBO implements DataElement {
  …
  private DataElement parent;
  …
  public DataElement getParent() { return parent; }
  public void setParent(DataElement aDataElement) { parent = aDataElement; }
  …
}
```

- 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 { … }
```



Frank Buchli

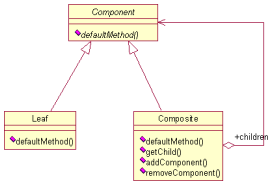
Design Pattern: Composite
Das Composite

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> { … }
```



Frank Buchli

Design Pattern
Factory

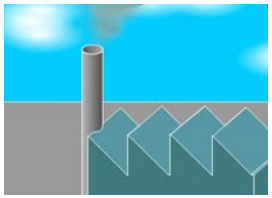
zühlke
empowering ideas

Problematik:

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

Lösung:

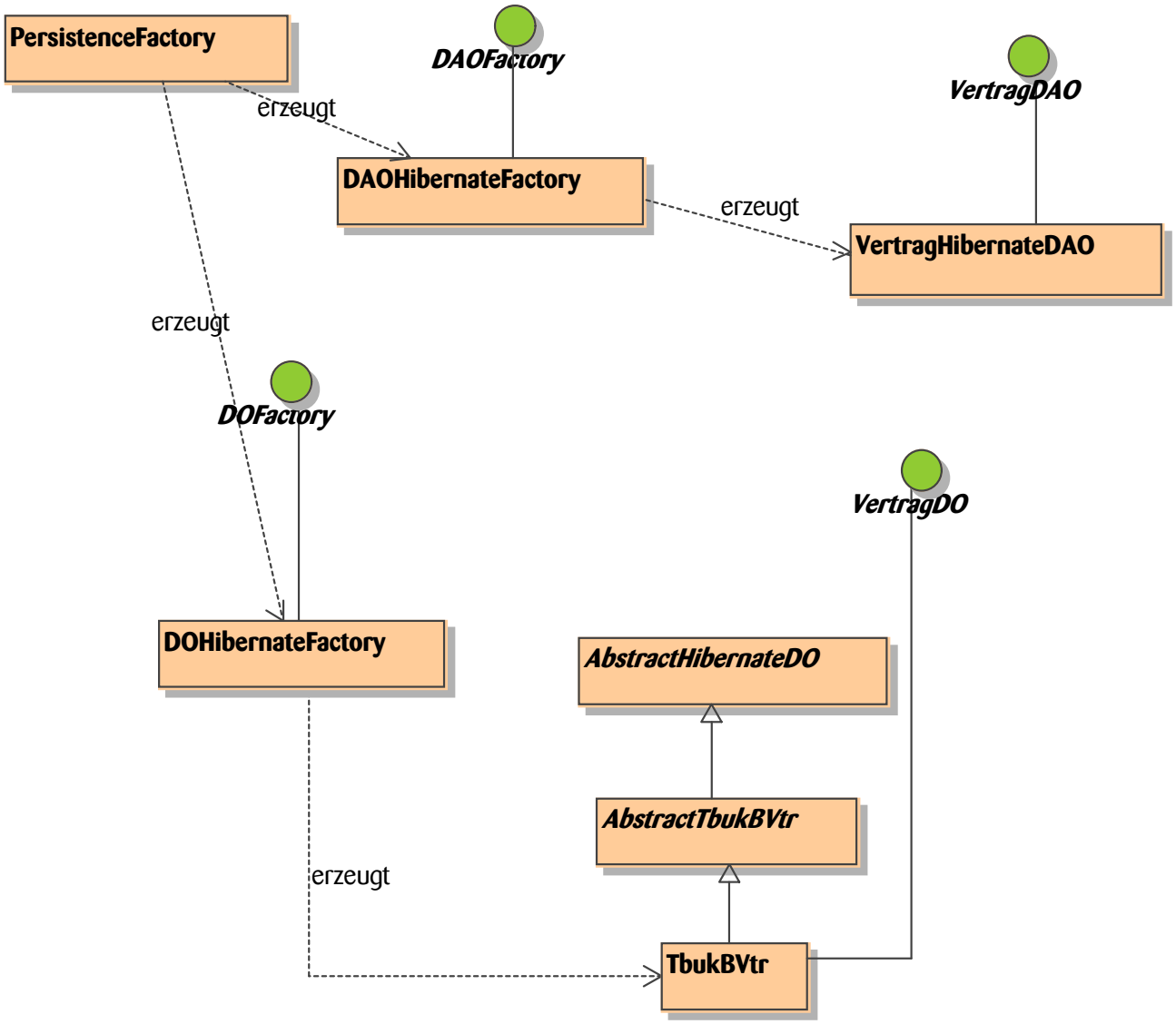
- „Oberes Layer“, das Business Layer, kommuniziert nur mit Interfaces.
- Es gibt eine **Persistenz-Factory** welche die konkreten Objekte des Persistenz-Layers (DAO & DO) kreiert.

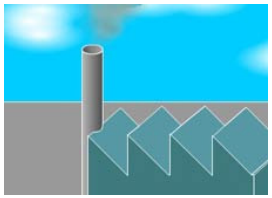


Frank Buchli

Design Patterns: Factory
UML

zühlke
empowering ideas





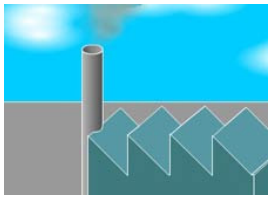
Frank Buchli

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();

}
```

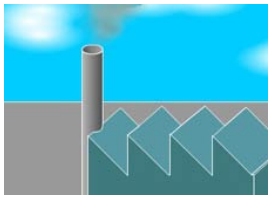


Frank Buchli

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();
    }

    ...

}
```



Frank Buchli

Design Pattern
Visitor

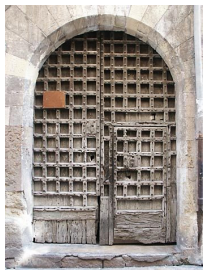
zühlke
empowering ideas

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.



Frank Buchli

Design Pattern: Visitor Interfaces

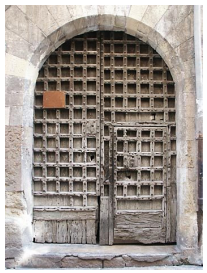
zühlke
empowering ideas

Visitor:

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

Visitable:

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



Frank Buchli

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);
        }
    }

    …

}
```

Frank Buchli

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;
  }

}
```

Frank Buchli

Design Pattern: Visitor
Anwendung

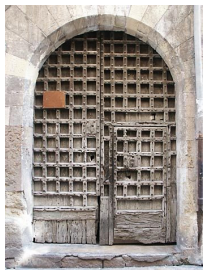
zühlke
empowering ideas

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)
```



Frank Buchli

Vielen Dank

zühlke
empowering ideas

Fragen?

… zum Projekt?

… zum „Leben als Informatiker“?

… zu Zühlke?

Frank Buchli