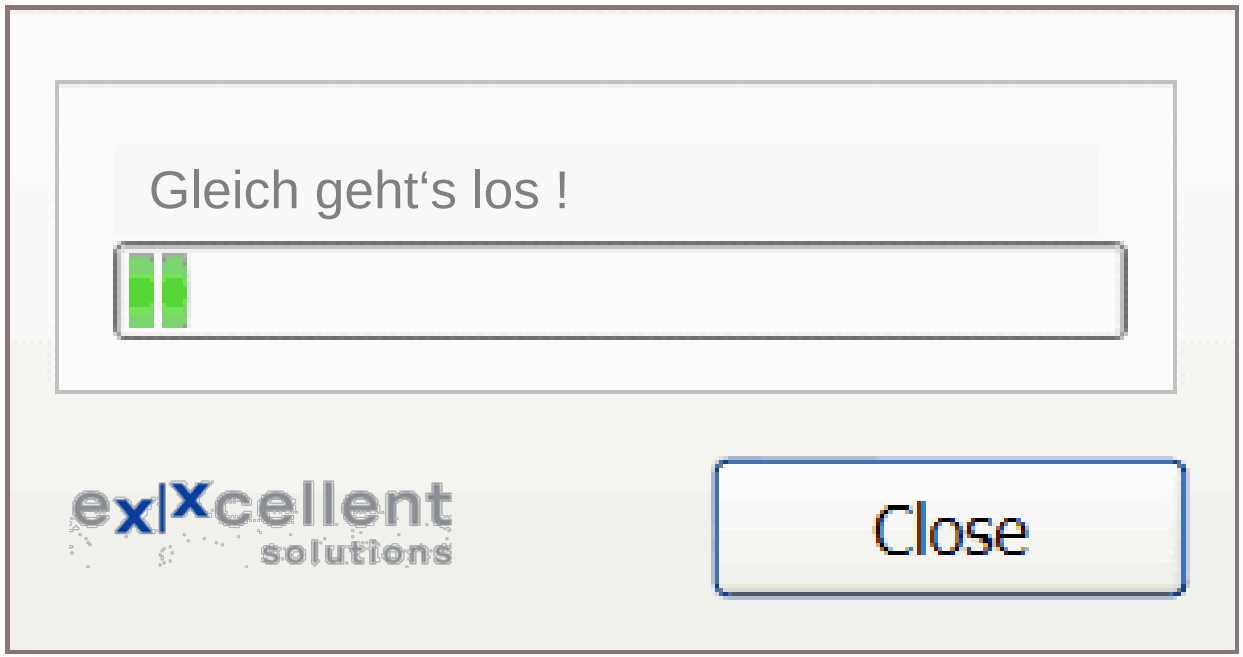
Herzlich Willkommen!



und

Auf dem Weg zu Unwartbarkeit

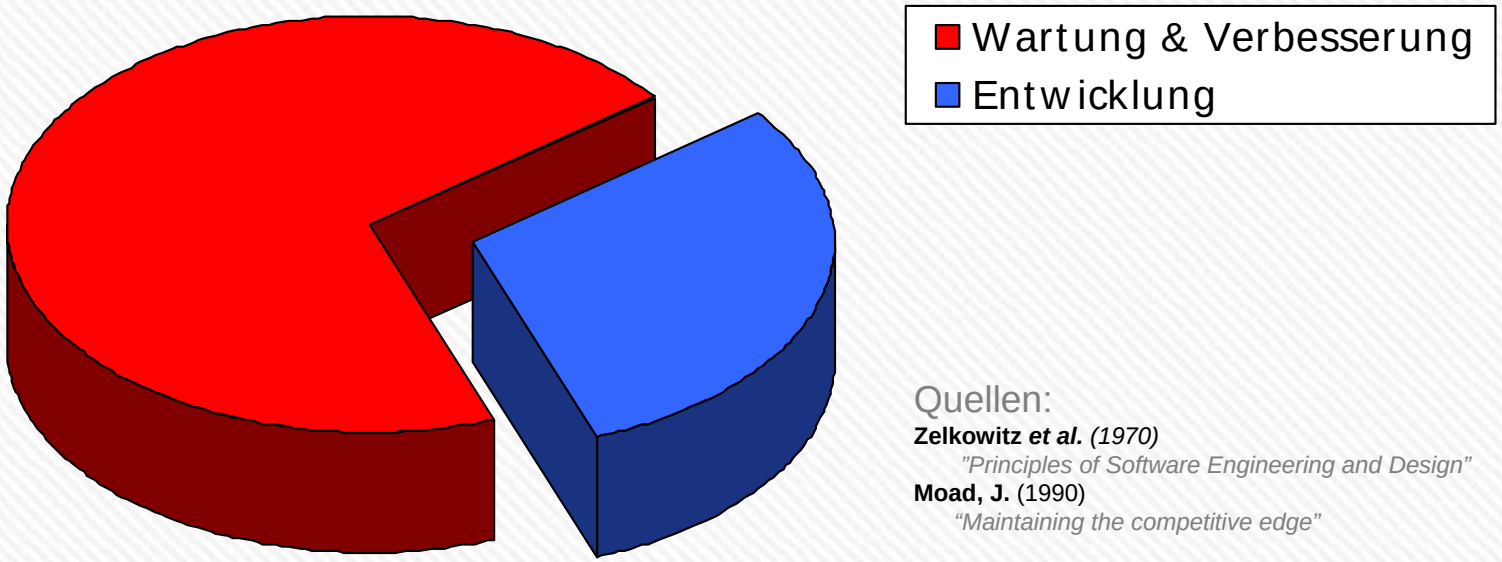
Die besten Strategien für den sicheren Erfolg!

Oliver Pehnke
Software Engineer

Benjamin Schmid
Software Architect

Anteil Wartungskosten in Softwareprojekten

~67% bis >90% der **Gesamtkosten**



Quellen:

Zelkowitz *et al.* *(1970)*
"Principles of Software Engineering and Design"

Moad, J. (1990)
"Maintaining the competitive edge"

- **Projektlaufzeit** überdauert oft Projekt-**Mitarbeiter**
- **Wartbarkeit** entscheidend für langfristigen Projekt**erfolg**

Die **Spielregeln** für Projekte

ex/xcellent solutions

1. Das Spiel kann anhand der **Prozess-Anleitung nicht gewonnen** werden – Improvisation erforderlich
2. Die **fixen Termine** und **Budgets** sind zu Beginn der Spielanleitung zu entnehmen
3. Nicht die Fachlichkeit, sondern **der Spielleiter entscheidet**
4. Wenn‘s kein Spass macht, machs‘d was falsch!

Strategie #1:

Visionen einbringen

Analyse- und Entscheidung

Gute Spezifikationen

Die Herausforderung:

- **vollständig** und **präzise**
- sind **nicht einfach** (auch graphisch)

Die Probleme:

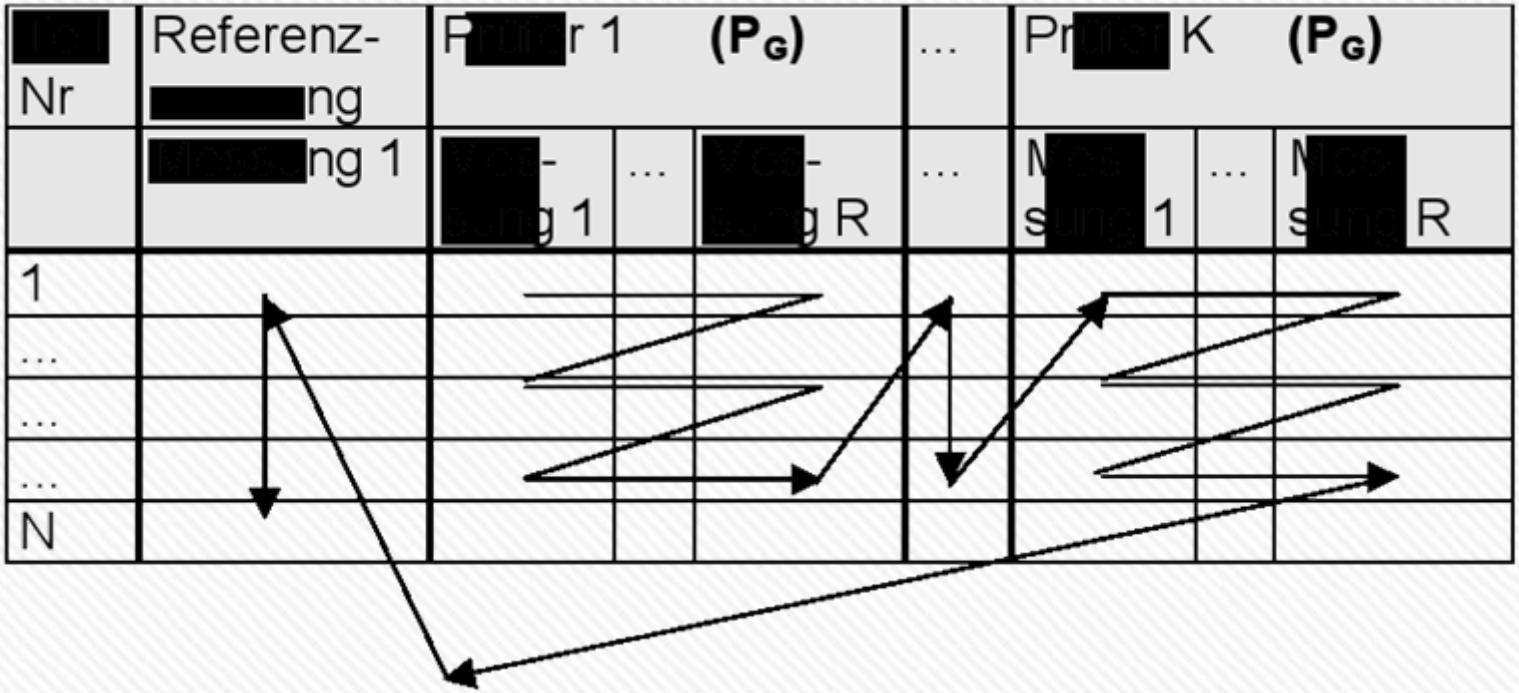
- **Kommunikation**
  - Kunde, Analyst, Entwickler haben eigene Vorstellungen
- **Konkretisierungen** nötig

Analyseergebnis ist die Basis von Allem

Klarheit & Eindeutigkeit reduzieren die Risiken

Graphische Spezifizikationen mal klar & eindeutig

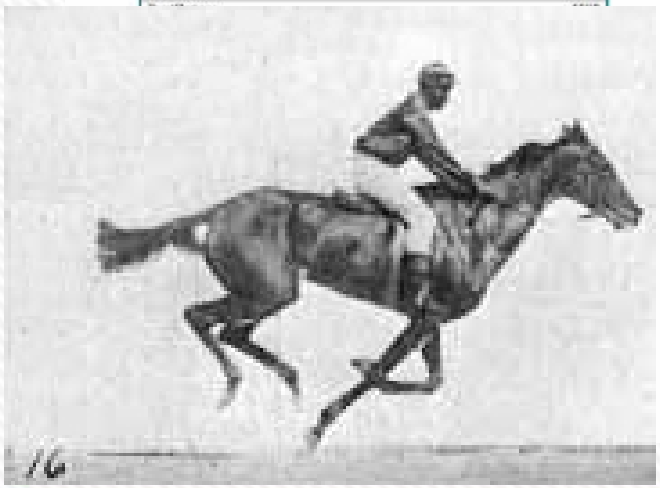
Attributive GC Studie



…und viel muß es sein

ex/xcellent solutions

ExControlAtom	4091
SlideListWithText	4080
AnimationInfoAtom	4081
InteractiveInfo	4082
InteractiveInfoAtom	4083
SlideList	4084
UserEditAtom	4085
CurrentUserAtom	4086
DateTimeMCAtom	4087
GenericDateMCAtom	4088
Header26CAtom	4089
Footer26CAtom	4090
ExMediaAtom	4100
ExVideo	4101
ExAviMovie	4102
ExMCMovie	4103
ExMIDIAudio	4109
ExCDAudio	4110
ExWAVAudioEmbedded	4111
ExWAVAudioLink	4112
ExOleObjStg	4113
ExCDAudioAtom	4114
ExWAVAudioEmbeddedAtom	4115
AnimationInfo	4116
RTFDateTimeMCAtom	4117
ProgTags	5000
ProgStringTag	5001
ProgBinaryTag	5002
BinaryTagData	5003



Appendix B: Miscellaneous Enumerated Types and Structures

```
#pragma pack(4)
//------------------------ Typer enumeration
-----------------------------------
//=========================================================================
-------

enum psrTypeCode                        // enumerates record types that
are saved
{
   PST_UNKNOWN                  = 0,   // should never occur in file
   PST_SubContainerCompleted    = 1,   // should never occur in file
   PST_IRRAtom                  = 2,   // Indexed Record Reference
   PST_PSS                      = 3,   // start of stream
   PST_SubContainerException    = 4,   // should never occur in file
   PST_ClientSignal1 = 6,              // should never occur in file
   PST_ClientSignal2 = 7,              // should never occur in file


   /* Application Saved State Information */
   PST_PowerPointStateInfoAtom = 16,

   /* Document & Slide */
   PST_Document             = 1000,
   PST_DocumentAtom         = 1001,
   PST_EndDocument          = 1002,
   // unused 1003
   PST_SlideBase            = 1004,
   PST_SlideBaseAtom        = 1005,
   PST_Slide                = 1006,
   PST_SlideAtom            = 1007,
   PST_Notes                = 1008,
   PST_NotesAtom            = 1009,
   PST_Environment          = 1010,
//   PST_SSlook               = 1011,
   PST_Scheme               = 1012,
   PST_SchemeAtom           = 1013,
   PST_DocViewInfo          = 1014,
   PST_SSlideLayoutAtom     = 1015,
   PST_MainMaster           = 1016,
   PST_SSSlideInfoAtom      = 1017,
   PST_SlideViewInfo        = 1018,
   PST_GuideAtom            = 1019,
   PST_ViewInfo             = 1020,
   PST_ViewInfoAtom         = 1021,
   PST_SlideViewInfoAtom    = 1022,
   PST_VBAInfo              = 1023,
   PST_VBAInfoAtom          = 1024,
   PST_SSDocInfoAtom        = 1025,
   PST_Summary              = 1026,
//   PST_Texture              = 1027,
   PST_VBASlideInfo         = 1028,
   PST_VBASlideInfoAtom     = 1029,
```

Strategie #2:

Die Wahl der richtigen Waffen

Technologien und Frameworks

Einbinden von Hoffnungen

- **SOA**
  „Wieder verwendbarer Baustein“
- **JUnit**
  „Applikation ist getestet“
- **Groovy**
  „Wir sind agil & flexibel“
- **MochiKit**
  „Es ist Web 2.0 und komfortabel“
- **CompanyFramework.jar**
  „Manfestierter Unternehmensstyle & -qualität“

XML ready!

ex/xcellent solutions

```
<?xml version="1.0" encoding="utf-8" ?>
<RealXML>
  <import tag="testdata" type="data" mode="update" />
- <fields>
    <field name="name" type="Char" target="person.data" />
    <field name="id" type="Char" target="person.id" />
    <field name="startyearyear" type="Char" target="person.start" />
    <field name="sex" type="Char" target="person.sex" />
    <field name="lastmodified" type="Char" target="person.lm" />
  </fields>
- <csvdata headers="false">
    <![CDATA[ "Fischer Tannenbaum",1,2003,"M",2006-04-23
    "Tannembaum Melanie",2,2001,"F",2002-01-03
    "Fritz Tannebaum",3,2007,"M",2007-03-02
    "Hans Buche",2,2001,"M",2002-05-01
    "Förster Tanja",2,2001,"F",2002-05-01
```

Frameworks: Den Rahmen finden

Gute Wahl:

- effizienter
- fehlerfreier
- wartbarer

Ungünstige Wahl:

- Einschränkungen
- Komplexität
- Technik-Selbstzweck

Agilitäts-Verlust › Effizienz-Gewinn ?

Strategien für den steinigen Weg

- Wahllos Hinzu
- Blind *„den Standard“*
- `StringUtils` neu erfinden

Strategie #3:

Die Rekrutierung

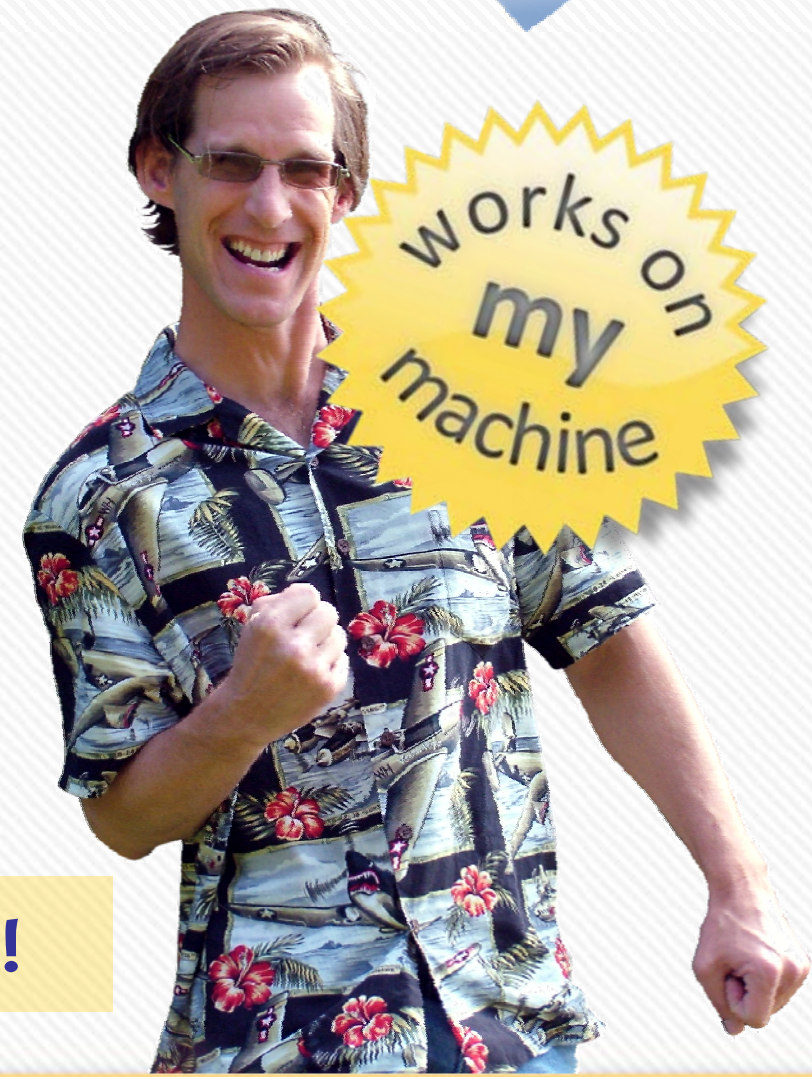
Team und Zusammenarbeit

ex/xcellent solutions

WOMM Certification

...in 4 easy steps

1. **Compile**
   (CVS/SVN update purely optional!)
2. **Launch!**
3. **Test** you code path
   (Unless code change is less than 5 lines or you are a **real** professional)
4. **Check-in!**



Welcome as WOMM solutionist!

Strategie #4:

Die hohe Kunst der Verschleierung

Namen und Code-Formatierung

Marry's Metamorphose

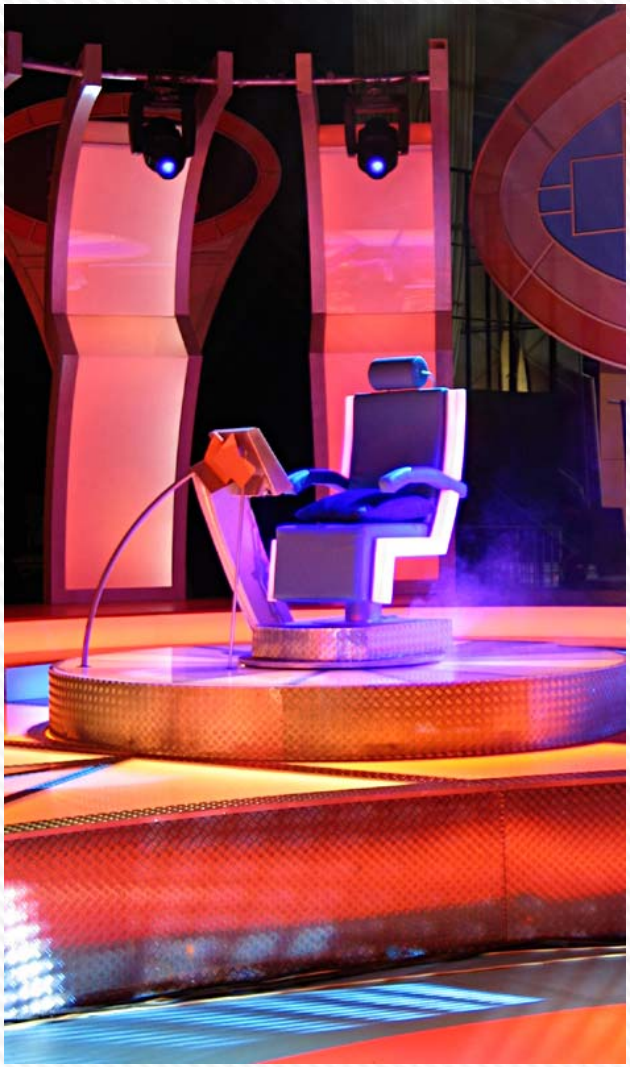
```
1  public class Util {
2      private double marypoppins = (9 / 5d);
3      private double starship = 1 / marypoppins;
4
5
6      public double toDaniel(double batman) {
7          return 32 +  batman * marypoppins;
8      }
9
10
11     public double toAnders(double v) {
12         return starship * (v - 32) ;
13     }
14 }
```

Falsche Fährten legen


```
 8        // "Syntax"-Optimierung
 9        int y = 3;
10        if ( a )
11           if ( b ) x=y;
12        else x=z;   // wann?
13
14        // Speicherplatz-Opt.
15        y = x * 2;
```

```
3   /* The BoolValue is a enumeration wrapper for boolean values. */
4    public enum BoolValue {
5        TRUE, FALSE, FILE_NOT_FOUND
6    }
```

Nun aber zur Preisfrage



```
 8        // "Syntax"-Optimierung
 9        int y = 3;
10        if ( a )
11
12
13
14
15
```

```
22        final int y = 3;
23        if (a) {
24            if (b) {
25                x = y;
26            } else {
27                x = z;
28            }
29        }
30        int dbl = 2 * x;
```

Wenn a und b false sind, dann ist x:

a) 42!

b) z

c) y

d) x

ex/xcellent solutions

Compiler übersetzen – Entwickler lesen!

- Lesbare Methoden, Klassen und Attributnamen
  - die **tun** was sie **sagen**
  - **konstante** Begrifflichkeiten
  - **griffige Namen**
- Namen und Bedeutung synchron halten
  - Namen sind **Orientierung** im Code und Design
- Java Coding Conventions

Marry's Metamorphose

```
1   public class TemperatureConverter {
2       private double FAHRENHEIT_FACTOR = (9 / 5d);
3       private double CELSIUS_FACTOR = 1 / FAHRENHEIT_FACTOR;
4
5       // Daniel Gabriel Fahrenheit Temperatur scale
6       public double toFahrenheit(double celsiusTemp) {
7           return 32 +  celsiusTemp * FAHRENHEIT_FACTOR;
8       }
9
10      // Anders Celsius Temperature Scale
11      public double toCelsius(double fahrenheitTemp) {
12          return CELSIUS_FACTOR * (fahrenheitTemp - 32) ;
13      }
14  }
```

Strategie #5:

The swiss-army knife of Java: Vererbung

OO-Konzepte

Professionelle Polymorphie in Java

```
20  public abstract class AbstractNavigationWizardStep implements NavigationStep {
21
```

```
10  public class NavigationWizardStep extends AbstractNavigationWizardStep {
```

```
59  public abstract class AbstractDokumentActionStep extends NavigationWizardStep
```

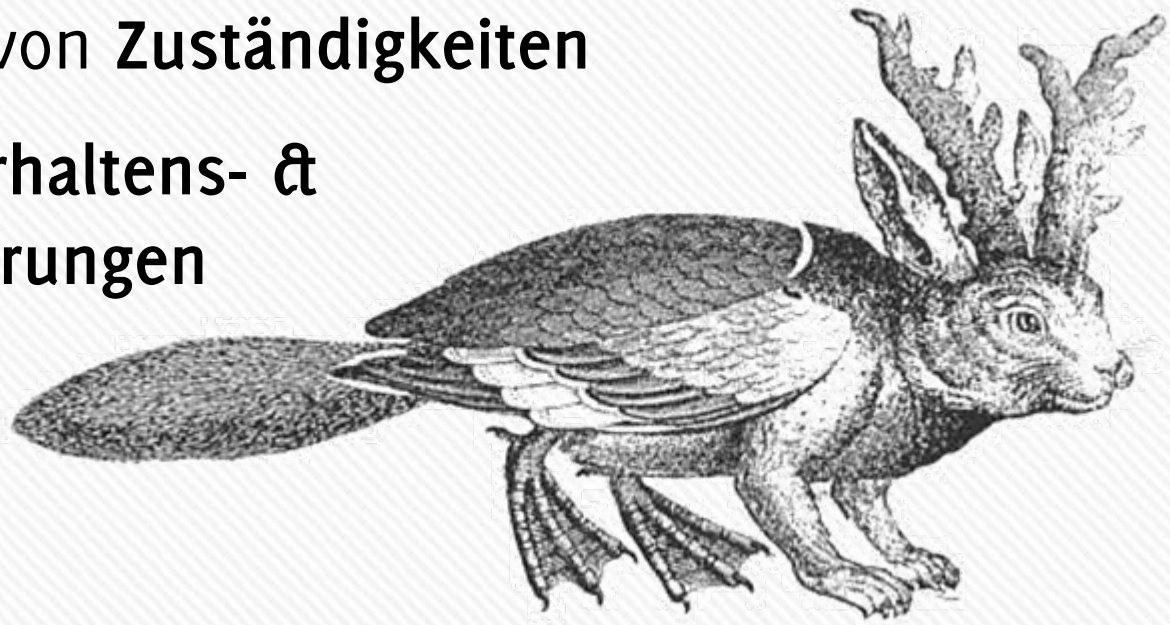
```
31  */
32  public class DokumentMaintainStep extends AbstractDokumentActionStep {
33
```

```
12  public class DokumentFaxStep extends DokumentMaintainStep {
13
14      public DokumentFaxStep() {
15          super(false, false, true, false);
16      }
17
18      public void setVisible(boolean visible) {
19          super.setVisible(true);
20          super.setVisible(false);
21          super.setVisible(visible);
22      }
```

Was bewirkt `setVisible()` eigentlich?

Extensive Nutzung von `extends` bewirkt

- **Hohe Komplexität** durch **verteilte** Logik innerhalb der Hierarchie
- Starke, nicht-lösbare **Abhängigkeiten**
- **Vermischung** von **Zuständigkeiten**
- **Versteckte** **Verhaltens- & Semantikänderungen**



ex/xcellent solutions

Verändern bzw. Hinzufügen von Verhalten in der OO

Vererbung:

```
public class DoMoreSet1<E> extends HashSet<E> {

    public DoMoreSet1(Set<E> elements) {
        addAll(elements);
    }

    public boolean add(final E element) {
        doMore(element);
        return super.add(element);
    }
```

Delegation bzw. Event/Listener:

```
public class DoMoreSet2<E> implements Set<E> {
    private final Set<E> delegee;

    public DoMoreSet2(final Set<E> delegee) {
        this.delegee = delegee;
    }

    public boolean add(final E element) {
        doMore(element);
        return this.delegee.add(element);
    }
```

A **ist** ein spezielles B

Enge, integrative Bindung

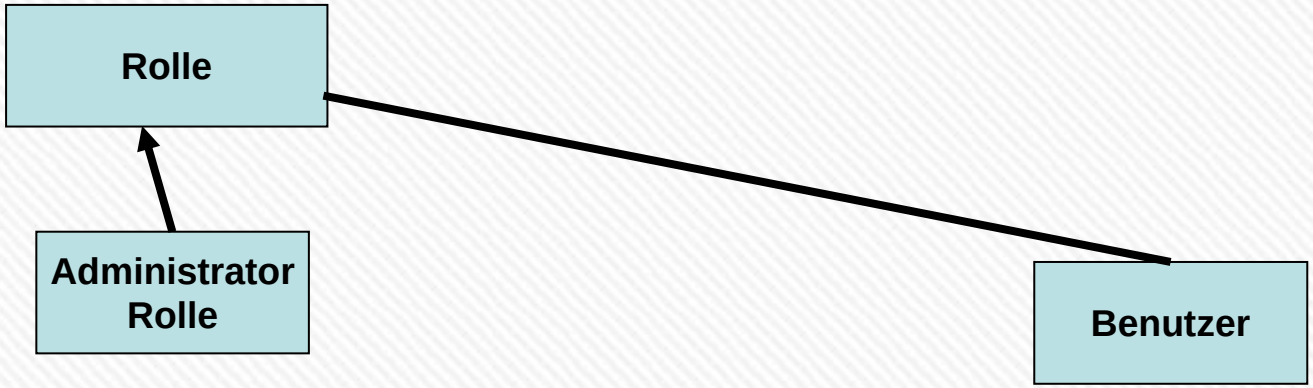
A **hat/nutzt/verwendet** B

Entkoppelt, einzeln wieder verwendbar

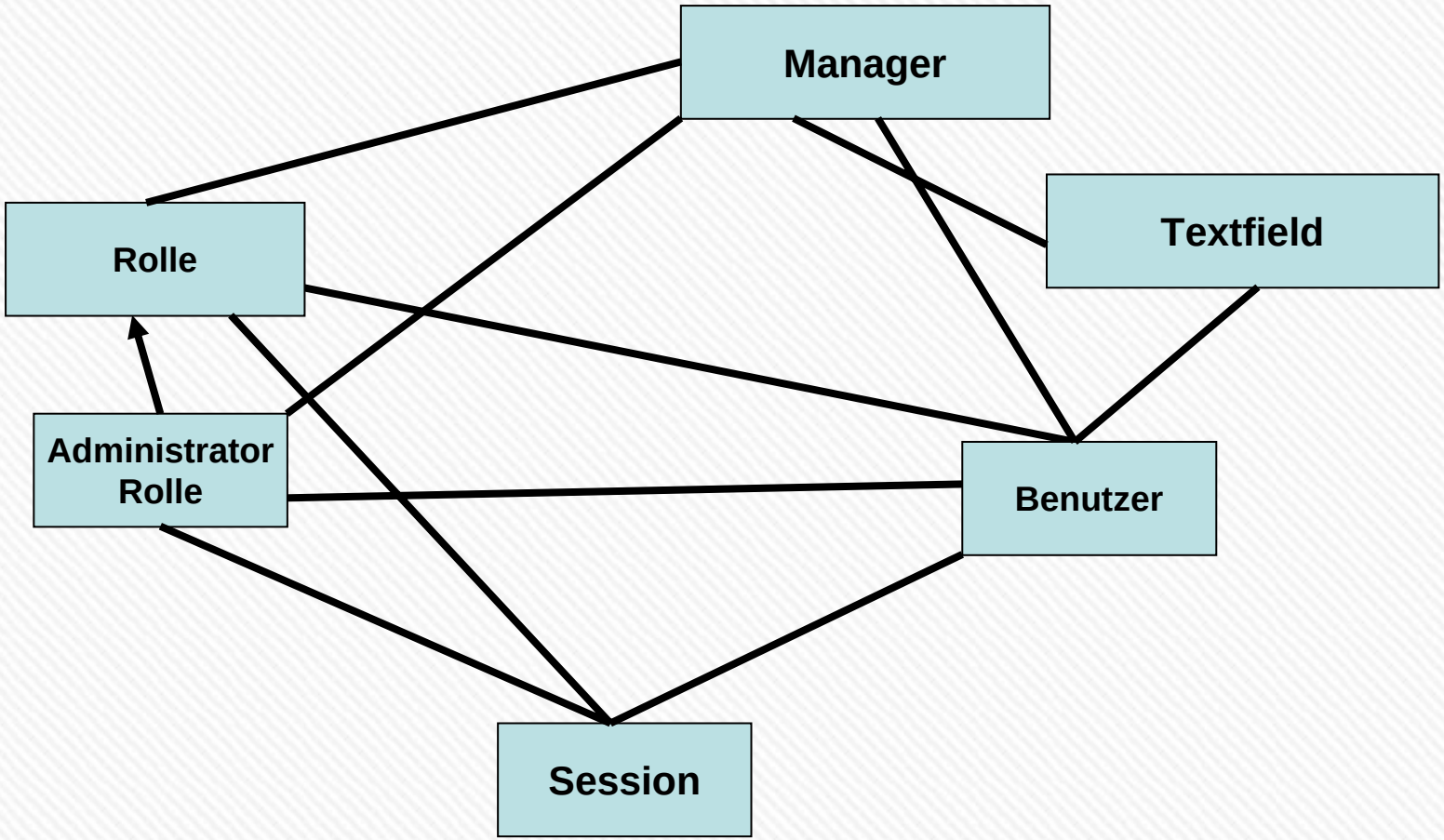
Strategie #6:
Gordische Seilkünste

Module und Komponenten

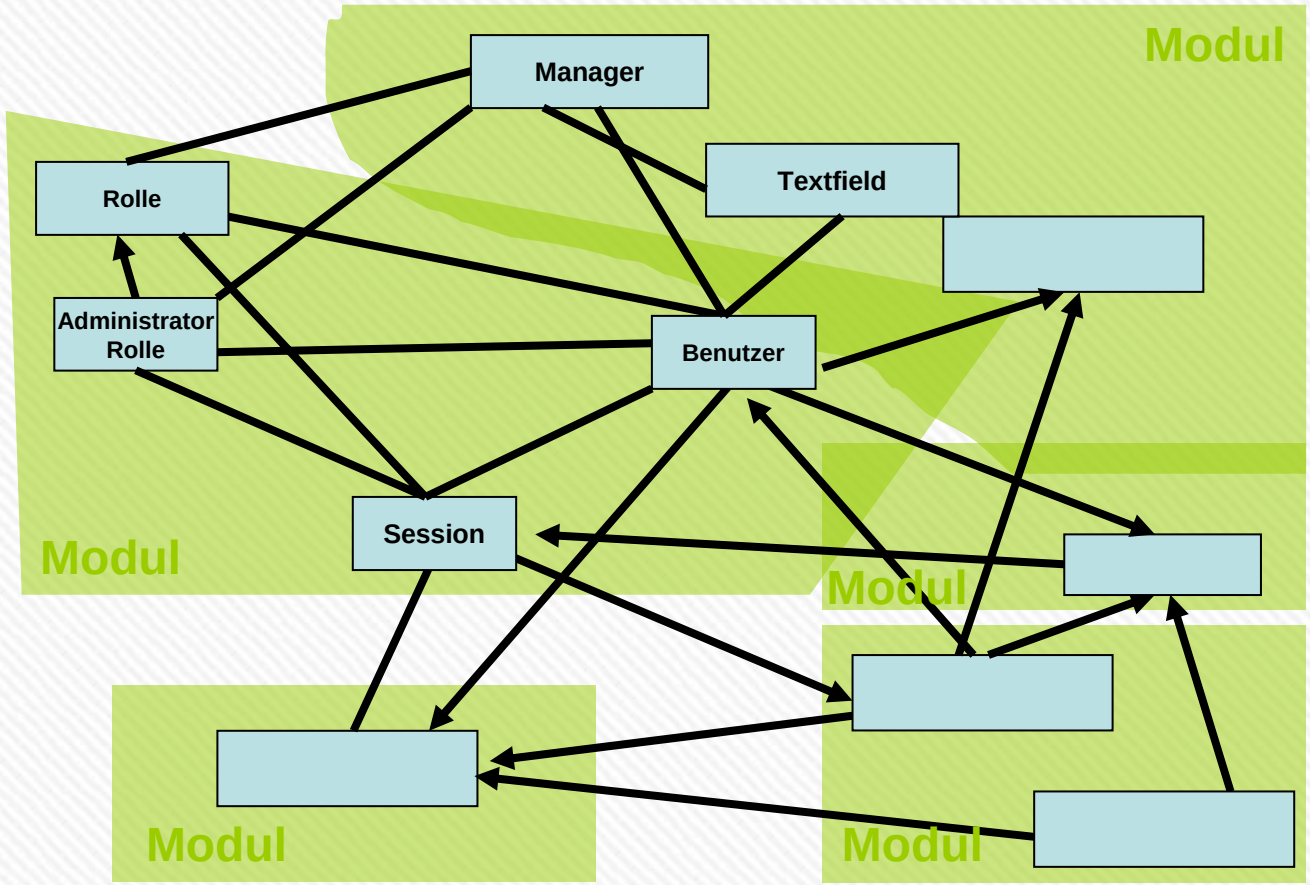
Intelligent Design...



Intelligent Design...

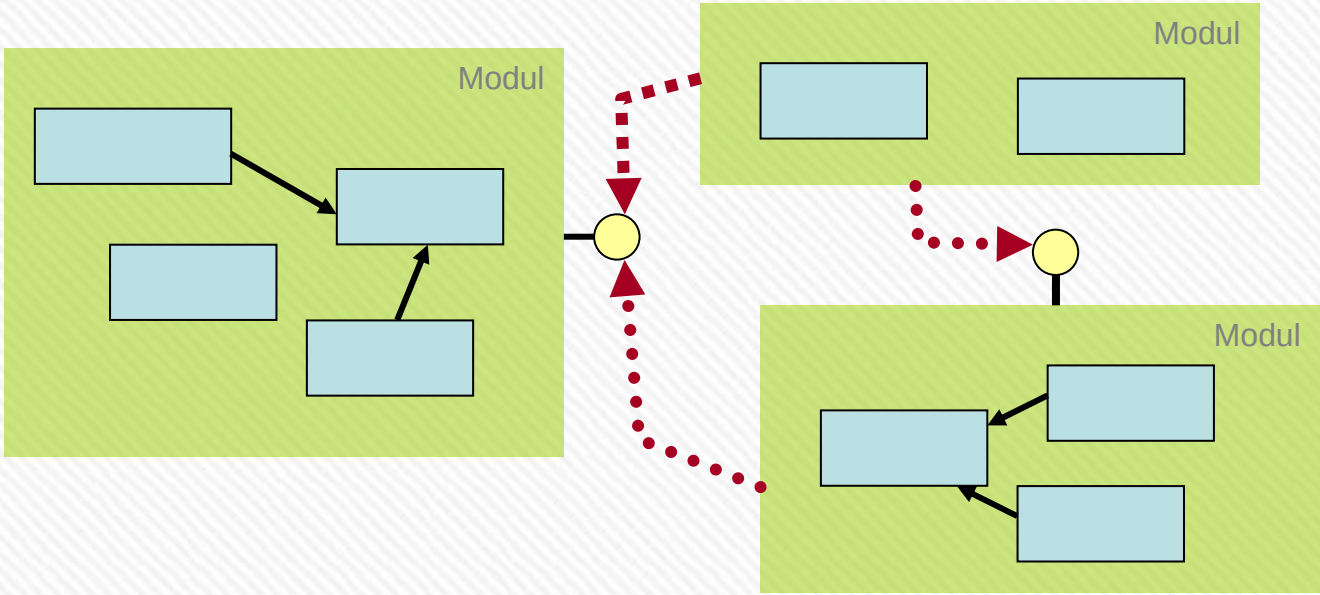


...und Grenzen anerkennen



Unberührte Natur

"A good modular, componentized design means minimizing the knowledge and dependency one part of the system has to have about another."



Was ist eine **gute Komponentaufteilung?**

Wie sehen **passenden Schnittstellen** aus?

Wegweiser für gute Modul-Einteilungen / Dekomposition

- **Fachliche** Ordnung
  („*Fax-Service*“, „*Kontoverwaltung*“)
- **Technische** Schichten
  (GUI-, Business-Layer)
- **Wenige**, unidirektionale **Abhängigkeiten**
- Klares Gedankenmodell bzgl. Frage:
  Was gehört rein, was nicht?
- Erfahrung

Strategie #6½:

Gordische Seilkünste II – Das Schwert

Schnittstellen

Nach AOP: **VOP**


```
1  import java.util.Vector;
2
3  public interface ManagerInterface {
4      /**
5       * Returns an Order vector or a vector of Order vectors, based on
6       * one or more parameter values. If there was an error retreiving
7       * the Order, then a standard Error vector is returned. (see docs)
8       *
9       * @param getOrSet       indicates what action to take.
10      *                       can be: "get", "set", "close", "find"
11      * @param orderNumber    number of the order
12      * @param orderId        id of the order
13      * @param customerNumber number of the customer
14      * @return vector of 18 elements or a vector of vectors or an error vector
15      */
16     Vector getSetOrder(String getOrSet, String orderNumber,
17                        String orderId, String customerNumber);
18 }
```

Trennungsschmerzen

Schnittstellen sind Rollen mit austauschbarer Besetzung!

Schnittstellen **einfach** halten!

- Komplexität gehört in die Implementierung
- **“If you’re in doubt – leave it out !”**
- Bei Klassen: `private`, etc. nutzen
- Neue Rollen erkennen und via Refactoring frühzeitig abbilden

Veröffentliche Schnittstellen sind Vertragsbestandteil einer Klasse und nachträglich schwer zu ändern

Strategie #7:
Mute the fire-alarm

Fehlerbehandlung

Nur nicht aufgeben

Nicht-deterministischer Rechenapparat

```
437     public Collection getAllGroupUsers(String groupName) throws RemoteException {
438         Exception e = null;
439         for (int i= 0; i < 3; i ++) {
440             try {
441                 return getRemoteObj().getAllGroupUsers(groupName);
442             } catch (Exception ex) {
443                 e = ex;
444             }
445         }
446         if (e != null)
447             throw new RemoteException("Error");
```

Lautlose Fehlerkorrektur

```
265         try {
266             return getHostName(url);
267         } catch (UnknownHostException e) {
268             return "www.error.com";
269         } catch (Exception ex) {
270             System.out.println("GET_SERVER_NAME");
271             ;  //To change body of catch statement use Options | File Templates.
272             System.exit(0);
273         }
```

Error-Handling: War was?

- Fehler immer **individuell, vollständig** und **korrekt behandeln** *oder* …
- … **einpacken** und **weitergeben**
  - In geeignete, deklarierte Exceptions
  - oder komfortable RuntimeException

Dass bedeutet

- Keine Details/Fehler verschlucken
- auch wenn **`throws`** nervt (lieber: RuntimeException)
- Fehlerhandling Alternativen, z.B. **`try {…} finally {…}`**

Diagnostikmöglichkeiten einbauen

- Logs sind der Schlüssel zur Laufzeitdynamik
- Nicht nur Details - auch **Eckpunkte dokumentieren**
  „Versuche Fax ID:4711 an Empfänger Y zu senden“
- Kein Sonderzeichen-Krieg; Log-**Levels/Kategorien**

Strategie #8:

Originell programmieren

Java Basics

Indirektion: Schlüssel zur OO

```
public class IntWrapper {
    private String m_sValue;

    public IntWrapper(int m_sValue) {
        this.m_sValue = Integer.toString(m_sValue);
    }

    public String getM_sValue() {
```

Masse gewinnen

```
     * Mit dieser abstrakten Methode kann der Focus beim Öffnen einem
     * bestimmten Kontrollelement zugeordnet werden.
     *
     * @return Diese Komponente erhlt nach dem fnnen des Dialogs den Focus
     */
    public abstract Component getComponentWhichGetsFocusUponDialogOpening();
```

```
public MultiSelectAttributeEnumComboBoxFormularSearchComponent
        (JComboBox cb, UiRepresenation uiRepresentation,
         boolean isNullable, ListCellRenderer listCellRenderer,
         TableCellRenderer tableCellRenderer) {
    super(null, null, isNullable);
```

Initialisierung mal anders

```
273         String tmpString = param + ",,,,,,,,,,";
274         String[] args = tmpString.split(",");
```

„richtig“ Null

```
168     static public final boolean isNull(double _value)
169     {
170       return (Math.abs(_value)  < 0.00001);
171
172     }
```

```
163     static public final boolean isVeryNull(double _value)
164     {
165       return (Math.abs(_value)  < 0.0000000001);
166     }
```

Instanzen zählen...

```
1   /**
2    * Ist selbsterklärend denk ich...
3    */
4   public class SimpleTimeMeasure {
5       private static long startVal = 0;
6       private static long lastMeasureVal = 0;
7       private static String measureName = "";
8
9       public static void start(String measureName2) {
10          startVal = System.currentTimeMillis();
11          measureName = measureName2;
12      }
13
14      public static synchronized void measure(String comment
15          long durationSinceLast = 0;
16          long actVal = System.currentTimeMillis();
17          long duration = actVal - startVal;
18
19          String output =  "#TIME_MEASURE: " + measureName +
20                  + comment + " # Complete Duration: " + dur
```

static Variablen/Mutables

- **Pitfall!**
  =
  Session.getSprache().berechne();

```
public static HashSet aktuelleZahlen
```

- Good style:
  „buh“;

```
private final static String FOO =
```

(Immutable und Konstante)

synchronized

- In der der Regel: **Vermeiden!**
  Besser: java.util.concurrent verwenden
- Wenn doch, erst mal Gegenprobe: `volatile` bekannt?

Strategie #9:
Qualität ‚fühlen' lernen – Testen vermeiden

Testen und Messen

Testen? Wir sind WOMM-zertifiziert!

**“I don't need to test my programs.
I use error-correcting RAM (ECC)”**

Gute Tests zu schreiben ist anspruchsvoll

- Nicht nur Pfade ablaufen, **Ergebnisse prüfen**
- **Nebenpfade** und **Randbereiche** bedenken
- Automatisierbar und **Regressionsfähig** halten
- Unbedarft gegen den **Vertrag der API testen**

Fehlern vorbeugen

- Konstruktive Mittel (‚Fehler durch Design verhindern‘)
- Code Robustheit (Fail gracefully)
- Diagnostik (‚schnelle Fehlerlokalisierung‘)

Strategie #10:

Die richtige Autobiographie

Dokumentation

…ist **selbsterklärend** – denk ich

Was dokumentieren

- **Code** (API zu 100%)
- **Design** (nach Entwurf)
- **Commits** (Nachvollziehbarkeit)

Wie dokumentieren

- Nicht was, sondern die **Hintergründe (Warum, Wie)**
- **Sagen** was **der Code nicht sagt**
- JavaDoc ausnutzen
  - **Zusammenhänge** mit `@link, @see`
  - **Parameter** (Nullable, Defaults) und **Rückgabewerte** (Typ)

ex/Xcellent solutions

Oliver Pehnke
Software Engineer

… eine Frage hätte ich da aber jetzt noch …

Benjamin Schmid
Software Architect

Vielen Dank für Ihre Aufmerksamkeit!

Besuchen Sie uns!
… Zertifizierung bei uns am Stand!

ex|Xcellent solutions

http://www.exxcellent.de

Quellen und Links zum Thema

The Daily WTF	http://worsethanfailure.com/
Tips for maintainable Java code	http://www.squarebox.co.uk/download/javatips.html
It Works on My Machine	http://jcooney.net/archive/2007/02/01/42999.aspx
Unmaintainable code	http://mindprod.com/jgloss/jgloss.html