Softwaretechnik

Kapitel 2

1. Wieso Software Engineering?
2. **Vom Einzelkämpfer zum Großprojekt**

Systematische Softwareentwicklung

3. Anforderungen und Test: *Die Basis des Projekts*
4. Entwurf: *Strukturen und nicht-funktionale Eigenschaften*
5. Entwürfe notieren mit UML: *Modelle im SE*
6. Design Patterns: *Entwurfserfahrungen nutzen*
7. Management: *Technik und Projektmanagement*

Softwaretechnik

Inhalt von Kapitel 2

1. Wieso Software Engineering?
2. **Vom Einzelkämpfer zum Großprojekt**

- ***Programmhygiene***
- ***Egoless programming***
- ***Programme für Menschen schreiben***
- ***Codekonventionen***
- ***Dokumentation***

Situation Einzelkämpfer

Allein
Keiner bezahlt
Keiner beteiligt sich
Selbst die Idee gehabt
Wird nicht weiterentwickelt

Muss keinem Kunden gefallen
Muss nicht zusammenpassen
Muss nicht verständlich sein
Muss man keinem zeigen
Muss nicht gelingen

Programmieren als Hobby
Auch schön. Aber nicht unser Thema.

Stellen Sie sich vor …

- **Sie haben eine neue Stelle**
- **Sie sollen Code entwickeln und pflegen**

- **Zu Anfang bekommen Sie das Programm eines Kollegen**
- **„Er hat sehr auf Speichereffizienz geachtet“, heißt es**
- **Er ist leider versetzt worden**
- **Nicht mehr verfügbar**

- **Viel Spaß …**

Was tut dieses Programmstück?

Erst verstehen, dann verbessern

```
public ProgObject DenAlgorythmus_Durchführenfür€€€(){ProgObject x, y;OtherClass
OtherObject;while (tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.
size() > 1) {x = tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.
    removeFirst();
y = tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.removeFirst();
OtherObject = new OtherClass();// subtract x and y values
OtherObject.getIntVar_toHaveANewValue(x.intVar() + y.intVar());OtherObject.setOne(x);
OtherObject.setZero(y);tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.
addFirst(OtherObject);
Collections.sort(tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers,
new myComparator());
);return
tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.getFirst();
)public class myComparator implements Comparator {public
    int compare(Object arg0, Object arg1) {return (((ProgObject
) arg0).intVar()        -           ((ProgObject) arg1).intVar());
 }
 }
```

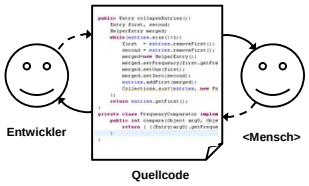
Wieso verstehen Sie das nicht?
Das Programm funktioniert doch!

Quellcode dient der Kommunikation

- **Syntax korrekt** → **der Compiler versteht**
- **Semantik korrekt** → **das Programm tut, was es soll**

- **Dennoch kann es unverständlich und unwartbar sein!**
- **Dann ist es in der Praxis nutzlos**

- **Fazit**
  - **Quellcode dient der Kommunikation mit dem Compiler**
  - **vor allem aber mit anderen Menschen**



Erster Versuch: Bessere Formatierung

- **Schon besser.**
- **Bezeichner sind nicht hilfreich, im Gegenteil**
  - **überlang, kryptisch, mit Sonderzeichen**
  - **bezeichnen nichts, sondern verwirren**

```
public ProgObject DenAlgorythmus_Durchführenfür€€€() {
    ProgObject x, y;
    OtherClass_OtherObject;
    while (tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.
        x = tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.
        y = tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.
        OtherObject = new OtherClass();
        // subtract x and y values
        OtherObject.getIntVar_toHaveANewValue(x.intVar() + y.intVar());
        OtherObject.setOne(x);
        OtherObject.setZero(y);
        tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.addF
        Collections.sort(tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ul
    }
    ;
    return tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeRe$ults_andOthers.g
}
```

Irreführend
Kein Hinweis!
FALSCH!
x, y: Koordinaten?
Bedeutung der intVar?

Zweiter Versuch: Bessere Bezeichner

Leere fehlt
Leere stört
Sieht wie Fehler aus
Gegen die Semantik

Fazit: Auch das Format ist wichtig
Leerzeilen, Leerzeichen, Tabs
Umbrüche, Einrückungen

```
    public Entry collapseEntries(){
Entry first, second;
      HelperEntry merged;
        while(entries.size(          )>1){
            first  = entries.removeFirst();second =     entries.removeFirst();
          merged=new HelperEntry();
            merged.setFrequency(first.getFrequency()
    + second.getFrequency());
            merged.setOne  (first)          ;
          merged.setZero( second);
      entries.addFirst  (    merged );
    Collections.sort(entries, new FrequencyComparator());
        );
    return entries.getFirst();
      }
    private class FrequencyComparator
implements Comparator{
        public int compare(Object arg0, Object arg1) {
            return ( ((Entry)arg0).   getFrequency()-((Entry)arg1).
                    getFrequency() );
}
    }
```


Format und Bezeichner besser

```
private Entry collapseEntries(){
    Entry first, second;
    HelperEntry merged;

    while(entries.size()>1){
        Collections.sort(entries, new FrequencyComparator());
        first  = entries.removeFirst();
        second = entries.removeFirst();
        merged=new HelperEntry();
        merged.setFrequency(first.getFrequency()+ second.getFrequency());
        merged.setOne(first);
        merged.setZero(second);
        entries.addFirst(merged);
    };
    return entries.getFirst();
}

private class FrequencyComparator implements Comparator{
    public int compare(Object arg0, Object arg1) {
        return ( ((Entry)arg0).getFrequency()-((Entry)arg1).getFrequency() );
    }
}
```

- **Keine Kommentare**
- **Alles public, keine Exceptions**
- **Verstehen Sie, was hier passiert – und warum?**

Kommentare sind kein Luxus

```
// ------------ private ------------------------

// Collapse list of entries into a code tree. Return its root.

private Entry collapseEntries(){
    Entry first, second;
    HelperEntry merged;

    if(entries.isEmpty()){
        return null;   // empty list, no tree can be built TODO throw ex
    }

    while(entries.size()>1){
        // Sort list by frequency, lowest frequency first
        Collections.sort(entries, new FrequencyComparator());

        first  = entries.removeFirst();
        second = entries.removeFirst();

        // merge elements and put result back into list
        merged=new HelperEntry();
        merged.setFrequency(first.getFrequency()+ second.getFrequency());

        // in Huffmann codes, it does not matter who gets "0" and who gets "1"
        merged.setOne(first);
        merged.setZero(second);

        // put result back (will be resorted if loop continues)
        entries.addFirst(merged);
    };
```

- **Schreiben Sie Programme zum Lesen – für Menschen!**
- **Dazu gehören**
  - **Bezeichner**
  - **Kommentare**
  - **Optische Verteilung**
- **Hintergrundinformation**
  - **„Huffmann Code"**

Fazit: Verständlich Programmieren

Sie wählen im Quellcode:
- **Klassennamen**
- **Methodennamen**
- **Leerzeichen**
- **Kommentare**
- **Einrückungen**
- **Variablentypen**
- **Schleifen**
- **Rückgabewerte**
- **…**

Ziele
- **Verständlichkeit**
- **Lesbarkeit**

```
// Collapse list of entries into a code tree. Return its r

private Entry collapseEntries(){
    Entry first, second;
    HelperEntry merged;

    if(entries.isEmpty()){
        return null;    // empty list, no tree can be buil
    }

    while(entries.size()>1){
        // Sort list by frequency, lowest frequency first
        Collections.sort(entries, new FrequencyComparator(

        first  = entries.removeFirst();
        second = entries.removeFirst();

        // merge elements and put result back into list
        merged=new HelperEntry();
        merged.setFrequency(first.getFrequency()+ second.g

        // in Huffmann codes, it does not matter who gets
        merged.setOne(first);
        merged.setZero(second);

        // put result back (will be resorted if loop conti
        entries.addFirst(merged);
    };
```


Software Engineering: Zoom out

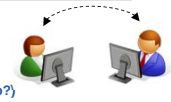
- **Vom Einzelkämpfer zum Team und Projekt**

- **Programmierstil:**
  subjektive Vorlieben und professionelles Verhalten
  - **Sie programmieren nicht vor allem für sich**
  - **Man soll nicht merken, wer es gemacht hat (egoless!)**
  - **Jemand anders macht damit weiter – evtl. nach langer Zeit**
  - **Software-Entwicklung ist viel mehr als Programmieren**
  - **Sie müssen zusammen erfolgreich sein**

Sie sind nicht allein.

- **Große Systeme kann man nicht allein entwickeln**
- **Entwicklung findet immer mit vielen Beteiligten statt**
  - Viele Kunden
  - Viele Entwickler
- **Koordination erforderlich**
  - Entwicklungsprozesse für die Abläufe
  - Versions- und Konfigurationsmanagement für die Technik
- **Unterstützung durch Werkzeuge**
  - z.B. CVS, Subversion oder Git

Richtlinien



- **Bezeichner:** machen knapp ihre Bedeutung klar
- **Format:** gewohnte Struktur erleichtert die Orientierung
- **Kommentare:** erklären den Quellcode (warum ist das so?)
- **Externe Dokumentation:** Beschreibt ausführlicher Hintergründe
- **Aufteilung und Struktur:** entspricht den Entwurfsentscheidungen
  - **Pakete:** Nicht vergessen!
  - **Anzahl von Klassen, Methoden, LOC, Kommentaren:** „ausgewogen“
  - **Vererbung:** einsetzen, aber nicht zu tief vererben
  - **Komplizierte Stellen im Programm** müssen erklärt werden
    - Wieso muss das gemacht werden?
    - Wieso wird es so gemacht?
    - Was bedeutet die Konstante, woher kommt die Formel?
    - Welche Funktion hat das im Algorithmus?
- **Möglichst macht man aber gleich gar nichts kompliziert**

Beispiele aus dem ProPra zur Diskussion

```
package de.unihannover.se.pp.Engine;

public class Account implements IAccount {
    private String name;
    private int balance;

    public Account(String name, int balance)
    {
        setName(name);
        setBalance(balance);
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        if(name == null)
            throw new IllegalArgumentException();
        if(name.equals(""))
            throw new IllegalArgumentException ("Der Kontoname sollte mindestens ein Zeichen lang sein!");
        this.name = name;
    }

    // Folgen noch getter und setter

    public int getBalance() {
        return balance;
    }
```

Sinnvoller Paketname

Gute Bezeichner

Position inkonsistent: {

Sogar sehr gut: Sonderfälle behandelt

Aufgabe
Implementierung für Interface IAccount
- Kontostand abrufen
- Inhabernamen abrufen und setzen

Bezeichner sorgfältig wählen

- **Konstanten benennen**
  - **Symbolische Namen:** PI statt 3.141592
  - **Groß geschrieben:** MEHRWERTSTEUER statt 16.0
- **Kurzbezeichner (nur) dort, wo sie üblich sind**
  - **Schleifenzähler:** i, j, k
  - **Koordinaten:** x, y
  - **Aber nicht aus Faulheit:** asrstz (Abschreibungsrabattsatz)
  - **Konvention:** Teilweise zur Vereinheitlichung erzwungen U2-MEA-PREM
- **Längere Bezeichner**
  - Drücken inhaltliche Bedeutung aus: steuersatz, kontonr
  - Hinweise auf Typ sind akzeptabel: kontoNrInt (nicht so gut)
  - Aber NICHT der Typ allein: float1
  - Nicht zu lang wählen: steuerlichAbsetzbarerPauschbetrag
  - Jedes Objekt soll EINE einzige Bedeutung haben: eindeutig benennen
  - Deutsch oder Englisch: kein Gemisch → stets Englisch!

Methodennamen

- **Bezeichnen, was der Empfänger tun soll:** *<Verb> (<Objekt>)*
  - **setName**
  - **fillRectangle**
  - **Update**
- **Methoden mit Rückgabe: Name bezeichnet Rückgabewert**
  - **isEmpty** implizierte Rückgabewerte: *ja oder nein*
  - **getName** nicht so schön, aber üblich (Rückgabe: name)
  - **conflictingRules** ermittelt Sammlung widersprüchlicher Regeln
    - ***Rückgabe: Rules (und zwar die mit Konflikt)***
- **Am besten lesen sich Aufrufe wie „normale Sätze“**
  - steuerformular.steuersatz(kunde.getEinkommen())
  - „Hallo Steuerformular, was ist der Steuersatz vom Einkommen des kunden?“

Vertiefung: Variablendefinition

- **Nur eine pro Zeile definieren, dahinter // Kommentar**

```
double steuersatz; // Durchschnittssatz, in Prozent
```

- **Initalisieren, wenn nicht stets erst eine Berechnung nötig**

```
double mehrwertsteuer = 19.0; //normaler Satz, in %
```

- **Möglichst lokal definieren**
  - Beispiel: Schleifenvariablen innerhalb der Schleife

```
for (int i = 1; i<10; i++) {…}
```

Java-Konventionen von Sun

- **Google: „Java code conventions“ (von Sun). Auszug:**

4.2 Wrapping Lines

When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Here are some examples of breaking method calls:

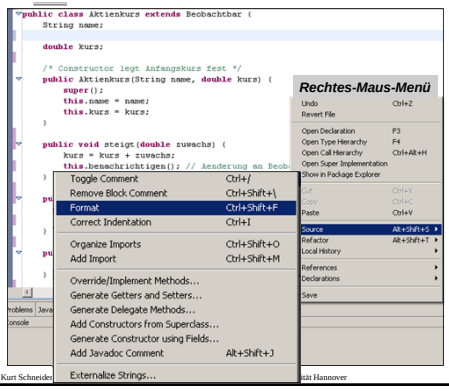
```
function(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);

var = function1(longExpression1,
                function2(longExpression2,
                          longExpression3));
```


Code-Konventionen

- **Wozu?**
  - Einheitlicher Code (egoless)
- **Welches sind die besten Konventionen?**
  - Eclipse, Sun, Firma XY, Ihre eigenen?
- **Welche sollte eine Firma verwenden, wie oft wechseln?**
  - Bekannte
  - Einfache
  - Prüfbare
- **Antwort: … vor allem immer dieselben!**

Eclipse hilft bei Java-Konventionen



ProPra-Beispiel zur Diskussion

stets anonym

Wenig „Luft“: Leerzeilen, Kommentare

```
package uebung01;
public class MoneyTransfer implements IMoneyTransfer {
    Account source;
    Account destination;
    int amount;
    public MoneyTransfer(Account source, Account destination, int amount) {
        this.source = source;
        this.destination = destination;
        if (amount >= 0)
          this.amount = amount;
        else
          this.amount = -1;
    }
    public void execute() {
        if (amount == -1)
          System.out.println("Buchung nicht erfolgreich - negativer Betrag!");
        else {
          source.setBalance(source.getBalance() - amount);
          destination.setBalance(destination.getBalance() + amount);
          System.out.println("Buchung erfolgreich!");
        }
    }
}
```

Paketname entspricht nicht der Norm

Abfrage bereits im Konstruktor. Wieso?

Fehler implizit durch Code gekennzeichnet, keine Reaktion

Scheinbar unsinnig enge Abfrage: was ist mit -200?

Anregung:
Zeigen Sie sich gegenseitig Ihre Programme und reden Sie darüber!

Dokumentation im Code

- **Kopfkommentare („header“)**
  - Bis zu einer Druckseite/Methode üblich
- **Struktur und Gestalt des Codes**
  - Code ist Dokument, muss ***lesbar*** gemacht werden
  - Einrückungen, Kommentare, Bezeichner
- **Namen von Variablen, Klassen, Methoden usw.**
  - Häufig Namenskonventionen
  - Dennoch sprechende Benennung
- **Kommentare**
  - Wo, wie viel, welche Angaben (v.a. Kopfkommentare)?
    - Im Projekt standardisieren!
  - Nicht beschreiben, was man sieht („5 addieren“, „Exception werfen“)
    - Eher *warum* es geschieht und *warum auf diese Weise*

Wichtig wieder:
Rationale
(Begründung)

Arten von Dokumentation

- **Für den Benutzer**
  - Manual, Bedienungsanleitung
  - Roll-Out-Plan
  - Wartungsunterlagen, technische Dokumentation
- **Für den Entwickler**
  - Codekommentare
  - Technische Dokumentation
  - Hintergrundinformation
- **Für das Projekt und das Management**
  - Produktdokumentation
  - Projektdokumentation (Ablauf, Zeitpläne, Organisatorisches)
  - Prozessdokumentation (Zwischendokumente, Prozessangaben)

Javadoc

Schnitt-Stellen (APIs)

Verlinkte Dokumentation, aus Kommentaren generiert

„doc comments“: /** <docu> */

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be l
 * that draw the image will incre
 *
 * @param  url  an absolute URL g
 *         name the location of t
 * @return      the image at the
 * @see         Image
 */
public Image getImage(URL url, S
    try {
        return getImage(new URL(
    } catch (MalformedURLExcepti
        return null;
    }
}
```

getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument is a specifier that is relative to the `url` argument.
This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

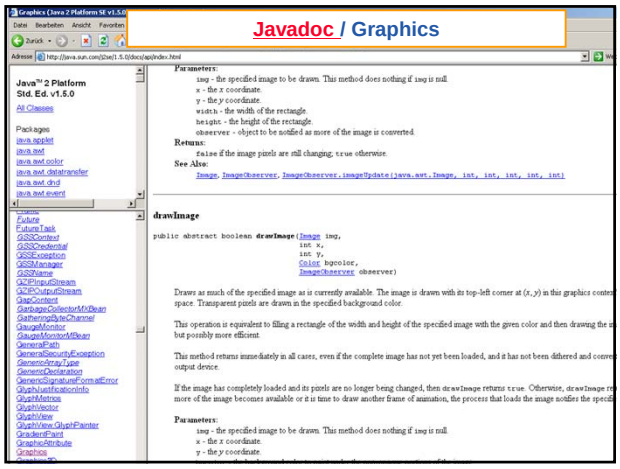
Parameters:
`url` - an absolute URL giving the base location of the image
`name` - the location of the image, relative to the `url` argument

Returns:
the image at the specified URL

See Also:
`Image`

Kurt Schneider

Javadoc / Graphics



Externe JavaDoc-Dokumentation erzeugen

1. JavaDoc-Kommentare einfügen

```
import java.util.Collections;

/**
 * This class can derive a Huffmann code from given character frequencies.
 *
 * @author Kurt Schneider
 */

public class HuffmannCode {
    LinkedList<Entry> entries;          // Original entries, modified during algorithm
    LinkedList<CodeEntry> codes;        // Resulting entries with code (no intermediate entries)
    CodeEntry entry;

    public HuffmannCode(){
        super();
        entries = new LinkedList<Entry>();
        codes   = new LinkedList<CodeEntry>();
    }

    /**
     * Include a new character.
     * This character will be included when codebook is created.
     *
     * @param newChar       character to find a code for
     * @param frequency     number of occurences
     */
    public void addChar(char newChar, int frequency) {
        entry = new CodeEntry();
```

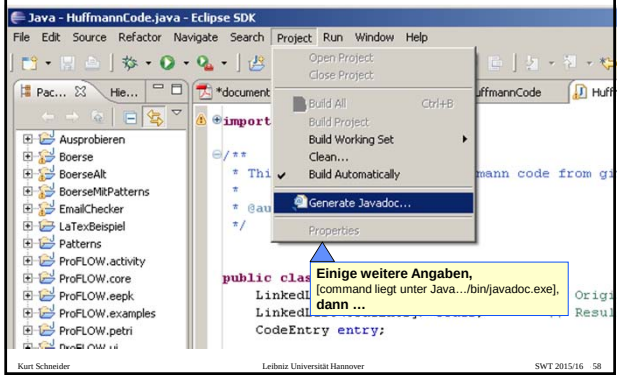
JavaDoc an die richtige Stelle schreiben

„normale Kommentare“ werden ignoriert

Es gibt Schlüsselwörter (mit @)

Externe JavaDoc-Dokumentation erzeugen

2. JavaDoc generieren



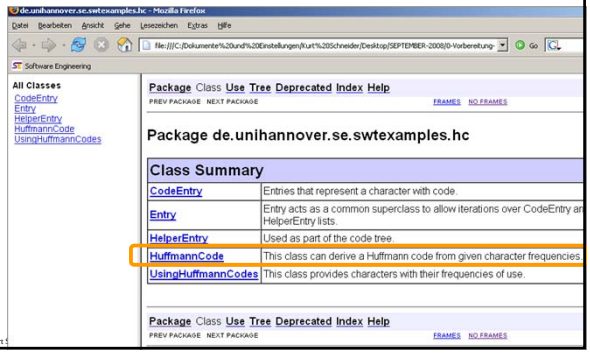
Einige weitere Angaben,
[command liegt unter Java…/bin/javadoc.exe],
dann …

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 58

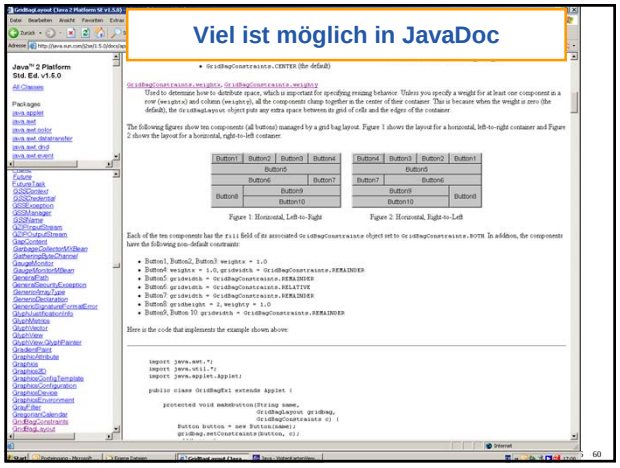
Externe JavaDoc-Dokumentation erzeugen

3. Erzeugte Dokumentation verwenden

Ein ganzer Dokumentenbaum wird erzeugt; z.B. index.html:



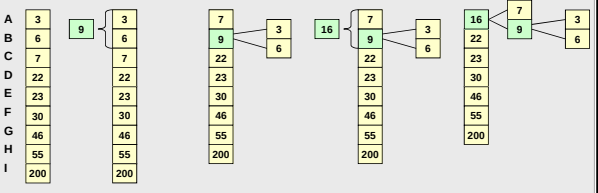
Viel ist möglich in JavaDoc



Externe Dokumentation
Beispiel: Entwurfsidee Huffmann-Code

- Teilweise als JavaDoc aus dem Programm generiert
- Teils separat geschrieben (nicht im Programm: Beispiel, Theorie)

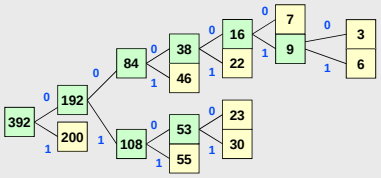
„Einträge nach Häufigkeit sortieren."



„Die zwei kleinsten streichen, ihre Summe in einen neuen Eintrag stecken und einsortieren."

„Diesen Schritt wiederholen, bis aus der Liste ein Baum geworden ist."

Externe Dokumentation
Beispiel: Entwurfsidee Huffmann-Code



„Diesen Schritt wiederholen, bis aus der Liste ein Baum geworden ist."

„Jeweils 0 und 1 an Verzweigung schreiben. Egal, was wohin."

Huffmann-Code

A	3	0 0 0 0 1 0
B	6	0 0 0 0 1 1
C	7	0 0 0 0 0
D	22	0 0 0 1
E	23	0 1 0 0
F	30	0 1 0 1
G	46	0 0 1
H	55	0 1 1
I	200	1

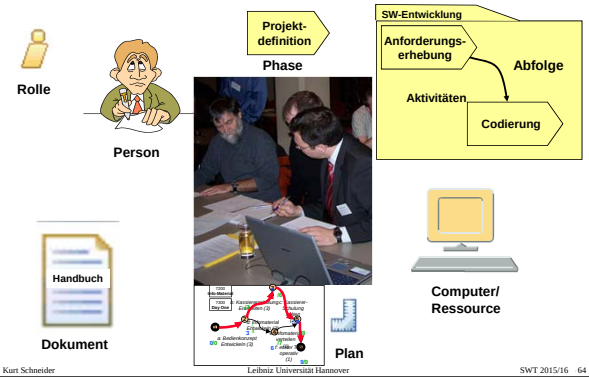
„Der 01-Pfad ist der Code für den Buchstaben"

- Auch interessant: Beweis, dass dies zu optimalem Code führt
- Wichtige Entwicklerdokumentation: Empfohlene Datenstruktur

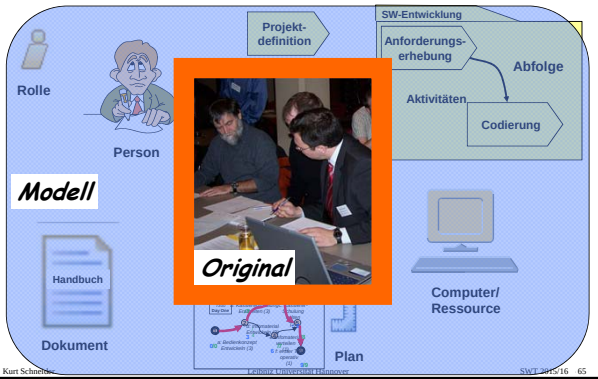
Beyond Coding

- Gut geschriebene Programme sind w€rtvoll.
  - Lesbare Programme kann man pflegen und erweitern.
  - Bezeichner, Format, Kommentare – und Dokumente
- Aber ein professionelles Softwareprojekt beginnt viel früher
  - Was soll eigentlich programmiert werden?
  - Wie strukturiert man den Code?
  - Wie können viele Experten dabei zusammenarbeiten?
  - Wie stellt man sicher, dass die Qualität stimmt?
- Der Schlüssel: Systematische Vorgehensweise!
  - Aktivitäten und Elemente, die sich ergänzen und stützen
  - Explizit, erlernbar, planbar, wiederholbar, optimierbar

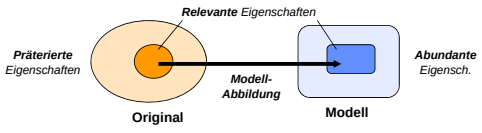
Softwareentwicklung planen
und „modellieren"



Modelle sind Voraussetzung für Planung



Grundbegriff „Modell"

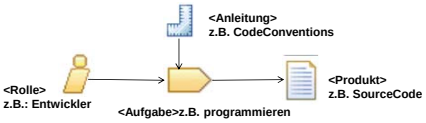


- Fragen zu Charakterisierung und Verständnis
  - Was ist das Original?
  - Modell für wen?
  - Modell zu welchem Zweck (welche Operationen)?
  - Welche Eigenschaften sind dafür relevant?

Wie modelliert man Software-Entwicklung?

SPEM / Syntax

- **SPEM heißt**
  - „Software & Systems Process Engineering Meta-Model“
  - Dt.: Metamodell für Entwicklungsprozesse der Software- und Systemtechnik
  - Version 2.0 vom April 2008
- **Historie**
  - Es gibt verwirrend viele Ablaufnotationen
  - Ziel: ordentlich definiertes Austauschformat für Prozessbeschreibungen
- **SPEM ist kein Prozess – sondern eine Modellierungsnotation dafür**



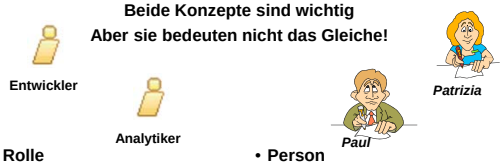
SPEM-Notation (Auszug)

- **Work Product Definition**
  - Dokument oder anderes Produkt
- **Role Definition**
  - Bündel von (erwarteten) Fähigkeiten u. Kompetenzen
  - „AKV: Aufgaben, Kompetenzen, Verantwortlichkeiten“
- **Task Definition**
  - Arbeitsvorgang; kann unterteilt werden
- **Category**
  - Zum Strukturieren von Elemente (in eine Hierarchie)
- **Guidance**
  - Anleitende Info.: Templates, Checklisten, Vorgaben

Person und Rolle

Immer genau unterscheiden

Beide Konzepte sind wichtig
Aber sie bedeuten nicht das Gleiche!



- **Rolle**
  - Auf Zweck hin definiert: Aufgabe erfüllen
  - Profil vorausgesetzt: Kenntnisse, Fähigkeiten
  - Definierte Aufgaben, Zuständigkeiten, Verantwortg.
  - Mit Erwartungen verbunden
  - Von einer oder mehreren Personen auszufüllen
- **Person**
  - Individuum mit persönlicher Geschichte: ist, wie sie ist
  - Hat ein Profil, passt mehr oder weniger zu Rolle
  - Wird Rolle zugewiesen, über-nimmt deren Aufgaben usw.
  - Erfüllt Erwartung mehr/weniger
  - Kann eine, mehrere oder keine (definierte) Rolle wahrnehmen

Modelle: für wen und zu welchem Zweck?

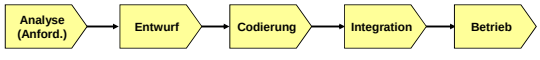
Ein Beispiel

Modell-element	Original	Relevante Eigenschaften	Für wen, wozu?
Aktivität	Bestimmte Tätigkeit	Ziel, Ergebnis, Dauer, Teilnehmer	Projektleiter, Mitarbeiter: Plan, Aufgaben
Rolle	Bündel von Aufgaben, Rechten u. Pflichten	Aufgaben, Zuständigkeiten Verantwortg.	Alle Projekt-beteiligte: wissen, wer was tun muss
Dokument	Papier- oder elektron. Dokument	Zugänglich, lesbar, meist mit definierter Struktur	Autor und Leser: was ist zu leisten/zu erwarten
Person	Individueller Mensch	Individuum; Fähigkeiten, Krankheiten …	Projektleiter: Rollen zuweisen, Eigen-heiten beachten
Beziehungen	Abhängigkeit, Kenntnisse …	(verschieden)	(vielfältig: meist um Übersicht zu wahren

Ordnung im Softwareprojekt

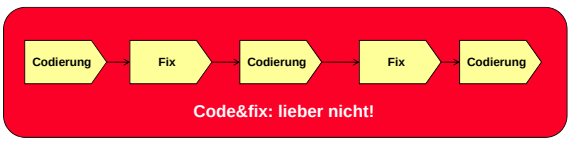
durch Prozesse und Prozessmodelle

Aktivitäten und Abläufe



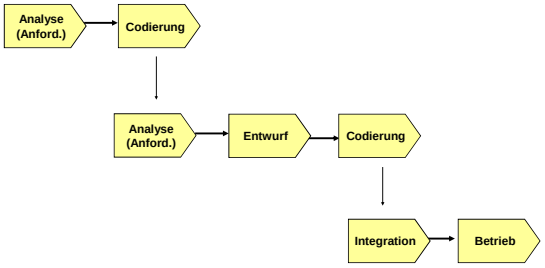
„Normaler Ablauf“

es gibt aber Alternativen!



Code&fix: lieber nicht!

Beispiel für eine sinnvolle Variante

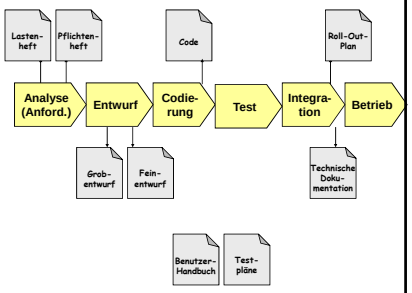


Prototyping – auch davon gibt es noch Unter-Varianten

Grundlegender Entwicklungsprozess

Basis für diese Vorlesung

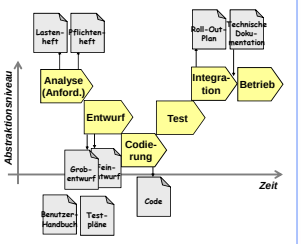
- **Diese Aktivitäten und Dokumentarten muss man kennen**
- **Später kann man einiges variieren**



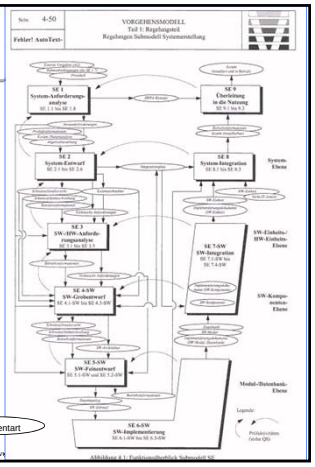
Dokumente sind Teil von „Software“ (vgl. IEEE-Def)

V-Modell 97

Ein weithin bekanntes Entwicklungsmodell



Leicht andere Modellsyntax: Dokum.-Art = Dokumentart



Schematisierte Darstellung

Ausblick: Test-Arten auf verschiedenen Ebenen

Beginnt beim System

Software ist unser Fokus

Integrieren und Testen