

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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*

Leibniz Universität Hannover

SWT 2015/16 · 31

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Inhalt von Kapitel 2



1. Wieso Software Engineering?

2. Vom Einzelkämpfer zum Großprojekt

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

Leibniz Universität Hannover

SWT 2015/16 · 32

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The diagram features a logo for "SOFTWARE ENGINEERING" with "SE" in large letters. The title "Situation Einzelkämpfer" is at the top. A red box contains the text "Programmieren als Hobby" and "Auch schön. Aber nicht unser Thema." Below the text are two columns of characteristics:

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
---	--

A small illustration of a person sitting at a computer is on the right.

At the bottom, the names "Kurt Schneider", "Leibniz Universität Hannover", and "SWT 2015/16 · 33" are listed.

Wer für sich alleine programmiert hat kaum etwas mit dem professionellen SW-Entwickler zu tun. Schon nach den Kriterien von Parnas ist eine Person allein kein SE.

Die obigen Eigenschaften der Allein-Arbeit machen deutlich, wie groß die Unterschiede zu einem echten Softwareprojekt sind.

Das gilt übrigens auch dann, wenn der oder die allein Arbeitende sehr versiert sind und sehr effiziente Algorithmen einsetzen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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 ...

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 34

Die Charakterisierung des Kollegen soll andeuten: Da wurde an dem gespart, was wirklich am billigsten ist: Der Speicherplatz für ein paar Kommentare usw. kostet nichts. Dagegen kann es Stunden (und damit tausende von Euro) kosten, Überlegungen nachzuvollziehen, die jemand nicht beschrieben bzw. dokumentiert hat.

Selbst auf der Ebene allein Arbeitender muss man manche Dinge beachten, um sich nicht selbst später ein Bein zu stellen.

Schon auf der einfachsten Ebene, beim reinen Programmieren, muss man auf verständliche Programme achten.

Natürlich müssen sie auch noch effiziente Algorithmen und passende Datenstrukturen verwenden; aber das lernen Sie in anderen Vorlesungen.

Aus ingenieurmäßiger Sicht ist es von sehr hoher Bedeutung, dass andere (und Sie selbst nach einiger Zeit!) nachvollziehen können, worum es in einem Programm geht.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Was tut dieses Programmstück?

Erst verstehen, dann verbessern

```
public ProgObject DenAlgorithmus_DurchführenfürEEE() {ProgObject x, y;OtherClass  
OtherObject;while (tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.  
size() > 1) {x = tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.  
removeFirst();  
y = tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.removeFirst();  
OtherObject = new OtherClass();// subtract x and y values  
OtherObject.getIntVar_toHaveANewValue(x.intVar() + y.intVar());OtherObject.setOne(x);  
OtherObject.setZero(y);tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.  
addFirst(OtherObject);  
Collections.sort(tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers,  
new myComparator());  
};return  
tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_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!**

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 35

Dieses Programmstück funktioniert – der Compiler kann es ausführen.

Das ist erstaunlich, weil vieles für Menschen unverständlich ist: Die seltsamen Namen mit Sonderzeichen, die fehlenden und unsinnigen Einrückungen usw.

Offenbar ist es ein Unterschied, ob der Compiler oder wir Menschen ein Programm „verständlich“ finden.

In den nächsten Folien werden unterschiedliche Aspekte verbessert – damit man sieht, wie viel man allein mit ein wenig Formatierung und vernünftigen Bezeichnern erreichen kann. Diese Einsicht gilt auch für weniger drastische Fälle: kaum werden Sie mit solchen Programmen konfrontiert werden; aber man wird sich im Beruf schon manchmal fragen, wieso die Autoren der SW nicht ein wenig mehr an spätere Leser und Kollegen denken konnten.

Wer es seinen Kolleginnen und Kollegen unnötig schwer macht, hat noch nicht begriffen, worum es in der echten SW-Entwicklung geht.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The diagram shows a central box labeled "Quellcode dient der Kommunikation". Inside, there's a list of points:

- 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

Two icons represent the communication ends: "Entwickler" (Developer) on the left and "<Mensch>" (User) on the right. Arrows point from each icon to the central text box. Below the text box is a box labeled "Quellcode" containing some Java-like pseudocode:

```
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());
        merged.setFirst(first);
        merged.setZero(second);
        entries.addFirst(merged);
        Collections.sort(entries, new P1());
    }
    return entries.getFirst();
}

private class FrequencyComparator implements Comparable<Object> {
    public int compare(Object arg0, Object arg1) {
        return ((Entry)arg0).getFrequency()
    }
}
```

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 36

Quellcode sollten Sie probeweise auch einfach als Erklärungstext verstehen. Wenn Leser nicht nachvollziehen können, was der kommentierte und gut strukturierte Code tut, dann hat der Autor etwas falsch gemacht. Dieser Fehler ist gravierend!

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

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; Kein Hinweis! Irreführend  
    OtherClass OtherObject;  
    while (tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.  
        x = tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.  
        y = tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.  
        OtherObject = new OtherClass();  
        FALSCH! // subtract x and y values x, y: Koordinaten?  
        OtherObject.getIntVar_toHaveANewValue(x.intVar() + y.intVar()); Bedeutung der intVar?  
        OtherObject.setOne(x);  
        OtherObject.setZero(y);  
        tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.addF  
        Collections.sort(tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.  
    }  
};  
return tableOfContentsWhichDescribeBoth_ThoseThingsthatCanBeResults_andOthers.c  
}
```

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 - 37

Schon durch richtige Zeilenumbrüche und Einrückungen (d.h. durch Formatierung) wird das Programmstück viel übersichtlicher.

Sie können jetzt recht leicht erkennen, welche Kontrollstrukturen darin vorkommen und wie sie geschachtelt sind.

Worum es genau geht, sieht man aber noch nicht. Daran sind hauptsächlich die extrem unschönen Bezeichner schuld.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Zweiter Versuch: Bessere Bezeichner

```
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();  
    }  
}
```

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 38

Hier hat man die schlechte Formatierung belassen, aber bessere Bezeichner eingeführt.

Auch das bringt schon einiges, aber allein die paar Leerzeilen und -zeichen machen aus einem leichtverständlichen Programm einen schwer verständlichen Verhau.

Sie sind als angehende SW-Profis dafür verantwortlich, die kleine Mühe der Formatierung und Bezeichnervergabe auf sich zu nehmen, um den Nutzen Ihrer Programme im Projektteam deutlich zu vergrößern. Wenn Sie allein etwas (jetzt) verstehen, sagt das gar nichts. Andere müssen es verstehen, das zeigt die Qualität.

Auch hier sieht man wieder: seltsame Einrückungen sind zwar für den Compiler kein Problem, aber für menschliche Leser sehr.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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<Entry> {
    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?

Kurt Schneider

SWT 2015/16 · 39

Das sieht schon viel besser aus.

Aber wenn Sie dieses Programm warten (verändern, korrigieren) sollten, würden Ihnen immer noch wichtige Informationen fehlen:

- Worum geht es überhaupt? Was soll mit dem Programm erreicht werden?
- Was hat sich der Autor oder die Autorin bei bestimmten Stellen gedacht?
- Wo stecken besondere Tricks, wo hat man eine Vereinfachung vorgenommen oder etwas besonders Intelligentes getan?

Wenn man das nicht weiß, besteht die große Gefahr, dass bei der Wartung all die Überlegungen unentdeckt bleiben und die guten Ideen wieder kaputt gemacht werden.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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 Huffman 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
 - „Huffman Code“

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 40

Kommentare im Code müssen nicht lang sein.

Man sollte sich aber klar über die Sprache werden: Englisch? Das versteht jeder, manche tun sich aber beim Schreiben etwas schwer.

Oder Deutsch/Spanisch/Finnisch? Das schreibt sich zwar leicht, aber die meisten der Sprachen sind nur für wenige Kollegen in einem globalen Projekt verständlich.

Fazit: Wenn Sie ein sehr kleines Projekt bearbeiten, das sicher nur von Kollegen gepflegt werden soll, die gut Deutsch sprechen (oder Russisch oder Portugiesisch...), dann können Sie diese Sprache für die Kommentare wählen. Aber Sie müssen sich darüber klar sein: Wenn das Programm versehentlich doch so gut wird, dass es weiter entwickelt wird, dann sind Sie mit einer anderen Sprache als Englisch ziemlich in der Klemme.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Fazit: Verständlich Programmieren

Sie wählen im Quellcode:

- Klassennamen
- Methodennamen
- Leerzeichen
- Kommentare
- Einrückungen
- Variablenarten
- Schleifen
- Rückgabewerte
- ...

Ziele

- Verständlichkeit
- Lesbarkeit

```
// Collapse list of entries into a code tree. Return its root entry.
private Entry collapseEntries(){
    Entry first, second;
    HelperEntry merged;

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

    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());
        merged.setOne(first);
        merged.setZero(second);

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

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

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 41

Sie müssen viele Entscheidungen treffen.

Sie sollen wissen, was davon abhängt: Die Verständlichkeit und Wartbarkeit.

Und Sie sollen wissen, wie die Sachen richtig zu machen sind.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



The slide features a logo for "Software Engineering" with stylized letters "SE" and the words "SOFTWARE ENGINEERING". The title "Software Engineering: Zoom out" is displayed in blue. Below the title is a bulleted list: "**• Vom Einzelkämpfer zum Team und Projekt**". An illustration shows seven stylized human figures in various colors (blue, orange, green, black) working at computer monitors. Another bulleted list follows: "**• Programmierstil:** subjektive Vorlieben und professionelles Verhalten" with five sub-points: "– 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", and "– Sie müssen zusammen erfolgreich sein". At the bottom of the slide, the names "Kurt Schneider", "Leibniz Universität Hannover", and "SWT 2015/16 · 42" are listed.

- 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

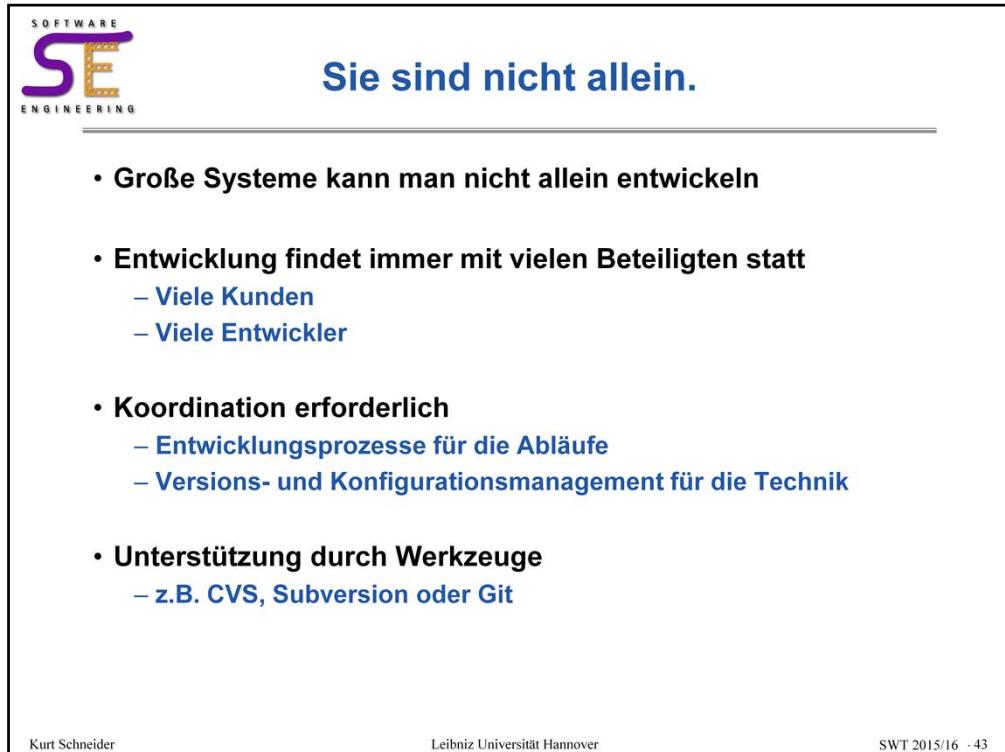
Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 42

Sie haben schon gemerkt: Es geht darum, dass sich ein Team versteht, auch über den Quellcode.

Dagegen ist es nicht wichtig, möglichst kompliziert aussehenden Code zu schreiben. Das tun nur Hobbyprogrammierer.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



The slide features a logo for 'SOFTWARE ENGINEERING' with the letters 'SE' in large, stylized letters. Below the logo is the slogan 'Sie sind nicht allein.' A horizontal line separates the logo from the main content. The main content is a bulleted list:

- 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

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 43

In jedem echten Projekt gehen Sie mit mehreren Entwicklern und Kunden um.
Das verändert vieles.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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**



Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 44

Dies sind wichtige Richtlinien, wie Sie auf Codeebene professionell arbeiten können.

Damit verringern Sie Unverständnis und Missverständnisse im Team.

Aber auch als Alleinprogrammierer profitieren Sie davon, wenn Sie später Ihre eigenen Programme ansehen und weiterentwickeln wollen.

Der Satz „machen Sie nichts komplizierter als unbedingt nötig“ ist ganz wichtig!
Dann brauchen Sie auch nicht viel zu erklären.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The slide features the SE Software Engineering logo at the top left. The title 'Beispiele aus dem ProPra zur Diskussion' is centered. Below the title is a Java code snippet for an 'Account' class:

```
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;
    }
}
```

Annotations on the right side of the code include:

- 'Sinnvoller Paketname' (meaningful package name) pointing to the package declaration.
- 'Gute Bezeichner' (good names) pointing to the class and method names.
- 'Position inkonsistent: {' (inconsistent position: {) pointing to the opening brace of the constructor.
- 'Aufgabe Implementierung für Interface IAccount' (task: implementation for interface IAccount) with sub-points: - Kontostand abrufen, - Inhabernamen abrufen und setzen.
- 'Sogar sehr gut: Sonderfälle behandelt' (Even very good: special cases handled) pointing to the validation logic in the setName method.

At the bottom left is the author's note 'Kurt Schneider'. At the bottom center is 'Leibniz Universität Hannover'. At the bottom right is 'SWT 2015/16 · 45'.

Ein paar Verstöße und Diskussionsanstände, die aus echten Lösungen im Programmierpraktikum stammen.

Namen werden nur angegeben, wenn es sich um rein positive Programmbeispiele zeigt, daher hier nicht.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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!

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 46

Bezeichner sind wichtig, das haben Sie vorne gesehen.
Hier die Tipps, wie Sie gute Bezeichner finden.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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?“

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 47

Es gibt unterschiedliche Ansichten über gute Methodennamen.

Oben meine Ansicht: man sollte den Code fast wie Sätze lesen können. Das ist am besten gewährleistet, wenn Operationen ohne Rückgabewert als Befehle formuliert sind.

Diese Befehle richten sich als objekt-orientierte Nachrichten an das empfangende Objekt. Wenn man „<Objekt> <Aufforderung>“ gut lesen kann, ist es schon einmal kein ganz schlechter Name – wenn er nicht zu lang usw. ist.

Ruft man dagegen eine Funktion auf, so sollte der Aufruf (der Methodename) so heißen wie der Rückgabewert.

Das wäre „`conflictingRules name taxRate`“. Meist werden aber die „Getter“ mit einem vorgestellten „get“ formuliert: „`getName`, `getTaxRate`“.

Diesen Kompromiss sollte man eingehen, also das get schreiben, obwohl sich der Satz dann nicht mehr so gut liest.

Tipp: Lesen Sie „get“ als „gib mir“, dann ist es wieder in Ordnung – und stimmt mit dem überein, wie sehr viele Leute programmieren.

Das ist gut für die Verständlichkeit.

Boolean Rückgabewerte sollte man durch die entsprechende Bedingung ausdrücken: „`isEmpty` hasBrother isFull“ usw.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Vertiefung: Variablendefinition

2

- Nur eine pro Zeile definieren, dahinter // Kommentar
`double steuersatz; // Durchschnittssatz, in Prozent`
- Initialisieren, 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++) {...}`

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 48

Berechnung nötig heißt: Wenn eine komplizierte Formel zur Initialisierung nötig ist, kann die auch nach der Definition kommen. Feste Werte besser schon in der Variablendeclaration zuweisen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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));
```

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 49

Sie können sich viel umfangreichere Richtlinien für „guten Code“ aus dem Internet herunterladen.

Wenn Sie „Java code conventions“ suchen, finden Sie z.B. zum Zeilenumbruch die obigen Richtlinien.

Sogar solche scheinbare Kleinigkeiten sind geregelt.

Man könnte es auch anders regeln.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Code-Konventionen

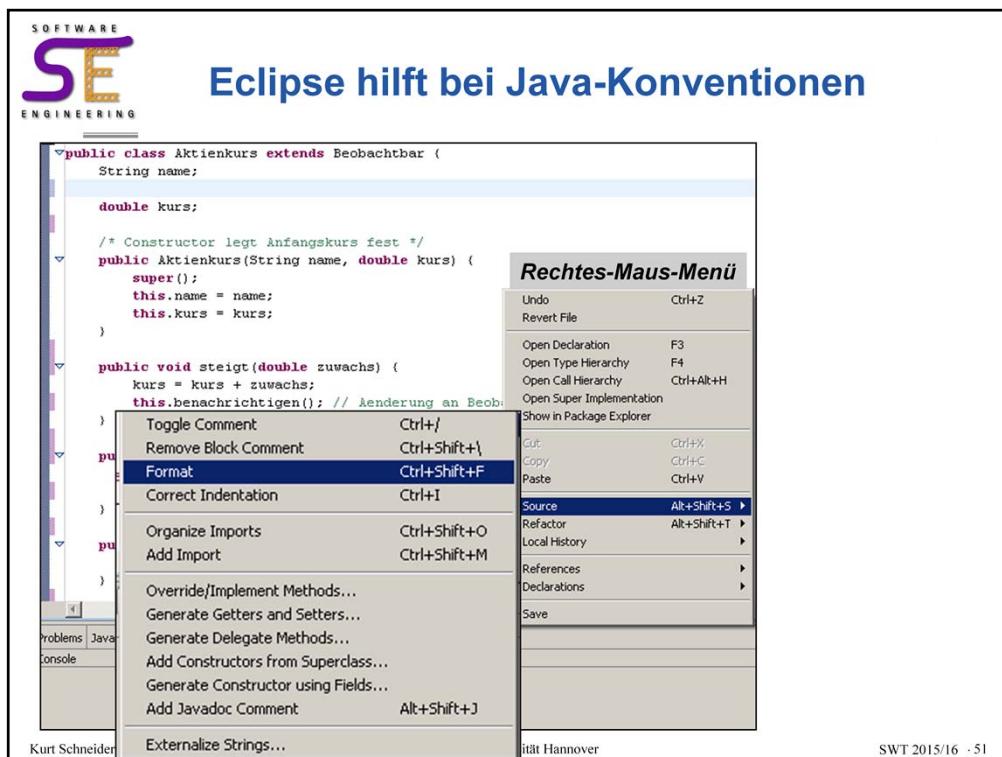
- Wozu?
 - Einheitlicher Code (egoless)
- Welches sind die besten Konventionen?
 - Eclipse, Sun, Firma XY, Ihre eigenen?
- Welche sollte eine Firma verwenden, wie oft wechseln?
 - Bekannte
 - Einfache
 - Prüfbare
- Antwort: ... vor allem immer dieselben!

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 50

Das wichtigste ist aber: Innerhalb eines Projekts, möglichst einer Firma, noch besser: einer Community muss man sich einig sein, wie es gemacht werden soll. Dann sind die Details nicht so wichtig. Streiten Sie nicht darüber.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Wenn Sie eine Entwicklungsumgebung wie Eclipse verwenden, ist die Formatierung des Codes schon vorgegeben.

Eine Firma könnte hier ihre Richtlinien hinterlegen, und daran müsste man sich natürlich halten.

Die automatisierte Formatierung durch einen sogenannten Pretty-Printer hat den wesentlichen Vorteil, dass sie so schnell und einfach ganz präzise herzustellen ist. Diesen Vorteil sollte man nutzen, und auf ein paar Feinheiten verzichten, die man gerne anders hätte.

ODER: Sie modifizieren die Regeln, das geht in Eclipse.

Nur müssen Sie dann darauf achten, dass alle Kollegen, auch an anderen Standorten stets die gleichen Regelsätze verwenden.

Das ist Bürokratie, richtig. Aber die ist hier wichtig. Das müssen Sie in den Griff bekommen, damit Sie mit wenig Mühe viel hinbekommen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The screenshot shows a Java code editor with annotations:

```
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!");
        }
    }
}
```

- A red arrow points to the package declaration with the annotation: "Paketname entspricht nicht der Norm".
- A yellow box at the top right says: "Wenig „Luft“: Leerzeilen, Kommentare".
- A red arrow points to the constructor's parameter list with the annotation: "Abfrage bereits im Konstruktor. Wieso?".
- A red arrow points to the assignment of amount = -1 with the annotation: "Fehler implizit durch Code gekennzeichnet, keine Reaktion".
- A red arrow points to the if-statement in execute() with the annotation: "Scheinbar unsinnig enge Abfrage: was ist mit -200?".
- A yellow box on the right says: "Anregung: Zeigen Sie sich gegenseitig Ihre Programme und reden Sie darüber!".

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 52

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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)

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 53

Vielfach wird Dokumentation als Produktdokumentation oder Dokumentation im Code verengt.

Die obigen Elemente sind dabei besonders wichtig.

Ganz wichtig: Dokumentation geht über codenahe Teile hinaus!

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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)

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 54

Alle diese Dokumentationsarten fallen in den meisten Projekten an – oder sollten zumindest erstellt werden.

Sie sehen viele unterschiedliche Aspekte, den das Management braucht ganz andere Angaben (Planung, Zeitplan, Aufwand, tatsächlicher Aufwand usw.), während Entwickler das Programm verstehen und Fehler beheben müssen.

Benutzer müssen ein Programm vor allem bedienen, aber auch installieren und auf viele Arbeitsplätze verteilen können (roll-out). Dazu gehört die Fähigkeit, die Zwischenzeit zu bewältigen – wenn nur ein Teil der Mitarbeiter das (neue) Programm hat, die anderen noch keines oder die ältere Version. Das muss alles technisch und organisatorisch geplant und bewältigt werden.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The screenshot shows a Javadoc-generated HTML page. At the top left is the logo "SOFTWARE ENGINEERING" with "SE" in large letters. To the right is the title "Javadoc". In the top right corner is a box labeled "Schnitt-Stellen (APIs)". Below the title, a section titled "Verlinkte Dokumentation, aus Kommentaren generiert" is shown. On the left, there is a block of Java code with Javadoc comments. A callout box highlights the "doc comments" syntax: `/**`. On the right side, the generated documentation for the `getImage` method is displayed. It includes the method signature `public Image getImage(URL url, String name)`, a detailed description, parameters (`url` and `name`), return information, and a "See Also" link to `Image`. The footer of the page says "Kurt Schneider".

Javadoc ist ein Werkzeug, um aus speziellen Kommentaren im Code eine separate, verlinkte Schnittstellendokumentation abzuleiten. Man sieht oben den Ausgangscode mit den spezifischen doc comments (`/**`) und dem daraus generierten Dokumentationsteil.

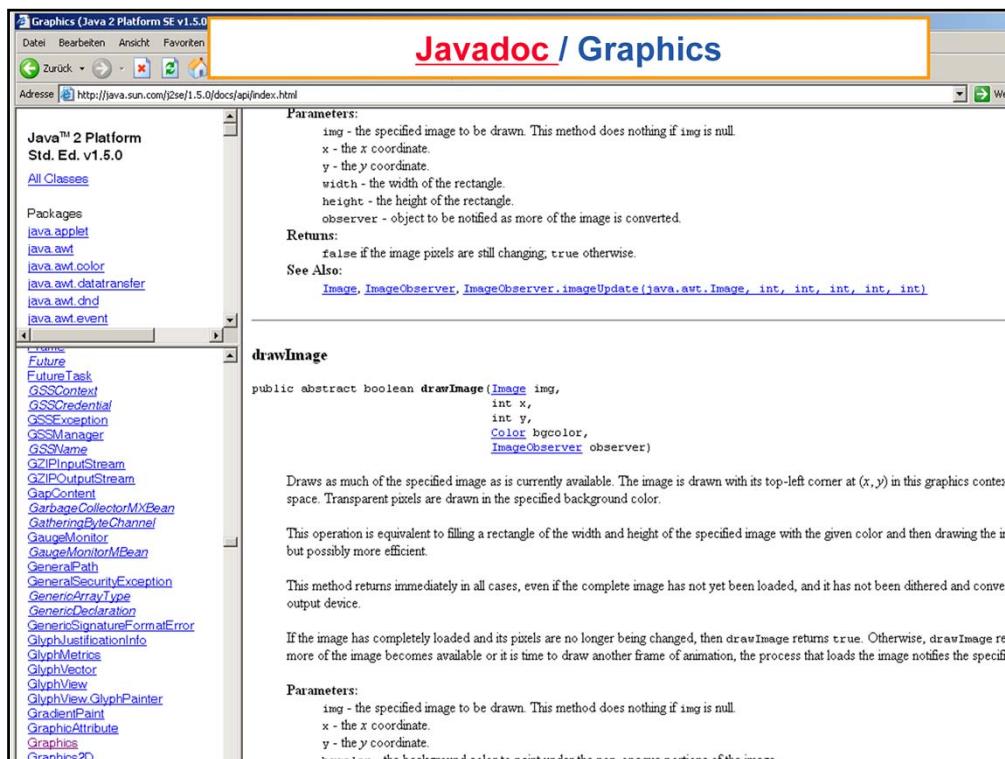
Insgesamt sind die Teile dieser erzeugten Dokumentation miteinander verknüpft, so dass man in einem Browser hin- und her-navigieren kann.

Wie ruft man JavaDoc auf?

- In Eclipse wählt man unter Projekte den Menüpunkt „generate JavaDoc“
- Dann erscheint ein Menü. Man muss folgendes auswählen:
 - Wo steht das JavaDoc-Kommando? Es befindet sich in der Java-JDK, also im Java Developer Kit unter bin/javadoc.exe
 - Diese Datei auswählen. Sie steht bei Java, NICHT bei eclipse.
 - Wofür soll Javadoc generiert werden? Typisch für public, denn der Rest sind Interna, die man nicht nach außen sehen oder kommentieren will.
 - Schließlich noch die Frage, wohin die HTML-Seiten generiert werden sollen. Das sollte ein eigener Ordner sein, denn es wird ziemlich viel generiert.
 - Von der generierten index.html aus kann man alle generierten Inhalte durchsuchen und betrachten.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Ein weiteres Beispiel, das die Signatur der Schnittstelle zeigt.

Sehr schöner, umfangreicher java-Doc-Code, der sehr lesbare Schnittstellenbeschreibungen zu generieren erlaubt.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The screenshot shows a Java code editor with the following code:

```
SOFTWARE  
SE ENGINEERING  
Externe JavaDoc-Dokumentation erzeugen  
1. JavaDoc-Kommentare einfügen  
JavaDoc an die richtige Stelle schreiben  
„normale Kommentare“ werden ignoriert  
Es gibt Schlüsselwörter (mit @)  
import java.util.Collections;  
  
/**  
 * This class can derive a Huffmann code from given character frequencies.  
 *  
 * @author Kurt Schneider  
 */  
  
public class HuffmannCode {  
    LinkedList<Entry> entries;  
    LinkedList<CodeEntry> codes;  
    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();  
    }  
}
```

Annotations in the code:

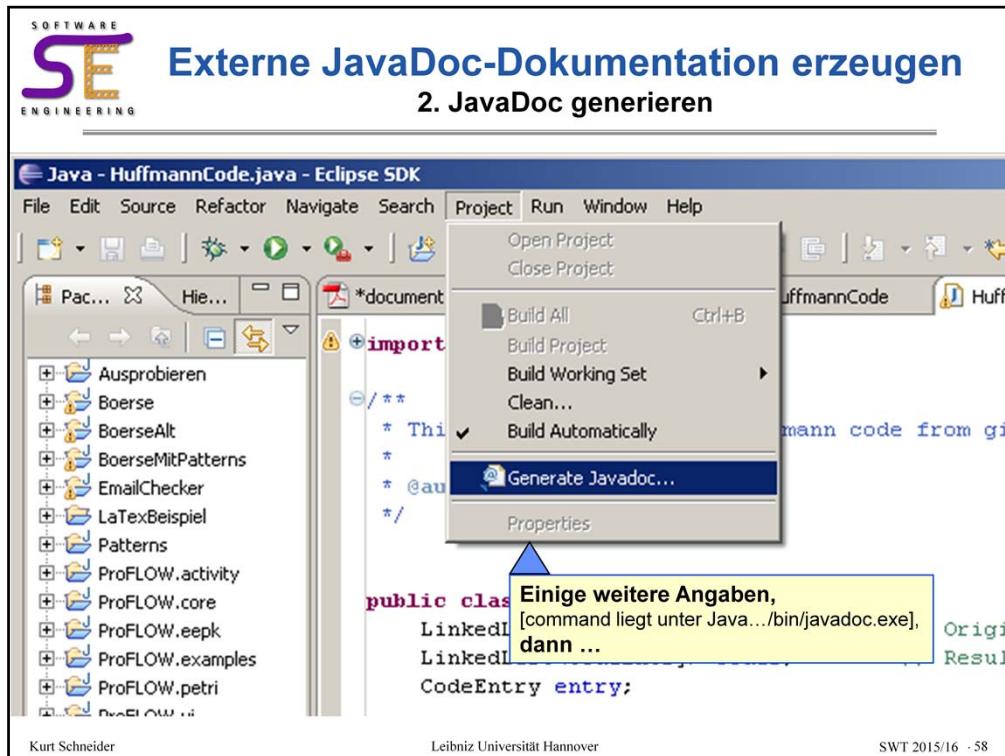
- JavaDoc an die richtige Stelle schreiben**: Points to the first JavaDoc block at the top of the class.
- „normale Kommentare“ werden ignoriert**: Points to the regular comments within the class body.
- Es gibt Schlüsselwörter (mit @)**: Points to the parameter annotations (@param).

Hier sehen Sie, wie man JavaDoc-Kommentare einfügt und dann die scöne formatierte Schnittstellenbeschreibung generiert.

Sie ist miteinander verlinkt und erlaubt es, von „außen“ in den Code hineinzuklicken.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Mit diesem Kommando erzeugen Sie die html-Darstellung.

Dabei müssen noch wenige Angaben gemacht werden, wohin der erzeugte html-Code gelangen soll und wie das Java-Doc-Kommando heißt.

Sie müssen seinen Pfad im Eclipse-Verzeichnis zeigen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

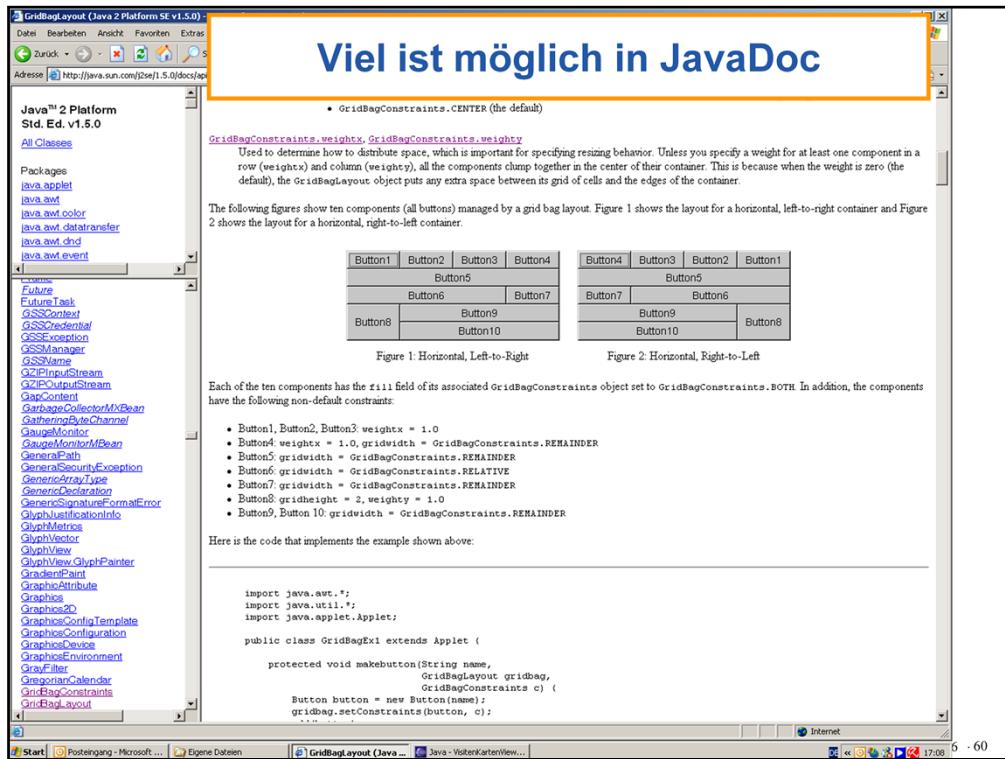
The screenshot shows a JavaDoc-generated HTML page for the package `de.unihannover.se.swtexamples.hc`. The main content is a **Class Summary** table:

Class Summary	
CodeEntry	Entries that represent a character with code.
Entry	Entry acts as a common superclass to allow iterations over CodeEntry and HelperEntry lists.
HelperEntry	Used as part of the code tree.
HuffmannCode	This class can derive a Huffmann code from given character frequencies.
UsingHuffmannCodes	This class provides characters with their frequencies of use.

Das kommt dann bei dem Code heraus, der oben in verschiedenen Varianten durchgesprochen wurde: Der HuffmannCode mit den dazugehörigen Hilfsklassen, sehr schön aufbereitet.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



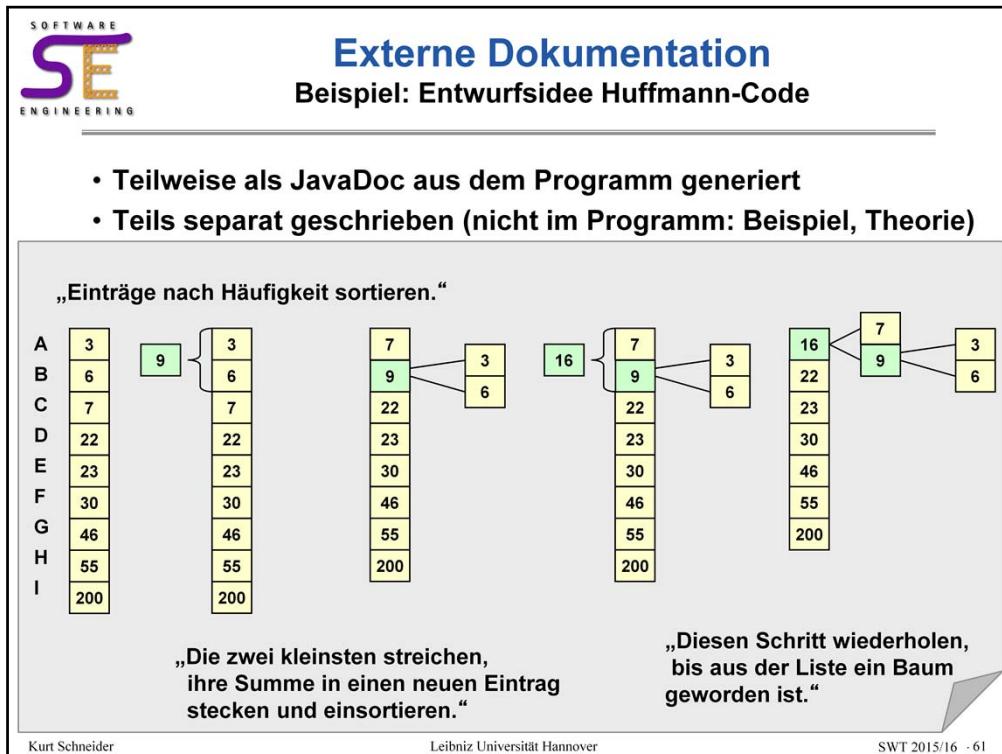
Hier sieht man einen Ausschnitt aus dem sehr ausführlichen Kopfkommentar der Klasse `GridBagLayout`. Auch der Code hier gehört zum Kommentar, er zeigt, wie das Beispiel oben erzeugt werden kann.

Da es nicht sinnvoll (oder möglich) ist, so komplizierte Dinge wirklich in den Quellcode zu schreiben, kann man auch separate Dateien einbinden.

An der Seite sehen sie die Klassen, die man anklicken kann.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Was Ihnen immer noch fehlte, war die Einführung, was der Code tun sollte, im Überblick.

Daher hier die Beschreibung der Idee von HuffmannCodes:

Man möchte für eine Liste gegebener Buchstaben (links, mit Häufigkeiten, direkt daneben) eine binäre (0-1)-Codierung erhalten, so dass man typische Texte mit möglichst wenigen Bits übertragen kann. Also: kurzer Code für häufige Buchstaben, längere Codes für seltene Buchstaben führt zu einem „kurzen“ Erwartungswert.

Dafür gibt es einen Algorithmus, der oben gezeigt ist.

Das Dokumentensymbol im Hintergrund zeigt, dass dieser Algorithmus und sogar das obige Beispiel natürlich ganz wichtig für das Verständnis des Programms ist, das diesen Algorithmus umsetzt. Also muss man das auch dokumentieren (Entwicklerdokumentation). Muss das in Code-Kommentaren geschehen? Eher nicht, denn den Überblick gewinnt man besser, bevor man sich die Details ansieht. Auch würden die Programme durch sehr umfangreiche Einleitungskommentare sehr umfangreich und auseinandergesogen.

Also: Ein Fall für externe Dokumentation.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Externe Dokumentation
Beispiel: Entwurfsidee Huffmann-Code

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

Diagramm: Eine hierarchische Baumstruktur zur Erstellung eines Huffmann-Codes. Die Wurzel ist der Wert 392. Sie verzweigt sich in 192 (0) und 200 (1). 192 verzweigt sich in 84 (0) und 108 (1). 84 verzweigt sich in 38 (0) und 46 (1). 108 verzweigt sich in 53 (0) und 55 (1). 38 verzweigt sich in 16 (0) und 22 (1). 53 verzweigt sich in 23 (0) und 30 (1). 16 verzweigt sich in 7 (0) und 9 (1). 9 verzweigt sich in 3 (0) und 6 (1).

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

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

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

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

Sogar der Beweis wäre interessant, obwohl man ihn nicht braucht, um den gegebenen Algorithmus zu programmieren.

In Programmkommentaren hat er aber sicher nichts verloren. Wieder Entwicklerdokumentation, vermutlich für das Qualitätsmanagement.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Beyond Coding

- Gut geschriebene Programme sind wertvoll.
 - 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



Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 63

Wir haben mit dem Programmcode begonnen und dort überlegt, wie man in einem professionellen Projekt am besten zusammen arbeiten kann.

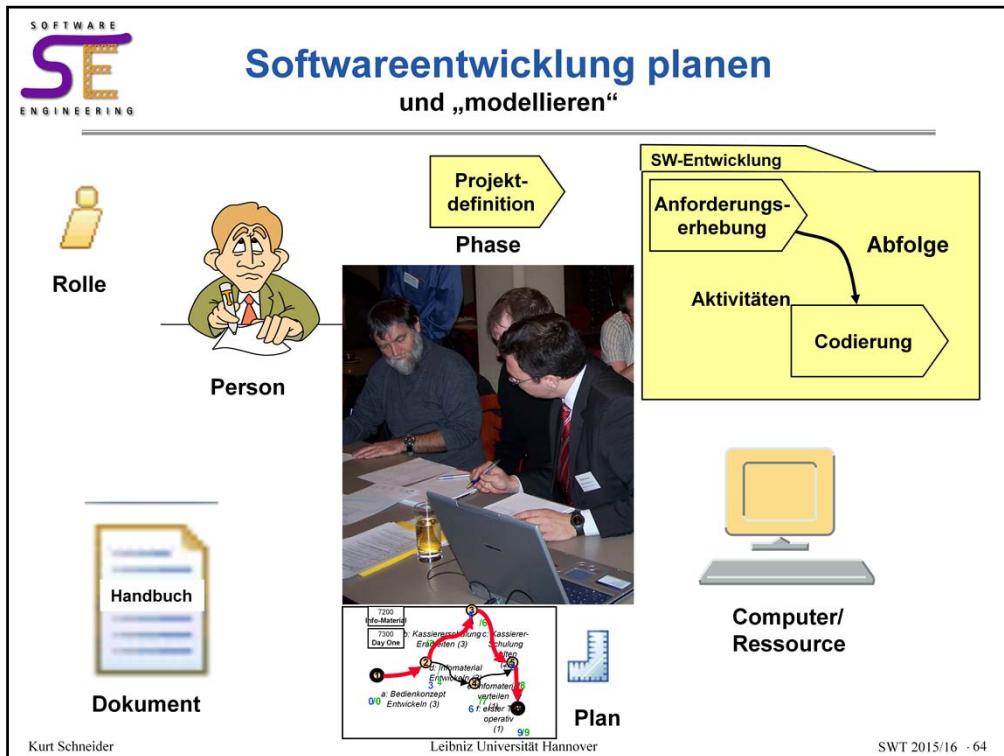
Aber die Frage geht natürlich weit über das eigentliche Programmieren hinaus. Schon die Klärung von Anforderungen, Strukturierung und Architektur bestimmen noch viel stärker, wie gut das Programm – und das Projekt laufen wird.

Es hat sich bewährt, erfolgreiche Abläufe und Techniken immer wieder einzusetzen. Man plant also vorher ein, nach welchem System man vorgehen will.

Das System sollte aufgeschrieben sein (explizit), dann können es neue Mitarbeiter erlernen. Projektleiter können anhand der Erfahrungen im System realistisch planen; erfolgreiche Pläne kann man bei Folgeprojekten anpassen und wieder verwenden. Und vielleicht sogar noch ein wenig weiter optimieren.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

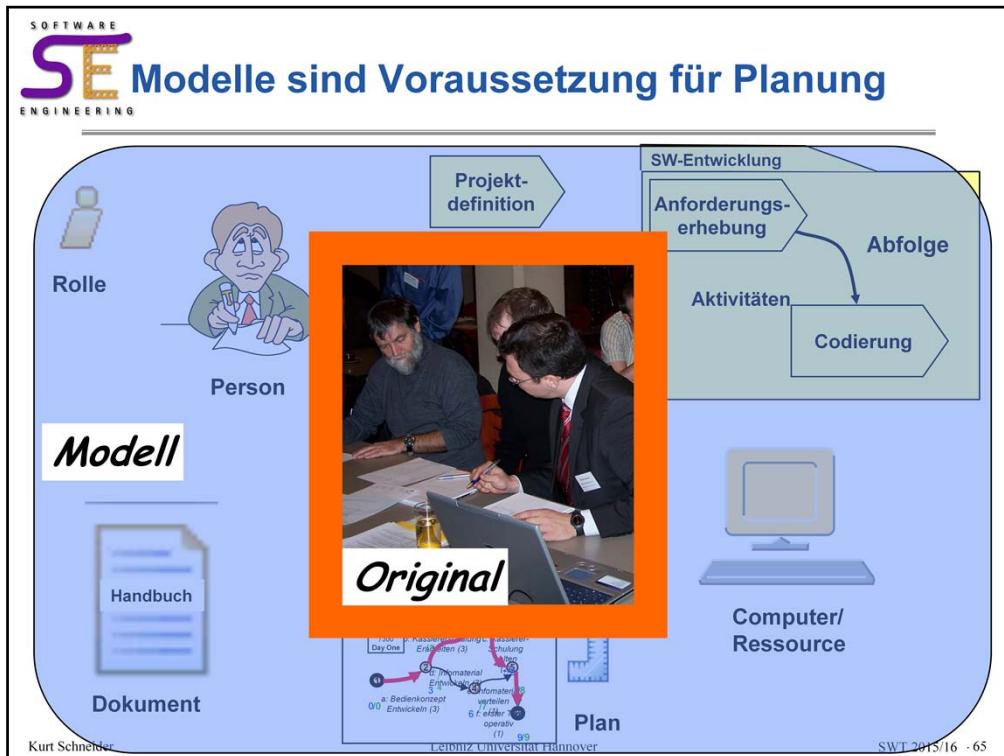
Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Zur Softwareentwicklung gehören ganz unterschiedliche Personen und Dinge.
Wir werden sie später genauer kennenlernen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Wichtig ist aber, wie man sie betrachtet: Was in der Wirklichkeit vorkommt, nennen wir das Original. Davon machen wir Modelle, um einfacher darüber sprechen und damit umgehen zu können.

Auf dem Foto sind wirkliche Personen zu sehen.

Die Modelle (außen) sind dagegen simple Symbole. Man kann aber mit Hilfe der Symbole bestimmte Dinge diskutieren und sie dann ins Original zurückübertragen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Grundbegriff „Modell“

Original

Modell

Relevante Eigenschaften

Präterierte Eigenschaften

Modell-Abbildung

Abundante Eigensch.

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

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 66

Diese Essenz der „Modelltheorie von Stachowiak“ ist für Informatiker unverzichtbar!

Es gibt so viele Modelle – man sollte sich bei jedem die oben stehenden Fragen stellen.

Erst mit der Antwort darauf kann man das Modell richtig verstehen und beurteilen, ob es gut oder schlecht ist.

Das bezieht sich nämlich darauf, ob es für die Zielperson den Modellzweck gut oder schlecht unterstützt.

Außer den relevanten Attributen, die abgebildet werden, gibt es immer auch präterierte (die man nicht abbildet) und abundante (die das Modell hat, obwohl sie keine Entsprechung im Original haben). Beide Fälle können für Original bzw. Modell nützlich sein. Man kann aber nicht abundante Eigenschaften (z.B.: Material eines Plastikautos) auf das Original zurückübertragen: Das ist aus Blech, das Material spielte offenbar keine Rolle, war „irrelevant“. Aber Plastik ist billig, hat keine scharfen Kanten und ist damit „für Kinder zum Zweck des Spielens“ nützlich, also gut.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

**SOFTWARE
ENGINEERING**

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

```
graph LR; Role["<Rolle>  
z.B.: Entwickler"] --> Task["<Aufgabe>z.B. programmieren"]; Task --> Product["<Produkt>  
z.B. SourceCode"]; Instruction["<Anleitung>  
z.B. CodeConventions"] --> Task;
```

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 67

Zuerst muss man wissen, was man überhaupt modellieren will, was also als relevant für die eigenen Zwecke gilt.

Die nächste Frage heißt dann: wie modellieren wir das (ein Softwareprojekt = Original)? Dafür gibt es eine standardisierte Beschreibungssprache, von der in dieser Vorlesung die wichtigsten Symbole immer wieder verwendet werden.

Unten ist davon ein kleines Beispiel zu sehen. Auf dem Internet findet man die formale Beschreibung von SPEM als UML-subset. Das ist jedoch sehr trockene Lektüre. So genau ist es hier nicht nötig.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



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 Elementen (in eine Hierarchie)
- Guidance
 - Anleitende Info.: Templates, Checklisten, Vorgaben

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 68

Dies sind die fünf Symbole, die uns zunächst ausreichen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

**SOFTWARE
SE
ENGINEERING**

Person und Rolle

Immer genau unterscheiden

Beide Konzepte sind wichtig
Aber sie bedeuten nicht das Gleiche!



Entwickler



Analytiker



Paul



Patrizia

- **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, übernimmt deren Aufgaben usw.
 - Erfüllt Erwartung mehr/weniger
 - Kann eine, mehrere oder keine (definierte) Rolle wahrnehmen

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 69

Nicht verwechseln!

Oft braucht man in Softwareprojekten beides, muss dann aber besonders genau unterscheiden, was gerade gemeint ist: Eine konkrete Person, oder die Rolle, die sie gerade spielt (und die auch von einem Stellvertreter ausgefüllt werden kann).

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

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, Eigenheiten beachten
Beziehungen	Abhängigkeit, Kenntnisse ...	(verschieden)	(vielfältig: meist um Übersicht zu wahren)

Kurt Schneider

Leibniz Universität Hannover

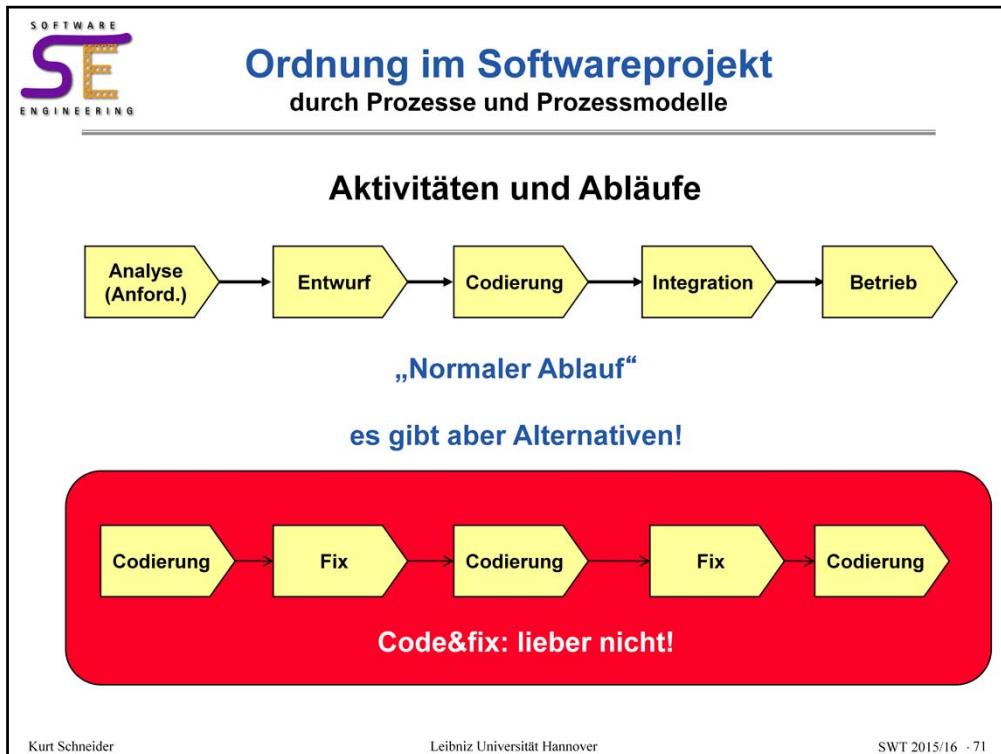
SWT 2015/16 · 70

Ein paar Beispiele, die noch dazu aus dem Bereich der Softwareprojekte stammen.

Daran sollen Sie üben können, die Fragen an Modelle zu stellen und sie präzise zu bearbeiten.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



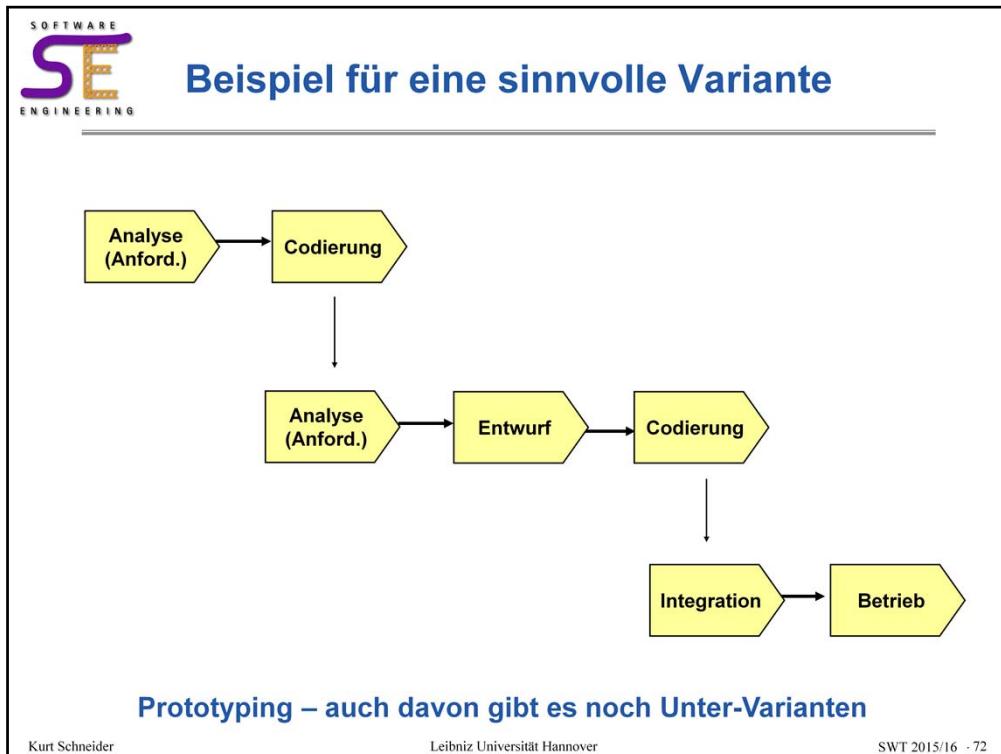
Diese Aktivitäten hängen „irgendwie“ voneinander ab und werden oft in der gezeigten Reihenfolge durchlaufen.

DemwohntheinegewisseinnereLogikinne.

Trotzdem gibt es viele Varianten, die die logischen Erfordernisse beachten und dennoch nicht nur einfach die obige Abfolge zeigen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

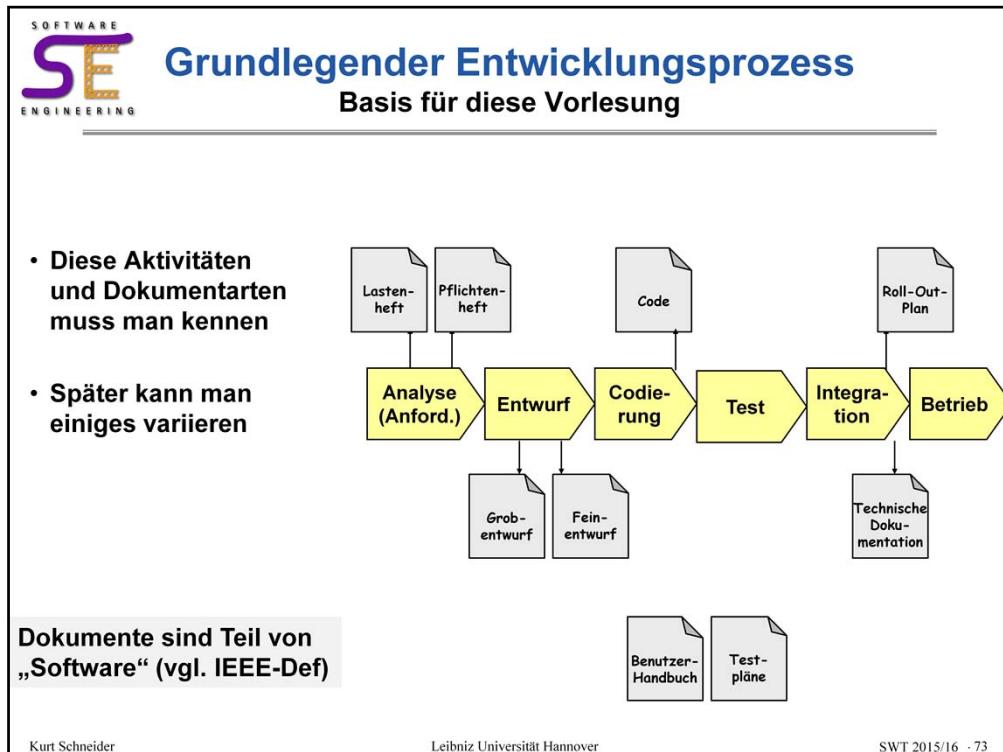
Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



So etwas ist in Ordnung: Man analysiert, programmiert dann gleich und nutzt das Ergebnis als „Prototyp“, um die Analyse zu vertiefen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



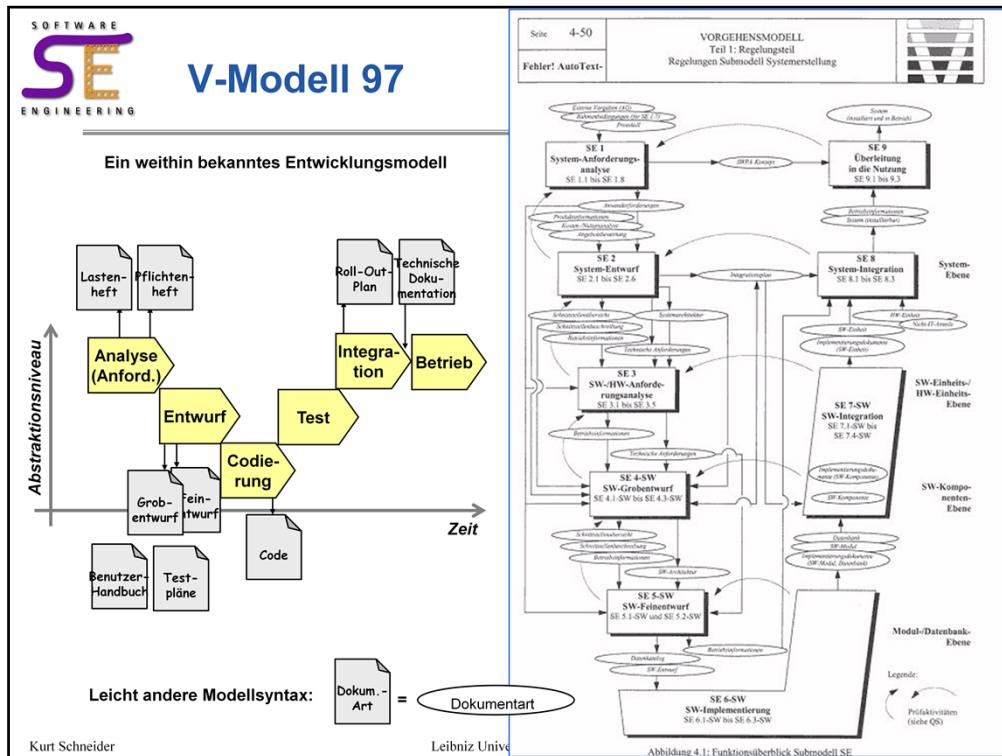
Auf der Basis der Definition von „Software“ muss man sich auch fragen, welche Dokumente (neben Code) hier eine Rolle spielen.

Die Folie zeigt eine Auswahl typischer Dokumentarten mit den allgemein üblichen Bezeichnungen. Es gibt für alle Variationen.

Vergessen Sie aber die beiden am Rand nicht!

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

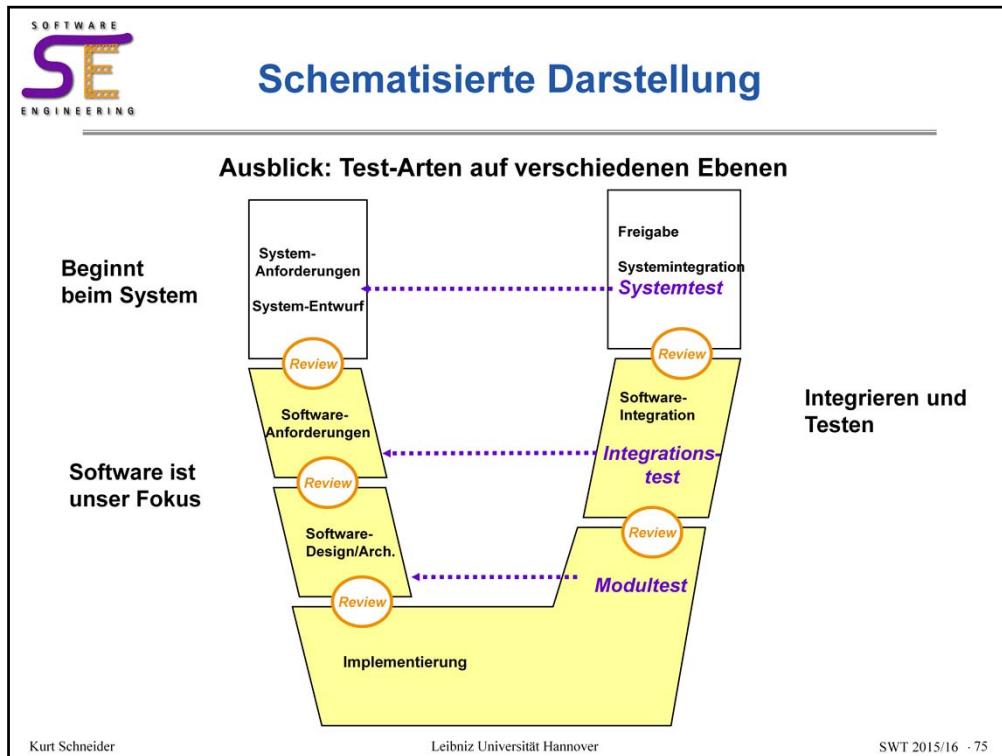
Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Hier zwei Darstellungen des selben bzw. eines ähnlichen Prozesses, aber mit der Achse „Abstraktionsgrad“ dazu. Wenn man die einführt, sieht man eine Art „V“. Danach hat dieser Prozess seinen Namen, V-Modell. Er ist sehr bekannt; inzwischen gibt es eine neuere Version, aber die alte rechts ist noch viel bekannter.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Um die Übersichtlichkeit zu wahren wird der V-Modell-Prozess mitunter auch etwas detaillärmer und schematischer dargestellt, so wie hier. Das ist praktisch eine abgekürzte Schreibweise für den komplizierten Gesamtprozess.