

- Software Richtlinien
- Typisierung
- Vertragsbasierte Programmierung
- Fehlertolerante Programmierung
- Portabilität
- Dokumentation

Portabilität entspricht Plattformunabhängigkeit.

## Portierungsszenarien in der Praxis:

- Architekturportierung (auf andere Hardware)
- Betriebssystemportierungen
- Systemportierungen (Portierung auf andere Geräteklasse)
- Sprachportierungen

Es existieren verschiedene Ebenen, um die Portabilität eines SW Systems zu erhöhen:

- **Portabilität auf Implementierungsebene**  
Wie kann ich ein portables Programm schreiben?
- **Auf Sprachebene**  
Wie kann die Programmiersprache bzw der Compiler in Richtung höhere Portabilität bewegt werden?
- **Auf Systemebene**  
Anpassung der Umgebung an das Programm

*Hier nicht weiter behandelt. Details siehe z.B.*

*Dirk W. Hoffmann: Software-Qualität, 2 Auflage, Springer Vieweg*

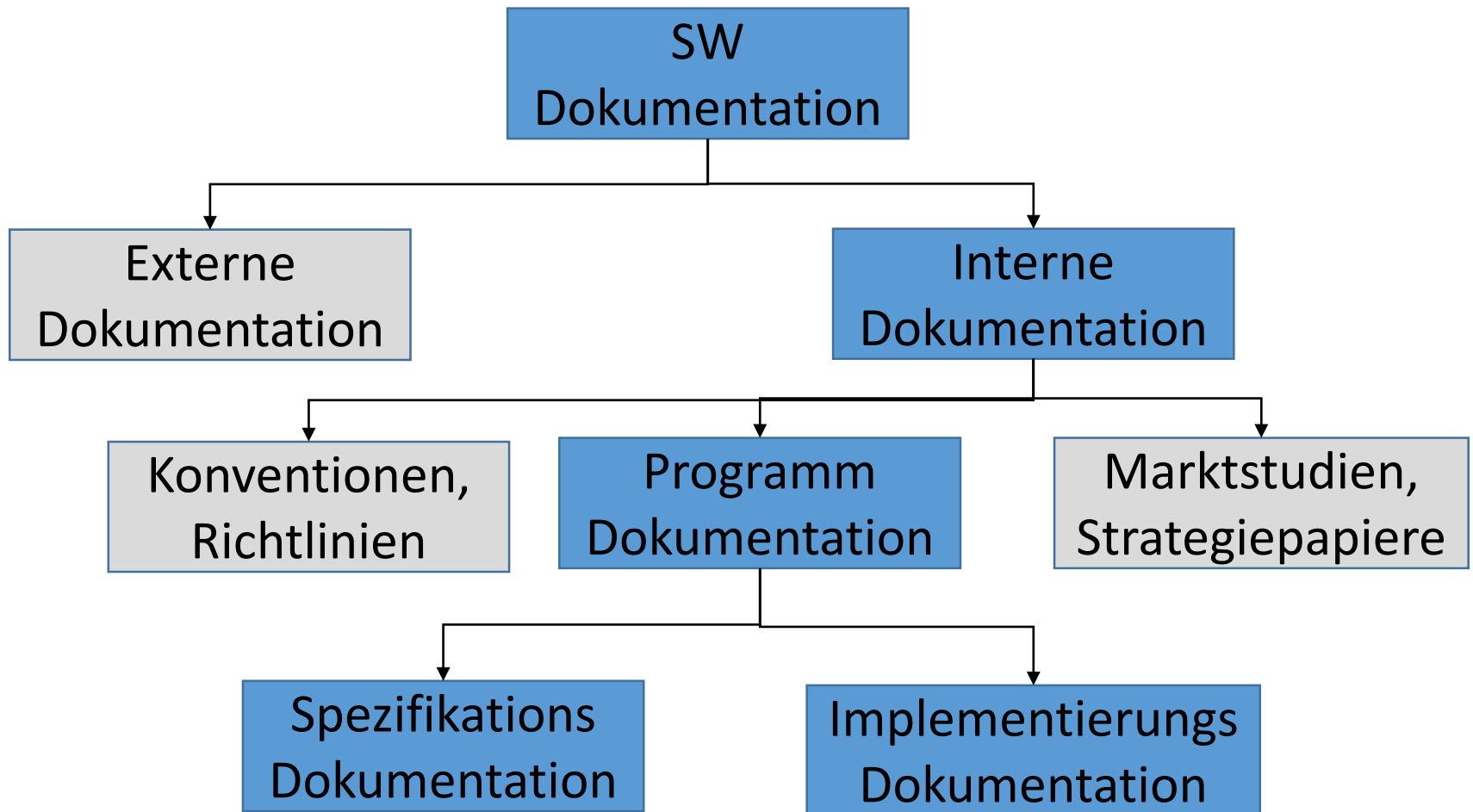
- Software Richtlinien
- Typisierung
- Vertragsbasierte Programmierung
- Fehlertolerante Programmierung
- Portabilität
- Dokumentation

- **Externe Dokumente**  
werden an den Kunden ausgeliefert.
- **Interne Dokumente**  
Nicht für den Kunden zugänglich.

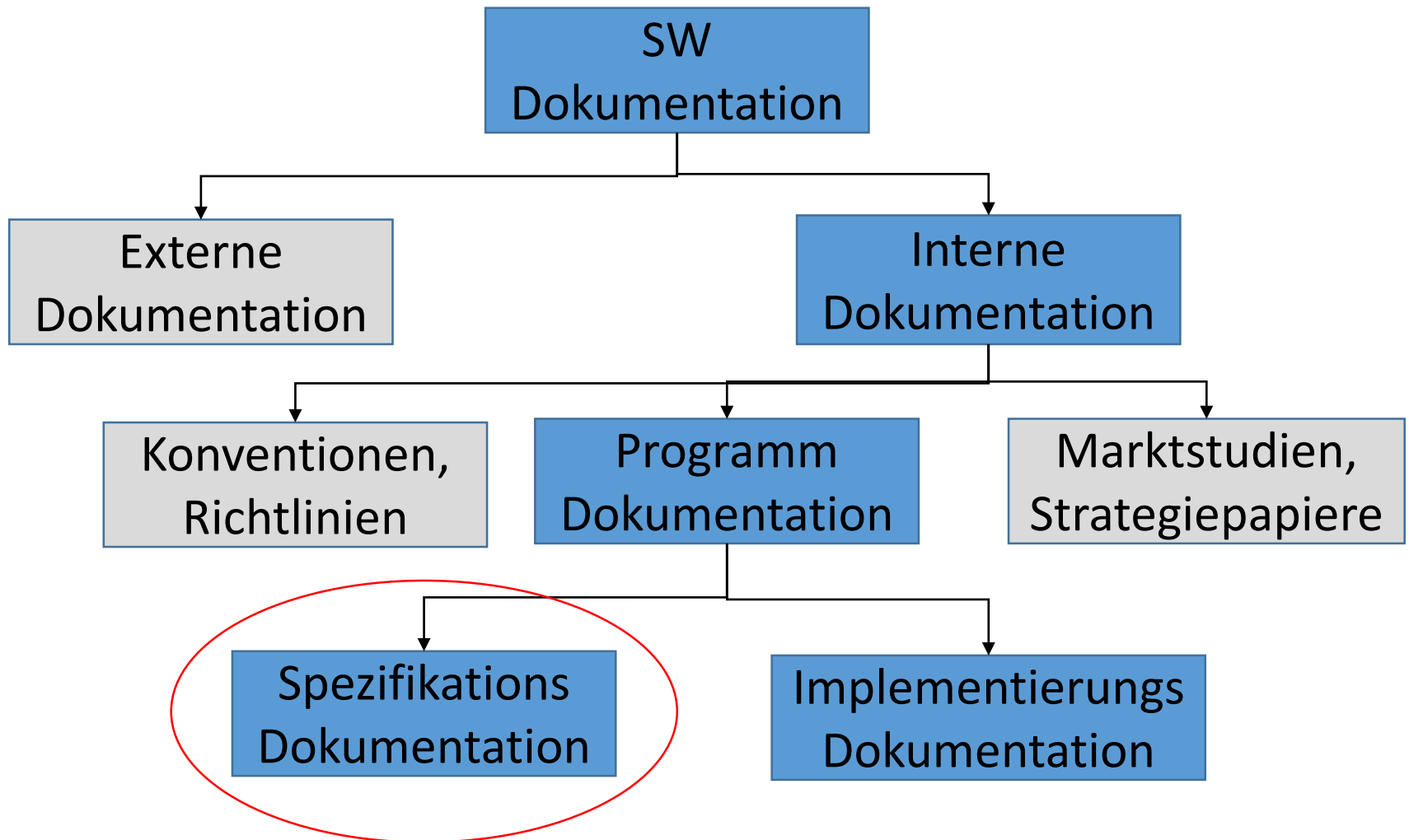
**Hier besprochen: Programmdokumentation (Teil der internen Dokumentation)**

**NB:** Terminologie ist nicht einheitlich: Hier wird die Terminologie von D. Hoffmann verwendet, der Spezifikationsdokumente auch als Dokumentation betrachtet.

# Dichotomie der SW Doku - Ausschnitt



# Dichotomie der SW Doku - Ausschnitt



## Kriterien

- Vollständig
- Eindeutig
- Widerspruchsfrei
- Verständlich



## Drei Varianten:

- Informal
- Semiformal
- Formal

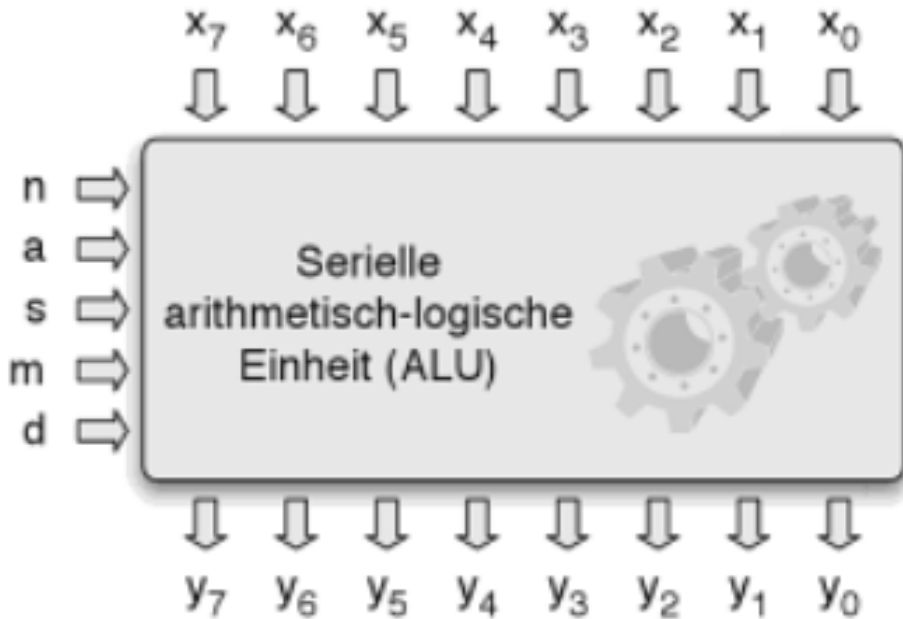
Umgangssprachlich formuliert.

## Offene Fragen durch:

- Implizite Annahmen
- Ignorieren von Sonderfällen
- Sprachliche Ungenauigkeiten

# Beispiel für informelle Spezifikation

## Beispiel ALU



### Legende

#### Ein / Ausgabe:

$(x_7, \dots, x_0)$  : Eingabestrom

$(y_7, \dots, y_0)$  : Ausgabestrom

#### Steuerleitungen:

$n$  : Negation

$a$  : Addition

$s$  : Subtraktion

$m$  : Multiplikation

$d$  : Division

# Informelle Spezifikation - Bsp

Die serielle arithmetisch-logische Einheit (ALU) berechnet aus dem seriellen Eingabestrom ( $x_7, \dots, x_0$ ) den Ausgabestrom ( $y_7, \dots, y_0$ ). Die von der ALU ausgeführte arithmetische Operation wird durch die Steuerleitungen  $n, a, s, m, d$  bestimmt. Für  $n=1$  (negate) negiert die ALU den Eingabewert. Für  $a=1$  (add) berechnet sie die Summe, für  $s=1$  (subtract) die Differenz, für  $m=1$  (multiply) das Produkt und für  $d=1$  (divide) den Quotienten der letzten beiden Eingabewerte.

## Beispiel – offene Fragen

1. Wie werden negative Werte dargestellt?
2. Wie verhält sich die Schaltung bei numerischen Überläufen?
3. Was passiert, wenn alle Steuerleitungen gleich 1 sind?
4. Was passiert, wenn alle Steuerleitungen gleich 0 sind?
5. Wie wird die Division durch 0 behandelt?
6. Welche Ausgabe liegt zum Zeitpunkt 0 an?
7. Welche Werte sind „die letzten beiden“ genau?
8. Was genau bedeutet „der Quotient“?

# Spezifikationsdokumentation Semiformale Spezifikation - Bsp

- **Eingabe**
  - $x[t]$ : Eingabe zum Zeitpunkt  $t$  in 8 Bit-Zweierkomplementdarstellung
- **Ausgabe**
  - $y[t]$ : Ausgabe zum Zeitpunkt  $t$  in 8-Bit Zweierkomplementdarstellung

- **$n, a, s, d, m$ ; Steuerleitungen**

- **Verhalten:**

$$y[0] = \begin{cases} -x[0] & \text{für } n=1 \text{ und } a=s=m=d=0 \\ 0 & \text{sonst} \end{cases}$$

$$y[n+1] = \begin{cases} -x[n+1] & \text{für } n=1 \text{ und } a=s=m=d=0 \\ x[n]+x[n+1] & \text{für } a=1 \text{ und } n=s=d=m=0 \\ x[n]-x[n+1] & \text{für } s=1 \text{ und } a=n=d=m=0 \\ x[n] \times x[n+1] & \text{für } m=1 \text{ und } n=s=a=d=0 \\ x[n] / x[n+1] & \text{für } d=1 \text{ und } n=a=s=m=0 \\ 0 & \text{sonst} \end{cases}$$

# Weiteres Beispiel

Anforderung an ein Programm in einer Musikschule, um Leihinstrumente zu verwalten:

„Die Anzahl der Saiteninstrument ist doppelt so hoch wie die Anzahl der Blasinstrumente. Die Summe der Saiten- und der Blasinstrumente liegt zwischen 5 und 15.“

→ Formulierung in mathematischen Formeln:

$$\text{Anzahl}(\text{SaitenInstr.}) = 2 \times \text{Anzahl}(\text{Blasinstrumente})$$

$$5 < \text{Anzahl}(\text{SaitenInstr.}) + \text{Anzahl}(\text{Blasinstrumente}) < 15$$

**Aufgabe: Programmieren Sie folgende Zahlenreihe bis zu Ihrem 8. Element:**

**Spezifikation:**

- Die Zahlenreihe beginnt mit einer 3.
- Jede Zahl der Folge ist um 1 grösser als die Hälfte der nächsten Zahl.

**Bessere Spezifikation:**

$$Z(1) = 3$$

$$Z(n+1) = 2 \text{ mal } (Z(n) - 1)$$



# Umsetzung in C

```
#include <stdio.h>
```

```
int main() {
```

```
    int zahl = 3;
```

```
    int ende = 0;
```

```
    int zaehler;
```

```
    printf("wie weit soll es denn gehen?\n");
```

```
    scanf("%d", &ende);
```

```
    printf("die %ite Zahl ist %i\n", 1, zahl);
```

```
    for(zaehler = 0; zaehler < ende-1; zaehler ++){
```

```
        zahl = 2 * (zahl-1);
```

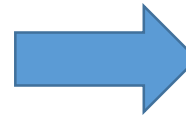
```
        printf("die %ite Zahl ist %i\n", zaehler+2, zahl);
```

```
    }
```

```
    getchar();
```

```
    return 0;
```

```
}
```



```
wie weit soll es denn gehen?  
8  
die 1te Zahl ist 3  
die 2te Zahl ist 4  
die 3te Zahl ist 6  
die 4te Zahl ist 10  
die 5te Zahl ist 18  
die 6te Zahl ist 34  
die 7te Zahl ist 66  
die 8te Zahl ist 130
```

## Weiteres Beispiel

Das Laufwerk von Nicks Computer hat die doppelte Kapazität wie das von Alfs Computer. Zusammen haben sie 240 GB.

Schreiben Sie in Programm, das die die Kapazität der einzelnen Laufwerke berechnet.

**Textuell schwer zu lösen. Mathematisch sehr einfach:**

(1)  $x + y = 240$

(2)  $x = 2 * y$

**→ aus (1):  $3y = 240 \rightarrow y = 80$**

**In (2) →  $x = 160$**

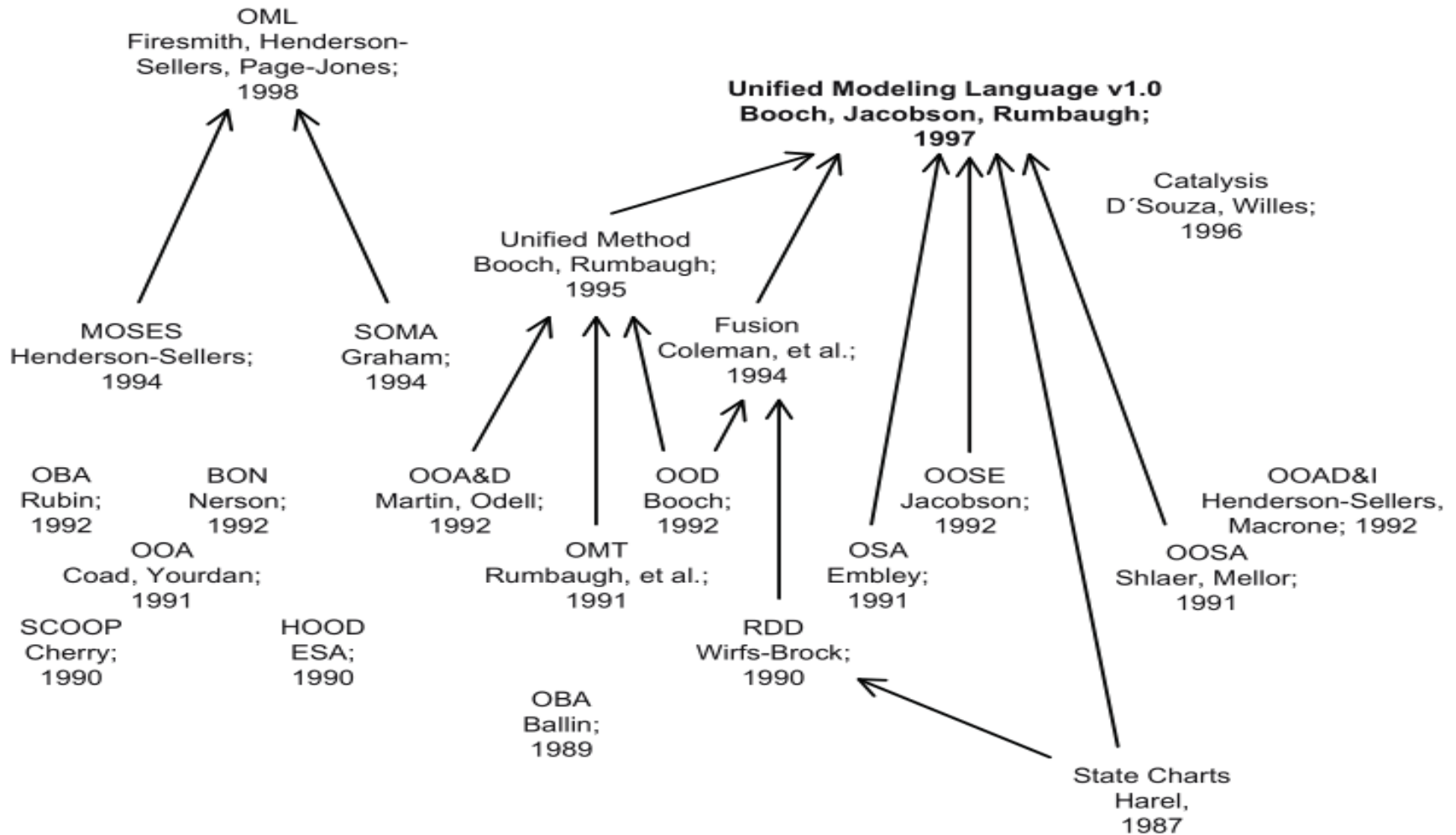
- Verwendung definierter Formalismen.
- Vorteil: Zweideutigkeiten vermieden, dennoch gut lesbar.

Prominentes Beispiel einer semi formalen Spezifikationssprache: **UML**

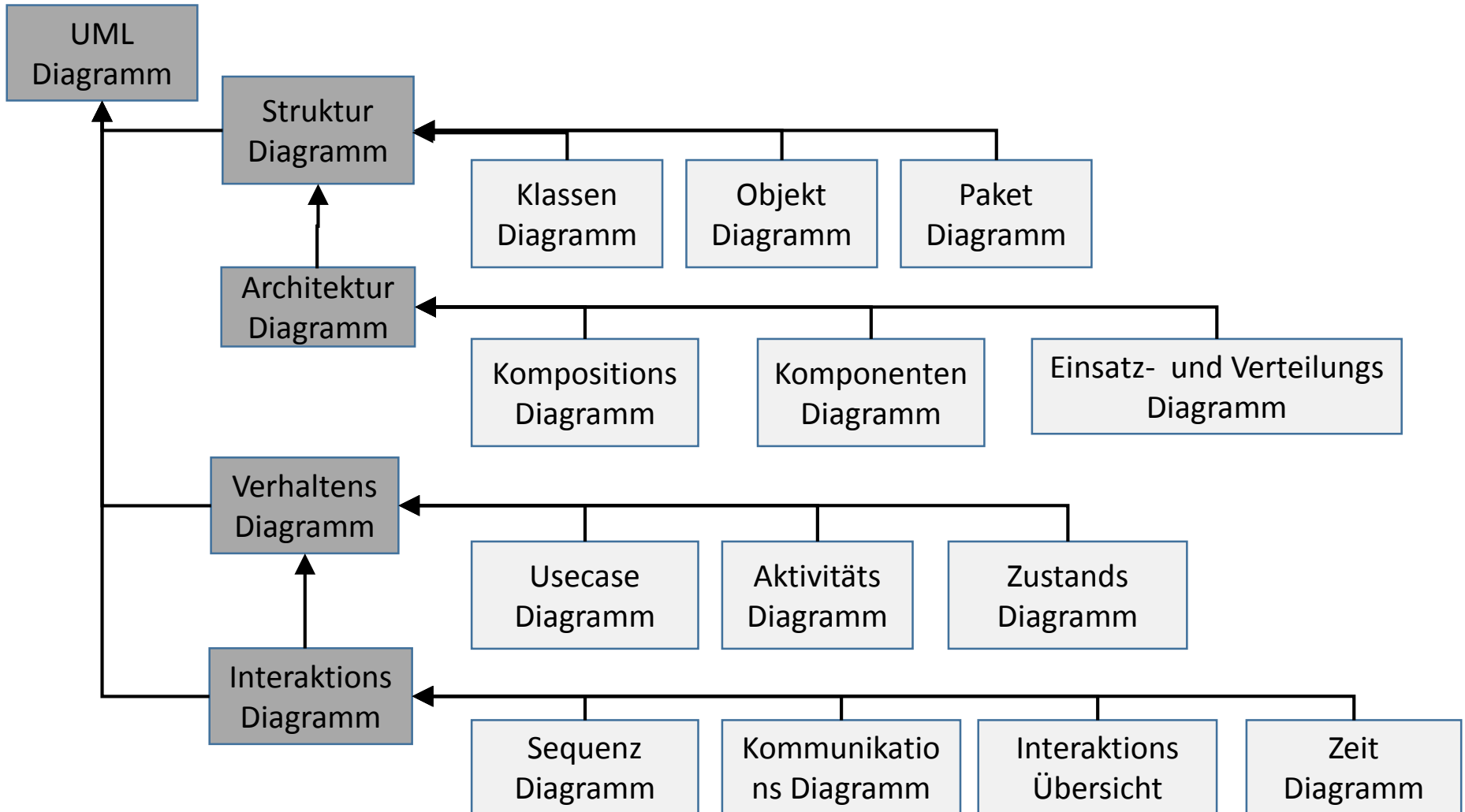
# UML Diagramme

Die Unified Modeling Language, kurz UML, ist eine grafische Darstellungsform zur **Visualisierung, Spezifikation, Konstruktion** und **Dokumentation** von (Software) Systemen. Sie bietet ein Set an standardisierten Diagrammtypen, mit denen komplexe Sachverhalte, Abläufe und Systeme einfach, übersichtlich und verständlich dargestellt werden können.

# Uml Historie



# UML Diagrammtypen



- ➔ **Übungsbeispiel zu Activity Diagramm ➔**  
Bsp. zu Materialwirtschaft.
- ➔ **Übungsbeispiel zu Usecase Diagramm ➔**  
Bsp. zu Fahrradshop.
- ➔ **Übungsbeispiel Klassen Diagramm ➔ Bsp.**  
Videoverleih.
- ➔ **Übungsbeispiel Zustandsdiagramm ➔ Bsp**  
Tresor

- Lässt keinen Interpretationsspielraum.
- Wird schnell extrem komplex, sogar bei einfachen Sachverhalten.
- Spielt in der Praxis kaum eine Rolle.

## Beispiele:

- Spezifikationssprache Z
- Vienna Development Method

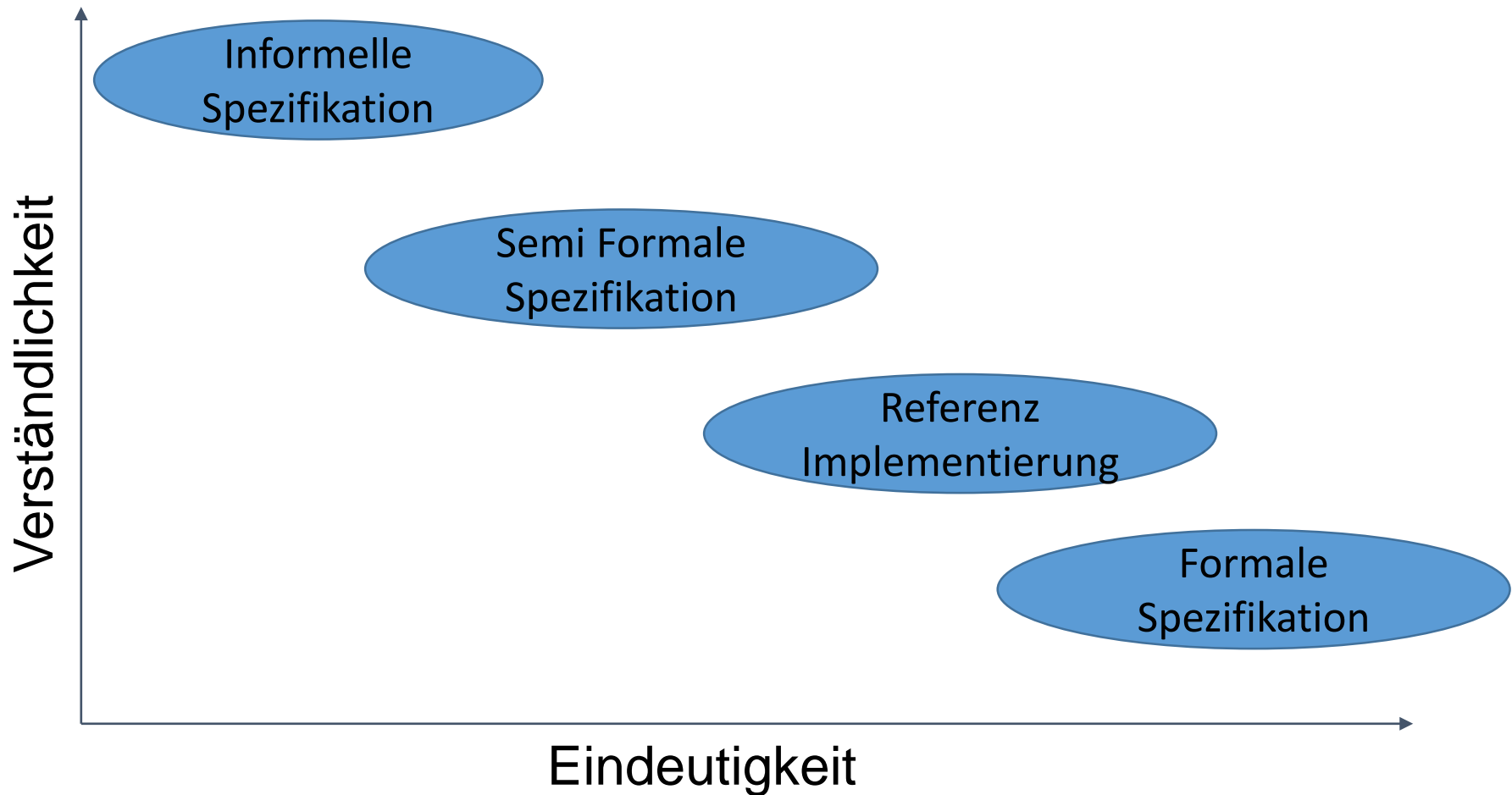


Spezifikation wird durch ein Programm ersetzt, dessen Verhalten als richtig definiert wird.

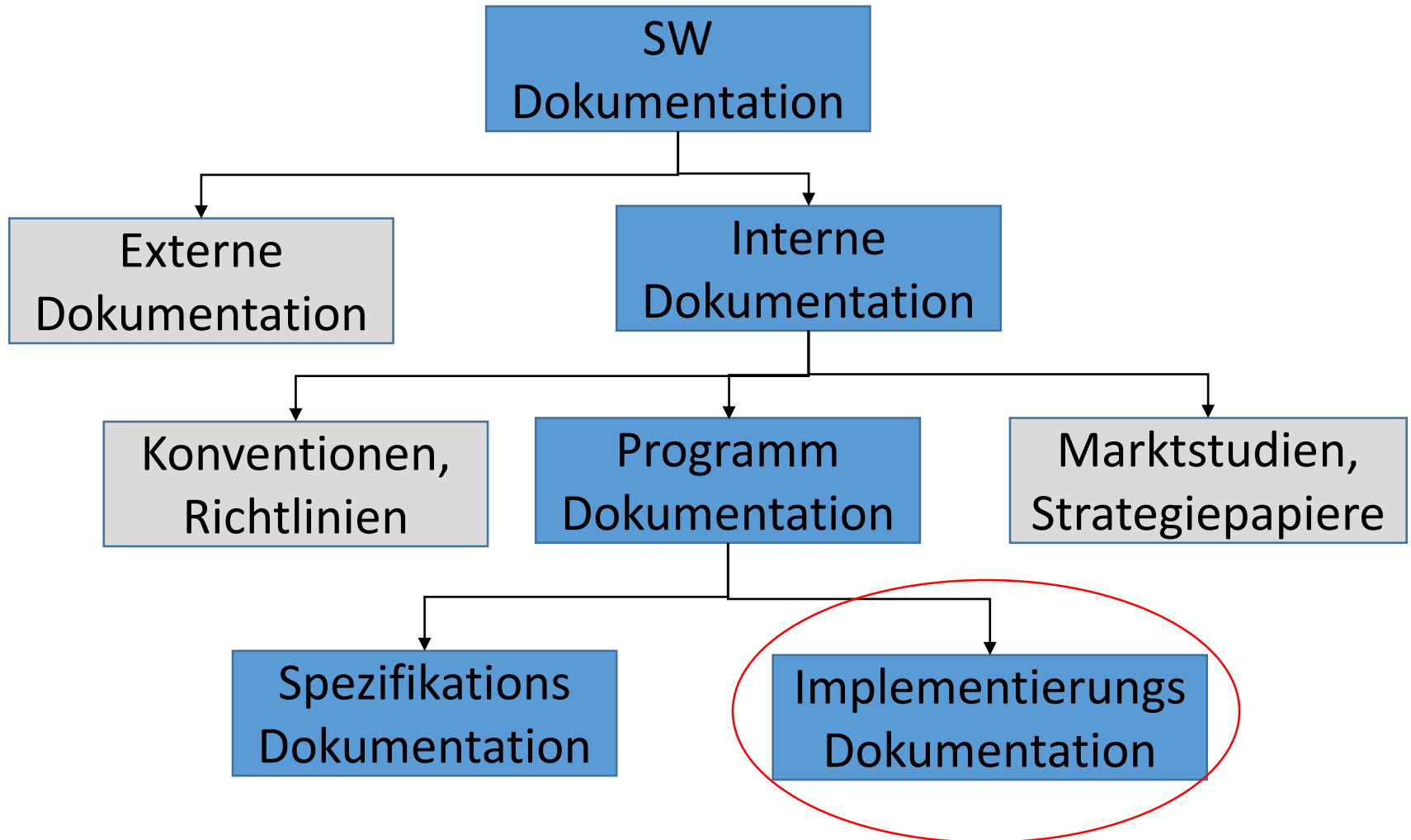
➔ Wenn sich das neue entwickelte Programm genauso verhält wie die Referenzimplementierung, dann entspricht sie der Spezifikation.

- Anwendung der Referenzimplementierung  
z.B. beim Refactoring: Neues Verhalten ist  
gleich altem Verhalten.
- Referenzimplementierung oftmals nicht  
ausreichend, da zwar das Verhalten  
nachprüfbar ist, aber nicht, warum sich ein  
System so verhalten soll.

# Spezifikationstechniken im Vergleich



# Dichotomie der SW Doku



# Implementierungsdokumentation

Beschreiben, **wie** die Spezifikation umgesetzt wurde.

**Oftmals:** „Our code is our documentation“

➔ Nicht befriedigend, Nicht ausreichend

**Hier behandelt:** Integrierte Dokumentation, d.h. Code Dokumentation.

## Teil des Agilen Manifests:

...

Funktionierende Software mehr als  
umfassende Dokumentation

...

➔ In agilen Projekten führt das oft dazu, dass  
Dokumentation nicht Teil der Iteration und der  
DoD ist

# Code Doku - Bsp

```
int ack(int n, int m)
{
    while (n!=0) {
        if (m==0) {
            m=1;
        } else {
            m=ack(m, n-1);
        }
        n--;
    }
    return m+1;
}
```

Wo ist der Fehler?

# Code Doku – Bsp Doku als Selbstzweck

```

/*
Funktion int ack(int n, int m)
Autor: Irgendwer
Datum: 15.3.2013
Revision History:
    12/15/2006: While Iteration eingefügt
    13/01/2008: Funktionsname geändert
*/
int ack(int n, int m)
{
    /*solange die erste Variable nicht null ist ...*/
    while (n!=0)
    {
        /*teste zweite Variable auf 0 */
        if (m==0)
        {
            m=1;
        }
        else
        {
            /*hier erfolgt der rekursive Aufruf*/
            m=ack(m,n-1);
        }
        n--;
    }
    /*Liefere Ergebnis zurueck*/
    return m+1;
}

```

Diese Information ist im  
Source Verwaltungstool  
enthalten.

Trivial und überflüssig

Wo ist der Fehler?  
Immer noch schwierig  
zu finden



# Sinnvolle Doku - Bsp

```
/*  
Funktion int ack(int n, int m)  
Autor: Irgendwer  
Beschreibung: Berechnet die Ackermann Funktion  
Definition der Ackermann Funktion:  
(1) ack(0,m) = m+1;           m>=0  
(2) ack(n,0) = ack(n-1,1);     n>0  
(3) ack(n,m) = ack(n-1,ack(n,m-1)); m,n>0
```

Hinweis: Die Ackermann wächst stärker als die Exponentialfunktion  
Bsp:  
ack(3,1)=13  
ack(4,1)=65533  
ack(5,1) = ca 20stellige Zahl

```
*/  
int ack(int n, int m)  
{  
    while (n!=0)  
    {  
        if (m==0)  
        {  
            /*Fall (2)*/  
            m=1;  
        }  
        else  
        {  
            /*Fall (3)*/  
            m=ack(m,n-1);  
        }  
        n--  
    }  
    /*Fall (1)*/  
    return m+1;  
}
```

Spezifikation, was die  
Funktion tut

Finden Sie den  
Fehler jetzt?



Rekursiver Aufruf ist falsch:  
statt ack(m,n-1) müsste es  
heißen: ack(n,m-1)

# Regeln für die Kommentierung von Quellcode

1. Make the code as clear as possible to reduce the need for comments!
2. Never repeat information in a comment that is already available in the code!
3. Where a comment is required, make it concise and complete!
4. Use proper grammar and spelling in comments!
5. Make comments visually distinct from the code!
6. Structure comments in headers so that information can be automatically extracted by a tool!

*ADA (1995): A DA 95 Quality and Style: Guidelines for Professional Programmers. Software Productivity Consortium, SPC-94093-CMC, Version 01.00.10*

- Vermeiden Sie Dokumentation als Selbstzweck.
- Dokumentation zielt auf einen Adressaten – denken Sie daran, welche Information dieser Adressat benötigt.
- Es kann sinnvoll sein, zu dokumentieren, warum etwas nicht implementiert ist.

Bis **jetzt**: Behandlung dessen, welche Codeteile wie dokumentiert werden.

**Jetzt**: Wer hat in welcher Form Zugriff auf die Dokumentation.

# Wer benötigt Code Dokumentation?

- Jeder, der den Code benutzt
  - ➔ Entwickler, die Ihre Komponenten/Klassen etc benutzen
  - ➔ Dokumentation ist relevant für den Nutzer, ohne dass er direkt in den Code schauen muss (z.B. weil er ein jar File bei sich einbindet).
- Jeder, der den Code weiterentwickelt
  - ➔ Entwickler, die auf Ihrem Code aufbauen

# Bsp für magere API Dokumentation

All Classes

**Packages**

- org.apache.lucene
- org.apache.lucene.analysis
- org.apache.lucene.analysis.token
- org.apache.lucene.codecs
- org.apache.lucene.codecs.com
- org.apache.lucene.codecs.lucene
- org.apache.lucene.codecs.lucene

All Classes

- AfterEffect
- AfterEffect.NoAfterEffect
- AfterEffectB
- AfterEffectL
- AlreadyClosedException
- Analyzer
- Analyzer.GlobalReuseStrategy
- Analyzer.PerFieldReuseStrategy
- Analyzer.ReuseStrategy
- Analyzer.TokenStreamComponent
- AnalyzerWrapper
- ArrayUtil
- AtomicReader
- AtomicReaderContext
- Attribute
- AttributeImpl
- AttributeReflector
- AttributeSource
- AttributeSource.AttributeFactory
- AttributeSource.State
- Automaton
- AutomatonProvider
- AutomatonQuery
- AveragePayloadFunction
- BaseCompositeReader
- BasicAutomata

## grow

```
public static byte[] grow(byte[] array)
```

## shrink

```
public static byte[] shrink(byte[] array,
    int targetSize)
```

## grow

```
public static boolean[] grow(boolean[] array,
    int minSize)
```

## grow

```
public static boolean[] grow(boolean[] array)
```

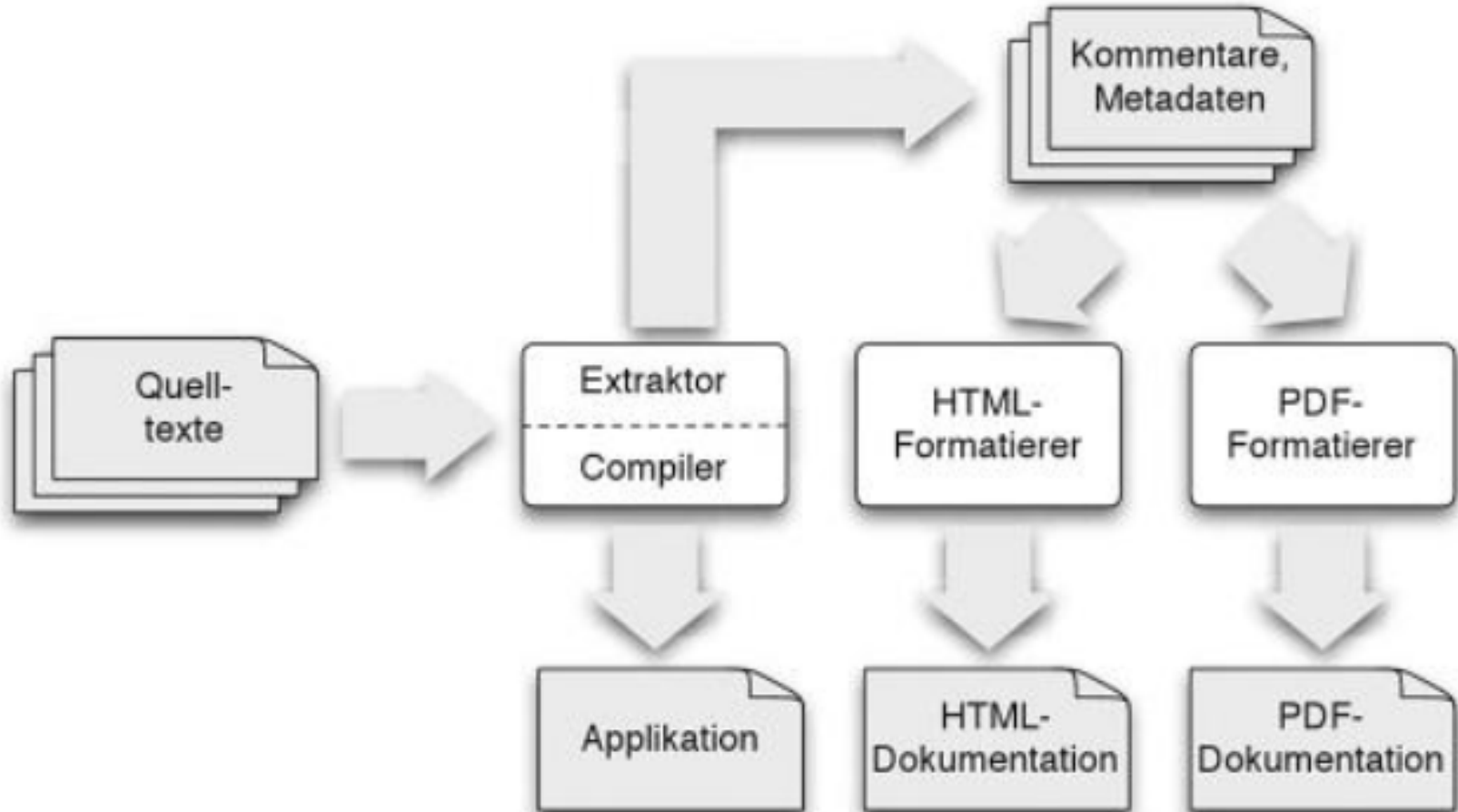
## shrink

```
public static boolean[] shrink(boolean[] array,
    int targetSize)
```

## grow

```
public static char[] grow(char[] array,
    int minSize)
```

# Extraktion von Dokumentation



- ➔ Dokumentation ist auch von Personen lesbar, die keinen Zugriff auf den Code haben.
- ➔ Dokumentation ist ohne Code lesbar.
- ➔ Dokumentation ist in aufbereiteter Form lesbar (HTML oder Pdf).
- ➔ Querverweise sind im Kommentar leichter nachzuverfolgen.

Bsp: Java SE 6, API Specification:

<http://docs.oracle.com/javase/6/docs/api/>

# Bsp: Java SE 6, API Specification

<http://docs.oracle.com/javase/6/docs/api/>

**Java™ Platform  
Standard Ed. 6**

[All Classes](#)

Packages

- [java.applet](#)
- [java.awt](#)
- [java.awt.color](#)
- [java.awt.datatransfer](#)

**All Classes**

- [AbstractAction](#)
- [AbstractAnnotationValueV](#)
- [AbstractBorder](#)
- [AbstractButton](#)
- [AbstractCellEditor](#)
- [AbstractCollection](#)
- [AbstractColorChooserPan](#)
- [AbstractDocument](#)
- [AbstractDocument.Elemen](#)
- [AbstractElementVisitor6](#)
- [AbstractExecutorService](#)
- [AbstractInterruptibleChann](#)
- [AbstractLayoutCache](#)
- [AbstractLayoutCache.Nod](#)
- [AbstractList](#)
- [AbstractListModel](#)
- [AbstractMap](#)
- [AbstractMap.SimpleEntry](#)
- [AbstractMap.SimpleImmute](#)
- [AbstractMarshallerImpl](#)
- [AbstractMethodError](#)

Standard Edition 8 Documentation  
acle.com/javase/8/docs/

**Overview** Package Class Use **Tree** [Deprecated](#) [Index](#) [Help](#)

PREV NEXT [FRAMES](#) [NO FRAMES](#)

**Java™ Platform, Standard Edition 6  
API Specification**

This document is the API specification for version 6 of the Java™ Platform, Standard Edition.

See:  
[Description](#)

Packages	
<a href="#">java.applet</a>	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
<a href="#">java.awt</a>	Contains all of the classes for creating user interfaces and for painting graphics and images.
<a href="#">java.awt.color</a>	Provides classes for color spaces.
<a href="#">java.awt.datatransfer</a>	Provides interfaces and classes for transferring data between and within applications.
<a href="#">java.awt.dnd</a>	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
<a href="#">java.awt.event</a>	Provides interfaces and classes for dealing with different types of events fired by AWT components.
<a href="#">java.awt.font</a>	Provides classes and interface relating to fonts.
<a href="#">java.awt.geom</a>	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometr



## Java unterstützt 3 Typen von Kommentaren:

- `/* text */`: Der Compiler ignoriert alles zwischen `/*` und `*/`
- `//text`: Der Compiler ignoriert alles von `//` bis Zeilenende
- `/**`  
documentation  
`*/`  
sogenannter „doc comment“: wird von javadoc genutzt.

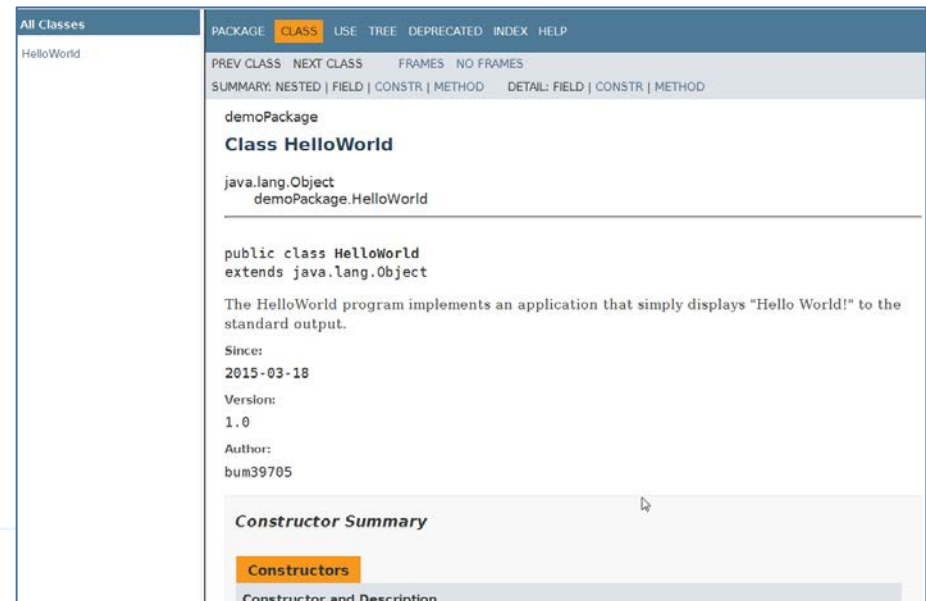
# Bsp javadoc

```
package demoPackage;
/**
 * The HelloWorld program implements an application that
 * simply displays "Hello World!" to the standard output.
 *
 * @author Michael Bulenda
 * @version 1.0
 * @since 2015-03-18
 */
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

Javadoc aus eclipse generiert

Docu: <http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html>  
<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>  
 Siehe auch [http://openbook.rheinwerk-verlag.de/javainsel9/javainsel\\_05\\_014.htm#mj1b008761e7e60dd49fcab7bf8de0d2cd](http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_05_014.htm#mj1b008761e7e60dd49fcab7bf8de0d2cd)

# Was ist Javadoc?

Javadoc ist ein Werkzeug, das eine standardisierte Dokumentation für Java unterstützt.

Javadoc ist integraler Bestandteil des JDK.  
Es ist nach der Installation im gleichen Verzeichnis wie der Java Compiler javac.

# Wie funktioniert javadoc?

Javadoc liest Deklarationen und Doc Comments aus den Quellcodedateien.

Es können auch weitere Dateien eingebunden werden.

Aus diesen Informationen wird eine HTML basierte Dokumentation erzeugt.

## Doc Comments:

```
/**
```

```
*Das ist ein Doc Comment
```

```
*/
```

# Welche Dateien werden von javadoc berücksichtigt?

Javadoc erstellt die Dokumentation aus folgenden Dateien:

- Quellcodedateien
- Package Bemerkungen
- Overview Bemerkungen
- diverse weiteren Dateien

Doc Comments in Quellcodedateien können vor

- Klassen
- Konstruktoren
- Datentypen
- Methoden stehen.

Doc Comments werden aber nur beachtet, wenn sie genau vor einer Deklaration stehen!

Ausgelesen als HTML → es kann HTML Syntax enthalten sein

# Package Bemerkungen

Für jedes Package kann eine Beschreibung geschrieben werden.

Es muss eine Datei erstellt werden die den Namen `package.html` trägt. Diese Datei muss in das Verzeichnis des Packages abgelegt werden.

Die Datei `package.html` benötigt kein Doc Comments, sie enthält nur HTML.

Javadoc bindet diese Datei automatisch ein.

# Overview Bemerkungen

In diese HTML Datei kommen Beschreibungen die für die ganze Anwendung gelten.

Sie ist wie die Datei package.html ohne Doc Comments.

Diese Datei braucht keinen speziellen Namen. Wenn Javadoc ausgeführt wird, muss dieser Dateiname angegeben werden.



# Einbindung weiterer Dateien

Diese Dateien werden von Javadoc in das Ausgabeverzeichnis der Dokumentation kopiert.

Dies können weitere HTML Dateien sein, Applets, Beispielcode aber auch Grafiken die in die Dokumentation eingebunden werden.

Diese Dateien müssen in einem Verzeichnis unterhalb des Packages abgelegt sein, welches ../doc-files/ heißen muss.

Zugriff kann innerhalb der Doc Comments so erfolgen:

```
/**
```

```
* This button looks like this:
```

```
* 
```

```
*/
```

# Javadoc Tags

Es gibt spezielle Schlüsselwörter welche als Tags bezeichnet werden. Tags werden mit einem beginnenden @ gekennzeichnet.

author

exception

linkplain

see

serialField

value

deprecated

inheritDoc

param

serial

since

version

docRoot

link

return

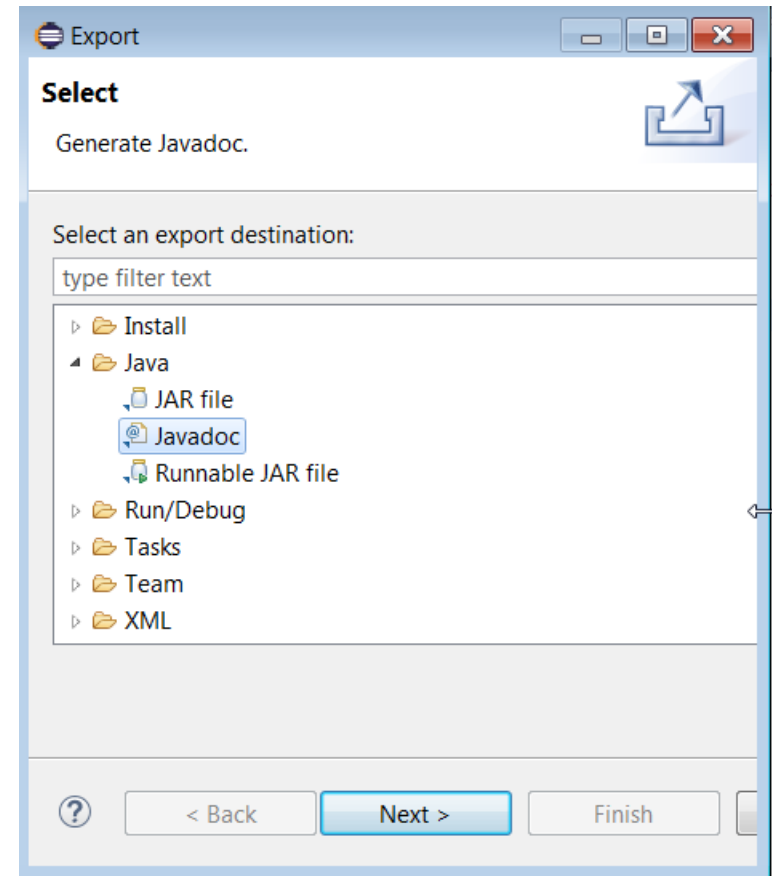
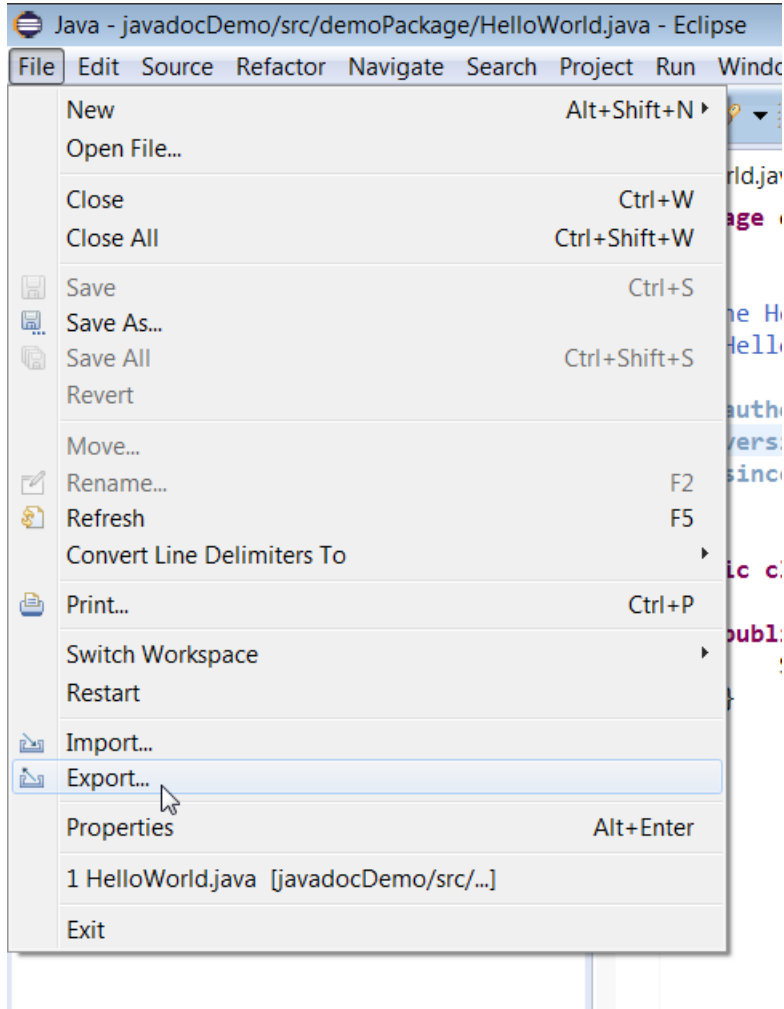
serialData

throws

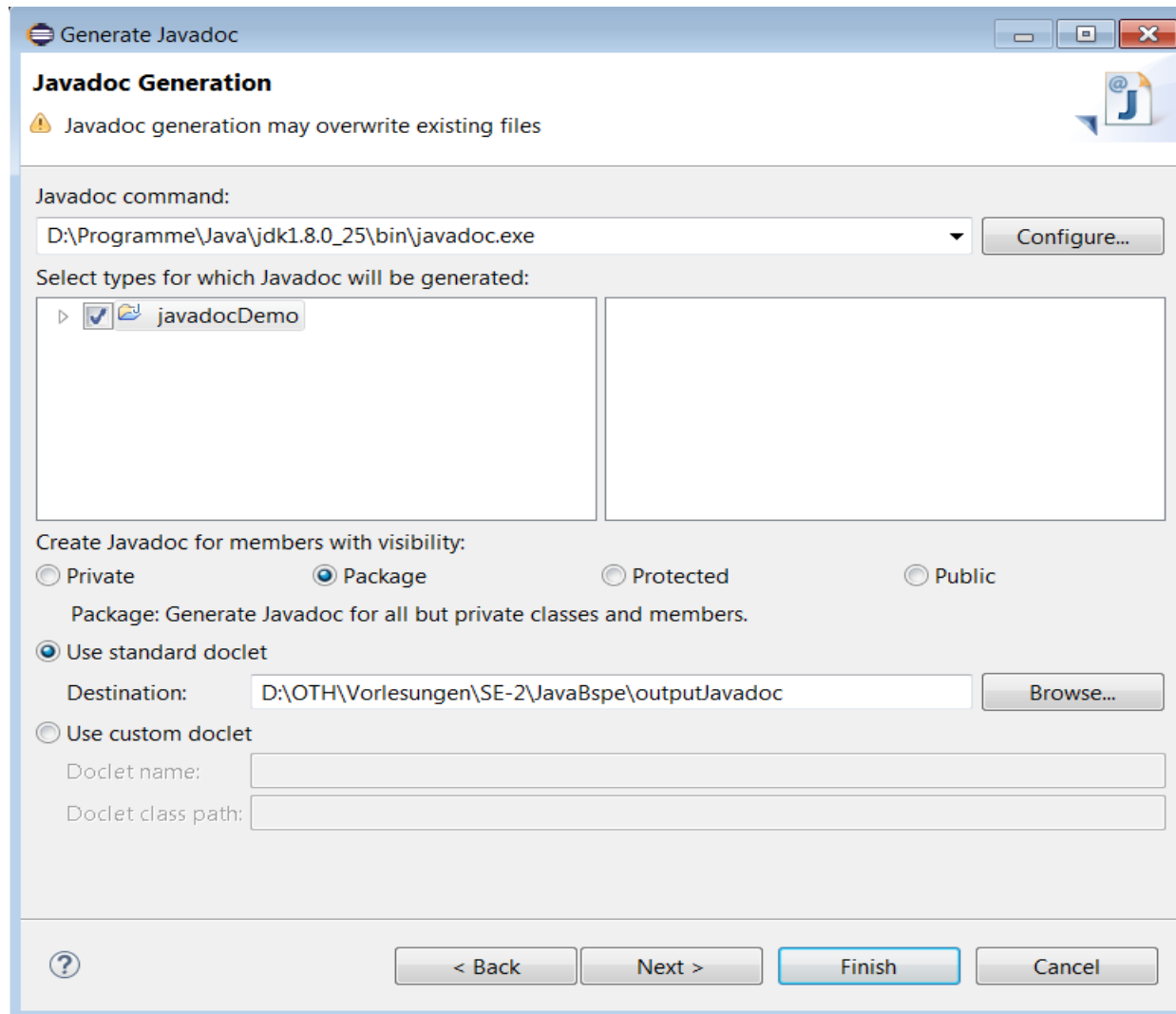
# Die wichtigsten Javadoc Tags erklärt

Kommentar	Beschreibung	Beispiel
<code>@param</code>	Beschreibung der Parameter	<code>@param a A Value.</code>
<code>@see</code>	Verweis auf ein anderes Paket, einen anderen Typ, eine andere Methode oder Eigenschaft	<code>@see java.util.Date @see java.lang.String#length()</code>
<code>@version</code>	Version	<code>@version 1.12</code>
<code>@author</code>	Schöpfer	<code>@author Christian Ullenboom</code>
<code>@return</code>	Rückgabewert einer Methode	<code>@return Number of elements.</code>
<code>@exception/@throws</code>	Ausnahmen, die ausgelöst werden können	<code>@exception NumberFormatException</code>
<code>{@link Verweis}</code>	Ein eingebauter Verweis im Text im Code-Font. Parameter wie bei <code>@see</code>	<code>{@link java.io.File}</code>
<code>{@linkplain Verweis}</code>	Wie <code>{@link}</code> , nur im normalen Font	<code>{@linkplain java.io.File}</code>
<code>{@code Code}</code>	Quellcode im Code-Zeichensatz – auch mit HTML-Sonderzeichen	<code>{@code 1 ist &lt; 2}</code>
<code>{@literal Literale}</code>	Maskiert HTML-Sonderzeichen. Kein Code-Zeichensatz	<code>{@literal 1 &lt; 2 &amp;&amp; 2 &gt; 1}</code>
<code>@category</code>	Für Java 7 oder 8 geplant: Vergabe einer Kategorie	<code>@category Setter</code>

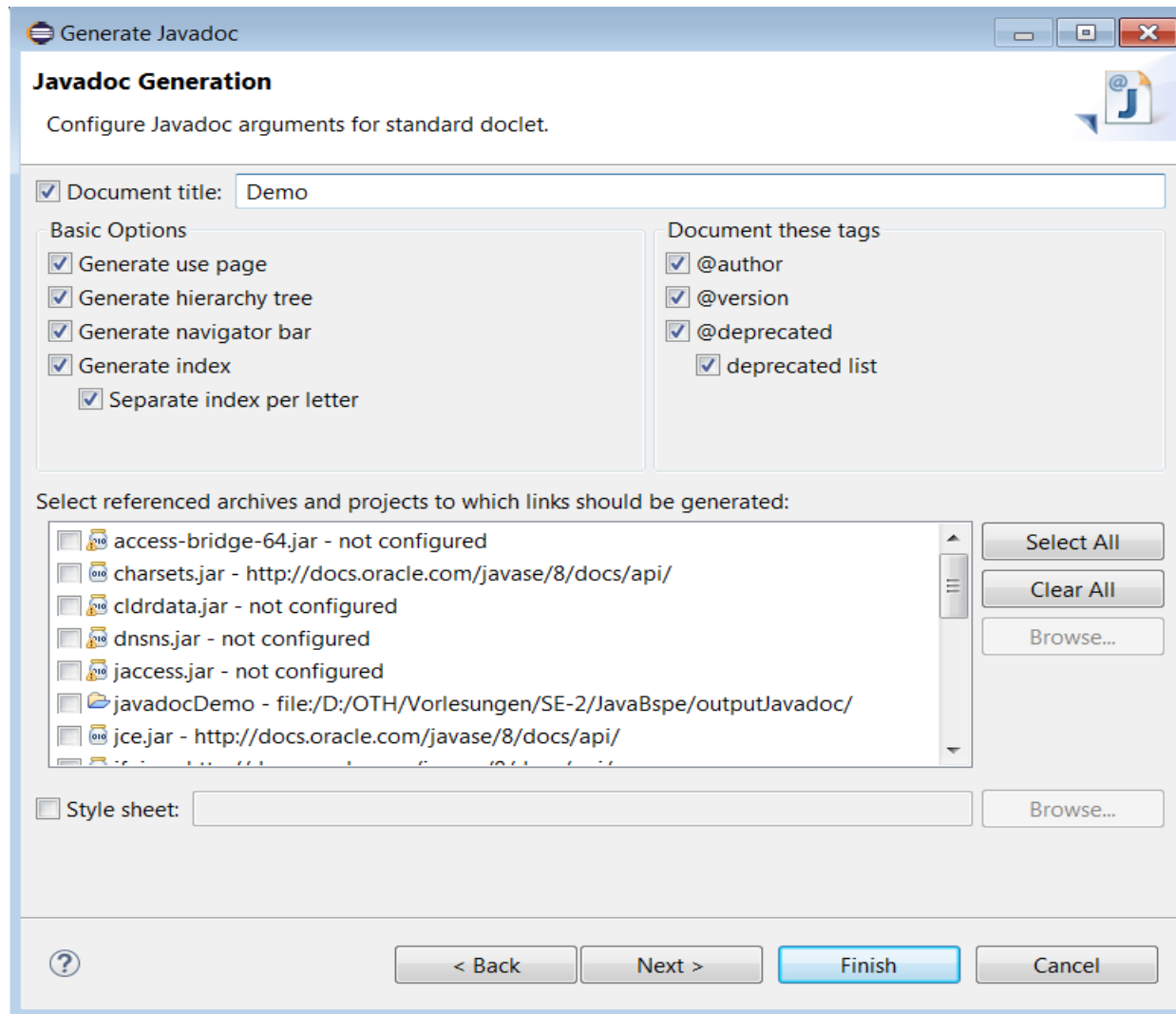
# Aufruf von javadoc in Eclipse



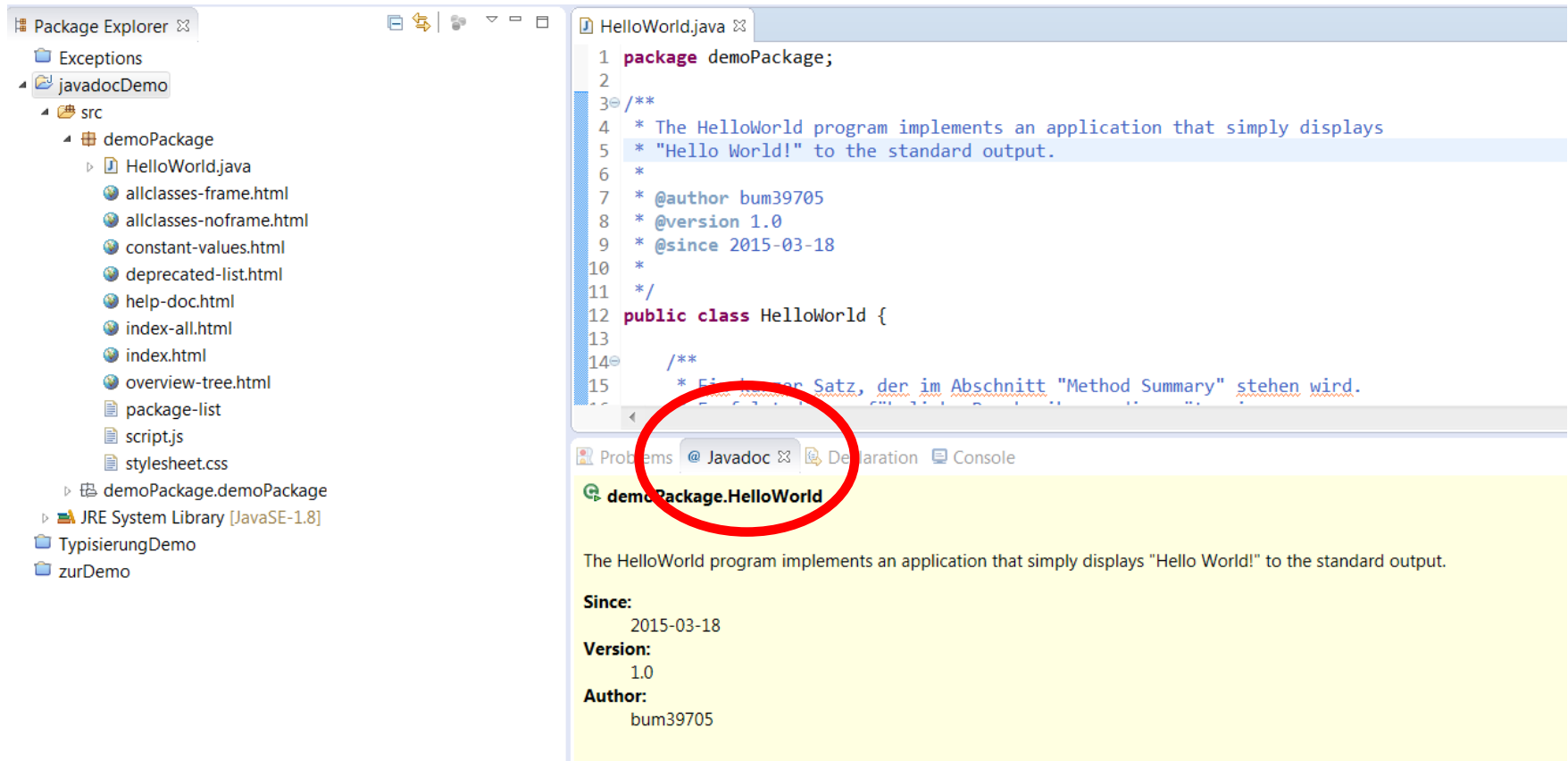
# Aufruf von javadoc in Eclipse



# Aufruf von javadoc in Eclipse



# Vorschau von javadoc in Eclipse



The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure, including the 'javadocDemo' package and its 'src' directory. The 'HelloWorld.java' file is selected. The main editor shows the source code of 'HelloWorld.java', which includes a package declaration, a class comment, and a public class definition. A red circle highlights the 'Javadoc' tab in the bottom right corner, which displays the generated javadoc for 'demoPackage.HelloWorld'. The javadoc content includes the class comment and the 'Since', 'Version', and 'Author' tags.

```

1 package demoPackage;
2
3 /**
4  * The HelloWorld program implements an application that simply displays
5  * "Hello World!" to the standard output.
6  *
7  * @author bum39705
8  * @version 1.0
9  * @since 2015-03-18
10 *
11 */
12 public class HelloWorld {
13
14     /**
15      * Ein kurzer Satz, der im Abschnitt "Method Summary" stehen wird.
16      */
17 }

```

**demoPackage.HelloWorld**

The HelloWorld program implements an application that simply displays "Hello World!" to the standard output.

**Since:**  
2015-03-18

**Version:**  
1.0

**Author:**  
bum39705

# Ausgabe anpassen

Javadoc verwendet Standardmässig das Standard Doclet um die Ausgabe zu erzeugen.

Wenn man eine andere Ausgabe möchte, kann man ein anderes Doclet verwenden (bestehende oder selber eins schreiben)

➔ Unter <http://www.doclet.com/> finden Sie eine Liste mit Doclets



# Requirements for Writing Java API Specifications (Oracle)

- ➔ <http://www.oracle.com/technetwork/java/javase/documentation/index-142372.html>
- ➔ Vorgaben und Beispiele für
  - ➔ Top Level Specification
  - ➔ Package Specification
  - ➔ Class/Interface Specification
  - ➔ Fields Specification
  - ➔ Method Specification

# Styleguide für javadoc

Siehe Unter

<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#styleguide>

Beispiele unter

<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#examples>

# Nützliche Links

How to Write Doc Comments for the Javadoc Tool:

<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>

Javadoc reference pages:

<http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html#javadocdocuments>

Javadoc reference Guide:

<http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html>

## Doxygen

- <http://www.doxygen.org>
- erstellt Dokumentation für C++,C,IDL, Java und C#

## Doc++

- <http://docpp.sourceforge.net>
- erstellt Dokumentation für C++,C,IDL und Java

# Vergleich Javadoc - Doxygen

## Doxygen:

- Klassendiagramme
- dependency graphs, inheritance diagrams, and collaboration diagrams
- Optionales Source Code Browsing
- Zusätzlicher tag Support
- Output in Tex Format

## Javadoc

- Speziell für Java designed
- Integraler Bestandteil der Sprache → keine weitere Installation nötig

Maven Integration mit beiden, Javadoc ignoriert unbekannte tags  
→ gleichzeitig verwendbar.