Software Engineering II

5 Dokumentation

SS 2020

Prof. Dr. Dirk Müller

Übersicht

- Motivation
- Arten
- Technische Umsetzungen
- Standards
- Spezielle Programmierparadigmen
  - *Literate Programming*
  - *Elucidative Programming*
- *Javadoc*
- Agile Methoden und Dokumentation
- Testgetriebene Entwicklung (TDD) und Dokumentation
- Zusammenfassung

Motivation

- Das System soll schnell und einfach nutzbar sein.
  - Idealfall: selbsterklärende Nutzerschnittstelle, jedoch Realität: Nutzerdokumentation
  - i. d. R. zuerst Installation nötig (Ausnahme: *Web*-Anwendung im *Browser*) => Wie?
- Das System soll systematisch und elegant änder- und erweiterbar sein.
  - Modularisierung
  - Objektorientierung
  - Entwicklerdokumentation
- Das System soll zur Wiederverwendung mit anderen Systemen interagieren können.
  - saubere Schnittstellen zum Import/Export von Daten

Arten von Dokumentation

- Nutzerdokumentation
  - Erklärung der Anwendung des Programms für den Nutzer, insbesondere der Nutzerschnittstelle (meist Kommandozeile oder GUI, aber inzwischen z. B. auch Spracheingabe)
- Installationsdokumentation
  - Vorbedingungen an HW-/BS-/SW-Umgebung incl. Standardbibliotheken und Laufzeitsysteme
  - Installation, Einspielen von *Updates*, De-Installation

für (Rolle der) Administratoren

- Entwicklerdokumentation
  - Beschreibung des Quellcodes für Entwickler zur Wartung/Evolution
- Datendokumentation
  - Semantik, Formate, Datentypen, Beschränkungen (Wertebereich)
  - innere: interne Realisierung, nur für Entwickler sichtbar
  - äußere: für Anwender incl. Import-/Exportschnittstellen
- Testdokumentation (Testfälle)
- Entwicklungsdokumentation (z. B. Lasten- und Pflichtenheft)
- *Marketing*-Dokumentation

Technische Umsetzungen

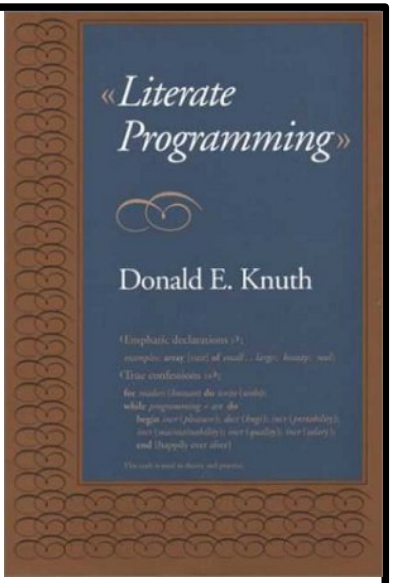
- klassisches gedrucktes Handbuch
- `readme.txt`
- Anzeige über spezielle Kommandozeilenparameter
  - `<Kommando> --help`
  - `<Kommando> /?`
- Aufrufen eines Hilfesystems an der Kommandozeile
  - `man <Kommando>`
  - `info <Kommando>`
  - `help <Kommando>`
- `manual.html` bzw. `manual.pdf`
- integrierte Handbuch-Hilfe, typischerweise mittels `<F1>`
- integrierte kontextsensitive Hilfe, z. B. `<Shift> + <F1>`
- *Website* mit *Patches*, Versionsankündigungen, etc.
- geführte Tour als Szenario mit Grundfunktionalitäten

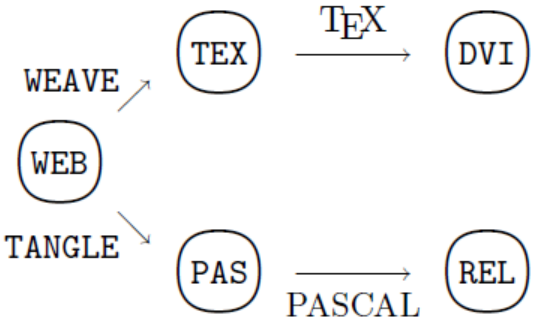
Entwicklerdokumentation

- beschreibt/erklärt Architektur, Entwurf, Implementierung
  - muss ständig aktualisiert (nachgeführt) werden
  - Ziele: Konsistenz zum Quellcode, schnelle Verfügbarkeit
    => kein klassisches Handbuch, sondern elektronisch
- Quellcode
  - Verbalisierung: Wahl der Bezeichner (Variablen, Funktionen, Prozeduren, Methoden) verständlich für Menschen
  - lokale Kommentare: genau da, wo nötig und hilfreich (nicht zu viele, nicht zu wenige), nicht in Extra-Dokument
  - selbstdokumentierende Konzepte und Sprachen (*Ada* vs. *C++*)
- Dokumentationswerkzeuge
  - Erzeugung von Hypertext-Übersichten aus dem Quelltext, z. B. *Javadoc*, *Doxygen*
  - Visualisierungen in IDEs, z. B. Datenstrukturen im *Debugger*
- Skizzen, Diagramme, Anmerkungen in Extra-Dateien
  - Datenhaltung zusammen mit zugehörigem Quellcode

Literate Programming

- seit ca. 1980 von *Donald E. Knuth* vorgeschlagen + zur Implementierung des *TeX*-Systems genutzt [2]
- Motivation
  - Programmierer angehalten, nicht nur guten Quellcode, sondern auch gute Dokumentation zu liefern
- Prinzip
  - gemeinsame Quelle für Quelltext + Dokumentation
  - natürliche Sprache und Code-Schnipsel in beliebiger Reihenfolge gemischt
  - zwei Präprozessoren
- Vorteile
  - bessere Qualität von SW, auch Portabilität
  - erstklassige Entwicklerdokumentation als zweites Produkt





Quelle: [2]

Was macht dieses *C*-Programm?

```
#include <stdio.h>
int b,w,l,t,L;main(int c,char**v){FILE*f=fopen(v[1],"r");for(;(c=getc(f))
!=-1;++b){t++;if(c=='\n'){l++,w++;L=t>L?t:L,t=0;}else if(c==' '||c=='\t')w++;}
L=t>L?t:L;printf("%d (%d) %d %d %d\n",l,L,w,b,b);}
```

- schlecht lesbar, sogar kryptisch
- Wie sieht es mit der Behandlung von Randwerten aus?
- Ist es portabel oder macht es plattformspezifische Annahmen?
- Kann ich es als Filter (`stdin` als Eingabe und `stdout` als Ausgabe) verwenden, also es in einer *Pipe* nutzen?

Bsp. für *Literate Programming: wc in noweb*

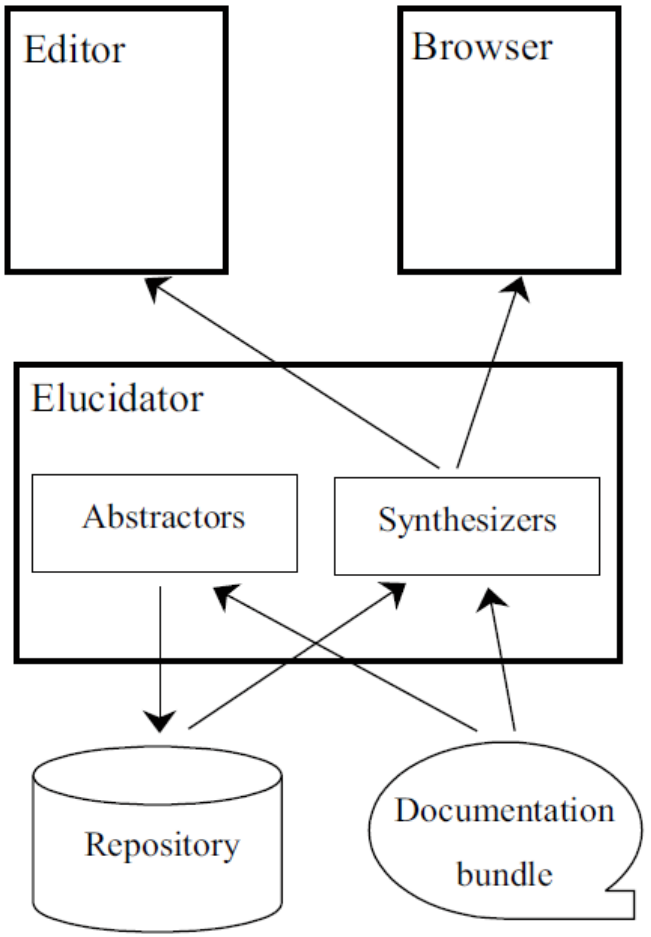
```
The present chunk, which does the counting that is <tt>wc</tt>'s
<i>raison d'etre</i>, was actually one of the simplest to write.
We look at each character and change state if it begins or ends a word.
<<Scan file>>=
while (1) {
     <<Fill [[buffer]] if it is empty; [[break]] at end of file>>
     c = *ptr++;
     if (c > ' ' && c < 0177) {
       /* visible ASCII codes */
       if (!in_word) {
         word_count++;
         in_word = 1;
       }
       continue;
     }
     if (c == '\n') line_count++;
     else if (c != ' ' && c != '\t') continue;
     in_word = 0;
       /* c is newline, space, or tab */
}
@
```

```
Buffered I/O allows us to count the
number of characters almost for
free.
<<Fill [[buffer]] if it is empty;
[[break]] at end of file>>=
if (ptr >= buf_end) {
       ptr = buffer;
       c = read(fd, ptr, buf_size);
       if (c <= 0) break;
       char_count += c;
       buf_end = buffer + c;
}
@
```

Quellen: [4][7]

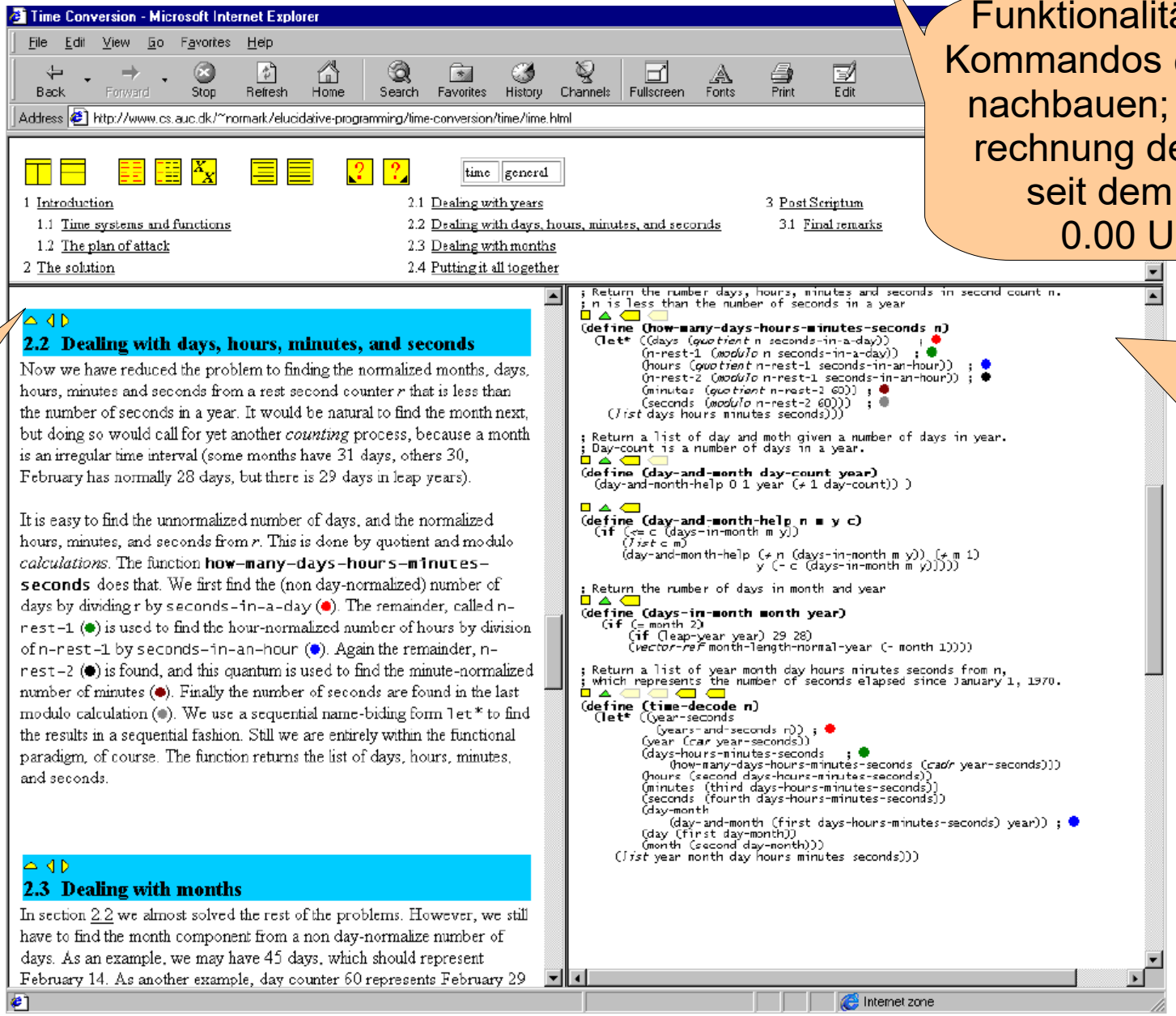
Elucidative Programming

- Variante des *Literate Programming*, 2000 (also fast 20 Jahre später) von *Kurt Nørmark* vorgeschlagen [3]
- Trennung von Quellcode und Dokumentation in zwei verschiedene Dateien, die aber wechselseitig verlinkt sind
- weniger extrem als *Knuth*'sches *Literate Programming*
- für durchschnittlichen Programmierer leichter umsetzbar
- Umgebungen für *Scheme*, später auch für *Java*



Quelle: [3]

Bsp. für *Elucidative Programming*



Funktionalität des *Linux*-Kommandos `date` teilweise nachbauen; speziell Umrechnung der Sekunden seit dem 1.1.1970 0.00 Uhr UTC

Dokumentationsfenster

Programmfenster (Quellcode); hier Sprache *Scheme* verwendet

Quelle: [3]

Javadoc

- parst *Java*-Programme nach *Javadoc*-Kommentaren und erstellt damit dann eine HTML-Dokumentation der API
- seit 1995, *De-facto*-Industriestandard zur Dokumentation von *Java*-Klassen, in JDK enthalten
- Syntax: Block-Kommentar (von `/*` und `*/` umschlossen) erhält durch zweiten Stern Sonderbedeutung
- *Tags* mit dem Zeichen `@` eingeleitet
- `@author` Autor des Quellcodes
- `@version` Version d. Quellcodes, z. B. `1.39, 02/28/97`
- `@param` Parameter einer Methode
- `@return` Rückgabewert einer Methode
- `@see` Querverweis zu anderem Element

Bsp. für *Javadoc*

```
/**
 * 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 loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
}
```

Präfix-Stern seit *Java 1.4* optional

Tags in üblicher Reihenfolge

getImage

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

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute URL. The `name` 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`

Quelle: [5]

Nutzerdokumentation

- Einordnung ins Umfeld (Problemdomäne des Nutzers)
  - aka Methodendokumentation
  - Algorithmen, technisch-wissenschaftl./kaufmännische Verfahren
  - Motivation des Projekts aus inhaltlicher, nicht geschäftlicher, Sicht
- Gliederungsmöglichkeiten
  - Anfänger: Tutorial
    - Szenarien für typische Anwendungssituationen
    - Übungsaufgaben mit Lösungsvorschlägen
  - Fortgeschrittene: thematische Bedienungsanleitung
  - Profis: alphabetische Liste der *Features* mit Querverweisen
- Problembehebung, Fehleranalyse
  - häufig als Tabelle mit Zuordnung von Symptomen zu möglichen Ursachen und Lösungsansätzen
- FAQ: *frequently asked questions*, häufig gestellte Fragen
- Glossar mit Erklärung der Fachbegriffe

Erstellen einer Nutzerdokumentation

1. Analyse
2. Planung: eigentliche Erstellung der Nutzerdokumentation
3. Begutachtung des Entwurfs, iterative Verbesserung
4. Benutzbarkeits-Tests: empirisch in Form von Studien
5. Überarbeitung unter Nutzung der Ergebnisse der Schritte 3 und 4, um eine Endversion zu erhalten

Standards und Richtlinien

- ISO/IEC/IEEE 26512:2011 als aktueller internationaler Standard für Nutzerdokumentation
  - „*Systems and software engineering - Requirements for acquirers and suppliers of user documentation*“
  - enthält Checklisten für Käufer und Lieferanten von Nutzerdokumentation im Anhang
- Richtlinien laut [8]
  - Deckblatt mit Projektname, Dokumentname, Autor, Zeitpunkt der Erstellung/Änderung, Leserkreis, Vertraulichkeit, Copyright sowie Qualitäts- und Konfigurationsmanagement
  - Kurzfassung und/oder Schlüsselwörter
  - Gliederung in Kapitel, Abschnitte und ggf. Unterabschnitte Nummerierungsstil `<Kapitel>-<Seite>` (bessere Änderbarkeit)
  - Referenzhandbuch mit Index, sonst kaum nutzbar
  - falls auch für Anfänger, Glossar zur Begriffserklärung/-abgrenzung sinnvoll

Schreibstil

- grammatikalisch und orthografisch korrekt
  - bessere Lesbarkeit und Glaubwürdigkeit
- Genus: Aktiv statt Passiv (Tat- statt Leideform)
- ein Satz pro Fakt
- maximal 7 Sätze pro Absatz
- gemeinte Bedeutung mehrdeutiger Begriff im Glossar klarstellen

Inspektionen wie auf Quelltext durchführen

- nicht ausschweifend, Qualität vor Quantität
- komplexe Sachverhalte in verschiedenen Beschreibungen darstellen, Chance auf Verständnis durch Leser steigt
- Querverweise nicht nur mit Nummer, sondern auch mit Einschüben (Appositionen)

Quelle: [8]

Marketing-Dokumentation

- Werbung z. B. in Prospekten (für Messen), im Fernsehen, in Suchmaschinen oder in sozialen Netzwerken
- Zwecke
  - Faszination eines potenziellen Kunden/Nutzers, Ansprechen von Emotionen durch Aufgreifen von Trends oder Hypes, um künstlich einen Kaufwunsch hervorzurufen oder einen nur rudimentär vorhandenen zu verstärken
  - Information über *Features* und Eigenschaften der Software
  - Erklärung der Vorteile (manchmal auch Nachteile) gegenüber Konkurrenzprodukten

Vergleichende Werbung war in Deutschland bis 2000 verboten, seitdem teilweise erlaubt.

- ohne Öffentlichkeitsarbeit kaum Verbreitung/Verkauf der Software möglich
  - aber: Kultprodukte benötigen manchmal kein Marketing mehr, werden Selbstläufer.

Agile Methoden und Dokumentation

- lauffähige Software als wichtiger als umfassende Dokumentation angesehen [1]
- häufiger, extremer Standpunkt:
  „Richtige Programmierer schreiben keine Dokumentation.“
- Quellcode und Tests als Ersatz für Dokumentation
  - *„The code is the documentation.“*

Erfolgs-Chancen des Ansatzes abhängig von Programmiersprache

- aber: Studien ergaben, dass Dokumentation bei agiler Entwicklung nicht als unnötig angesehen wird.
- Kernproblem ist Mangel an Motivation
  - erreichbar durch Ansätze, die auf agile Entwicklung zugeschnitten sind, z. B. Belohnungssysteme und *Gamification*

TDD und Dokumentation

- Ziel sind sich selbst dokumentierende Testfälle.
  - Testfälle formalisieren, was Codestücke leisten sollen.
- erreichbar durch feine Granularität
  - Module klein halten
  - Testfälle dann besser lesbar
  - agiles Prinzip der Babyschritte umgesetzt
- Das *Single*-*Source*-Prinzip wird gut unterstützt.
- extreme Sicht, dass Code und Tests allein als Dokumentation dienen können
- besser: Testfälle als ausführbare Spezifikation mit klassischer Dokumentation kombinieren

Quelle: [6]

Zusammenfassung

- mehrere Arten von Dokumentation
  - wichtigste: Nutzer- und Entwicklerdokumentation
- technische Umsetzungen mit Trend vom klassischen gedruckten Handbuch zu elektronischen (Dateien) bzw. sogar integrierten (z. B. kontextsensitive Hilfe) Ansätzen
  - Agilität (Aktualität, schnellere Reaktion auf Veränderungen mgl.)
  - bessere Nutzbarkeit (Querverweise, Volltextsuche, Animationen, Audiodaten)
- Entwicklerdokumentation
  - Kommentare im Quelltext oder extra (ggf. mit Verlinkung)
  - Verbalisierung: aussagekräftige Bezeichner, Wahl der Sprache
  - eigene Programmierstile: *Literate*/*Eludicative Programming*
  - agile Methoden mit weniger klassischer Dokumentation, teilweise Ersatz durch Quellcode und Tests
- Nutzerdokumentation
  - als Tutorial/thematisch/alphabetisch je nach Erfahrungsgrad
  - Standards mit Checklisten sowie Richtlinien verfügbar

Literatur

[1] http://www.agilemanifesto.org/, Download am 10.4.2014

[2] Knuth, Donald E. (1984). “Literate Programming” (PDF). The Computer Journal. British Computer Society. 27 (2): 97–111. doi:10.1093/comjnl/27.2.9, Download am 30.03.2017

[3] Kurt Nørmark. Elucidative Programming. Nordic Journal of Computing, 7(2):87-105, Summer 2000.

[4] Ramsey, Norman (May 13, 2008). “An Example of noweb“, http://www.cs.tufts.edu/~nr/noweb/examples/wc.html, Download am 31.03.2017

[5] Oracle: „How to Write Doc Comments for the Javadoc Tool“, http://www.oracle.com/technetwork/articles/java/index-137868.html, Download am 03.04.2017

[6] Scott Ambler: „Introduction to Test Driven Development (TDD)“, http://agiledata.org/essays/tdd.html, 2002-2013, Download am 03.04.2017

[7] Norman Ramsey, „Noweb — A Simple, Extensible Tool for Literate Programming“, 1989-2011, Download am 05.04.2017, http://www.cs.tufts.edu/~nr/noweb/index.html

[8] Ian Sommerville: „30 Documentation“, 2010, *Web*-Kapitel zum Software-Eingineering-Buch, Download am 18.04.2017, http://iansommerville.com/software-engineering-book/files/2014/07/Documentation.pdf