12.–15.09.2010
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Wer, Wie, Was
Wieso, Weshalb, Warum?

Kommentieren und Dokumentieren

Stefan Hildebrandt

consutling.hildebrandt.tk

Ziele von Software-Dokumentation

- Erleichterung der
  - Benutzung als API
  - Weiterentwicklung
  - Fehlerbehebung
- Adressaten:
  - Andre Entwickler
  - Der Autor selbst
- Unterstützung beim Review
- Benutzung der Software
- Installation und Betrieb des Softwaresystems
- Metaaufgabe:
  - Reflektion des Autors über sein Werk

Dokumentationsstellen

- Inline
- Paket, Klasse, Methode, Feld
- Word
- UML-Diagramme
- Wiki

- Entwurfs-/Planungsdokumentation
- Handbücher

Zu kommentierende Objekte

- Komponente / Bibliothek
  - API
  - Implementierung
- Anwendung / System
- Benutzerschnittstellen
- Installation
- Konfigurationsparameter
- Fehlersituationen
- ...

So süüt dat ut - Projektalltag

- Kunde, PL, Architekt, … schreibt vor, der gesamte Code muss kommentiert werden
  - Jede Methode?
    - Wird meistens so ausgelegt.
- Entwickler kommentieren nicht so gerne

  → Einführung automatischer Prüfungen
- Erst wird die Funktion, dann der Kommentar erstellt
  - Funktionales Ergebnis wird abgenommen, bevor der Kommentar erstellt ist

    → Kein Kommentar
- Bei Änderungen / Bugfixes werden die Kommentare selten angepasst

So süüt dat ut - Projektalltag

- Code Reviews inkl. Kommentaren finden selten statt
  - Kaum Überprüfung der Lesbarkeit und Aktualität des Codes und der Dokumentation

~~Machen lassen!~~

```
/**
 * Set length.
 * @param length length
 */
public int setLength(int length) {
   this.length = length;
}
```

- Kann nur generieren, was allgemeint bekannt ist
- Automatische Generierung der Grundstruktur
  - Wird selten weiter ausgefüllt
  - Wird als Kommentar betrachtet
    - Es erfolgt keine weitere Bearbeitung!
- Ergebnis
  - Sinnlose Verschandlung des Codes

  → Nur bei Bedarf Grundstruktur erzeugen lassen

Zwang!

- Automatische Prüfung mit PMD/Checkstyle

```
//length in mm
private int length;
/**
 * sets the width
 * @param length
 */
public void setLength(int length) {
   this.length = length;
}
/**
 * .
 * @return -
 */
public int getLength() {
  return length;
}
```

→ Manuelle, inhaltliche Prüfung notwendig

→ Demotivation der Entwickler

→ Mehr Arbeit als vorher

Implementierung

```
//Maximum retry count
private static String MAX;

private static String MAX_RETRIES;

//length in mm
private static int length;

private static int lengthInMm;
```

Beispiel

```
/**
 * This class contains various methods for manipulating arrays (such as sorting and
 * searching).  This class also contains a static factory that allows arrays to be
 * viewed as lists.
 * <p>The methods in this class all throw a <tt>NullPointerException</tt> if
 * the specified array reference is null, except where noted.
 *<p>The documentation for the methods contained in this class includes
 * briefs description of the <i>implementations</i>.  Such descriptions should
 * be regarded as <i>implementation notes</i>, rather than parts of the
 * <i>specification</i>.  Implementors should feel free to substitute other
 * algorithms, so long as the specification itself is adhered to.  (For
 * example, the algorithm used by <tt>sort(Object[])</tt> does not have to be
 * a mergesort, but it does have to be <i>stable</i>.)
 *
 * <p>This class is a member of the
 * <a href="{@docRoot}/../technotes/guides/collections/index.html">
 * Java Collections Framework</a>.
 *
 * @author  Josh Bloch
 * @author  Neal Gafter
 * @author  John Rose
 * @version 1.71, 04/21/06
 * @since   1.2
 */
public class Arrays {
```

Beispiel

```
/**
 * Sorts the specified array of longs into ascending numerical order.
 * The sorting algorithm is a tuned quicksort, adapted from Jon
 * L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function",
 * Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November
 * 1993).  This algorithm offers n*log(n) performance on many data sets
 * that cause other quicksorts to degrade to quadratic performance.
 *
 * @param a the array to be sorted
 */
public static void sort(long[] a) {
  sort1(a, 0, a.length);
}
```

Beispiel

```
/**
 * Sorts the specified array of objects into ascending order, according to the
 * {@linkplain Comparable natural ordering} of its elements.  All elements in
 * the array must implement the {@link Comparable} interface.  Furthermore, all
 * elements in the array must be <i>mutually comparable</i> (that is, <tt>e1.
 * compareTo(e2)</tt> must not throw a <tt>ClassCastException</tt> for any
 * elements <tt>e1</tt> and <tt>e2</tt> in the array).
 * <p> This sort is guaranteed to be <i>stable</i>:  equal elements will not
 * be reordered as a result of the sort.
 * <p>The sorting algorithm is a modified mergesort (in which the merge is
 * omitted if the highest element in the low sublist is less than the
 * lowest element in the high sublist).  This algorithm offers guaranteed
 * n*log(n) performance.
 *
 * @param a the array to be sorted
 * @throws  ClassCastException if the array contains elements that are not
 *       <i>mutually comparable</i> (for example, strings and integers).
 */
   public static void sort(Object[] a) {
       Object[] aux = (Object[])a.clone();
       mergeSort(aux, a, 0, a.length, 0);
   }
```

Beispiel

```
/**
 * Src is the source array that starts at index 0
 * Dest is the (possibly larger) array destination with a possible offset
 * low is the index in dest to start sorting
 * high is the end index in dest to end sorting
 * off is the offset to generate corresponding low, high in src
 */
private static void mergeSort(Object[] src,
                  Object[] dest,
                  int low,
                  int high,
                  int off) {
  int length = high - low;
  // Insertion sort on smallest arrays
  if (length < INSERTIONSORT_THRESHOLD) {
    for (int i=low; i<high; i++)
      for (int j=i; j>low &&
        ((Comparable) dest[j-1]).compareTo(dest[j])>0; j--)
          swap(dest, j, j-1);
          return;
    }
```

Beispiel

```
  // Recursively sort halves of dest into src
  int destLow  = low;
  int destHigh = high;
  low  += off;
  high += off;
  int mid = (low + high) >>> 1;
  mergeSort(dest, src, low, mid, -off);
  mergeSort(dest, src, mid, high, -off);
  // If list is already sorted, just copy from src to dest.  This is an
  // optimization that results in faster sorts for nearly ordered lists.
  if (((Comparable)src[mid-1]).compareTo(src[mid]) <= 0) {
        System.arraycopy(src, low, dest, destLow, length);
        return;
  }
  // Merge sorted halves (now in src) into dest
  for(int i = destLow, p = low, q = mid; i < destHigh; i++) {
    if (q >= high || p < mid && ((Comparable)src[p]).compareTo(src[q])<=0)
      dest[i] = src[p++];
    else
      dest[i] = src[q++];
    }
}
```

Beispiel

```
public static void mergeSort(Object[] src, Object[] dest,
                             int low, int high, int off) {
    if (isLengthLowerInsertationThreshold(low, high)) {
        insertationSort(dest, low, high);
    } else {
        Borders borders = new Borders(low, high, off);
        mergeSortFirstHalfRecursiv(src, dest, borders);
        mergeSortSecondHalfRecursive(src, dest, borders);
        if (isAllreadySorted(src, borders)) {
            copyAllreadySortedSrcToDestination(src, dest, borders);
        } else {
            mergeHalvesToDestination(src, dest, borders);
        }
    }
}
```

Wer, wie, was?

- Die Informationen sollten in gut strukturiertem Code aus dem Code und VCS ersichtlich sein
  → redundante Informationen
  - Verletzung des DRY-Prinzips
  - Zusätzlicher Aufwand, Quelle für Inkonsistenzen
- Verursacht hohe Kosten
  - Beim Einstig in eine Klasse muss eventuell viel Doku gelesen werden, die potentiel nicht aktuell / fehlerhaft ist.
  - Im Fehlerfall muss Code und Kommentar geprüft und verglichen werden
    - Doppelter Aufwand
  - Bei Änderungen müssen alle Kommentare in dem Umfeld angepasst werden

Beispiel

```
public static void mergeSort(Object[] src, Object[] dest,
                             int low, int high, int off) {
    //simple insertation sort results in faster sorts for small arrays
    if (isLengthLowerInsertationThreshold(low, high)) {
        insertationSort(dest, low, high);
    } else {
        Borders borders = new Borders(low, high, off);
        mergeSortFirstHalfRecursiv(src, dest, borders);
        mergeSortSecondHalfRecursive(src, dest, borders);
        if (isAllreadySorted(src, borders)) {
            //if allready sorted, we can speedup by ony copy values
            copyAllreadySortedSrcToDestination(src, dest, borders);
        } else {
            mergeHalvesToDestination(src, dest, borders);
        }
    }
}
```

Wieso, weshalb, warum?

- Geht nur selten aus dem Code hervor
  - Bei entsprechenden Entscheidungen bei der Entwicklung unbedingt dokumentieren
  - Nach Fertigstellung einer Methode, diese im Hinblick auf Unklarheiten lesen.
  - Falls im Review / Pairprogramming Unklarheiten auftreten, sollte es sofort dokumentiert werden
- Anworten sind bei der Wartung sehr nützlich

Auswirkung vieler Methoden

- Performance
  - Auto inline
  - Compiler
- StackOverflow-Heuristik
  - Liefert bei starker Rekursion eventuell Fehlalarm
- Lesbarkeit
  - Methoden unterschiedlicher Granularität
  - Sortieren der Methoden
- Testbarkeit
  - Deutlich besser

Beispiel

public RenderedOp createNS(String opName,
ParameterBlock args,
RenderingHints hints)
Creates a RenderedOp which represents the named operation, using the source(s) and/or parameter(s) specified in the ParameterBlock, and applying the specified hints to the destination. This method should only be used when the final result returned is a single RenderedImage. However, the source(s) supplied may be a collection of rendered images or a collection of collections that at the very basic level include rendered images.

The supplied operation name is validated against the operation registry. The source(s) and/or parameter(s) in the ParameterBlock are validated against the named operation's descriptor, both in their numbers and types. Additional restrictions placed on the sources and parameters by an individual operation are also validated by calling its OperationDescriptor.validateArguments() method.

JAIallows a parameter to have a null input value, if that particular parameter has a default value specified in its operation's descriptor. In this case, the default value will replace the null input.

JAIalso allows unspecified tailing parameters, if these parameters have default values specified in the operation's descriptor. However, if a parameter, which has a default value, is followed by one or more parameters that have no default values, this parameter must be specified in the ParameterBlock, even if it only has a value of code>null.

Beispiel

The rendering hints associated with this instance of JAI are overlaid with the hints passed to this method. That is, the set of keys will be the union of the keys from the instance's hints and the hints parameter. If the same key exists in both places, the value from the hints parameter will be used.

This version of create is non-static; it may be used with a specific instance of theJAIclass. All of the static create() methods ultimately call this method, thus inheriting this method's error handling.

Since this method performs parameter checking, it may not be suitable for creating RenderedOp nodes meant to be passed to another host using the RemoteImage interface. For example, it might be necessary to refer to a file that is present only on the remote host. In such cases, it is possible to instantiate a RenderedOp directly, avoiding all checks.
Parameters:
opName - The name of the operation.
args - The source(s) and/or parameter(s) for the operation.
hints - The hints for the operation.
Returns:
A RenderedOp that represents the named operation, or null if the specified operation is in the "immediate" mode and the rendering of the PlanarImage failed.

Beispiel

Throws:
IllegalArgumentException - if opName is null.
IllegalArgumentException - if args is null.
IllegalArgumentException - if no OperationDescriptor is registered under the specified operation name in the current operation registry.
IllegalArgumentException - if the OperationDescriptor registered under the specified operation name in the current operation registry does not support rendered image mode.
IllegalArgumentException - if the specified operation does not produce a java.awt.image.RenderedImage.
IllegalArgumentException - if the specified operation is unable to handle the sources and parameters specified in args

Java[tm] Advanced Imaging API Documentation

Beispiel

```
public static RenderedOp rotateImage(RenderedOp image,
                                      float angle) {
  Interpolation interpolation = new InterpolationBicubic2(4);
  float centerY = (float) ((double) image.getWidth() / (double) 2);
  float centerX = (float) ((double) image.getHeight() / (double) 2);
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(image);                  // The source image
  pb.add(centerX);                      // The x origin
  pb.add(centerY);                      // The y origin
  pb.add(angle);                      // The rotation angle
  pb.add(interpolation); // The interpolation
  image = JAI.create("Rotate", pb, null);
  return image;
```

Beispiel

```
public static RenderedOp scaleImage(RenderedOp image,
                                    float scale,
                                    Interpolation interpolation) {
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(image);
  pb.add(scale);
  pb.add(scale);
  pb.add(0.0f);
  pb.add(0.0f);
  pb.add(interpolation);
  return JAI.create("scale", pb, null);
```

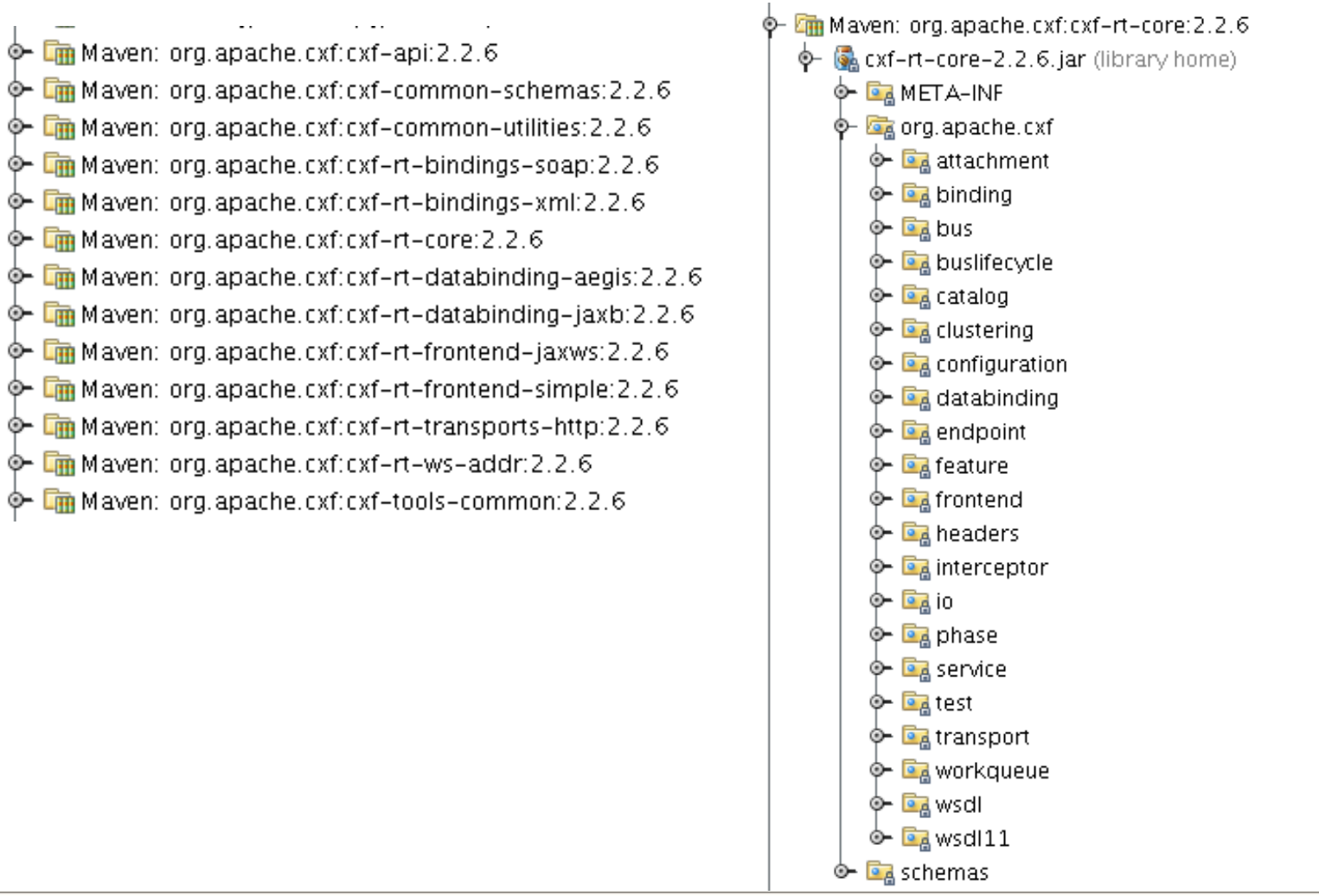
API / Componenten-Interface

- Beschreibung der Funktion
- Die Einflussnahme jedes Parameters auf die Funktion
  - Inklusive Hinweisen zur Bestimmung gültiger Werte
  - Inklusive dem Verhalten bei null
- Ergebnis der Funktion
- Nicht beschreiben
  - Wie die Methode intern funktioniert

Implementierungen

- Wer, wie, was
  - Durch den Programmcode beschreiben lassen
- Wieso, Weshalb, Warum
  - Sensibilisieren und Vorgaben machen
- Keine automatischen Prüfungen und Generierungen

Beispiel


Maven: org.apache.cxf:cxf-api:2.2.6
Maven: org.apache.cxf:cxf-common-schemas:2.2.6
Maven: org.apache.cxf:cxf-common-utilities:2.2.6
Maven: org.apache.cxf:cxf-rt-bindings-soap:2.2.6
Maven: org.apache.cxf:cxf-rt-bindings-xml:2.2.6
Maven: org.apache.cxf:cxf-rt-core:2.2.6
Maven: org.apache.cxf:cxf-rt-databinding-aegis:2.2.6
Maven: org.apache.cxf:cxf-rt-databinding-jaxb:2.2.6
Maven: org.apache.cxf:cxf-rt-frontend-jaxws:2.2.6
Maven: org.apache.cxf:cxf-rt-frontend-simple:2.2.6
Maven: org.apache.cxf:cxf-rt-transports-http:2.2.6
Maven: org.apache.cxf:cxf-rt-ws-addr:2.2.6
Maven: org.apache.cxf:cxf-tools-common:2.2.6
Maven: org.apache.cxf:cxf-rt-core:2.2.6
cxf-rt-core-2.2.6.jar (library home)
META-INF
org.apache.cxf
attachment
binding
bus
buslifecycle
catalog
clustering
configuration
databinding
endpoint
feature
frontend
headers
interceptor
io
phase
service
test
transport
workqueue
wsdl
wsdl11
schemas

Einstieg? - Wegweiser?- Konzepte?

- Große Projekte
  - Einstig für Dokumentation aufzeigen
  - Erläuterung von Zusammenhängen und übergreifender Konzepte
    - Einleitende Beschreibung untermalt mit Bildern
      - z.B. mit UML Klassen- und Sequenzdiagrammen
  - Einstiegspunkte für Nutzung und Entwicklung bieten
    - Mit Beispielen verdeutlichen
    - Konventionen erklären
- Wohin mit den Informationen?

Quellcode (Paket, Klasse)

- Pro
  - Konsistenz
  - Verknüpfung des Codes
  - Versionierbarkeit
- Contra
  - Gliederungsmöglichkeiten
  - Übersichtlichkeit
  - Editierbarkeit durch Nichtentwickler
  - Pflegekomfort

Plaintext im VCS

- z.B. HTML, Apt, Docbook, Latex
- Pro
  - Konsistenz (kann mit refactorisiert werden)
  - Versionierbarkeit
  - Übersichtlichkeit
- Contra
  - Pflegekomfort
  - Zusätzliche Sprache als Hürde für die Pflege

Non-Plaintext im VCS

- Word, Dokumentations-Tools,
- Pro
  - Pflegekomfort
  - Übersichtlichkeit
  - Pflegbarkeit durch Nichtentwickler
  - Versionierbarkeit
    - Version pro Codezweig / Release
- Contra
  - Konsistenz
  - Parallele Bearbeitung
    - Kaum mergebar

Wiki

- Pro
  - Pflegekomfort
  - Übersichtlichkeit
  - Pflegbarkeit durch Nichtentwickler
  - Relativ niedrige Schwelle bei Nichtentwicklern Änderungen vorzunehmen
  - Gute Durchsuchbarkeit
  - Versionierbarkeit
    - Konkurrierend editierbar (kleinteilig)
    - Änderungsprotokoll
- Contra
  - Konsistenz
  - Versionen für mehrere Releases

Zusammenfassung - Code-Dokumentation

- API
  - Verpflichtung zur Kommentierung des Vertrags
- Implementierung
  - Information-Hiding
  - Kein Kommentarzwang pro Methode
  - Verbesserung der Lesbarkeit des eigentlichen Codes
- Gesparte Zeit für Reviews oder Pairprogramming verwenden
- Refactoring-Möglichkeiten der IDEs nutzen

Zusammenfassung – Anwendung / System

- Beschreibung der Zusammenhänge
- Benutzerschnittstellen
- Installation, Konfigurationsparameter, Fehlersituationen, …
- Aufgaben
  - Editierbarkeit und Zugriff für alle Beteiligten
  - Sicherstellung der Konsistenz
  - Versionierung

Vielen Dank!

Stefan Hildebrandt

consulting.hildebrandt.tk

consulting.hildebrandt.tk

- Entwicklung und Entwicklungsunterstützung im Bereich JEE-Anwendungen
- Coaching (im Projekt) und Schulung
  - Frameworks und Tools
  - Pairprogramming
  - Test-Driven-Development
- Softwarearchitektur und Projektsetup
  - Buildprozess, CI
  - Frameworkauswahl und -integration
  - Automatisiertes Deployment
  - Betriebssetup Jboss, Tomcat, Jetty
- Leichtgewichtige Entwicklungsprozesse