Lecture 6

Documentation Generation

SOEN 6441, Summer 2018

René Witte
Department of Computer Science
and Software Engineering
Concordia University

Outline

Motivation

Software Documentation

Software development nightmares:

- Undocumented code
- Documentation not matching implementation

Classical Documentation

Documentation written and developed separate from source code

- No explicit connections
- Becomes easily outdated (traceability issue)

Solution: Documentation Generation

Maintain one file only, containing both code and documentation

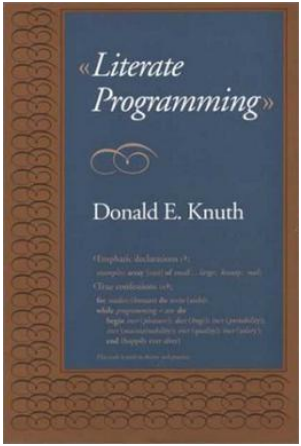
Literate Programming

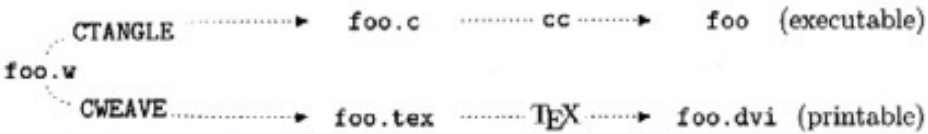
René Witte
Concordia University

Donald Knuth, 1992

"I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: "Literate Programming."

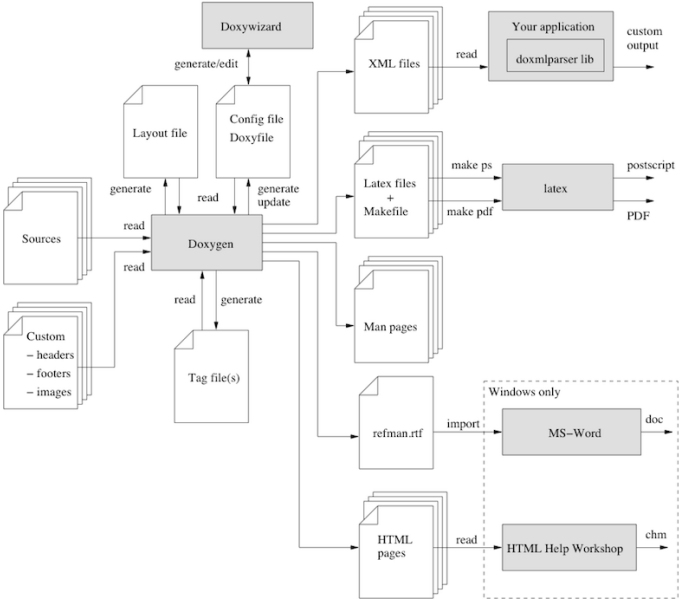
Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do."





Doxygen

Doxygen (http://doxygen.org/) supports C, C++, C#, Java, PHP, Python, Perl, . . .



René Witte

Doxygen Input Example (C++)

```
/**
 * Construct a vector normal to the polygon defined
 * by the given points using Martin Newell's algorithm.
 * The normal vector will be exact if the points lie in a plane,
 * otherwise it will be a sort of average value.  As with OpenGL,
 * the vector will point in the direction from which the points
 * are enumerated in a counter-clockwise direction.
 *
 * Unlike other functions, this function does \b not use
 * homogeneous coordinates.  The points are assumed to have
 * (x,y,z) coordinates; the w component is ignored.
 * \param points is an array of points.
 * \param numPoints is the number of points in the array.
 * \return the vector normal to the plane defined by \a points.
 * \note The vector is \b not a unit vector because it will probably
 * be averaged with other vectors.
 */
Vector(Point points[], int numPoints);
```

Doxygen Generated HTML Output

The array must have at least three components.

```
Vector::Vector ( Point points[],
                 int   numPoints
               )
```

Construct a vector normal to the polygon defined by the given points using Martin Newell's algorithm.

The normal vector will be exact if the points lie in a plane, otherwise it will be a sort of average value. As with OpenGL, the vector will point in the direction from which the points are enumerated in a counter-clockwise direction.

Unlike other functions, this function does **not** use homogeneous coordinates. The points are assumed to have (x,y,z) coordinates; the w component is ignored.

Parameters:

points is an array of points.

numPoints is the number of points in the array.

Returns:

the vector normal to the plane defined by *points*.

Note:

The vector is **not** a unit vector because it will probably be averaged with other vectors.

Javadoc

René Witte

Java Documentation

The Javadoc tool (originally written JavaDoc) has been part of the JDK since 1.0

- Documentation embedded in special Java comments
- The `javadoc` commands generates output based on a Doclet (typically HTML)

Java Comments

- Single-line comments: `// this is a comment`
- Multi-line comments: `/* ... */`
- Javadoc comments: `/** ... */`

Changes in Java 8

Java 8 has more strict checking of Javadoc through the doclint tool:

- Illegal HTML (e.g., unclosed tags or illegal tags)
- Broken references (e.g., non-matching `@param` name)

as well as others now result in an error (rather than a warning)

Javadoc Tags

Types of tags

- Stand-alone tags (document tags): start with an '`@`' (at-sign)
  - e.g., `@author`
  - must appear on their own line
- In-line tags: start with a '`{`' (curly brace)
  - e.g., `{@code}`
  - can be used within a larger description
- Standard HTML tags are also allowed (formatting, tables, etc.)
  - However, avoid tags interfering with Javadoc HTML output (e.g, headers)
  - Must conform to HTML4 (changed to HTML5 in Java 10)
    (Java 9 already has optional HTML5 support)

Some Javadoc Tags (https://en.wikipedia.org/wiki/Javadoc)

René Witte

Tag & Parameter	Usage	Applies to	Since
@author *John Smith*	Describes an author.	Class, Interface, Enum	
{**@docRoot**}	Represents the relative path to the generated document's root directory from any generated page.	Class, Interface, Enum, Field, Method	
@version *version*	Provides software version entry. Max one per Class or Interface.	Class, Interface, Enum	
@since *since-text*	Describes when this functionality has first existed.	Class, Interface, Enum, Field, Method	
@see *reference*	Provides a link to other element of documentation.	Class, Interface, Enum, Field, Method	
@param *name description*	Describes a method parameter.	Method	
@return *description*	Describes the return value.	Method	
@exception *classname description* **@throws** *classname description*	Describes an exception that may be thrown from this method.	Method	
@deprecated *description*	Describes an outdated method.	Class, Interface, Enum, Field, Method	
{**@inheritDoc**}	Copies the description from the overridden method.	Overriding Method	1.4.0
{**@link** *reference*}	Link to other symbol.	Class, Interface, Enum, Field, Method	
{**@linkplain** *reference*}	Identical to {@link}, except the link's label is displayed in plain text than code font.	Class, Interface, Enum, Field, Method	
{**@value** *#STATIC_FIELD*}	Return the value of a static field.	Static Field	1.4.0
{**@code** *literal*}	Formats literal text in the code font. It is equivalent to <code>{@literal}</code>.	Class, Interface, Enum, Field, Method	1.5.0
{**@literal** *literal*}	Denotes literal text. The enclosed text is interpreted as not containing HTML markup or nested javadoc tags.	Class, Interface, Enum, Field, Method	1.5.0
{**@serial** *literal*}	Used in the doc comment for a default serializable field.	Field	
{**@serialData** *literal*}	Documents the data written by the writeObject( ) or writeExternal( ) methods.	Field, Method	
{**@serialField** *literal*}	Documents an ObjectStreamField component.	Field	

Javadoc: Method Comment Example

```
/**
 * Validates a chess move.
 *
 * Use {@link #doMove(int theFromFile, int theFromRank, int theToFile, int theToRank)} to move a piece.
 *
 * @param theFromFile file from which a piece is being moved
 * @param theFromRank rank from which a piece is being moved
 * @param theToFile   file to which a piece is being moved
 * @param theToRank   rank to which a piece is being moved
 * @return            true if the move is valid, otherwise false
 * @since             1.0
 */
boolean isValidMove(int theFromFile, int theFromRank, int theToFile, int theToRank) {
    // ...body
}

/**
 * Moves a chess piece.
 *
 * @see java.math.RoundingMode
 */
void doMove(int theFromFile, int theFromRank, int theToFile, int theToRank)  {
    // ...body
}
```

Javadoc Writing Guidelines

How to Write Doc Comments for the Javadoc Tool

Additional writing rules (see full document for all guidelines):

- First Sentence: should be a summary explanation of the class/method/interface/member.
- Inherit comments from superclasses, rather than duplicating them
- 3rd Person: Use 3rd person (descriptive) not 2nd person (prescriptive); e.g., *"Gets the label"*, not *"Get the label"*
- Method descriptions begin with a verb phrase. A method implements an operation, so it usually starts with a verb phrase, e.g., *"Gets the label of this button"*
- Omit the subject for Class/interface/field descriptions and simply state the object, e.g., *"A button label"*, not *"This field is a button label"*
- Use `this` instead of "the" when referring to an object created from the current class, e.g., *"Gets the toolkit for this component"*, not *"... the component"*.
- Don't repeat API name: comment must add value, e.g., *"Gets the label text"* for a method `getLabelText()` is not useful.

See *How to Write Doc Comments for the Javadoc Tool,* http://www.oracle.com/technetwork/articles/java/index-137868.html

Javadoc in Eclipse: Generate Javadoc

René Witte


```
/**
 *
 * @param dice
 * @return
 */
public static boolean isThreeOfAKind(List<Die> dice) {
  // implementation not shown
}
```

Javadoc in Eclipse: View

```
catch (InvocationTargetException e) {
    e.fillInStackTrace();
    throw e.getTargetException();
```

Tasks | @ Javadoc | Search | Call Hierarchy | Problems

Javadoc - java.lang.Throwable.fillInStackTrace()

Fills in the execution stack trace. This method records within this `Throwable` object information about the current state of the stack frames for the current thread.

Returns:

a reference to this `Throwable` instance.

See Also:

java.lang.Throwable.printStackTrace()

Outline

Reading Material

Required

- Ian Sommerville, *Software Engineering*, Chapter 30 “Software Documentation” (Web Chapter), http://iansommerville.com/software-engineering-book/files/2014/07/Documentation.pdf
- How to Write Doc Comments for the Javadoc Tool, http://www.oracle.com/technetwork/articles/java/index-137868.html

References

- Java Platform, Standard Edition Javadoc Guide: https://docs.oracle.com/javase/10/javadoc/JSJAV.pdf

Literate Programming

- Original article by Donald Knuth: http://www.literateprogramming.com/knuthweb.pdf
- Literate Programming book: https://www-cs-faculty.stanford.edu/~knuth/lp.html
- Website: http://www.literateprogramming.com/