Regular Lecture 13
Software Documentation

Prof. Dr. Jasmin Blanchette

Chair of Theoretical Computer Science and Theorem Proving

Version of May 13, 2026

Generalities

Types of Software Documentation

Software documentation includes

- **end-user documentation**;
- **library documentation**;
- **implementation documentation**.

End-user documentation (e.g., user's manuals) targets your software's **end users**.

Library documentation targets programmers using your software as a library via its public **application programming interface** (**API**).

Implementation documentation targets programmers **maintaining** or **extending** your software.

End Users

Types of End User Documentation

There are three main types of end-user documentation:

- **getting-started guides**;
- **tutorials**;
- **reference manuals**.

Getting-Started Guides

Getting-started guides generally cover

- **obtaining** the software;
- **installing** the software;
- **testing** the installation;
- a **hello-world** example.

These guides are typically the **entry point** for new users.
They are generally designed to be read **sequentially**.

Tutorials

Tutorials introduce the reader to your software **step by step**.
Like getting-started guides, they are generally designed to be read **sequentially**.

Basic tutorials are used to introduce the software's **major features**.
More **specialized topics** are often covered in advanced tutorials.

Reference Manuals

Reference manuals provide **comprehensive documentation** for your software.
They are optimized for **searching**, not for reading sequentially.
Ample **cross-referencing** helps the reader navigate them.

For a command-line program, a reference manual would explain all the command-line options. Unix man pages are reference manuals.

For an application with a graphical user interface, a reference manual would document all the menu entries and include screenshots of the different views and dialog windows.

Libraries

Types of Library Documentation

There are two main types of library documentation:

- **tutorials**;
- **reference manuals**.

These are similar in purpose to end-user documentation, but they are aimed at programmers.

Tutorials

Library documentation usually includes programming tutorials.
These explain a topic—typically, a set of related classes or functions—with the help of code examples that show the **API in action**.

For example, the Java documentation includes a tutorial on graphical user inferface programming using the Swing framework.

Reference Manuals

If your software is a library, its public API should be **thoroughly documented**.
Every class, method, function, etc., should be covered in a reference manual.

Documentation Generators

The simplest way to document APIs is to use a documentation generation tool such as **Doxygen**, **Javadoc**, **QDoc**, or **Sphinx**.

- You can document your software's components—the classes, the methods, etc.—directly where they are defined, in specially labeled comments.
- The tool extracts the comments and generates documentation in common formats such as HTML.

With the documentation in the source code, when you modify the API, you are likely to remember to **update the documentation**.

Documentation generators also support **cross-referencing**.
Some are conveniently built into **integrated development environments** (e.g., Javadoc in Eclipse and IntelliJ).

Javadoc Example

```
/**
 * Returns the Unicode code point at the specified index. The
 * index refers to {@code char} values and ranges from
 * {@code 0} to {@code length() - 1}.
 *
 * @param index the index of the char value
 * @return the code point value of the character at the index
 * @throws IndexOutOfBoundsException if the index parameter is
 *         negative or else greater than or equal to the length
 *         of this string
 * @since 1.5
 */
public int codePointAt(int index) {
```
[1]

[1] Adapted from the Java 21 documentation at https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html#codePointAt(int).

Javadoc Example

codePointAt

`public int codePointAt(int index)`

Returns the Unicode code point at the specified index. The index refers to `char` values and ranges from `0` to `length() - 1`.

Parameters:
`index - the index of the char value`

Returns:
`the code point value of the character at the index`

Throws:
`IndexOutOfBoundsException - if the index parameter is negative or else greater than or equal to the length of this string`

Since:
1.5

Code Examples

Library documentation is illustrated with code examples.
These must be as **simple** as possible while demonstrating usage and patterns of use.

To avoid distracting the reader, choose consistent **naming** and **indentation conventions**. Generally avoid single-letter identifiers.

Make sure that your code **compiles**.
Ideally, write your code as a standalone application and **quote** the relevant passages.

Code Examples

Compare:

```
JPanel p = new JPanel(new GridLayout(0, 1));
p.add(cb);
p.add(gb);
p.add(hb);
p.add(tb);
add(p, BorderLayout.LINE_START);
```

```
JPanel checkPanel = new JPanel(new GridLayout(0, 1));
checkPanel.add(chinButton);
checkPanel.add(glassesButton);
checkPanel.add(hairButton);
checkPanel.add(teethButton);
add(checkPanel, BorderLayout.LINE_START);
```

The Inclusive *We*

When explaining code, you can choose between **third-person** forms and the **inclusive 'we'**. Compare:

> *Procedure* prime_the_change_buffer *sets* change_buffer *in preparation for the next matching operation.*
>
> *In procedure* prime_the_change_buffer*, we set* change_buffer *in preparation for the next matching operation.*

Do not mix and match, at least within a paragraph.

Implementation

Implementation Documentation

Implementation documentation describes your software's **internals**.
These documents tend to focus on its architecture and often include Unified Modeling Language (UML) diagrams. They usually include the rationale for key design decisions—e.g., the choices of tooling, algorithms, and data structures.

Implementation documentation usually takes the form of standalone documents, but following a discipline called **literate programming**, you can thoroughly document your software directly in the source code.

Literate Programming

With literate programming, the software and its implementation documentation are generated from a **single source**.

CWEB is an example of a literate programming language.
From a CWEB source file, you can obtain both a C source file and a LaTeX document.

CWEB and similar languages also provide powerful mechanisms for defining and using **macros**, giving you a lot of flexibility when presenting the code.

CWEB Example

```
@ Procedure |prime_the_change_buffer|
sets |change_buffer| in preparation for the next matching operation.
Since blank lines in the change file are not used for matching, we have
|(change_limit == change_buffer && !changing)| if and only if
the change file is exhausted. This procedure is called only when
|changing| is 1; hence error messages will be reported correctly.

@c
void prime_the_change_buffer()
{
  change_limit = change_buffer; /* this value is used if the change file ends */
  @<Skip over comment lines in the change file; |return| if end of file@>;
  @<Skip to the next nonblank line; |return| if end of file@>;
  @<Move |buffer| and |limit| to |change_buffer| and |change_limit|@>;
}
```
[1]

[1] Adapted from Donald E. Knuth and Silvio Levy, "The CWEB System of Structured Documentation (Version 3.64 – February 2002)," 2002.

CWEB Example

12. Procedure *prime_the_change_buffer* sets *change_buffer* in preparation for the next matching operation. Since blank lines in the change file are not used for matching, we have (*change_limit* $\equiv$ *change_buffer* $\wedge$ $\neg$*changing*) if and only if the change file is exhausted. This procedure is called only when *changing* is 1; hence error messages will be reported correctly.

```
void prime_the_change_buffer()
{
  change_limit = change_buffer;      /* this value is used if the change file ends */
  ⟨Skip over comment lines in the change file; return if end of file 13⟩;
  ⟨Skip to the next nonblank line; return if end of file 14⟩;
  ⟨Move buffer and limit to change_buffer and change_limit 15⟩;
}
```