



# 23. Framework Documentation

Prof. Uwe Aßmann  
TU Dresden

Institut für Software- und Medientechnik  
Lehrstuhl Softwaretechnologie  
11-1.0, 23.12.11



Design Patterns and Frameworks, © Prof. Uwe Aßmann

1

## References

- ▶ **Obligatory:**
  - M. Meusel, K. Czarnecki, W. Köpf. A model for structuring user documentation of object-oriented frameworks using patterns and hypertext. European Conference on Object-Oriented Programming. LNCS. Springer-Verlag, 1997.  
<http://www.springerlink.com/index/292mk7473w9m5910.pdf>
- ▶ **Other:**
- ▶ B. Minto. The Pyramid Principle. Part One: Logic in Writing. Pitman Publishing, London, 1991. First published by Minto International Inc. in 1987.
- ▶ G. Jimenez-Diaz, M. Gomez-Albarran. A Case-Based Approach for Teaching Frameworks.
- ▶ Andreas Bartho. Creating and Maintaining Tutorials with DEFT. ICPC 2009
- ▶ T. Vestdam. Generating Consistent Program Tutorials. Technical Report, University of Aalborg, Denmark.
- ▶ T. Vestdam. Pulling Threads Through Documentation. Technical Report, University of Aalborg, Denmark.
- ▶ T. Vestdam. Contributions to Elucidative Programming. PhD thesis, January 2003, University of Aalborg, Denmark.

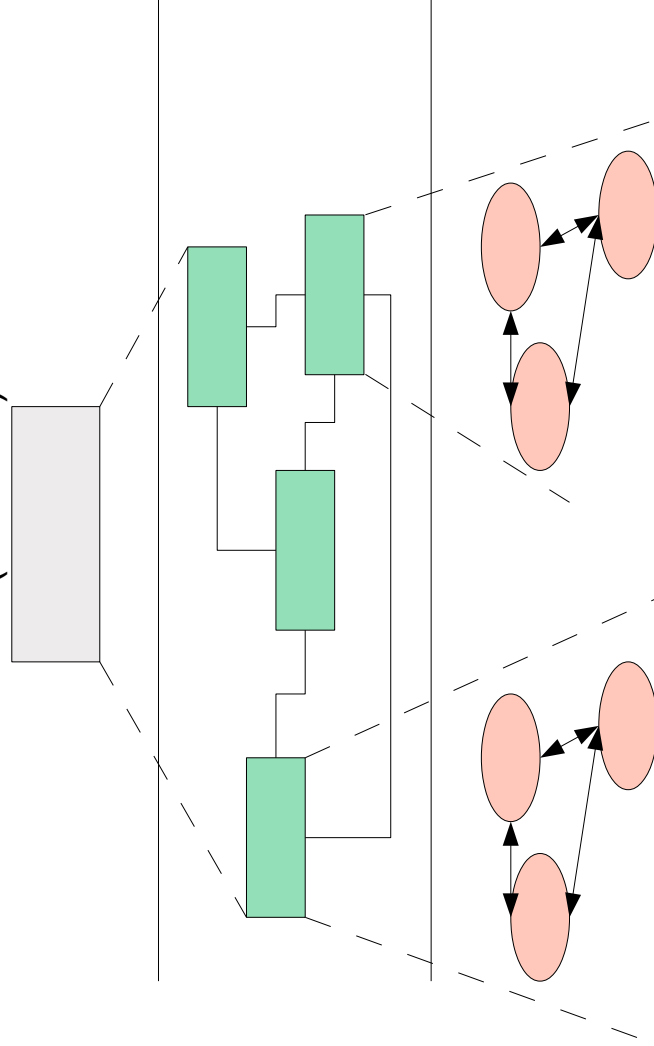


# Problem: How to Document a Framework?

- ▶ Framework understanding is hampered by many problems
  - Good documentation should help to solve them
- ▶ Lack of knowledge of domain of the framework
- ▶ Unknown mapping between domain concepts and framework classes
  - Often not 1:1, but n:m mappings
- ▶ Unknown framework functionality
  - Does this framework fit?
- ▶ Lack of knowledge of interactions between framework classes
  - Impact of instantiations cannot be estimated
- ▶ Lack of knowledge of the architecture of the framework
  - Framework integrity is related
- ▶ Multiple solutions possible with the framework
- ▶ Technical problems (platform knowledge, ..)

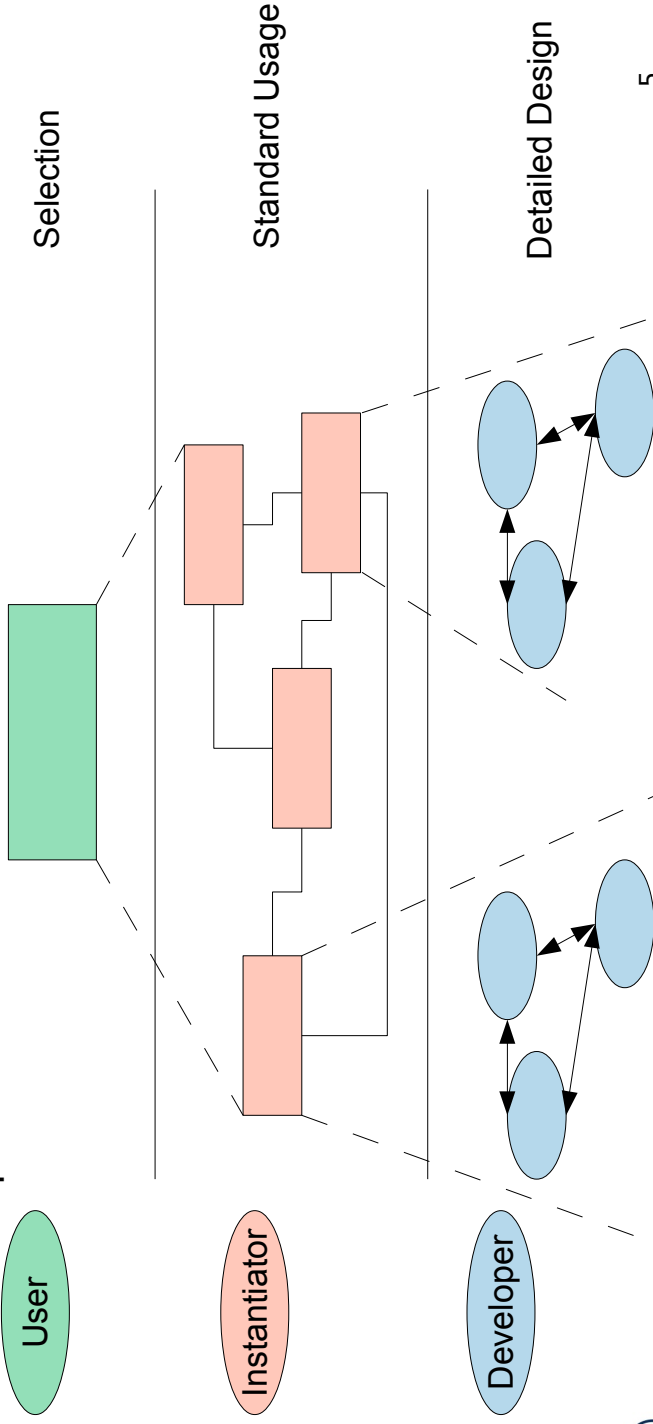
# The Pyramid Principle

- ▶ Documents (also documentation) should consist of several *abstraction levels*
- ▶ A top node is refined into lower levels [Minto]
- ▶ A *reducible* structure results (see ST-II)



# The Pyramid Principle in Framework Documentation

- ▶ Framework Selection: Does the framework address my problem?
- ▶ Framework Standard Usage: How to use it?
- ▶ Framework Detailed Design: How does it work? How to further develop it?



## Level 1: Framework Selection Sheet

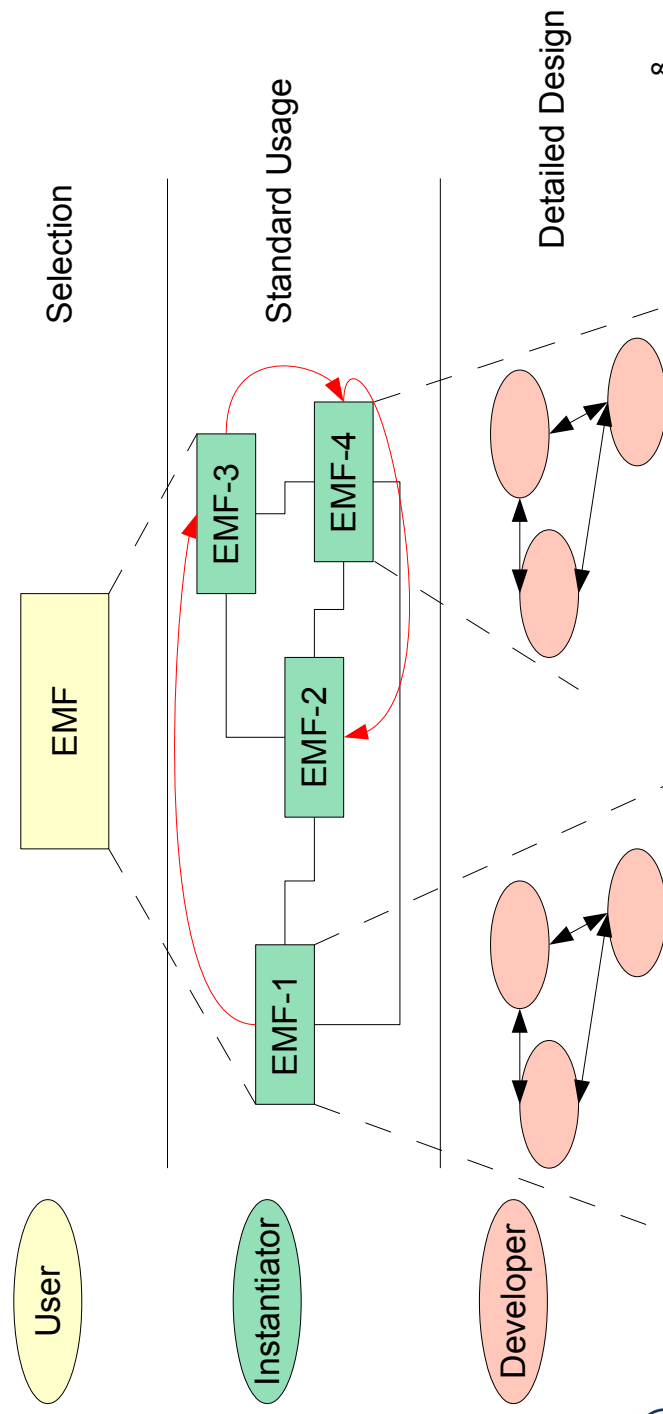
- ▶ Basically a short description (fact sheet), comparable to a Linux LSM:
  - **Name:** EMF (Eclipse Modelling Framework)
  - **Keywords:** modelling, editor, development environment, UML
  - **Problem description (application domain):** EMF facilitates the construction of graphic editors, providing basic functionality for diagrams, nodes, edges, including the workspace of an IDE
  - **Solution (features, design concepts):** EMF is an extensible framework, and itself an Eclipse plugin
  - **Examples (typical applications):** UML-EMF application
  - **Other related frameworks:** JDT (Java Development Tools)

# Level 2: Standard Use Cases with Application Patterns

- ▶ An *application pattern* is a standard usage pattern (use case) of a framework
- ▶ Example:
  - **Name:** EMF-1
  - **Short Description:** “Creating a Petri-Net Editor”
  - **Context:** “EMF is the eclipse-based modelling framework, which can be tailored towards more specific editors”
  - **Problem:** How can I draw a Petri-Net?
  - **Instantiation Explanation (Solution Explanation)**
    - This can be a petri net, statechart, activity diagram, or flowchart to describe the framework instantiation process. Description step by step:
      - “1) write a plugin.xml file
      - 2) write a Java Plugin class and name it in the plugin.xml
      - 3) describe the extended extension points in the plugin.xml
      - 4) load the .jar file into the eclipse plugin directory”
  - **Instantiation Chart (Instantiation Solution):** <<a chart showing the process>>
  - **Example applications:** PN Editor
  - **Design information:** << info about extension points, extended points>>
  - And many more.

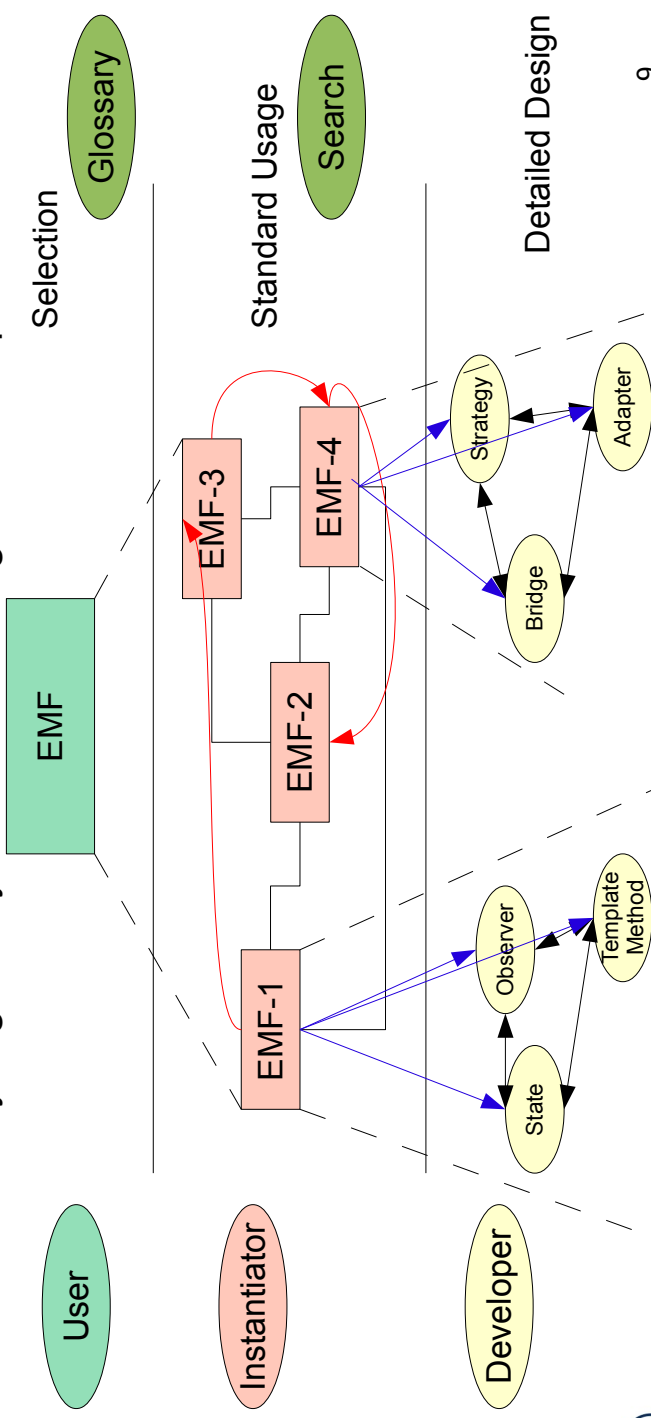
# Application Pattern Documentation is Threaded

- ▶ For a tutorial, the application patterns will be **threaded**



# Third Level: Detailed Design

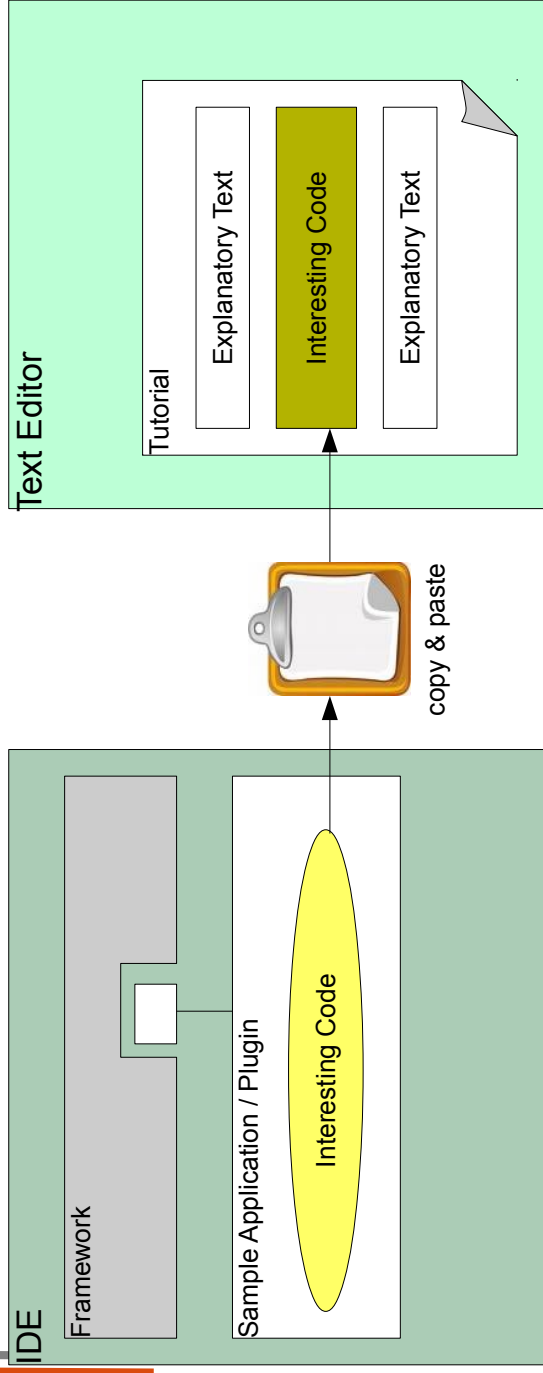
- ▶ On this level, the framework is documented by
  - Design patterns within the framework
  - Design patterns at the border of the framework (framework hook patterns)
- ▶ Additionally, a glossary and a search engine can be provided



# Realization with Elucidative Programming

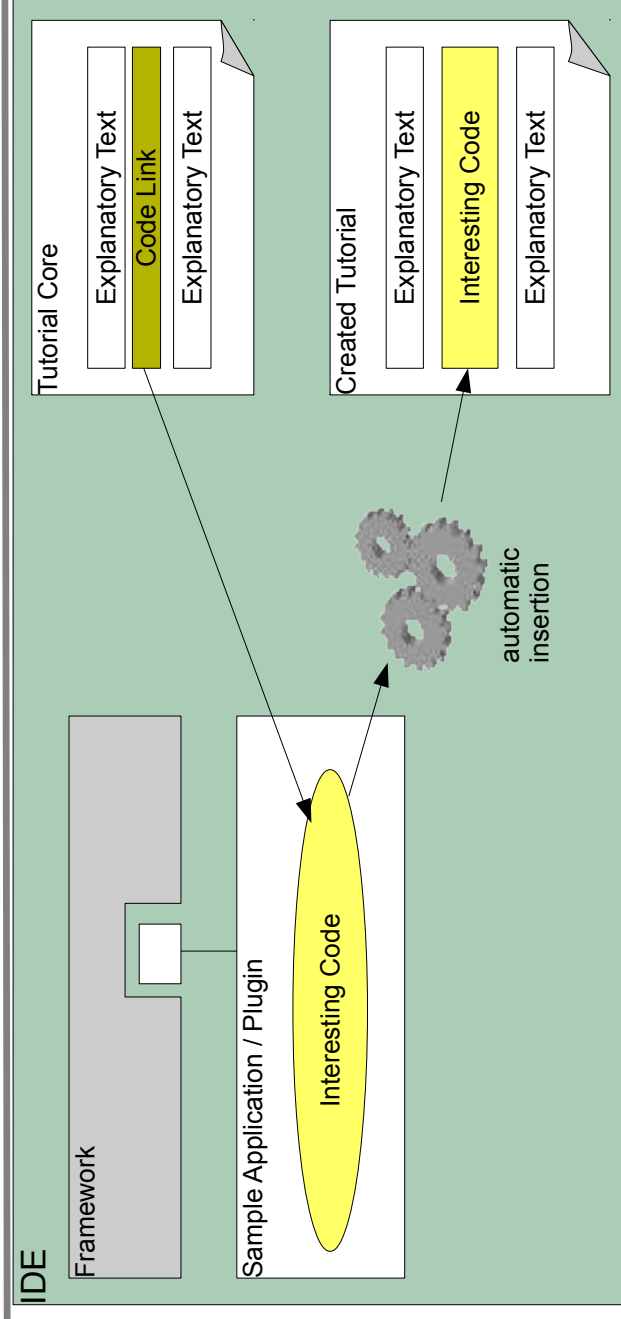
- ▶ **Elucidative programming** is programming by example
  - Basically cross-linked implementation documentation
  - Better form of literate programming (non-linear, but hypertext)
- ▶ 2 screens
  - Left: documentation
  - Right: source code
- ▶ A markup language marks up source code and puts fragments into the documentation
  - Crosslinking between source and documentation possible
- ▶ Documentation threads (as required for tutorials on level 2)
- ▶ Tools
  - Java elucidator <http://elucidator.sf.net>
  - Scheme elucidator
  - DocSewer tools for tutorial threads
    - DEFT <http://deftproject.org>

# Tutorial Creation – Conventional Approach



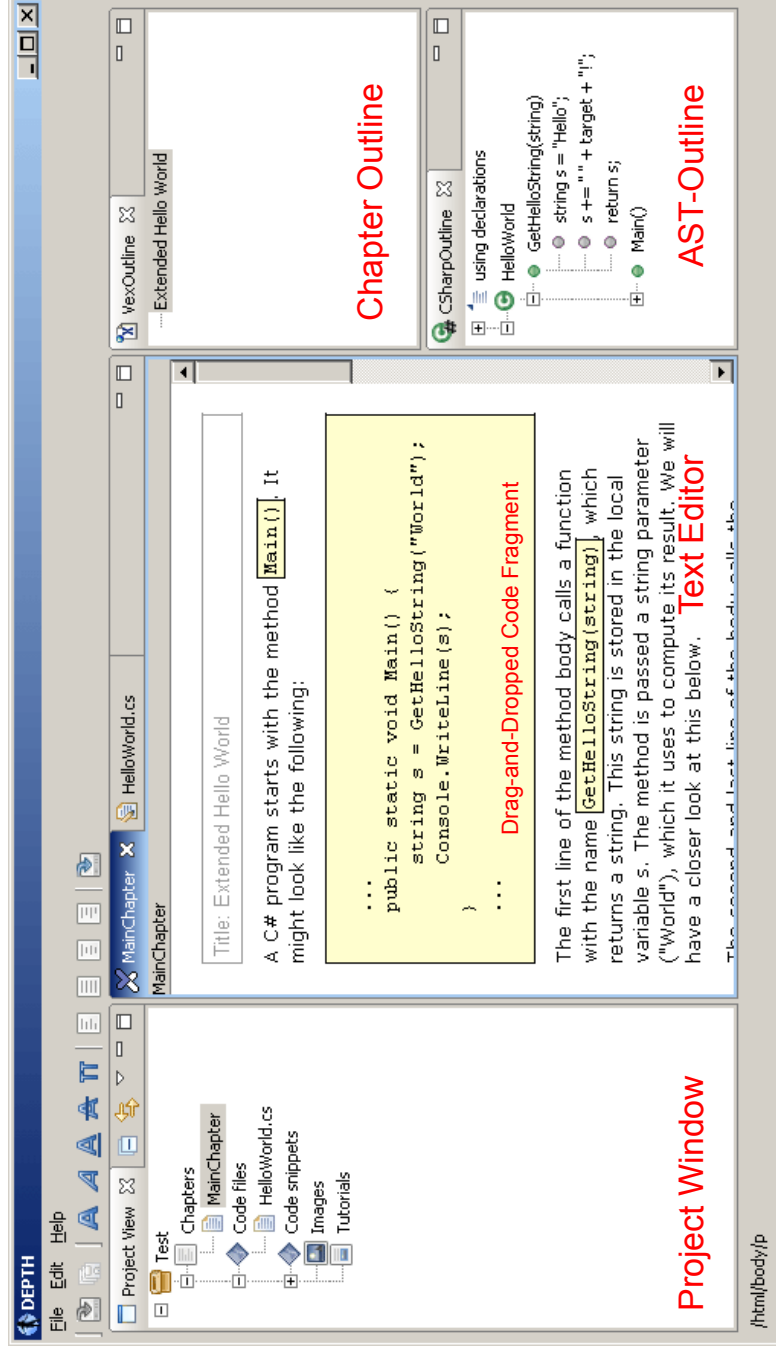
- ▶ Framework and Sample Plugin can be developed side by side
- ▶ Tutorial is detached and needs special treatment
  - code fragments are copied manually
  - documented code fragments can become inconsistent when framework and Sample Plugin evolve

# Solution - Tutorial Generation Environment



- ▶ Tutorial can be developed along with Framework and Sample Application
  - code not included directly, only linked
  - automatic tutorial update when original code changes

# Documenting HelloWorld with DEFT (Development Env. for Tutorials)



## Documenting HelloWorld

- ▶ write explanatory text
- ▶ embed code fragments via drag&drop
- ▶ set different styles for code fragments
  - code snippets
  - in-line fragments for variable-/method names
- ▶ select output format (HTML, PDF, ...)
- ▶ compile tutorial to output format



# HTML Output

Beschreibung des Frameset-Inhalts - Mozilla Firefox

Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe

W3C Validator zoom images in zoom images out linked images

file:///D:/output/helloWorld.html

A C# program starts with the method [Main\(\)](#). It might look like the following:

```
...  
public static void Main() {  
    string s = GetHelloString("World");  
    Console.WriteLine(s);  
}
```

**Code Fragment**

The first line of the method body calls a function with the name [GetHelloString\(string\)](#), which returns a string. This string is stored in the local variable `s`. The method is passed a string parameter ("World"), which it uses to compute its result. We will have a closer look at this below.

The second and last line of the body calls the method `Console.WriteLine(string)`, which is predefined in the C#

```
using System;  
  
public class HelloWorld {  
  
    static string GetHelloString(string target) {  
        string s = "Hello";  
        s += " " + target + "!";  
        return s;  
    }  
  
    public static void Main() {  
        string s = GetHelloString("World");  
        Console.WriteLine(s);  
    }  
}
```

**Complete Source Code**

# The End