23. Framework Documentation

Prof. Uwe Aßmann

TU Dresden

Institut für Software– und Multimediatechnik

Lehrstuhl Softwaretechnologie

12-1.0, 22.01.13

1

Design Patterns and Frameworks, © Prof. Uwe Aßmann

Prof. Uwe Aßmann, Design Patterns and Frameworks

---

Obligatory Literature

2

► 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

► Claas Wilke, Andreas Bartho, Julia Schroeter, Sven Karol, and Uwe Aßmann. Elucidative development for model-based documentation. In Carlo Furia and Sebastian Nanz, editors, Objects, Models, Components, Patterns – 50th International Conference, TOOLS, volume 7304 of Lecture Notes in Computer Science, pages 320-335. Springer Berlin / Heidelberg, 2012.

Prof. Uwe Aßmann, Design Patterns and Frameworks

References

3

► B. Minto. The Pyramid Principle. Part One: Logic in Writing. Pitman Publishing, London, 1991. First published by Minto International Inc. in 1987.

► G. Jimenz-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.

Prof. Uwe Aßmann, Design Patterns and Frameworks

Problem: How to Document a Framework?

4

► Framework understanding is hampered by many problems
  – Good documentation should help to solve them
  – Good framework contracts will help (trustworthy instantiation)
  – Good extension languages will help (framework composition)
► 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 the architecture of the framework
  – Framework integrity is related
  – Lack of knowledge of interactions between framework classes
  – Impact of instantiations cannot be estimated
  – Multiple solutions possible with the framework
  – Technical problems (platform knowledge, ..)

Prof. Uwe Aßmann, Design Patterns and Frameworks

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 course Softwaretechnologie-II)
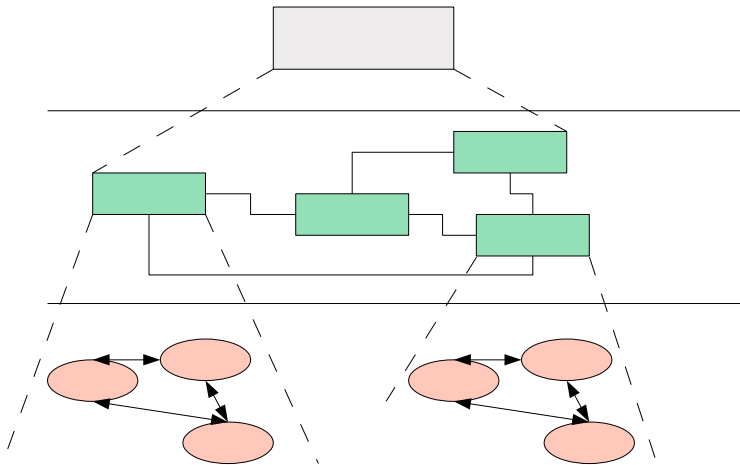
The Pyramid Principle in Framework Documentation

- ▶ 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?

User

Instantiator

Developer

Selection

Standard Usage

Detailed Design
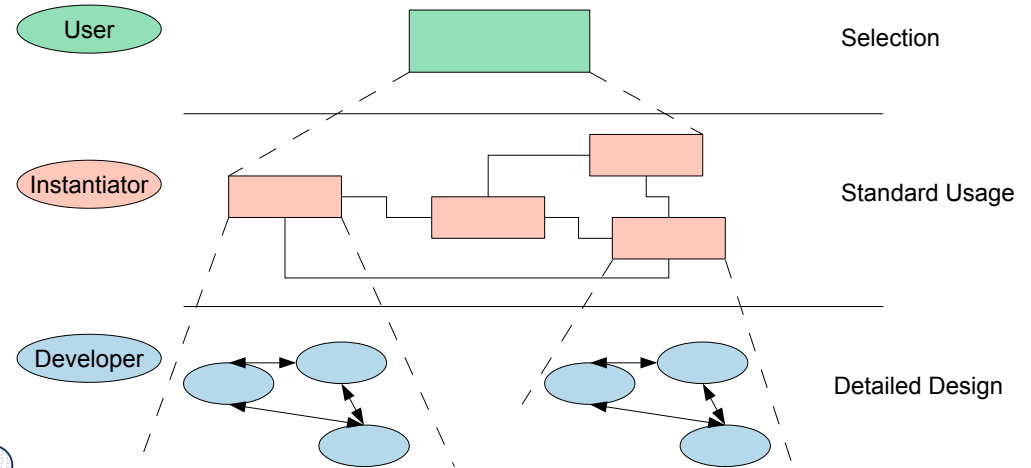
Level 1: Framework Selection Sheet

- ▶ 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.

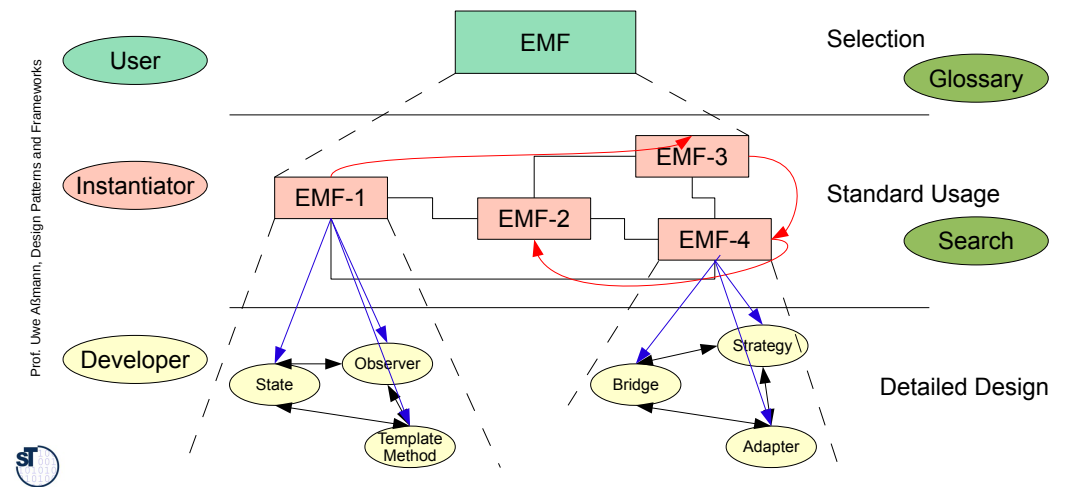# Application Pattern Documentation is Threaded

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

(diagram: User, Instantiator, Developer on left; EMF box at top with EMF-1, EMF-2, EMF-3, EMF-4; labels Selection, Standard Usage, Detailed Design)

Prof. Uwe Aßmann, Design Patterns and Frameworks
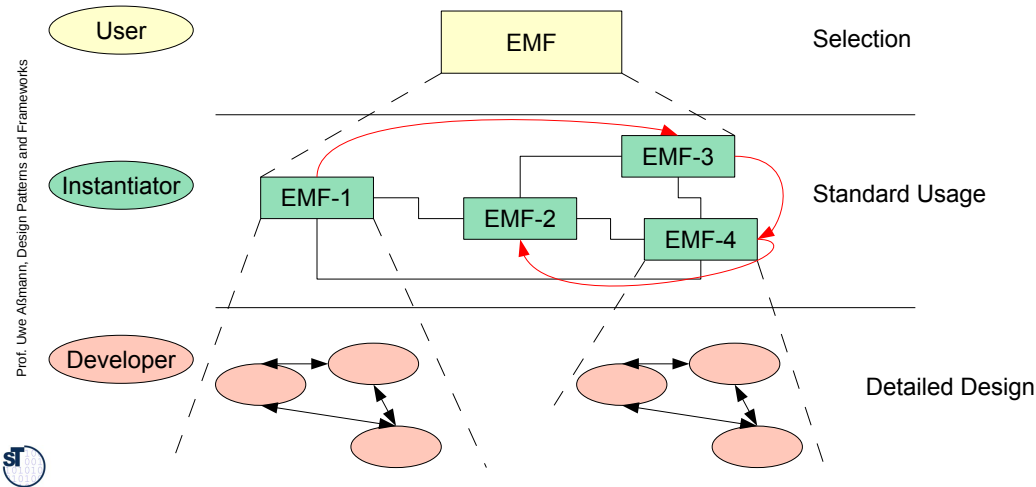
Third Level: Detailed Design

- ▶ 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

(diagram: User, Instantiator, Developer; EMF box with EMF-1, EMF-2, EMF-3, EMF-4; State, Observer, Template Method, Bridge, Strategy, Adapter; Glossary, Search; labels Selection, Standard Usage, Detailed Design)

Prof. Uwe Aßmann, Design Patterns and Frameworks

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

Prof. Uwe Aßmann, Design Patterns and Frameworks

Tutorial Creation – Conventional Approach

(diagram: IDE containing Framework, Sample Application / Plugin with Interesting Code; copy & paste to Text Editor containing Tutorial with Explanatory Text, Interesting Code, Explanatory Text)
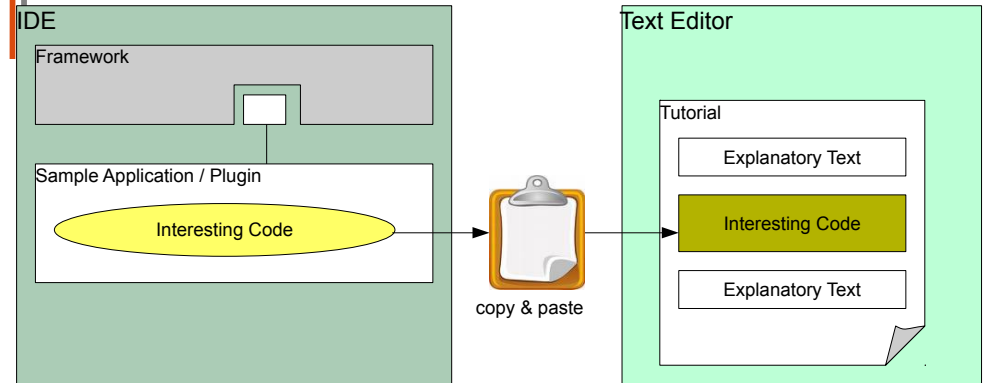
- ▶ Framework and Sample Plugin can be developed side by side
- ▶ Tutorial is detached and needs special treatment
  - – code fragments are copied manually
  - – documented code fragments can become inconsistent when framework and Sample Plugin evolve

Prof. Uwe Aßmann, Design Patterns and Frameworks

Solution - Tutorial Generation Environment

13

IDE

Framework

Sample Application / Plugin

Interesting Code

automatic insertion

Tutorial Core

Explanatory Text

Code Link

Explanatory Text

Created Tutorial
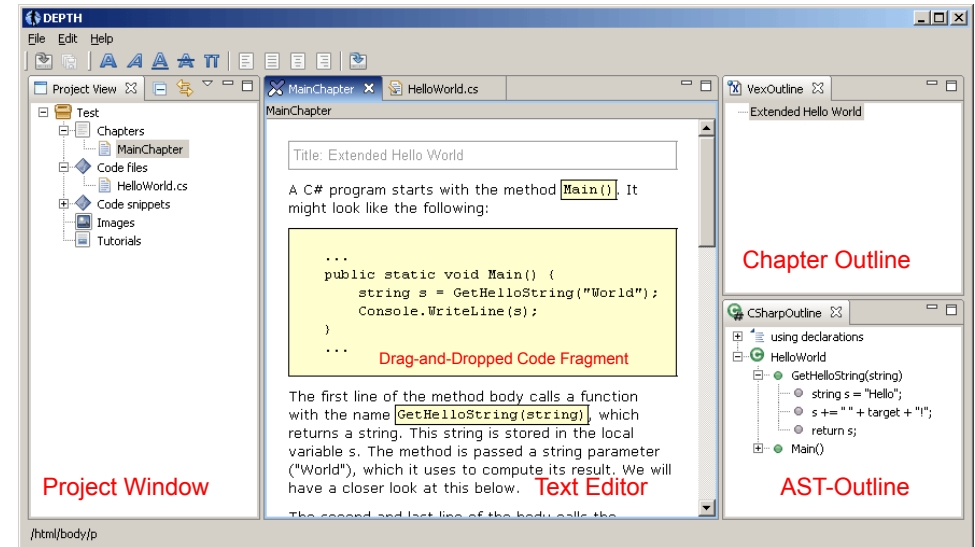
Explanatory Text

Interesting Code

Explanatory Text
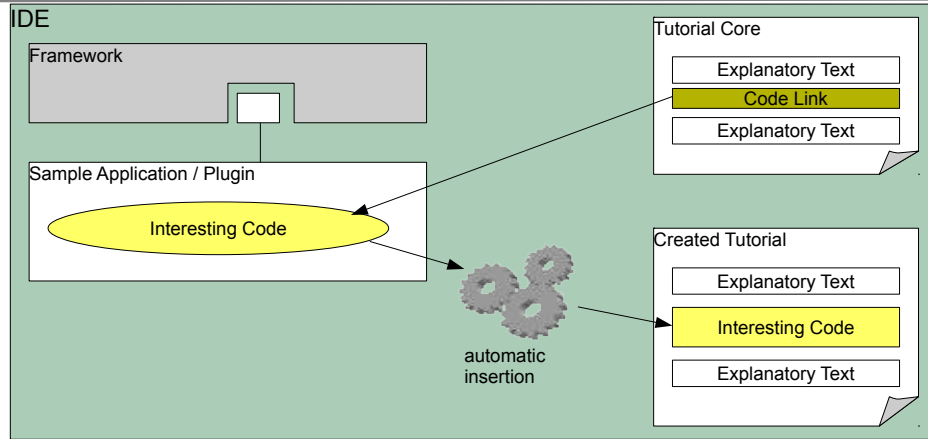
► Tutorial can be developed along with Framework and Sample Application
  – code not included directly, only linked
  – automatic tutorial update when original code changes

Documenting HelloWorld with DEFT (Development Env. for Tutorials)

14

DEPTH

File Edit Help

Project View ⊠

Test
  Chapters
    MainChapter
  Code files
    HelloWorld.cs
  Code snippets
  Images
  Tutorials

MainChapter

HelloWorld.cs

MainChapter

Title: Extended Hello World

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);
}
...
```

Drag-and-Dropped 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

/html/body/p

VexOutline ⊠

Extended Hello World

Chapter Outline

CSharpOutline ⊠

using declarations
HelloWorld
  GetHelloString(string)
    string s = "Hello";
    s += " " + target + "!";
    return s;
  Main()

Project Window   **Text Editor**   **AST-Outline**

■ http://deftproject.org

Documenting HelloWorld

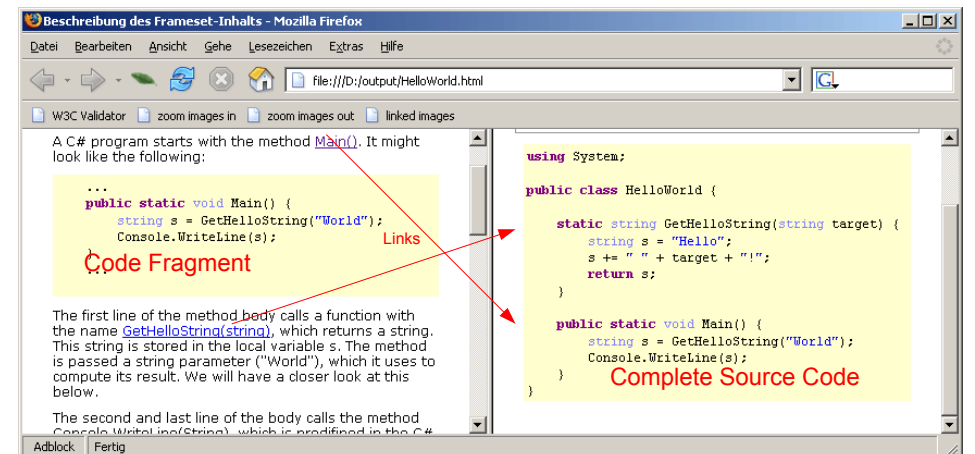15

► 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

16

Beschreibung des Frameset-Inhalts - Mozilla Firefox

Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe

file:///D:/output/HelloWorld.html

W3C Validator   zoom images in   zoom images out   linked images

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);
}
```

Links

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

Adblock   Fertig

Prof. Uwe Aßmann, Design Patterns and Frameworks