Merciless Refactoring with Eclipse

Martin Lippert, Bernd Schiffer
it-agile GmbH
{martin.lippert, bernd.schiffer}@it-agile.de
http://www.it-agile.de/

JAX 2006, Wiesbaden

Part 1: Daily Refactoring

- Part 1: Daily Refactoring
  - Quick fixes
  - Local refactorings
  - Small refactorings
  - Hands-on demonstrations
- Part 2: Large Refactorings
  - Large refactorings
  - Dependency management
  - Tools to detect and control refactorings
  - Some Demos

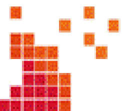
Contents 1/2

- Refactoring – a short introduction
- The classics:
  - Rename and Move
- Working with variables
  - Extract Local Variable
  - Convert Local Variable into Field
- Working with methods
  - Extract Method
  - Change Method Signature
  - Inline Method

Contents 2/2

- Working with types
  - Extract Interface
  - Infer Generic Type Arguments
- Combined refactorings:
  - Inline Constructor
- Links and books

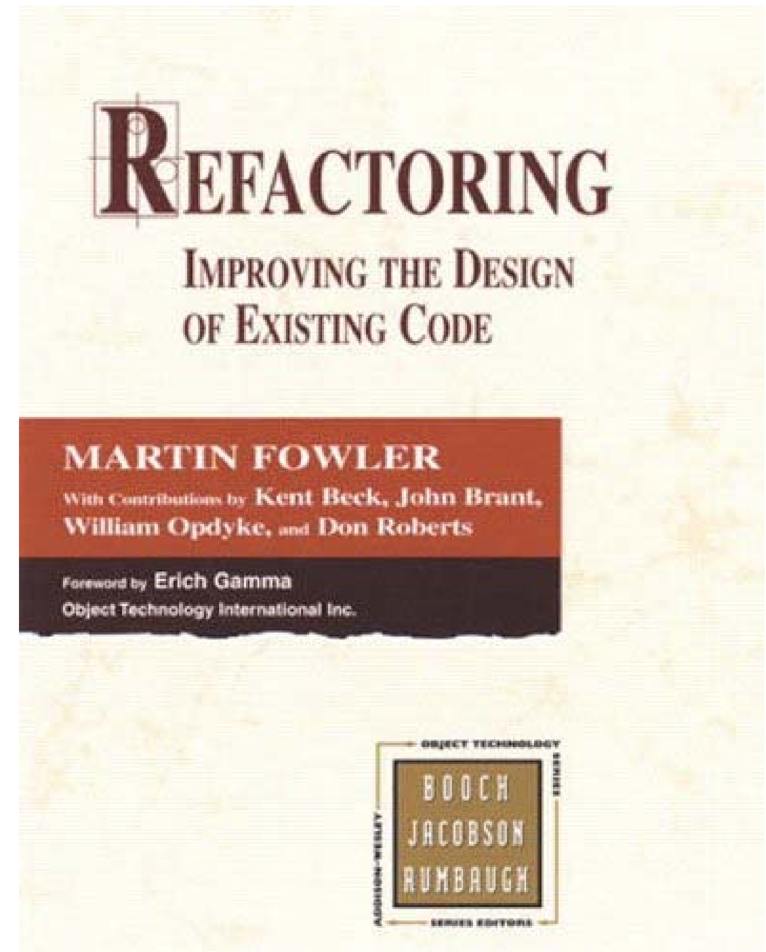
* The material provided here is based on Eclipse 3.1



What is refactoring?

- **„A change made to the internal structure of software to make it easier to unterstand and cheaper to modify without changing its observable behavior"**

[Fowler 99]



Refactoring mechanics

- [Fowler 99] describes detailed mechanics for each refactoring. These mechanics allow developers to realize the refactoring in small steps while reducing the danger of changing the behavior (introducing new bugs)
- Nevertheless some refactorings are expensive to implement:
  - Rename a method requires to adapt all references to this method manually
- The danger of introducing errors or changing the behavior still exists
  - A good test suite is required to be as safe as possible

Refactoring tools

- It is a good idea to automate as many refactorings as possible
- **But: The tool must ensure that it does not change the behavior of the system (or should warn about possible changes)**

- Smalltalk Refactoring Browser was the first tool that automated refactorings
  - Written by John Brant & Don Roberts

- Meanwhile most Java IDEs include refactoring support.
  - IDEs for other languages appear

Our goal

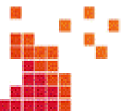
- **We want to refactor our systems by using the automated refactorings of Eclipse – and nothing else !!!**
  - Let Eclipse ensure that the behavior of our system does not change
  - Speed up the refactoring work

  - Identify the circumstances where we should be attentive while using the refactoring support of Eclipse

Refactoring in practice

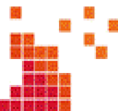
- Rather than talking about all the refactoring possibilities of Eclipse in theory, I would like to present them interactively

- The slides are the reference
  - You can find all refactorings explained in the slides
  - But I will not show all slides here



Refactoring: Rename

- Rename works on:
  - Packages
  - Classes
  - Methods
  - Parameters
  - Variables
- Automatically adapts all references to those elements, including:
  - File names
  - Folder names
  - Javadoc `@param` tags



Attention: “Rename in file” is different

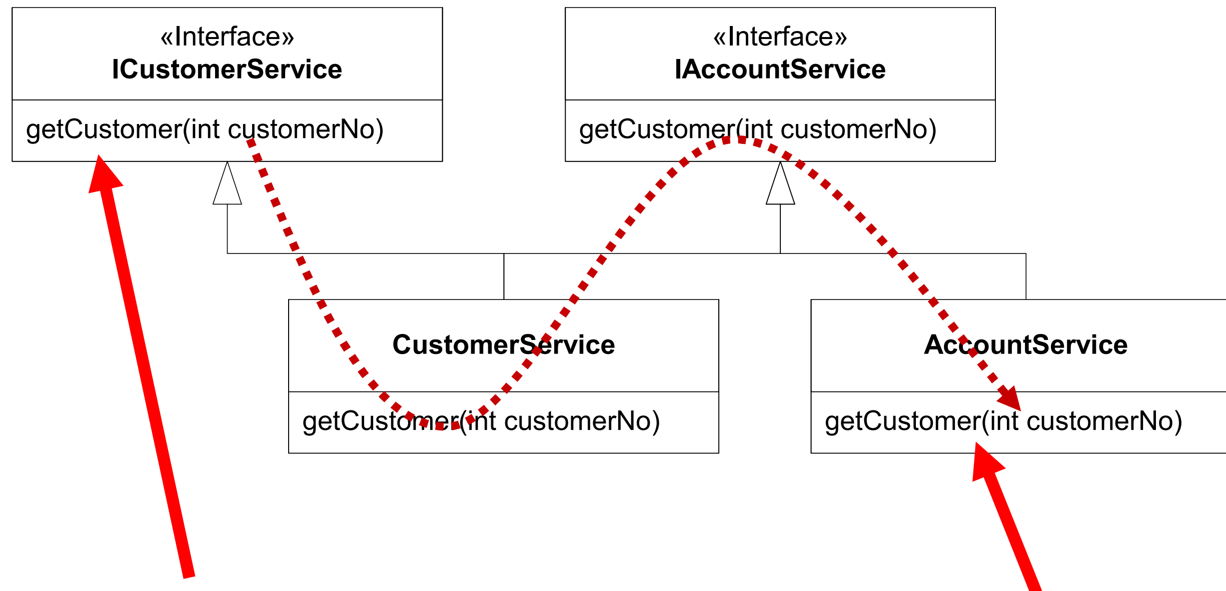
- The “Rename in file” feature is different from the rename refactoring:
  - “Rename in file” automatically updates all references to the selected element within the same file – **and nothing else.**
  - Does not check whether the element is used from outside and does not update those references

- **Never use Rename in file for non-local elements – otherwise you assume the risk of introducing errors and behavior changes**

- Use “Rename in file” only for local elements
  - Local variables
  - Parameters
  - Private attributes
  - Private methods
  - Private inner classes

Attention: Renaming of interface methods

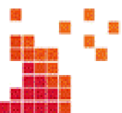
- If you rename a method in a class that implements identical methods from two or more interfaces, all definitions of that method in all implemented interfaces change (and therefore in all classes that implement those interfaces)



Method rename here means also changing the method name here

Rename and non-java sources ???

- The rename refactoring is able to find all references to a class name, for example, in Java files
  - By using the parser information
- What happens to class-references in non-java files?
  - Extension definitions in plugin.xml files?
  - JavaServer Pages?
  - XML configuration files (e.g. Spring)?

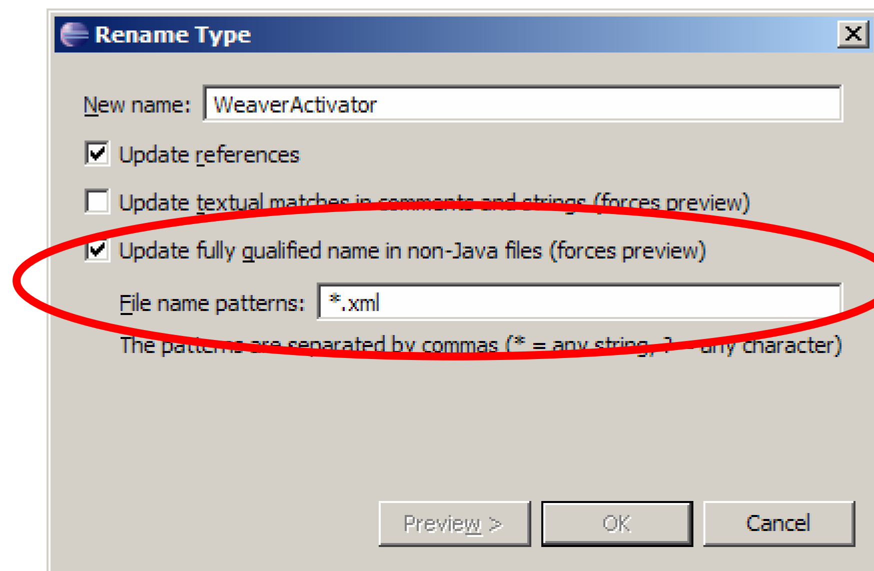


Rename and .xml files

- The rename refactoring of Eclipse is able to find class-references in any kind of file (e.g. .xml) if the class if fully qualified

- This works for:
  - plugin.xml
  - Spring config files

- This does not work for:
  - import-like class usages
  - method names



Refactoring: Move

- Works on:
  - Classes
  - Packages

- Automatically adapts all references to moved elements, including:
  - Import statements
  - Full-qualified class statements

Refactoring: Extract Local Variable

- Allows you to extract a statement into a local variable at a single keystroke
- Replaces **all occurrences** of the statement (within the same block) with the new local variable

- Seldom used refactoring because most people are used to cut&paste those statements into new variable declarations
- But this refactoring is extremely useful for everyday programming

Extract Local Variable

```
    if (wcp.getGeneratedClasses().length > 0) {
        for (int i = 0; i < wcp.getGeneratedClasses().length; i++) {
            String generatedClassName = wcp.getGeneratedClasses()[i];
            byte[] generatedClassBytecode = wcp.getGeneratedClassBytecode(generatedCla:
            result.addAdditionalClasses(generatedClassName, generatedClassBytecode);
        }
    }

:h (RuntimeExce
ystem.out.print
.printStackTrac
:h (Exception e
ystem.out.print
.printStackTrac

n result;

: the information about new dependencies out of the message
```

Extract Local Variable

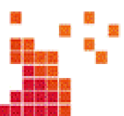
Variable name: generatedClasses

☑ Replace all occurrences of the selected expression with references to the local variable

☐ Declare the local variable as 'final'

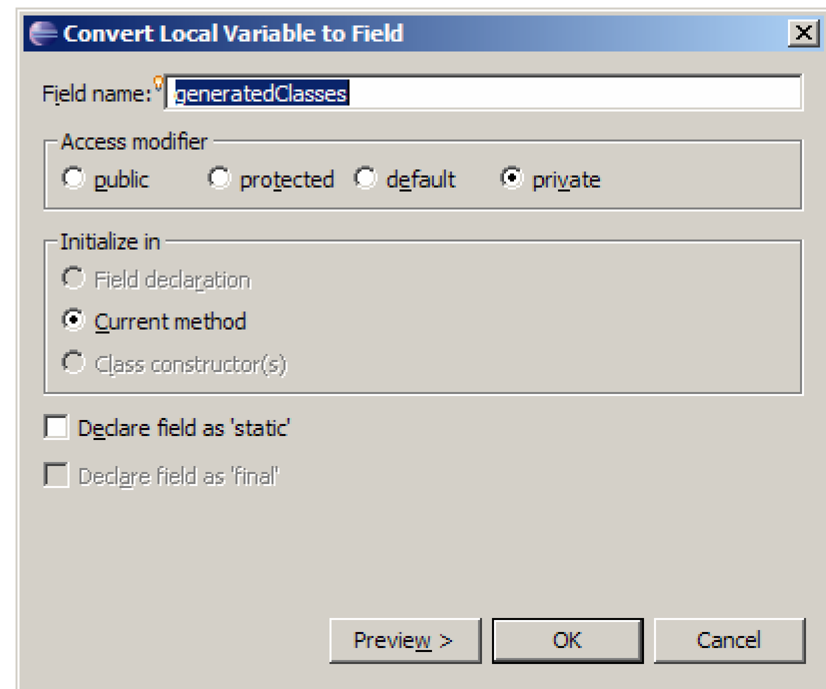
Signature Preview: String[] generatedClasses

Preview > | OK | Cancel



Refactoring: Convert Local Variable to Field

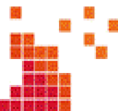
- Allows you to convert a local variable into a field of the surrounding class at a single keystroke
- Seldom used refactoring because most people are used to cut&paste those declarations from the local context into the field declarations part of a class
- But this refactoring makes it a lot easier



Refactoring: Extract Method

- Allows you to extract a code block into a separate method at a single keystroke:
  - Generates the necessary set of parameters
  - Create the correct return type
  - Warns you if more than one return value is necessary

- This is extremely useful to split large methods into smaller ones
- I also use this refactoring to experiment with different method splittings



Extract Method example

```
    try {
        WeavingClassFileProvider wcp = new WeavingClassFileProvider(className,bytecode);
        weaver.weave(wcp);

        if (wcp.isChanged()) {
            result.setAdditionalDependencies(getNewDependencies());
            result.setTransformedBytecode(wcp.

            String[] generatedClasses = wcp.ge
            if (generatedClasses.length > 0) {
                for (int i = 0; i < generatedC
                    String generatedClassName
                    byte[] generatedClassBytec
                    result.addAdditionalClasse
                }
            }
        }
    } catch (RuntimeException e) {
        System.out.println("problem during wea
        e.printStackTrace();
    } catch (Exception e) {
        System.out.println("problem during wea
        e.printStackTrace();
    }
    return result;
}

/**
 * Extract the information about new dependenc
 * handler from the aspectj weaving world. The
 * cleaned after this.
 *
 * @return The names of the bundles that are n
 */
```

Extract Method

Method name: handleDependencies

Access modifier: ( ) public ( ) protected ( ) default (•) private

Parameters:

Type	Name
TransformationResult	result
WeavingClassFileProvider	wcp

[Edit...] [Up] [Down]

☐ Add thrown runtime exceptions to method signature

☐ Generate method comment

☐ Replace duplicate code fragments

Method signature preview:

```
private void
handleDependencies(TransformationResult
```

[Preview >] [OK] [Cancel]

Attention: Extract Method

- If you extract a method from an anonymous inner class that exists inside a non-anonymous inner class, you have to take care:
  - If a method in the non-anonymous inner class exists with the same signature as your extracted method, Eclipse does not warn you about possible conflicts

- Extract the method into the anonymous inner class
  - Everything is fine
- Extract the method into the non-anonymous inner class
  - Results in compiler warnings because a method with the same signature already exists in that class
- Extract the method into the surrounding class
  - Results in possible behavior changes because the anonymous inner class calls the method with the same signature from the non-anonymous inner class and not the extracted one in the outer class

Refactoring: Change Method Signature

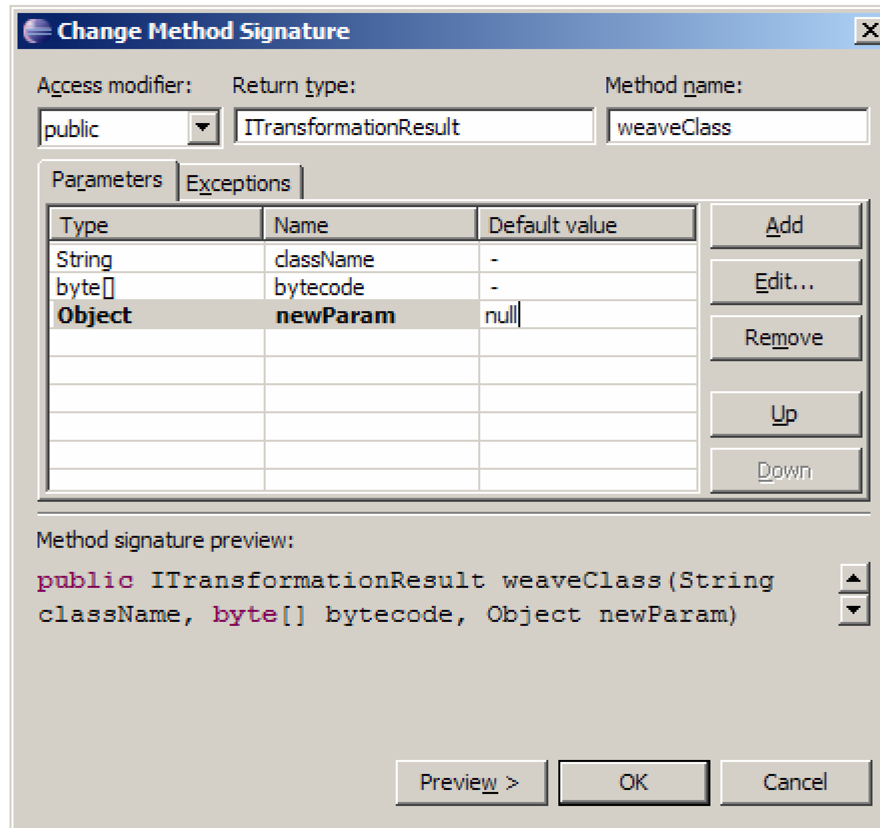
it-agile

- Allows you to change the signature of a method at a single click
  - Rename the method itself
  - Change the access modifier
  - Add, remove, rename and reorder parameters (including default values for new parameters)
  - Change the type of the return value or parameters
  - Add and remove exceptions

- Adapts all references to this method, if possible
  - Interfaces as well as implementing classes
  - Calls to this method

- **This is one of the most powerful refactorings within Eclipse (from my point of view)**

Refactoring: Change Method Signature



Inline Considered Helpful

- Inline refactoring replaces the invocation of the method with the method's code
- Eclipse warns you in case of overridden methods
- Seems like this refactoring creates duplicated code
- Extremely useful to remove deprecated calls:
  - Implement the old method by using the new methods
  - Then the implementation of the old method looks like the client code of the new method(s)
  - Inline old method to replace all invocations of the old method by invocations of the new method(s)

Best Practices: Inline Method

```
/**
 * @deprecated use druckeDokument instead
 */
public void drucke (String dok) {
  druckeDokument(new Dokument(dok));
}

public void druckeDokument (Dokument obj) {
  ... implementation ...
}
```

```
...
String meinDokument = ...;
...
meinDrucker.drucke(meinDokument);
...
```

Best Practices: Inline Method

```
/**
 * @deprecated use druckeDokument instead
 */
public void drucke (String dok) {
  druckeDokument(new Dokument(dok));
}

public void druckeDokument (Dokument obj) {
  ... implementation ...
}
```

```
...
String meinDokument = ...;
...
meinDrucker.druckeDokument(new Dokument(meinDokument));
...
```

Refactoring: Extract Interface

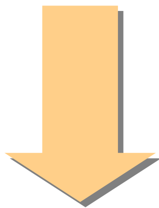
it-agile

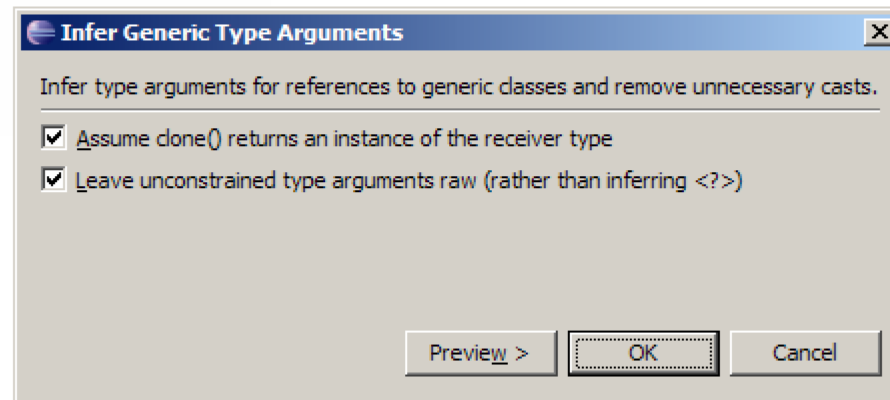
- Extract a new interface from an existing class very comfortable by selecting the appropriate methods.

- **The secret power of this refactoring is:**
  - Eclipse changes declarations in the client code from the class to the interface type where possible
  - You not just extract the interface type, you also use the new abstraction in the client code right away

Refactoring: Infer Generic Type Arguments

```
public void foo() {
    List list = new ArrayList();
    list.add("Hallo");
    list.add("Foo");
}
```






```
public void foo() {
    List<String> list = new ArrayList<String>();
    list.add("Hallo");
    list.add("Foo");
}
```

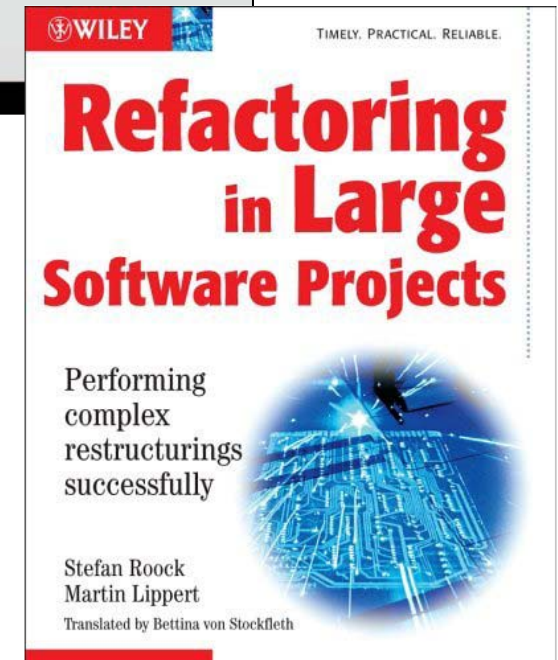
Advanced: Inline Constructor

- Problem: A constructor that is deprecated and uses this(..) to adapt invocations to a new constructor.
- But we cannot inline the constructor since the inline refactoring is allowed for methods only.

- **Solution:**
  - **1. Introduce Factory for the deprecated constructor.**
  - **2. Replace the body of the factory (to use the new constructor)**
  - **3. Inline the factory method.**

Some advertisement ☺

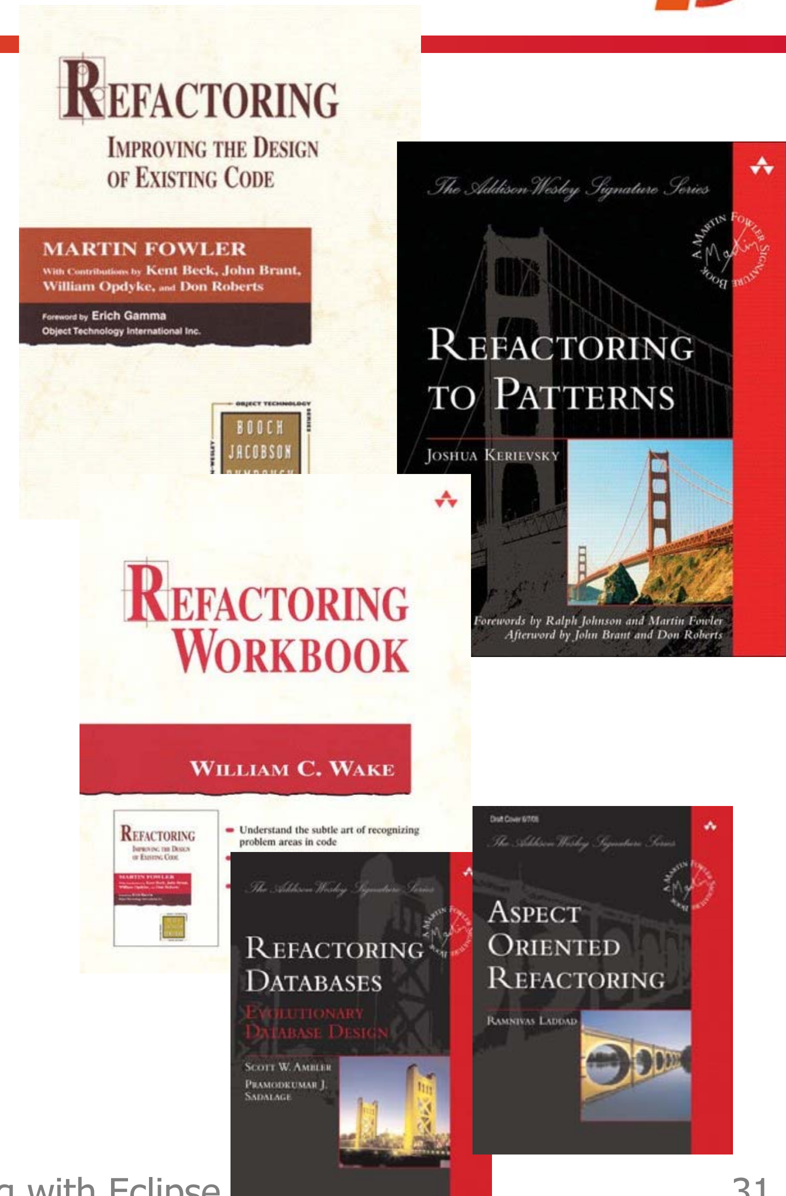
- Best practices for performing complex refactorings
- Covers:
  - Short introduction to refactoring
  - Architecture smells
  - Large refactorings
  - API-Refactorings
  - Database-Refactorings
  - Guest chapter: Finding and analyzing architecture smells
- "War Stories" from Sven Gorts, Berrin Ileri, Dierk König, Klaus Marquardt, Jens-Uwe Pipka, Markus Völter and Henning Wolf



Other books

- Martin Fowler: *Refactoring – Improving the Design of Existing Code*, Addison-Wesley, 1999
- Joshua Kerievsky: *Refactoring to Patterns*, Addison-Wesley, 2004
- William Wake: *Refactoring Workbook*, Addison-Wesley, 2003.

- On the road:
  - Ramnivas Laddad: *Aspect Oriented Refactoring*, Addison-Wesley, 2006
  - Scott W. Ambler, Pramodkumar J. Sadalage: Refactoring Databases: Evolutionary Database Design, Addison-Wesley, 2006



The end.

- **Thank you for your attention. Feedback is welcome!**
Martin Lippert: martin.lippert@it-agile.de
Bernd Schiffer: bernd.schiffer@it-agile.de

- Some interesting references:
  - http://www.refactoring.com/: Maintained by Martin Fowler, contains a lot of useful other references, articles, tools catalog, …
  - http://www.refactoring.be/: Refactoring Thumbnails as a visualization for refactorings
  - http://groups.yahoo.com/group/refactoring: Refactoring mailing list at Yahoo