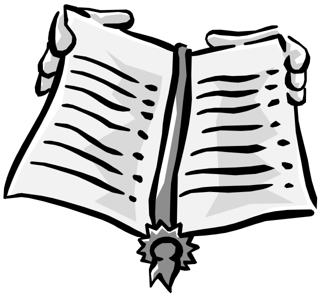
Refactoring

Michael Stal
Senior Principal Engineer
Siemens AG, Corporate Technology

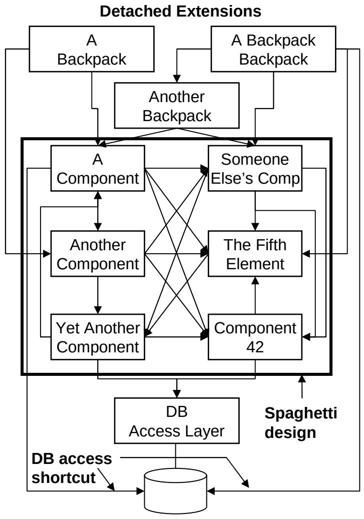
Michael.Stal@siemens.com

Content

- **Motivation**
- **Reengineering and Refactoring**
- **Refactoring within a Process**
- **Refactoring Examples**
- **Refactoring: Additional Issues**
- **Refactoring Tools**
- **Conclusions**
- **References**



Motivation



After a while, many architectures tend to look like this one ...

- The original architecture vision is hardly visible.
- Design flaws are scaffold by many small and local "corrections."
- Missing parts are attached via backpacks.

However:

Such an architecture is doomed to fail before it goes into implementation or operation, because it suffers from:

- developmental qualities like flexibility and maintainability.
- operational qualities like performance and scalability.

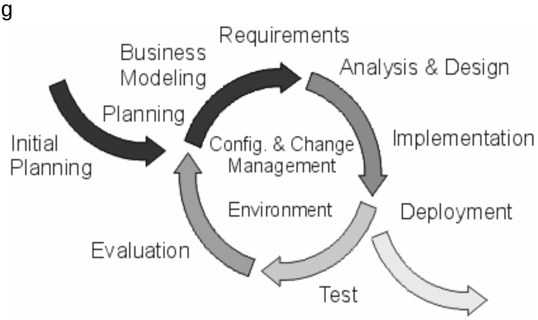
Refinements and Refactorings

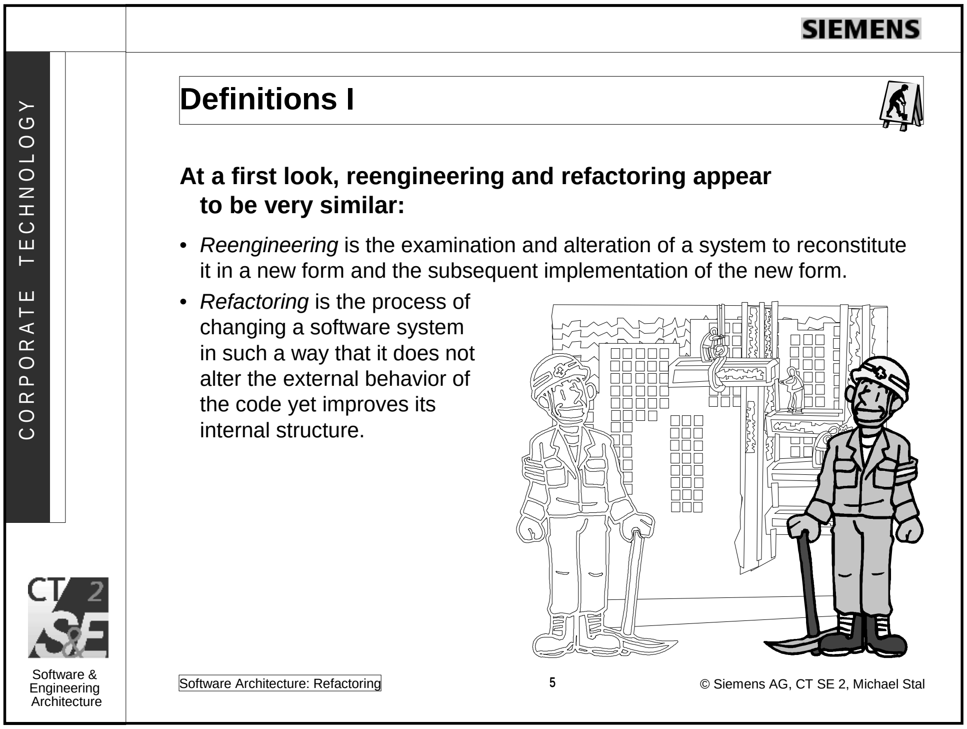
Therefore:

Create a software architecture step-wise via a number of well-defined, small increments. Each increment includes:

- top-down refinement activities to detail and complete the software architecture.
- bottom-up refactoring activities to garden and clean-up inconsistent or insufficient design decisions.

The process stops if the software architecture is complete and consistent in all its parts and details.

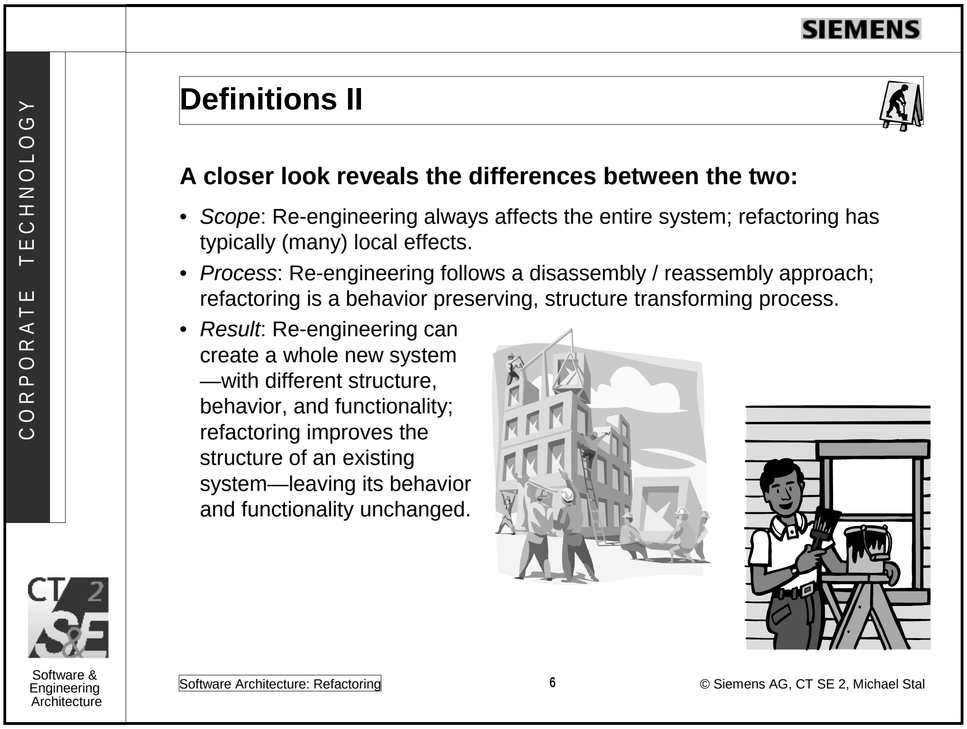




Definitions I

At a first look, reengineering and refactoring appear to be very similar:

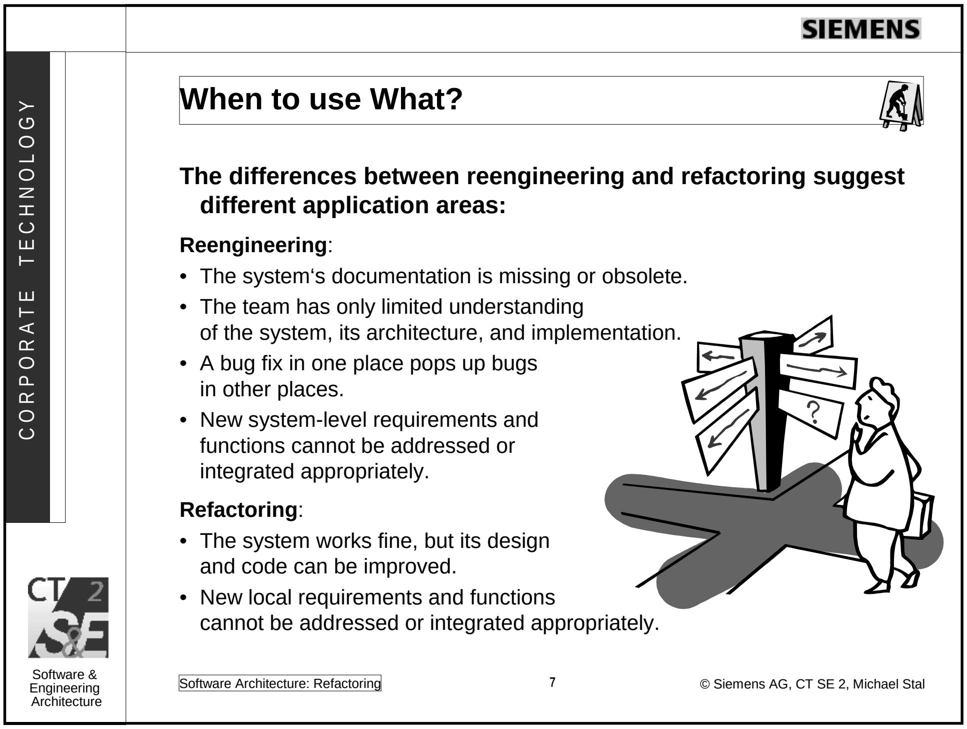
- *Reengineering* is the examination and alteration of a system to reconstitute it in a new form and the subsequent implementation of the new form.
- *Refactoring* is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.



Definitions II

A closer look reveals the differences between the two:

- *Scope*: Re-engineering always affects the entire system; refactoring has typically (many) local effects.
- *Process*: Re-engineering follows a disassembly / reassembly approach; refactoring is a behavior preserving, structure transforming process.
- *Result*: Re-engineering can create a whole new system—with different structure, behavior, and functionality; refactoring improves the structure of an existing system—leaving its behavior and functionality unchanged.



SIEMENS

CORPORATE TECHNOLOGY

When to use What?

The differences between reengineering and refactoring suggest different application areas:

Reengineering:

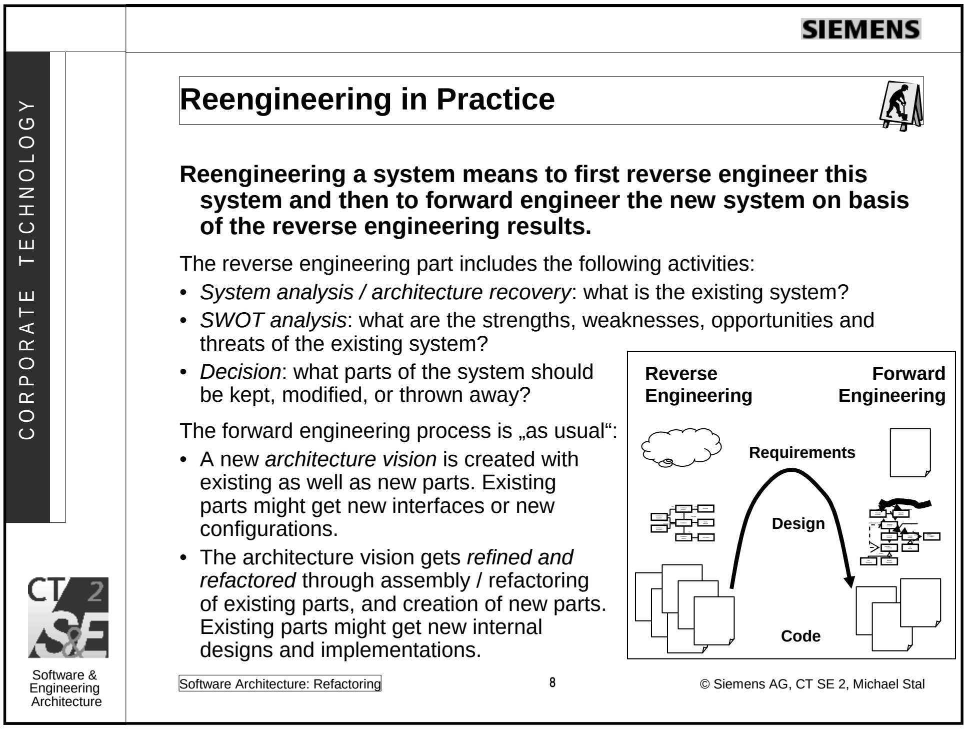
- The system‘s documentation is missing or obsolete.
- The team has only limited understanding of the system, its architecture, and implementation.
- A bug fix in one place pops up bugs in other places.
- New system-level requirements and functions cannot be addressed or integrated appropriately.

Refactoring:

- The system works fine, but its design and code can be improved.
- New local requirements and functions cannot be addressed or integrated appropriately.

CT 2 S&E

Software & Engineering Architecture



SIEMENS

CORPORATE TECHNOLOGY

Reengineering in Practice

Reengineering a system means to first reverse engineer this system and then to forward engineer the new system on basis of the reverse engineering results.

The reverse engineering part includes the following activities:

- *System analysis / architecture recovery*: what is the existing system?
- *SWOT analysis*: what are the strengths, weaknesses, opportunities and threats of the existing system?
- *Decision*: what parts of the system should be kept, modified, or thrown away?

The forward engineering process is „as usual“:

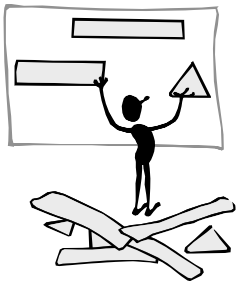
- A new *architecture vision* is created with existing as well as new parts. Existing parts might get new interfaces or new configurations.
- The architecture vision gets *refined and refactored* through assembly / refactoring of existing parts, and creation of new parts. Existing parts might get new internal designs and implementations.

CT 2 S&E

Software & Engineering Architecture

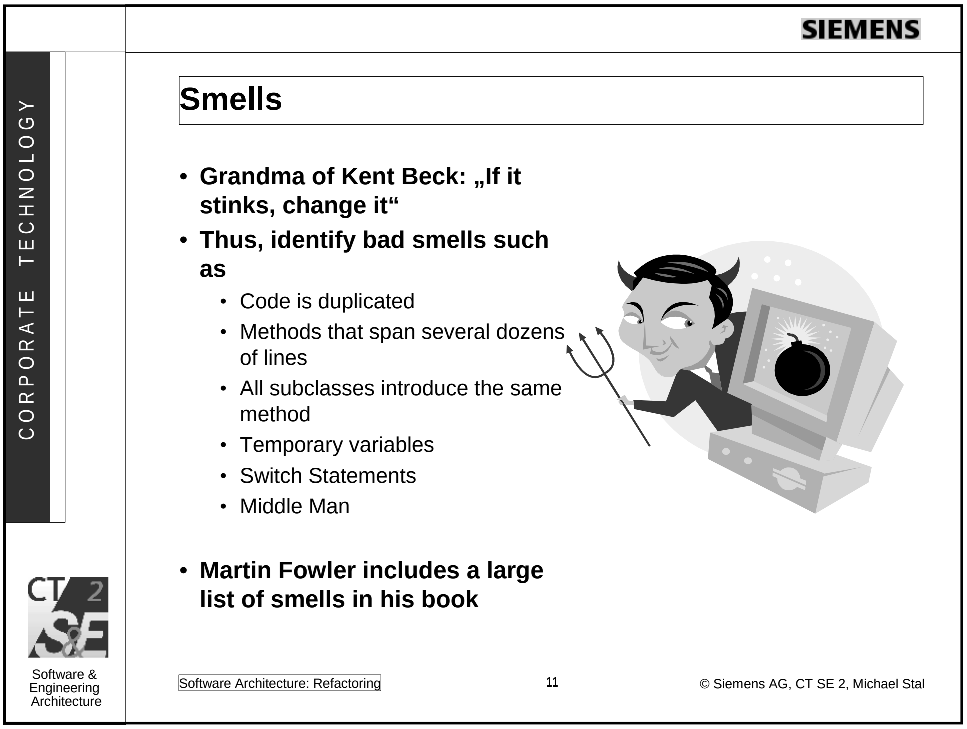
Refactoring in Depth

- **What is Refactoring?**
- **According to Martin Fowler it is**
  - *„... the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure“*
  - *„... a disciplined way to clean up code that minimizes the chances of introducing bugs“*



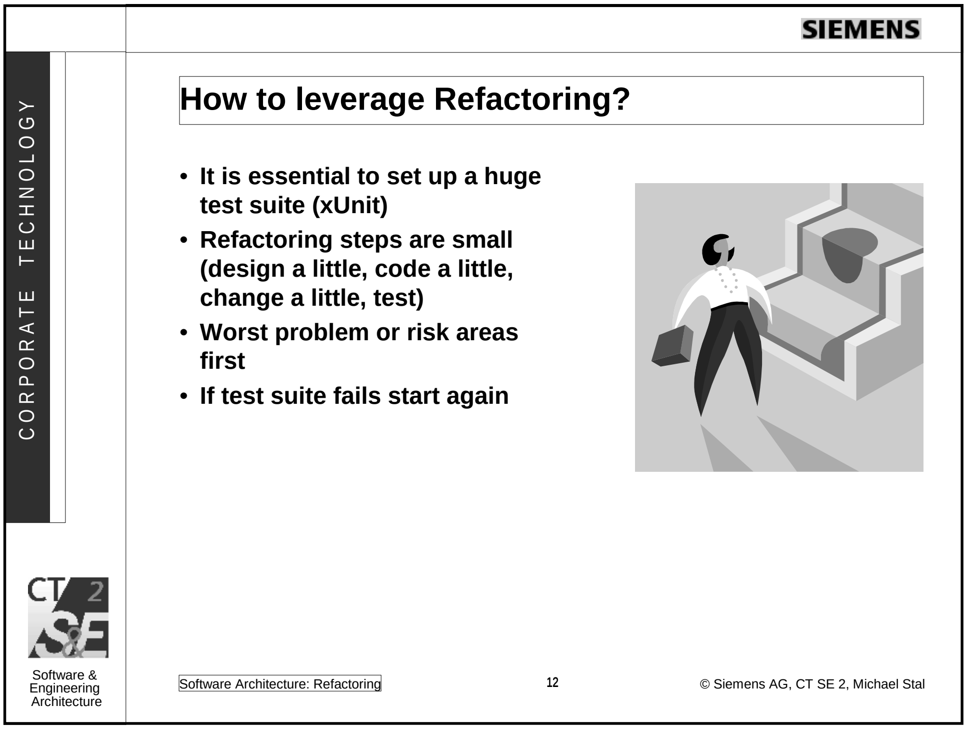
Reasons for Refactoring

- **Reasons to use Refactoring:**
  - Design improvement and maintenance
  - Better readability
  - Bugs
- **The Rules of Three:**
  - Refactor before adding new functionality. E.g., when structure prevents simple additions
  - Refactor when fixing bugs because refactoring helps to find the bug
  - Within Code Reviews to apply improvements



Smells

- **Grandma of Kent Beck: „If it stinks, change it“**
- **Thus, identify bad smells such as**
  - Code is duplicated
  - Methods that span several dozens of lines
  - All subclasses introduce the same method
  - Temporary variables
  - Switch Statements
  - Middle Man
- **Martin Fowler includes a large list of smells in his book**



How to leverage Refactoring?

- **It is essential to set up a huge test suite (xUnit)**
- **Refactoring steps are small (design a little, code a little, change a little, test)**
- **Worst problem or risk areas first**
- **If test suite fails start again**

Some Properties of Refactorings

- **A Refactoring reveals the following parts:**
  - *Name*, for example*: Extract Method*
  - *Summary (of Situation),* for example: Code fragment that can be grouped together. Turn the fragment into a method whose name explains the purpose of the method
  - *Motivation*. For example, use Extract Method when encountering long methods or replicated code
  - *Mechanics*, for example:
    - Create a new method and name it after the intention of the method
    - Copy extracted code from source to new target method
    - Scan extracted method for local variables
    - If one mutated local variable, make method as a query that returns that local variable's value. If more than one you might need to apply additional refactorings first (e.g., Split Temporary Variable)
    - Read-only local variables will be passed as parameters to new target method
    - Compile
    - Replace in source-code extracted code with call to new target method
    - Compile and test
  - *Examples*: illustrate usage (see previous slide)
  - Refactorings might be considered like patterns: forces, context, problem, solution
  - Most refactoring are reversible (see Inline Method refactoring)

Refactoring Examples: Extract Method

- **Note 1: examples are in Java/C# only for sake of brevity**
- **Note 2: the subsequent examples only show the mechanics for the same reason**

```
void printFormatted(string text) {
   System.out.println(„Copyright (c) 1006, Siemens AG“);
   System.out.println(„Author: Michael Stal“);
   printRest(text);
}
```

⇓

```
void printFormatted(string text) {
   printHeader();
   printRest(text);
}

printHeader() {
   System.out.println(„Copyright (c) 1006, Siemens AG“);
   System.out.println(„Author: Michael Stal“);
}
```

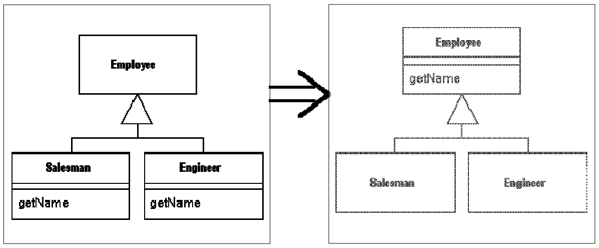
Refactoring Examples: Rename Method

```
void accCustDB() {  ... } // ???
```

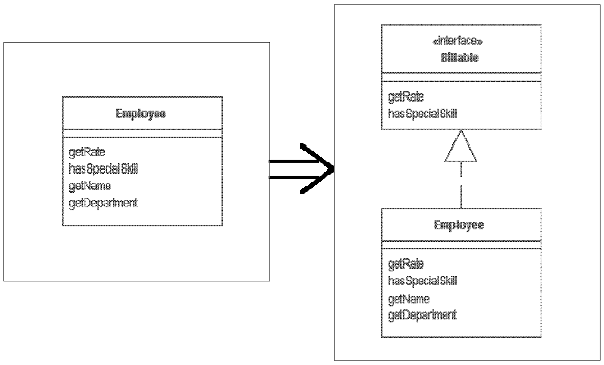
⇓

```
void accessCustomerDatabase() { ... }
```

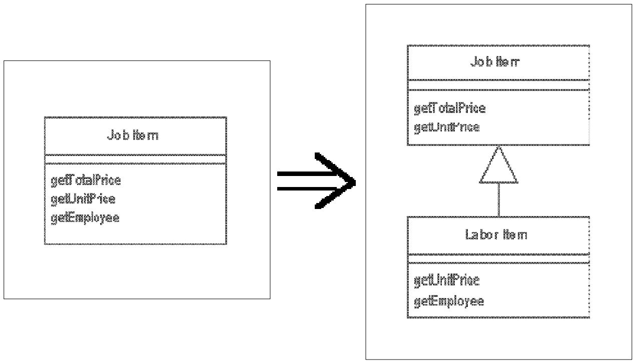
Refactoring Examples: Pull Up Method



Refactoring Examples: Extract Interface

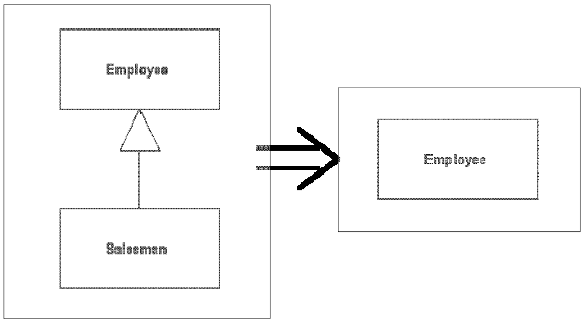


Refactoring Examples: Extract Subclass



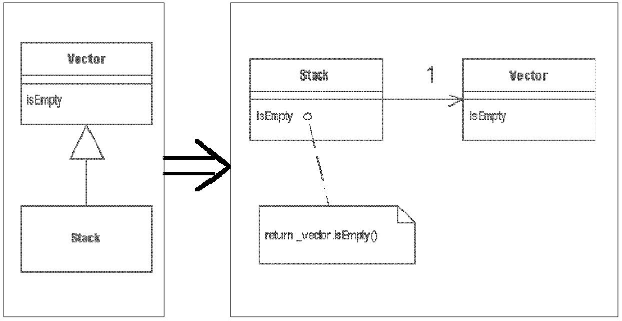
Refactoring Examples: Collapse Hierarchy

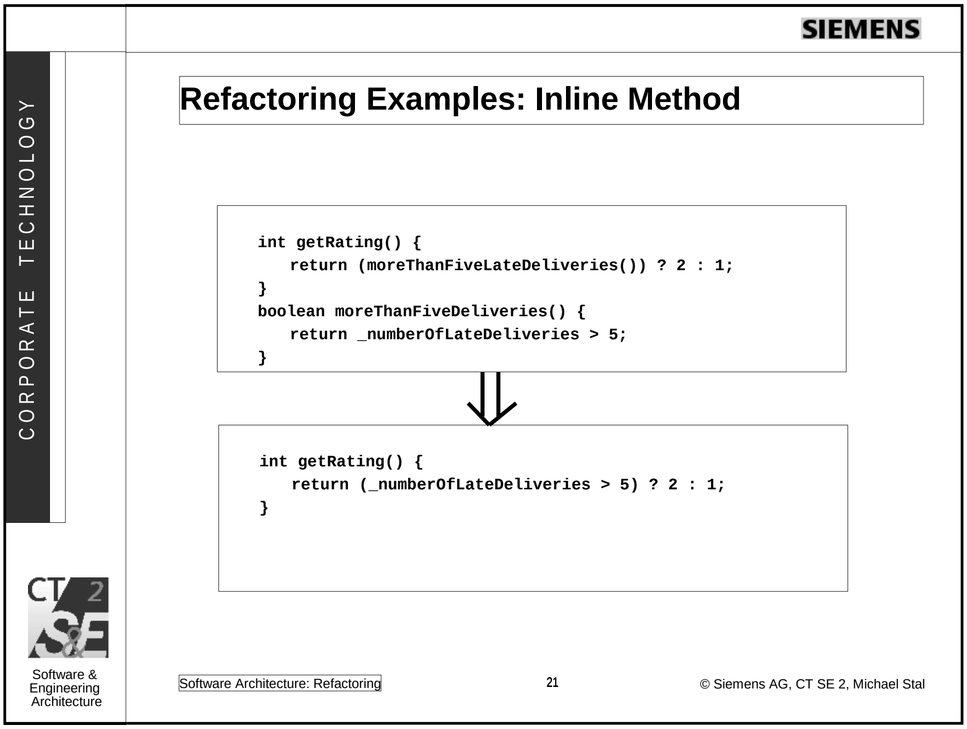
- If class and subclass don‘t differ too much



Refactoring Examples: Replace Inheritance with Delegation

- Create field for superclass, adjust methods to delegate, remove subclassing





SIEMENS

CORPORATE TECHNOLOGY

Refactoring Examples: Inline Method

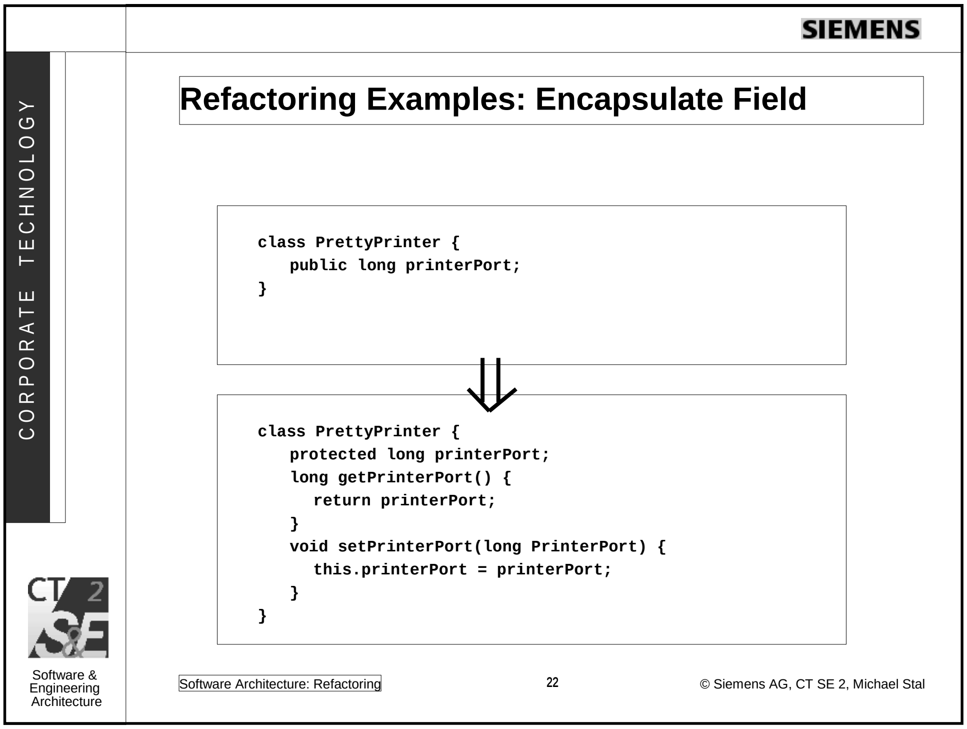
```
int getRating() {
   return (moreThanFiveLateDeliveries()) ? 2 : 1;
}
boolean moreThanFiveDeliveries() {
   return _numberOfLateDeliveries > 5;
}
```

⇓

```
int getRating() {
   return (_numberOfLateDeliveries > 5) ? 2 : 1;
}
```

Software & Engineering Architecture

Software Architecture: Refactoring 21 © Siemens AG, CT SE 2, Michael Stal



SIEMENS

CORPORATE TECHNOLOGY

Refactoring Examples: Encapsulate Field

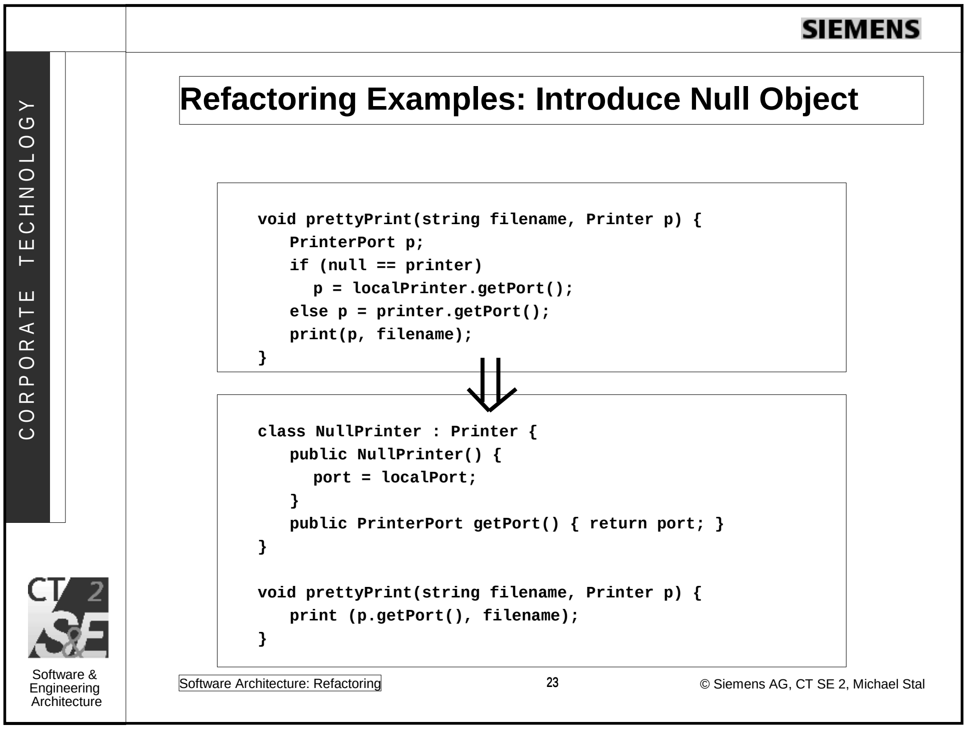
```
class PrettyPrinter {
   public long printerPort;
}
```

⇓

```
class PrettyPrinter {
   protected long printerPort;
   long getPrinterPort() {
     return printerPort;
   }
   void setPrinterPort(long PrinterPort) {
     this.printerPort = printerPort;
   }
}
```

Software & Engineering Architecture

Software Architecture: Refactoring 22 © Siemens AG, CT SE 2, Michael Stal



CORPORATE TECHNOLOGY

Refactoring Examples: Introduce Null Object

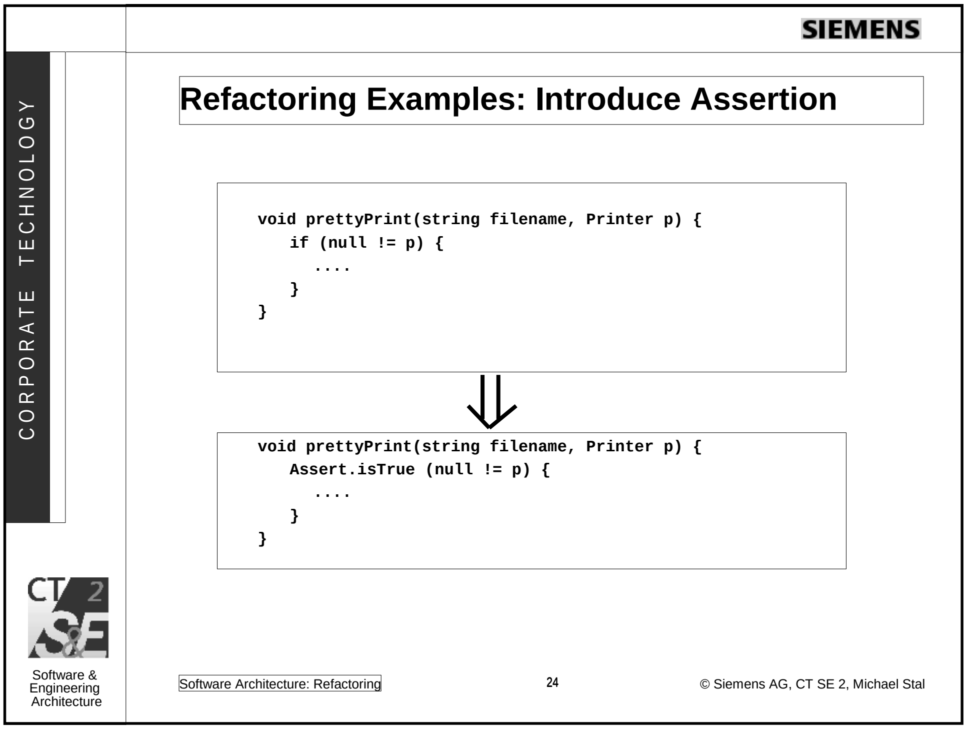
```
void prettyPrint(string filename, Printer p) {
   PrinterPort p;
   if (null == printer)
     p = localPrinter.getPort();
   else p = printer.getPort();
   print(p, filename);
}
```

⇓

```
class NullPrinter : Printer {
   public NullPrinter() {
     port = localPort;
   }
   public PrinterPort getPort() { return port; }
}

void prettyPrint(string filename, Printer p) {
   print (p.getPort(), filename);
}
```

Software & Engineering Architecture



CORPORATE TECHNOLOGY

Refactoring Examples: Introduce Assertion

```
void prettyPrint(string filename, Printer p) {
   if (null != p) {
     ....
   }
}
```

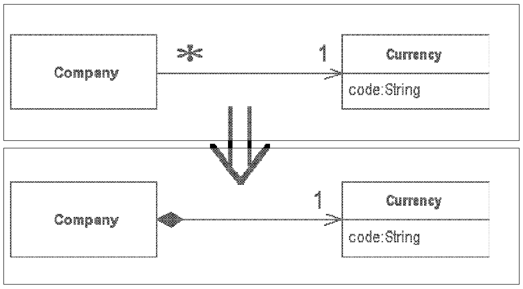
⇓

```
void prettyPrint(string filename, Printer p) {
   Assert.isTrue (null != p) {
     ....
   }
}
```

Software & Engineering Architecture

Refactoring Examples: Change Reference to Value

- If a type is immutable, small, difficult to manage



Refactoring Examples: Parametrize Method

```
class Servlet{
   public void handlePut() { }
   public void handleGet() { }
}
```

⇓

```
class Servlet{
   public void handle(ServiceType st) {
     ...
   }
}
```

Refactoring Examples: Replace Constructor with Factory Method

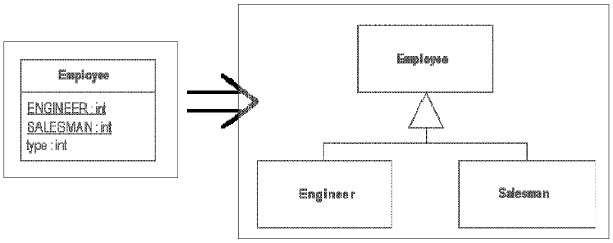
```
class PrettyPrinter {
   public PrettyPrinter(...) { }
}
```

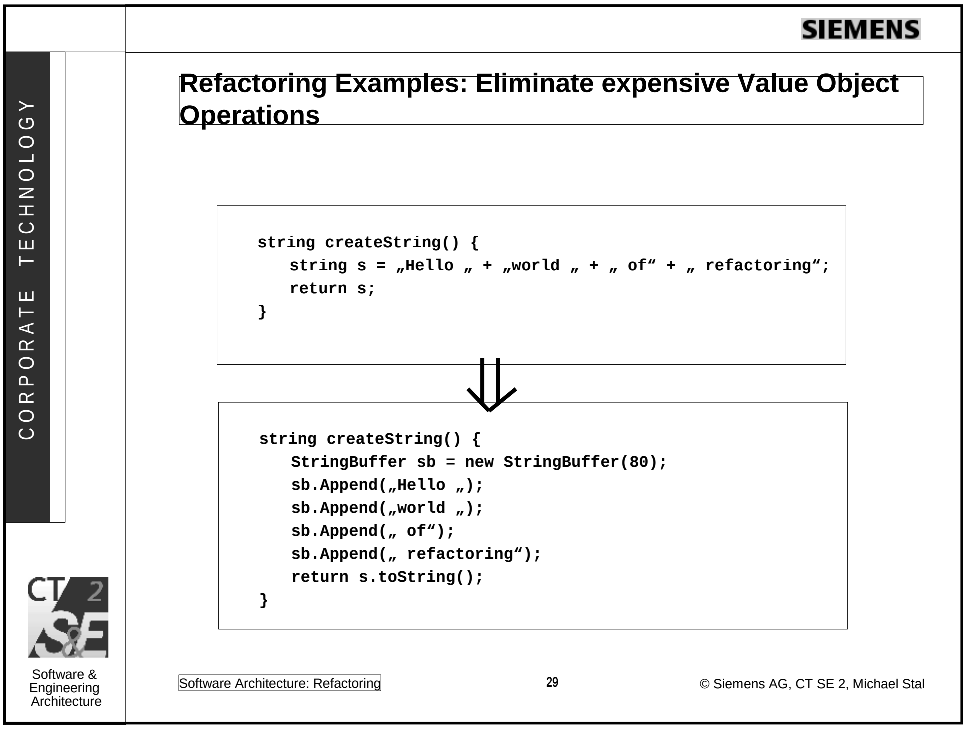
⇓

```
class PrettyPrinter {
   protected PrettyPrinter(...) { };
   PrettyPrinter create(...) {
     // do preprocessing
     return new PrettyPrinter(...);
     // do postprocessing
   }
}
```

Refactoring Examples: Replace Type Code with Subclasses

- Get rid of immutable type codes that affect class behavior





SIEMENS

CORPORATE TECHNOLOGY

Refactoring Examples: Eliminate expensive Value Object Operations

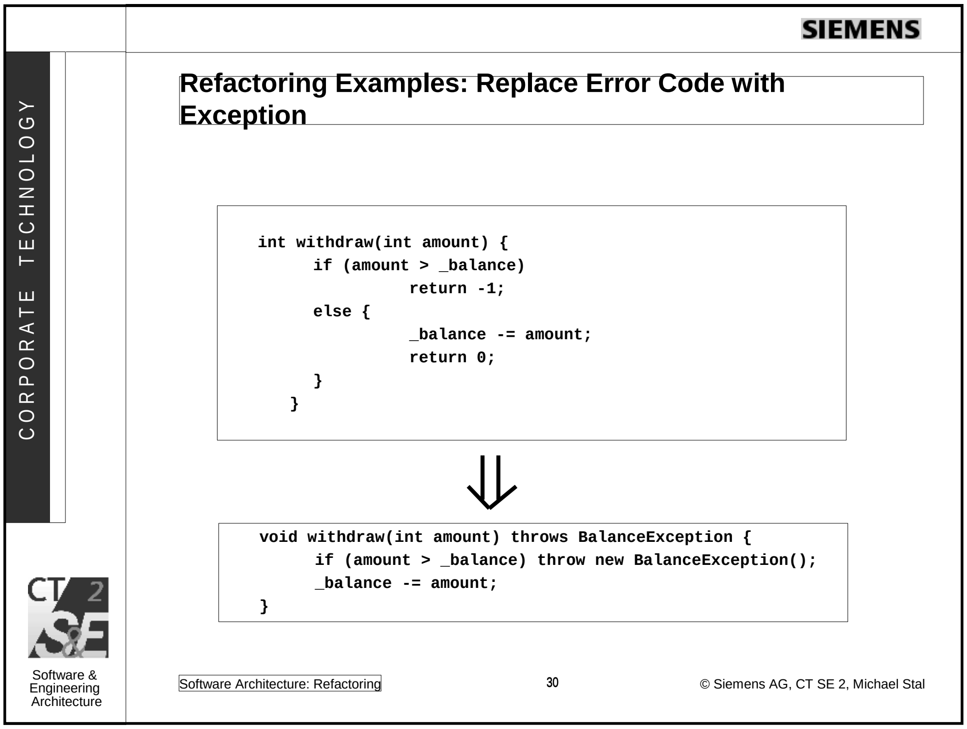
```
string createString() {
   string s = „Hello „ + „world „ + „ of“ + „ refactoring“;
   return s;
}
```

⇓

```
string createString() {
   StringBuffer sb = new StringBuffer(80);
   sb.Append(„Hello „);
   sb.Append(„world „);
   sb.Append(„ of“);
   sb.Append(„ refactoring“);
   return s.toString();
}
```

Software & Engineering Architecture

Software Architecture: Refactoring 29 © Siemens AG, CT SE 2, Michael Stal



SIEMENS

CORPORATE TECHNOLOGY

Refactoring Examples: Replace Error Code with Exception

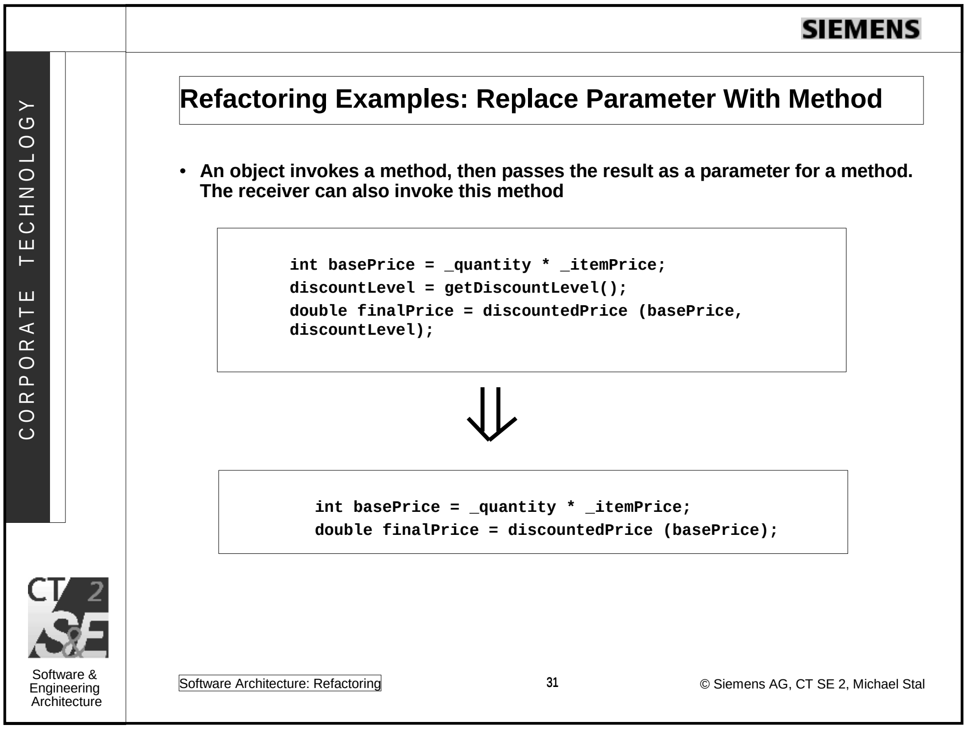
```
int withdraw(int amount) {
      if (amount > _balance)
                return -1;
      else {
                _balance -= amount;
                return 0;
      }
   }
```

⇓

```
void withdraw(int amount) throws BalanceException {
      if (amount > _balance) throw new BalanceException();
      _balance -= amount;
}
```

Software & Engineering Architecture

Software Architecture: Refactoring 30 © Siemens AG, CT SE 2, Michael Stal



CORPORATE TECHNOLOGY

Refactoring Examples: Replace Parameter With Method

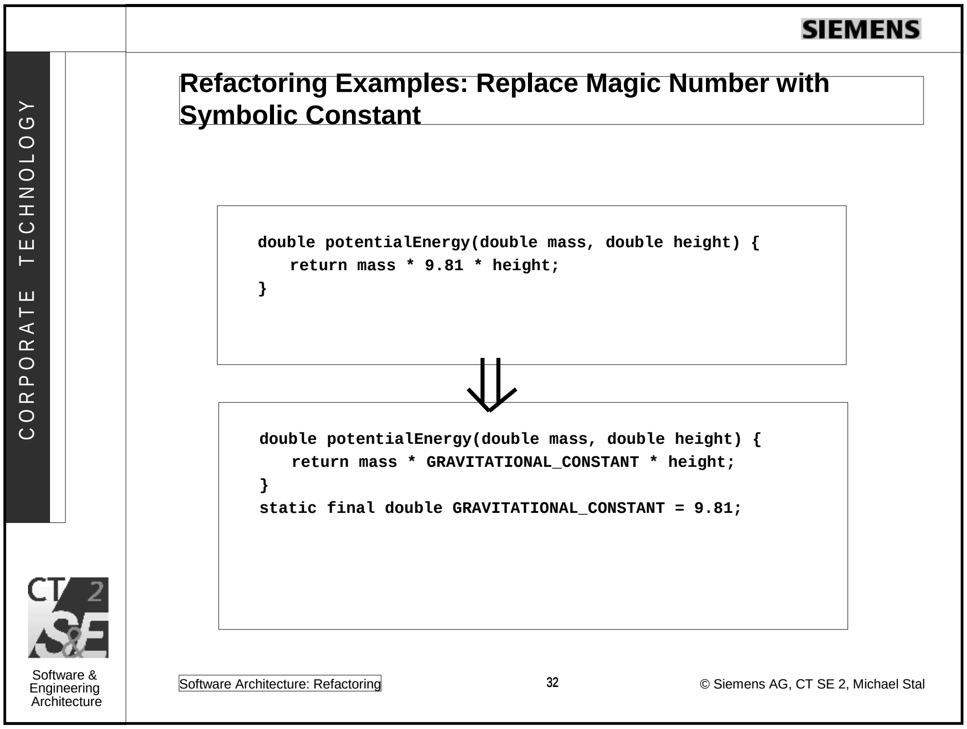
- An object invokes a method, then passes the result as a parameter for a method. The receiver can also invoke this method

```
int basePrice = _quantity * _itemPrice;
discountLevel = getDiscountLevel();
double finalPrice = discountedPrice (basePrice,
discountLevel);
```

⇓

```
int basePrice = _quantity * _itemPrice;
double finalPrice = discountedPrice (basePrice);
```

Software & Engineering Architecture



CORPORATE TECHNOLOGY

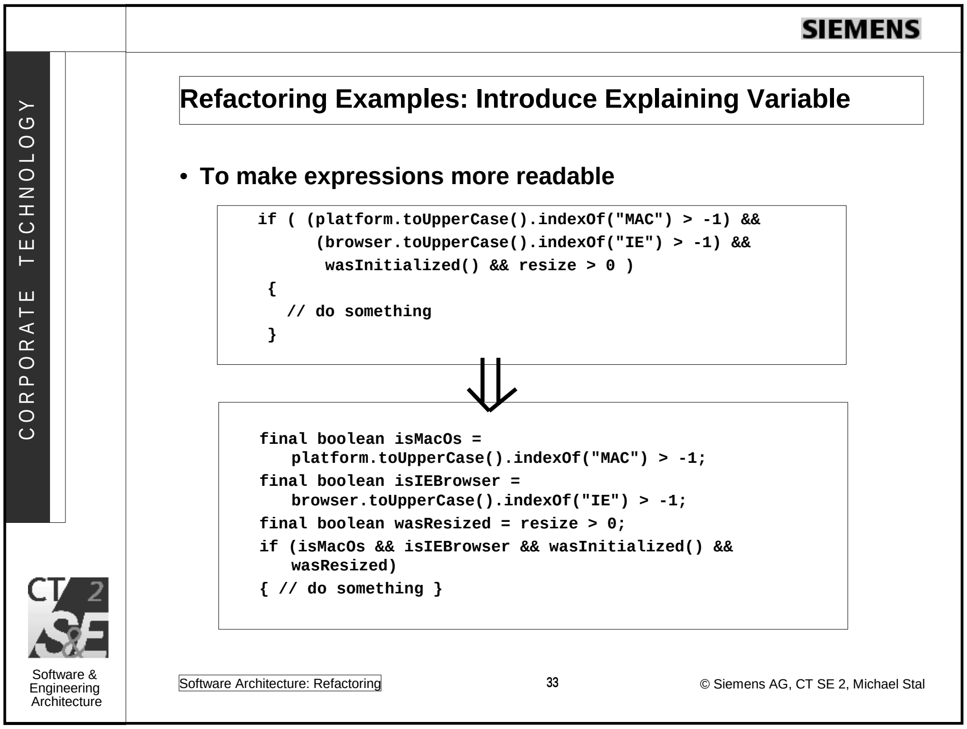
Refactoring Examples: Replace Magic Number with Symbolic Constant

```
double potentialEnergy(double mass, double height) {
   return mass * 9.81 * height;
}
```

⇓

```
double potentialEnergy(double mass, double height) {
   return mass * GRAVITATIONAL_CONSTANT * height;
}
static final double GRAVITATIONAL_CONSTANT = 9.81;
```

Software & Engineering Architecture



SIEMENS

CORPORATE TECHNOLOGY

Refactoring Examples: Introduce Explaining Variable

- To make expressions more readable

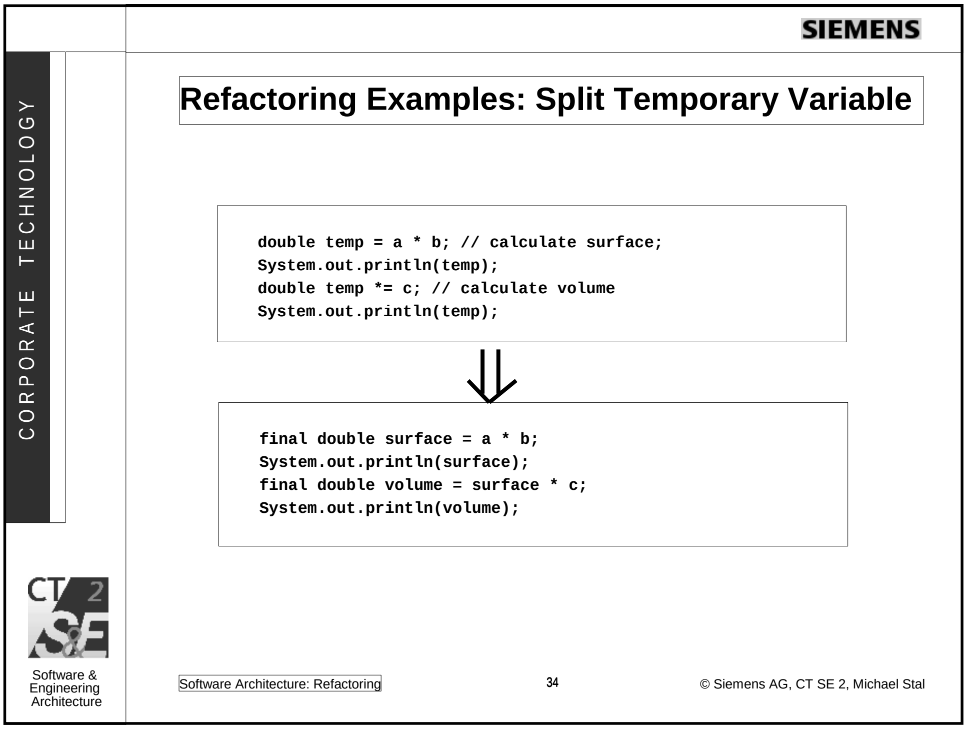
```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
      (browser.toUpperCase().indexOf("IE") > -1) &&
       wasInitialized() && resize > 0 )
 {
   // do something
 }
```

⇓

```
final boolean isMacOs =
   platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser =
   browser.toUpperCase().indexOf("IE") > -1;
final boolean wasResized = resize > 0;
if (isMacOs && isIEBrowser && wasInitialized() &&
   wasResized)
{ // do something }
```

Software & Engineering Architecture

Software Architecture: Refactoring 33 © Siemens AG, CT SE 2, Michael Stal



SIEMENS

CORPORATE TECHNOLOGY

Refactoring Examples: Split Temporary Variable

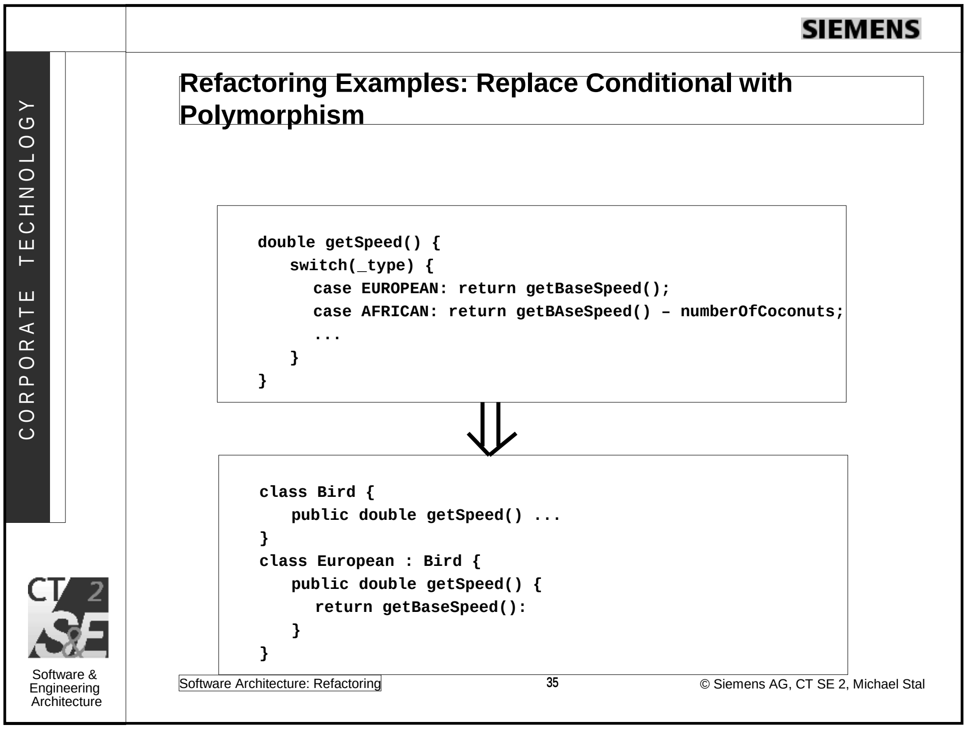
```
double temp = a * b; // calculate surface;
System.out.println(temp);
double temp *= c; // calculate volume
System.out.println(temp);
```

⇓

```
final double surface = a * b;
System.out.println(surface);
final double volume = surface * c;
System.out.println(volume);
```

Software & Engineering Architecture

Software Architecture: Refactoring 34 © Siemens AG, CT SE 2, Michael Stal



SIEMENS

CORPORATE TECHNOLOGY

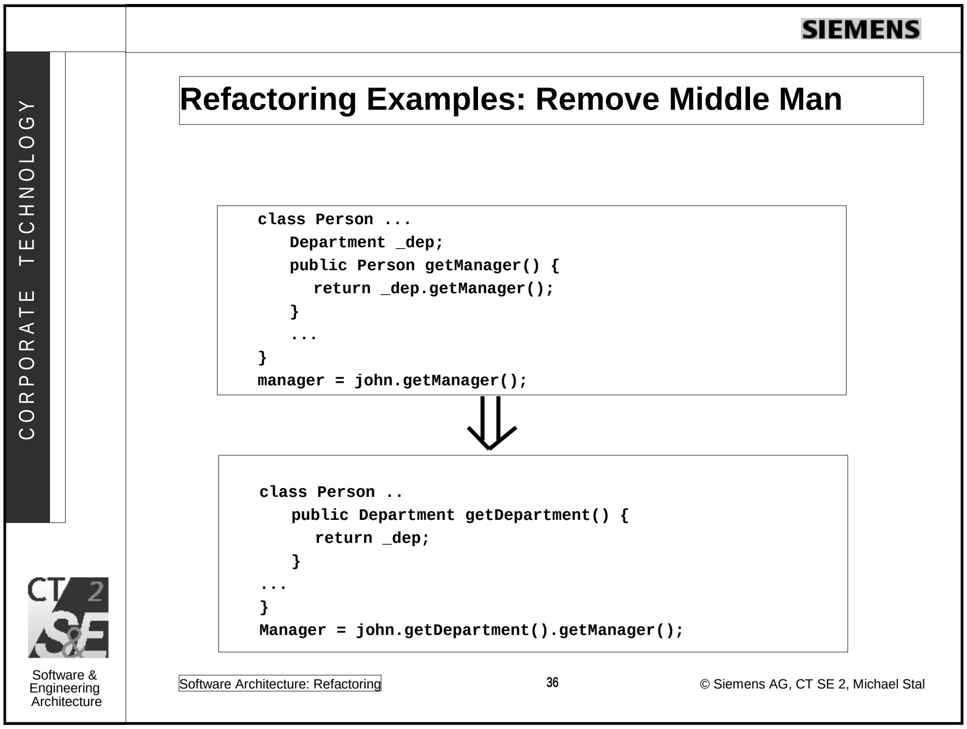
Refactoring Examples: Replace Conditional with Polymorphism

```
double getSpeed() {
   switch(_type) {
     case EUROPEAN: return getBaseSpeed();
     case AFRICAN: return getBAseSpeed() – numberOfCoconuts;
     ...
   }
}
```

⇓

```
class Bird {
   public double getSpeed() ...
}
class European : Bird {
   public double getSpeed() {
     return getBaseSpeed():
   }
}
```

Software & Engineering Architecture



SIEMENS

CORPORATE TECHNOLOGY

Refactoring Examples: Remove Middle Man

```
class Person ...
   Department _dep;
   public Person getManager() {
     return _dep.getManager();
   }
   ...
}
manager = john.getManager();
```

⇓

```
class Person ..
   public Department getDepartment() {
     return _dep;
   }
...
}
Manager = john.getDepartment().getManager();
```

Software & Engineering Architecture

Refactoring Examples: Move field

- **Field is more used in other class**

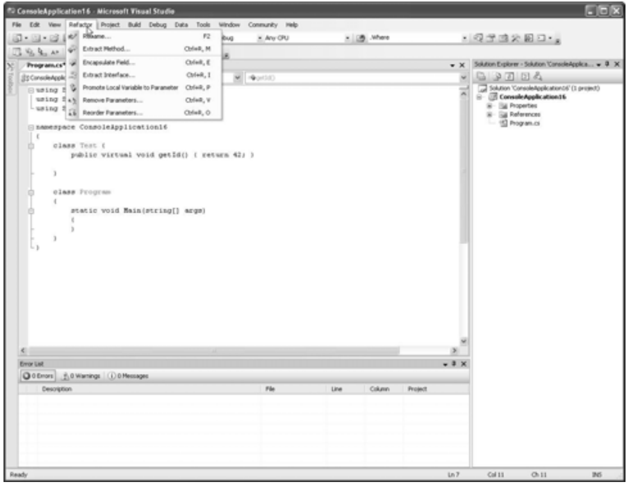
```
class Account ...
   private AccountType _type;
   private double InterestRate;
   double Calculate(double amount, int days) {
     return _interestRate * amount * days / 365;
   }
]
```

⇓

```
class AccountType ...
   private double _interestRate;
   void setInterestRate(double arg) {_interestRate = arg; }
   void getInterestRate() { return _interestRate; }
}
// in class Account:
   double Calculate(double amount, int days) {
     return _type.getInterestRate() * amount * days / 365;
   }
```

Refactoring and Tools

- **Most IDEs tightly integrate refactoring, e.g. VS.NET 2005:**


ConsoleApplication16 - Microsoft Visual Studio
File
Edit
View
Refactor
Project
Build
Debug
Data
Tools
Window
Community
Help
Rename... F2
Extract Method... Ctrl+R, M
Encapsulate Field... Ctrl+R, E
Extract Interface... Ctrl+R, I
Promote Local Variable to Parameter Ctrl+R, P
Remove Parameters... Ctrl+R, V
Reorder Parameters... Ctrl+R, O
Any CPU
Where
Program.cs*
Solution Explorer - Solution 'ConsoleApplica...
Solution 'ConsoleApplication16' (1 project)
ConsoleApplication16
Properties
References
Program.cs
Error List
0 Errors
0 Warnings
0 Messages
Description
File
Line
Column
Project
Ready
Ln 7
Col 11
Ch 11
INS

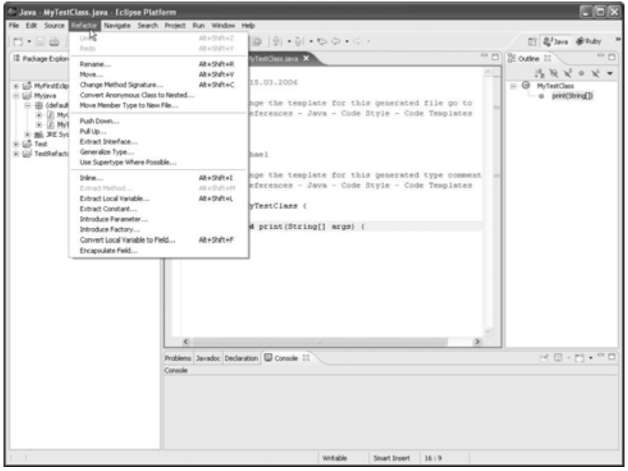
```
using S
using S
using S

namespace ConsoleApplication16
{
    class Test {
        public virtual void getId() { return 42; }
    }

    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Refactoring and Tools (cont‘d)

- **Eclipse 3.1 / Java IDE:**


Java - MyTestClass.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Undo Alt+Shift+Z
Redo Alt+Shift+Y
Rename... Alt+Shift+R
Move... Alt+Shift+V
Change Method Signature... Alt+Shift+C
Convert Anonymous Class to Nested...
Move Member Type to New File...
Push Down...
Pull Up...
Extract Interface...
Generalize Type...
Use Supertype Where Possible...
Inline... Alt+Shift+I
Extract Method... Alt+Shift+M
Extract Local Variable... Alt+Shift+L
Extract Constant...
Introduce Parameter...
Introduce Factory...
Convert Local Variable to Field... Alt+Shift+F
Encapsulate Field...
Package Explorer
MyFirstEclip
MyJava
(default
JRE Sys
Test
TestRefact
MyTestClass.java
Outline
MyTestClass
print(String[])
Java
Ruby
Problems Javadoc Declaration Console
Console
Writable Smart Insert 16 : 9

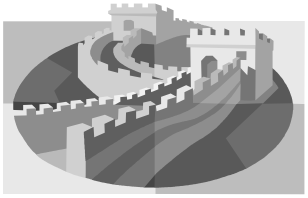
```
15.03.2006
nge the template for this generated file go to
eferences - Java - Code Style - Code Templates
hael
nge the template for this generated type comment
eferences - Java - Code Style - Code Templates
yTestClass {
d print(String[] args) {
```

Refactoring and Tools (cont‘d)

- **C++:**
  - Slickedit provides several refactorings for C/C++
  - Ref++: commercial add-in for Visual Studio
  - Xrefactory for C++: emacs plug-in
- **Additional Tool Support for:**
  - VB
  - Python
  - Haskell
  - Smalltalk
  - Self
  - Delphi

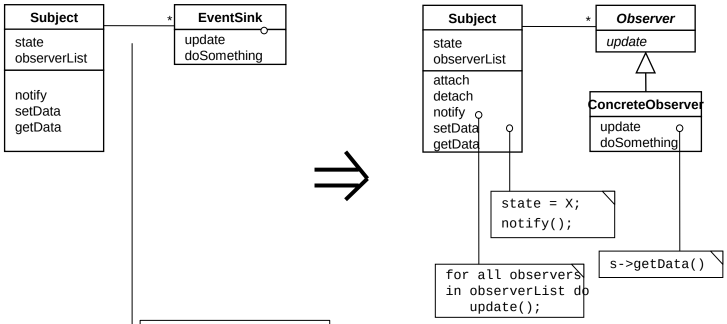
Pattern-based Refactoring

- **Two perspectives:**
  - Refactorings might be documented in kind of pattern form
  - Patterns might help to refactor on architectural level:
    - Replace your proprietary solution with a pattern that solves the same problem
    - Introduce symmetry and orthogonality by making sure the same problem is always solved using the same pattern/solution



Example: Applying Observer

- **Do it yourself:**
- **Observer Pattern**

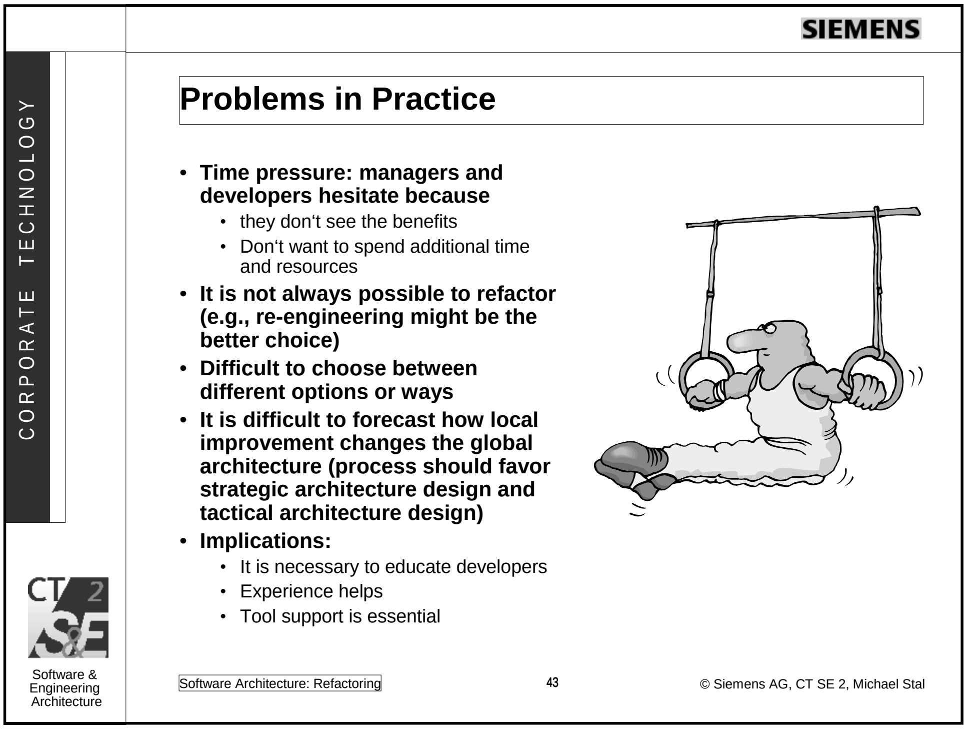


```
state = X;
notify();
```

```
for all observers
in observerList do
   update();
```

```
s->getData()
```

Subject contains hardwired list of interested components



CORPORATE TECHNOLOGY

Problems in Practice

- **Time pressure: managers and developers hesitate because**
  - they don‘t see the benefits
  - Don‘t want to spend additional time and resources
- **It is not always possible to refactor (e.g., re-engineering might be the better choice)**
- **Difficult to choose between different options or ways**
- **It is difficult to forecast how local improvement changes the global architecture (process should favor strategic architecture design and tactical architecture design)**
- **Implications:**
  - It is necessary to educate developers
  - Experience helps
  - Tool support is essential

Software & Engineering Architecture

CORPORATE TECHNOLOGY

Q&A

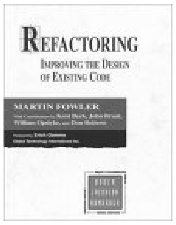
Software & Engineering Architecture

Conclusions

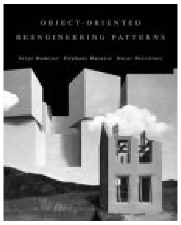
- **Refactoring improves the code/design without changing behavior. It more applies to the code**
- **Reengineering is a complete redesign of an architecture and might also change behavior. It applies also to the architecture**
- **Both methods are essential, but use the right one for the right purpose**
- **If refactoring is applied, make sure your environment is appropriate**
  - Your process should allow to design a little, code a little, test
  - Unit testing is extremely important
  - Refactoring without appropriate tools is tedious and error-prone
- **Visit http://www.refactoring.com/catalog/ for refactoring list**

References

References

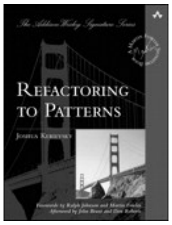


M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts: Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999



S. Demeyer, S. Ducasse, O. Nierstrasz: Object-Oriented Reengineering Patterns, Morgan Kaufmann, 2002

References (cont’d)



Kerievsky, Joshua: Refactoring to Patterns, Addison-Wesley, 2004

Web References

- **William C. Wake: „Refactoring Workbook“**
  http://xp123.com/rwb/RWB-draft3.PDF

- **Martin Fowler, „Refactoring Home Page“**
  http://www.refactoring.com

- **Joshua Kerievsky: „Refactoring to Patterns Home Page“**
  http://industriallogic.com/rtpdata/index.html