DAT159
Refactoring (Introduction)

Volker Stolz[1], with contributions by:
Larissa Braz[2], Anna M. Eilertsen[3],
Fernando Macías[1], Rohit Gheyi[2]

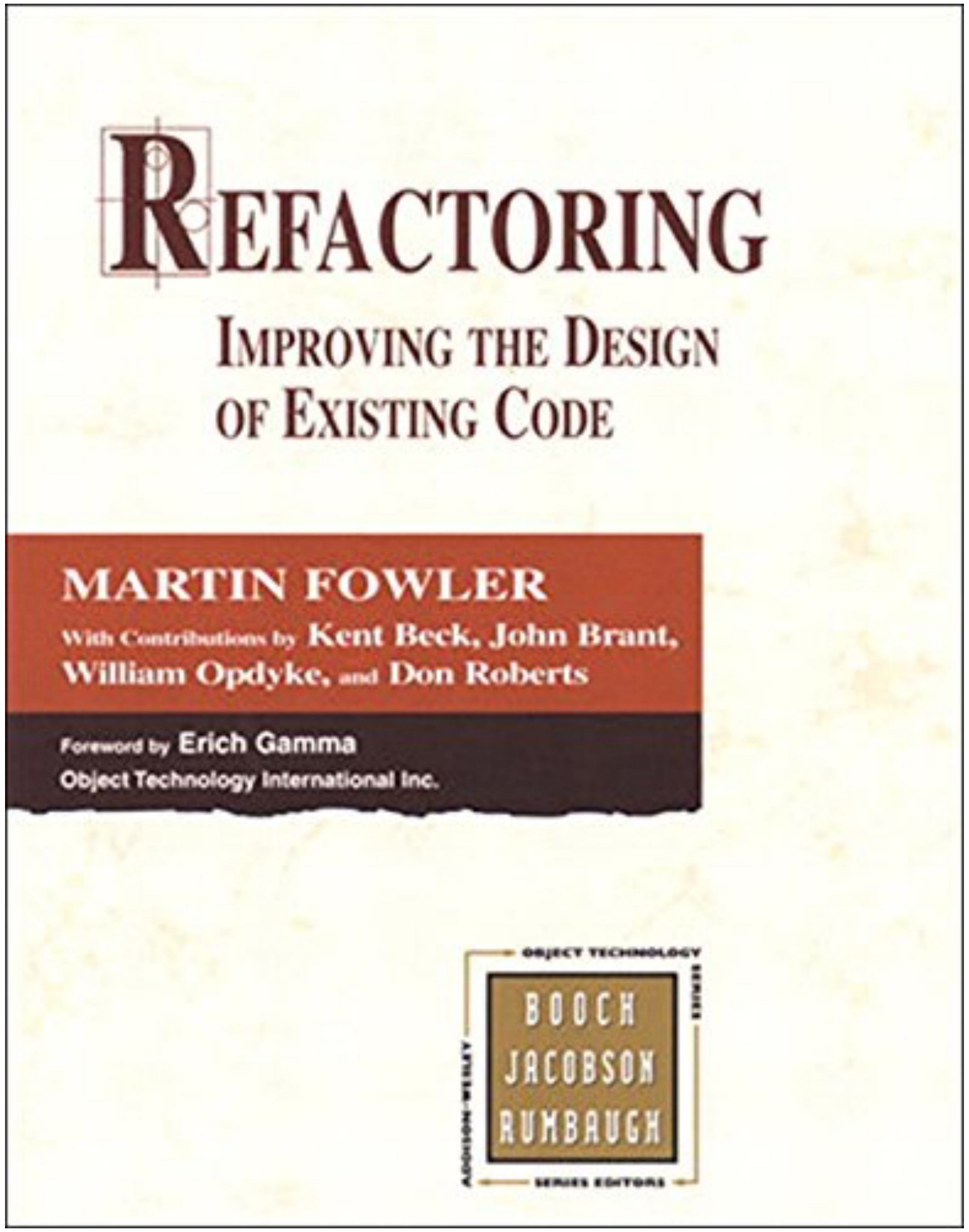
Western Norway University of Applied Sciences,
Universidade Federal de Campina Grande,
University of Bergen, Norway

Supported by the bilateral SIU/CAPES project “Modern Refactoring” 2017/18

Høgskulen
på Vestlandet

Overview: Refactoring

- What are refactorings?
- Common refactorings for different languages.
- Why refactor? What are source code metrics?
- What can go wrong?
- How to implement refactorings?


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

Overview

- 6+1 lectures
- 3 labs
- 1 oblig

Please bring your laptop!
(at least 1/group)

IDEs: Eclipse, IntelliJ

Languages: mostly Java, some C

Overview

- Guest lectures from Brazil!
  (SIU/CAPES project “Modern Refactoring”)
  (see changed schedule)
- Possible Bachelor projects…
- …and Master theses.

DAT159
Refactoring (Introduction)

Volker Stolz[1], with contributions by:
Larissa Braz[2], Anna M. Eilertsen[3],
Fernando Macías[1], Rohit Gheyi[2]

Western Norway University of Applied Sciences,
Universidade Federal de Campina Grande,
University of Bergen, Norway

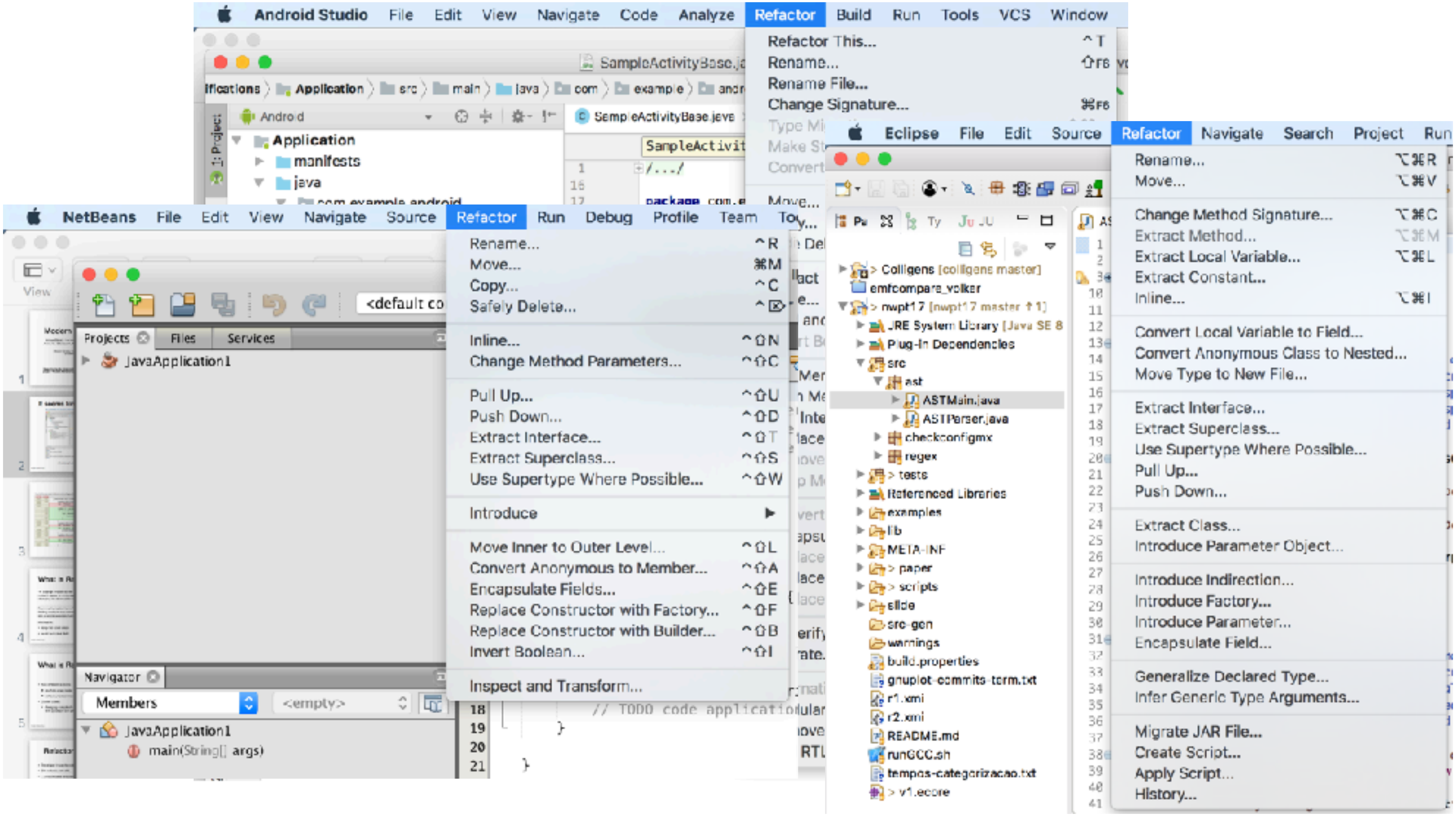
Supported by the bilateral SIU/CAPES project “Modern Refactoring” 2017/18

Høgskulen
på Vestlandet

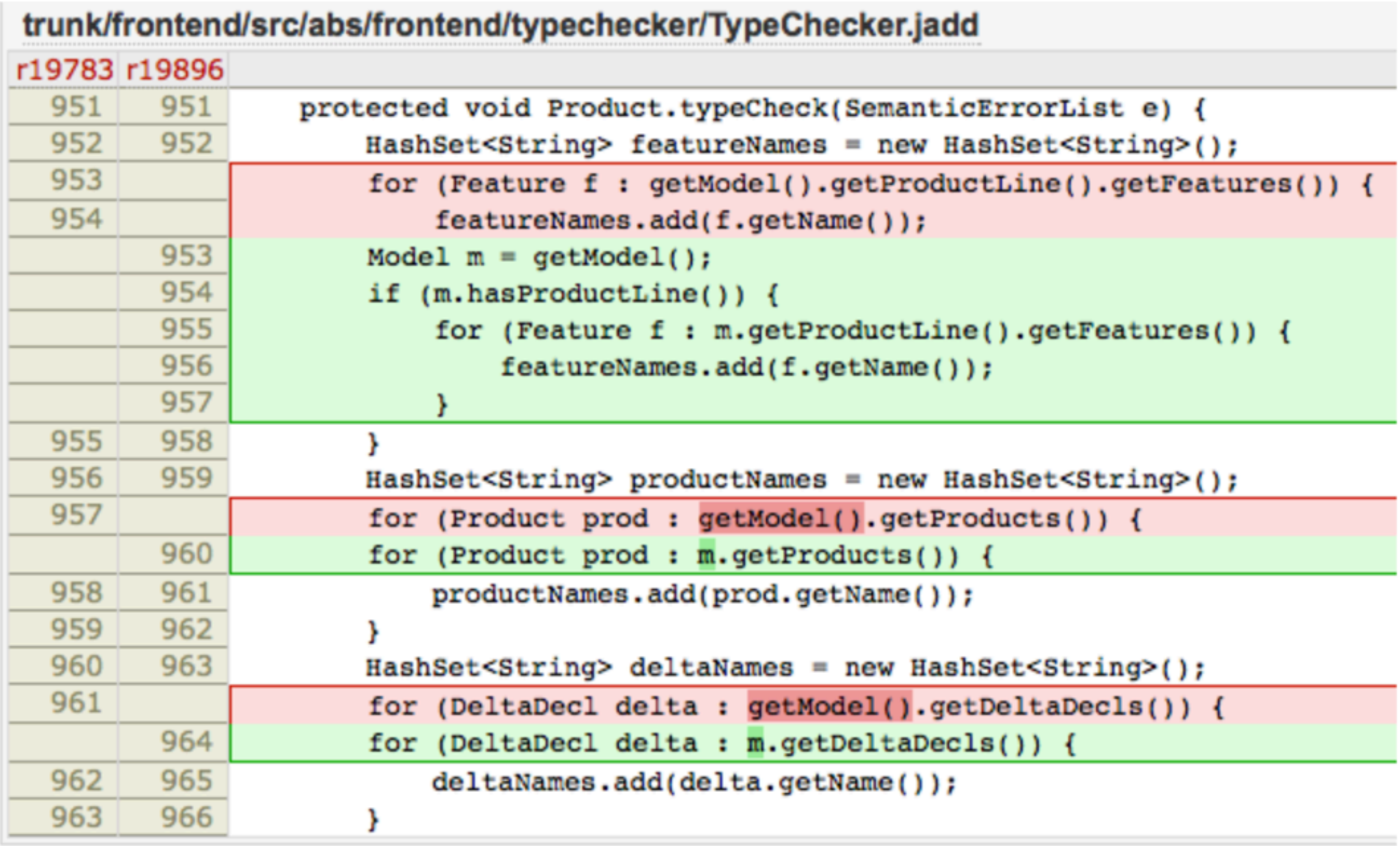
Overview

- What are refactorings? What are they good for?
- Examples in common IDEs
- Examples in common languages (Java, C/C++, …)
- Impact on software quality metrics
- Implementation of refactorings
- Formal treatment of refactorings

It seems kinda important…


Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window
Refactor This... ^T
Rename... ⇧F6
Rename File...
Change Signature... ⌘F6
Move...
NetBeans
File Edit View Navigate Source Refactor Run Debug Profile Team Tools
Rename... ^R
Move... ⌘M
Copy... ^C
Safely Delete... ^⌦
Inline... ^⇧N
Change Method Parameters... ^⇧C
Pull Up... ^⇧U
Push Down... ^⇧D
Extract Interface... ^⇧T
Extract Superclass... ^⇧S
Use Supertype Where Possible... ^⇧W
Introduce ▶
Move Inner to Outer Level... ^⇧L
Convert Anonymous to Member... ^⇧A
Encapsulate Fields... ^⇧E
Replace Constructor with Factory... ^⇧F
Replace Constructor with Builder... ^⇧B
Invert Boolean... ^⇧I
Inspect and Transform...
Eclipse
File Edit Source Refactor Navigate Search Project Run
Rename... ⌥⌘R
Move... ⌥⌘V
Change Method Signature... ⌥⌘C
Extract Method... ⌥⌘M
Extract Local Variable... ⌥⌘L
Extract Constant...
Inline... ⌥⌘I
Convert Local Variable to Field...
Convert Anonymous Class to Nested...
Move Type to New File...
Extract Interface...
Extract Superclass...
Use Supertype Where Possible...
Pull Up...
Push Down...
Extract Class...
Introduce Parameter Object...
Introduce Indirection...
Introduce Factory...
Introduce Parameter...
Encapsulate Field...
Generalize Declared Type...
Infer Generic Type Arguments...
Migrate JAR File...
Create Script...
Apply Script...
History...

(Everybody’s doing it; you should as well!)

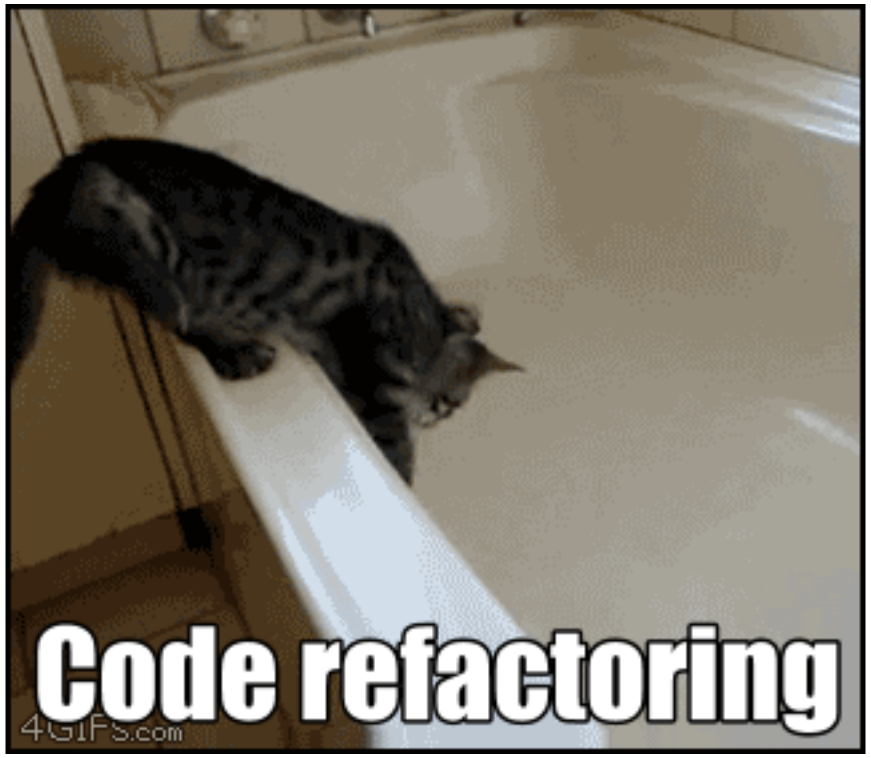


trunk/frontend/src/abs/frontend/typechecker/TypeChecker.jadd

```
r19783 r19896
  951    951      protected void Product.typeCheck(SemanticErrorList e) {
  952    952          HashSet<String> featureNames = new HashSet<String>();
  953                 for (Feature f : getModel().getProductLine().getFeatures()) {
  954                     featureNames.add(f.getName());
         953          Model m = getModel();
         954          if (m.hasProductLine()) {
         955              for (Feature f : m.getProductLine().getFeatures()) {
         956                  featureNames.add(f.getName());
         957              }
  955    958          }
  956    959          HashSet<String> productNames = new HashSet<String>();
  957                 for (Product prod : getModel().getProducts()) {
         960          for (Product prod : m.getProducts()) {
  958    961              productNames.add(prod.getName());
  959    962          }
  960    963          HashSet<String> deltaNames = new HashSet<String>();
  961                 for (DeltaDecl delta : getModel().getDeltaDecls()) {
         964          for (DeltaDecl delta : m.getDeltaDecls()) {
  962    965              deltaNames.add(delta.getName());
  963    966          }
```

Refactoring: ~~how to do it?~~ Why does everyone hate it?

YEAH... IF YOU COULD STOP REFACTORING WHILE WE TEST THINGS..
THAT'D BE GREAT



What is Refactoring? (1)

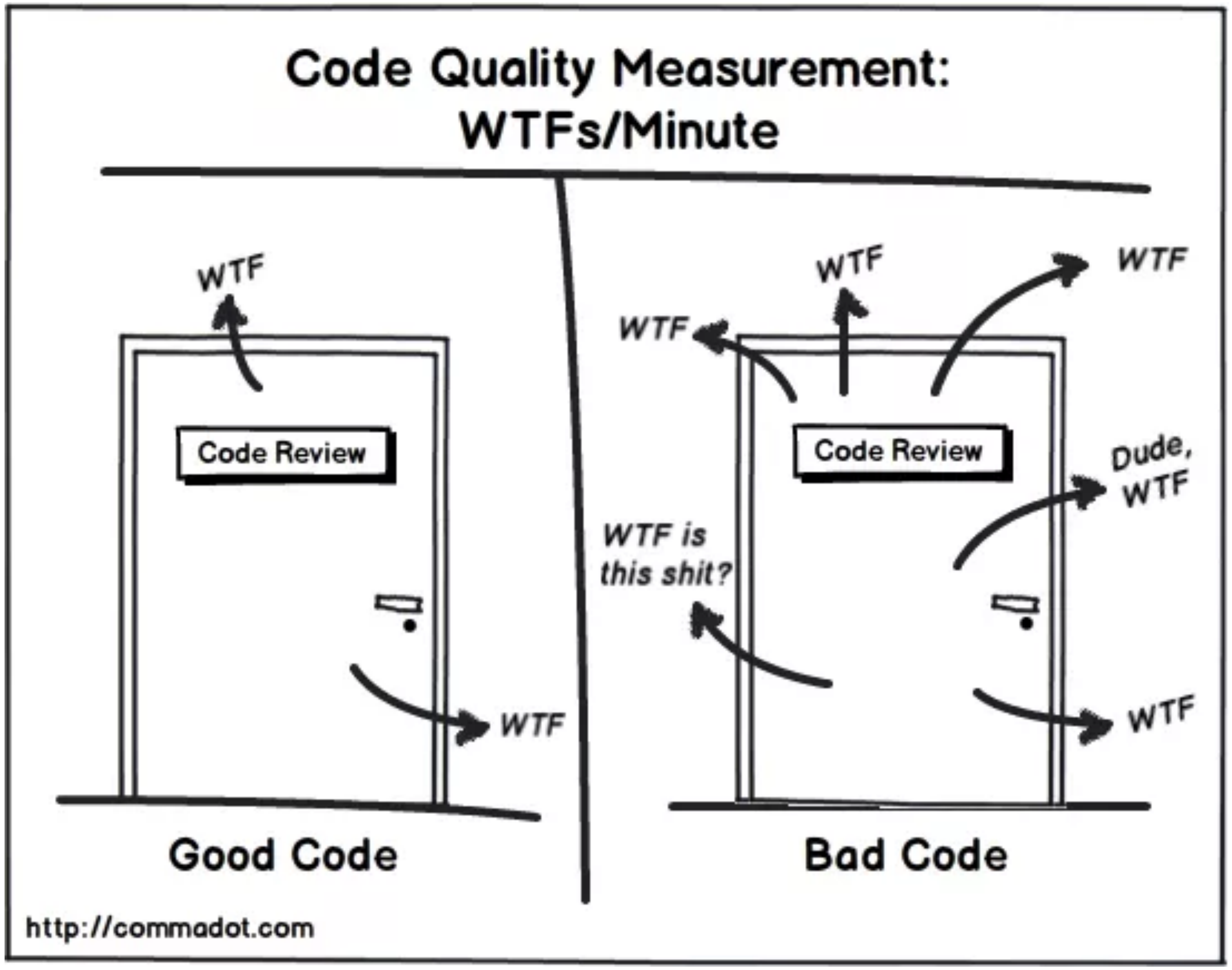
“A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour” [Fowler]

From mathematical term “factor”:
finding multiple occurrences of similar code and *factoring* it into a single reusable function

Motivation:

- keep the code clean
- avoid technical debt

Motivation

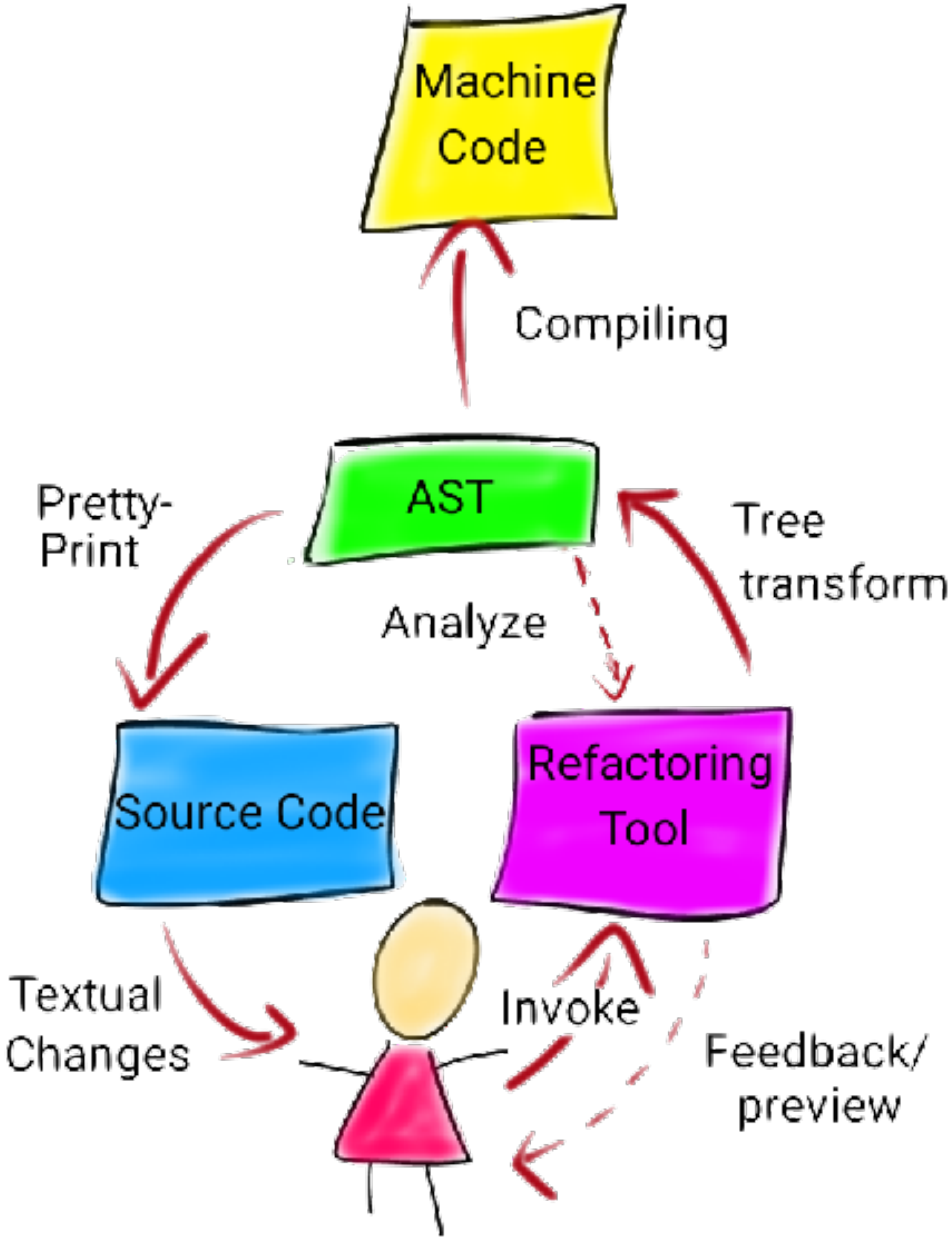

Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
Code Review
Dude, WTF
WTF is this shit?
WTF
Bad Code
http://commadot.com

What is Refactoring? (2)

- Two different schools:
  - *anything goes* (agile)
  - *behaviour preserving*
- Corner cases:
  - changing complexity class, e.g. replacing bubble sort with quicksort still a refactoring?

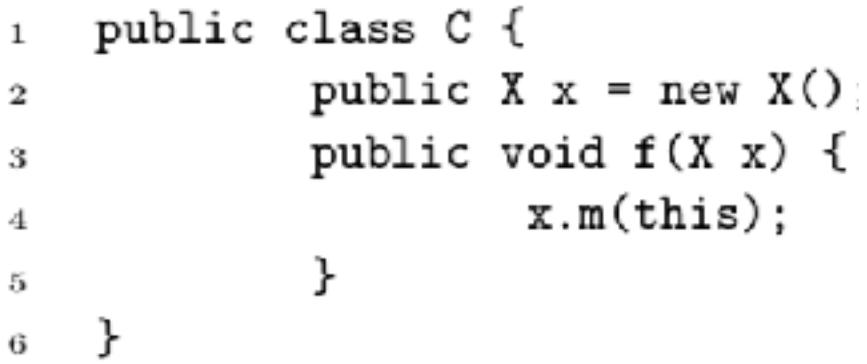
Refactoring Process

- Developer inspects code.
- She selects part of it…
- …and chooses refactoring action from menu.
- Refactorings usually modify the Abstract Syntax Tree (AST) in memory…
- … and then synchronize the source code file.

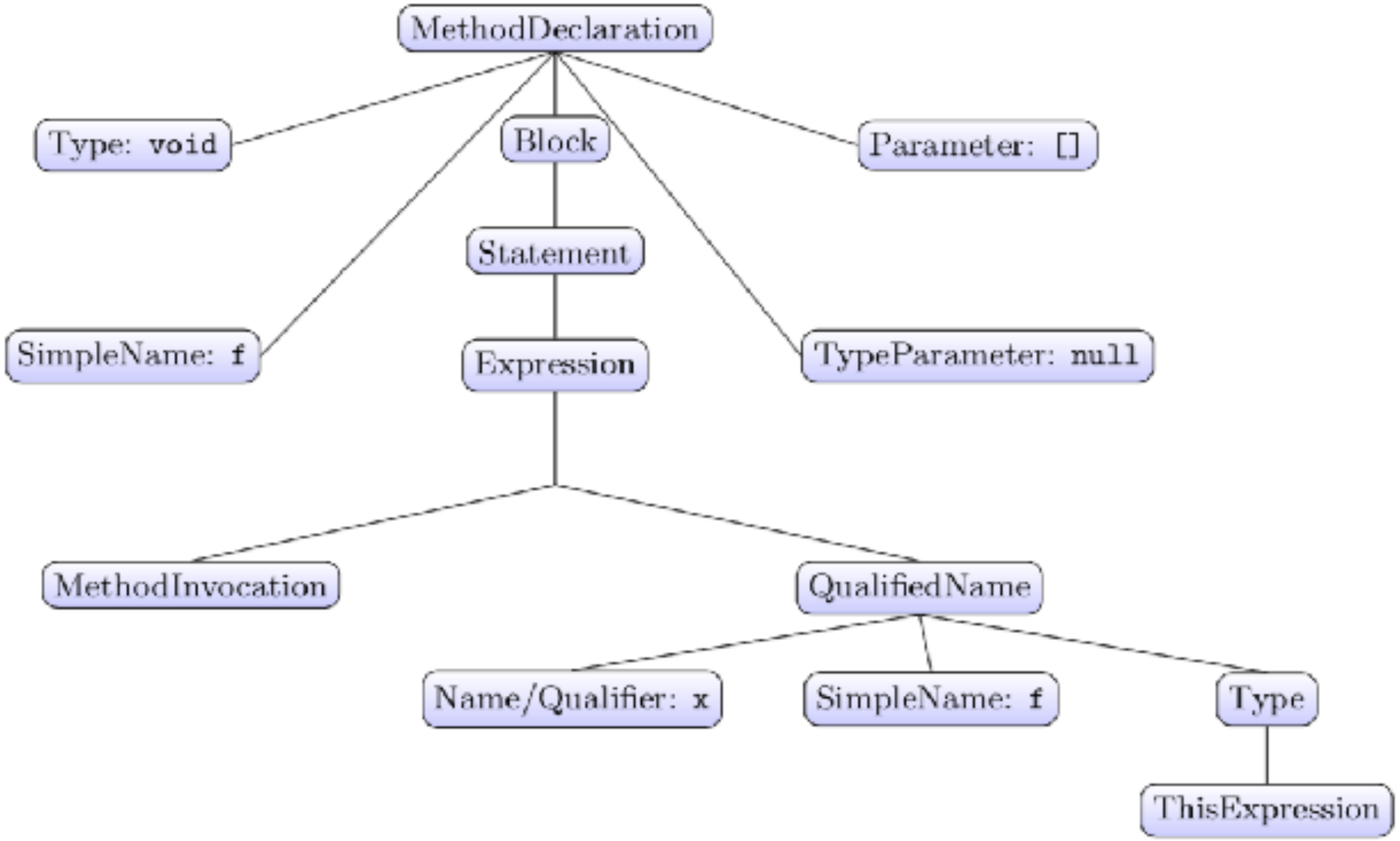


Abstract Syntax Tree (AST)

- In-memory representation of parsed source code
- Semantic information available (Where was this variable declared? What are the superclasses?)



```
public class C {
        public X x = new X()
        public void f(X x) {
                x.m(this);
        }
}
```

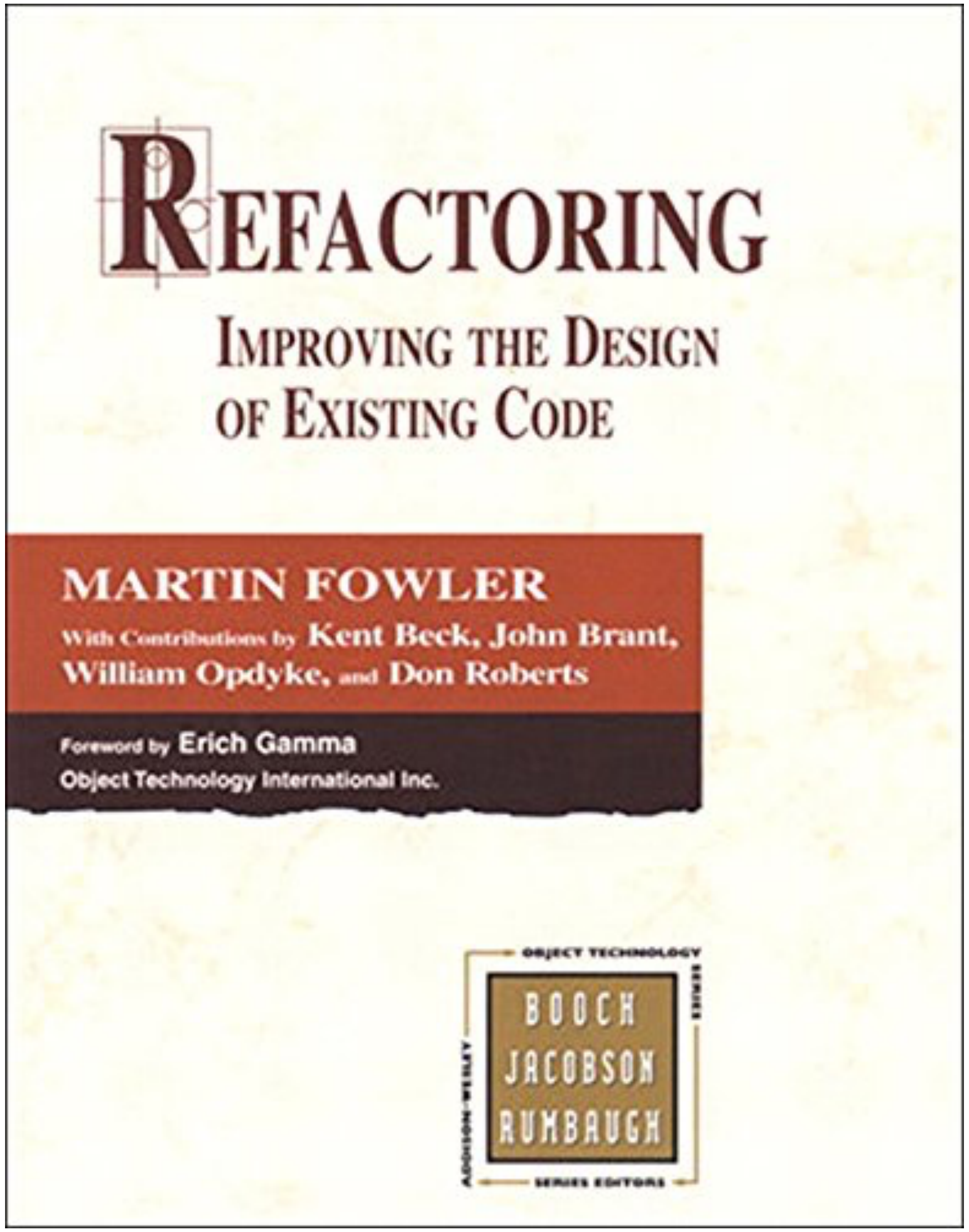


Refactoring: Origins

- Opdyke’s PhD thesis [1992]
- Smalltalk Refactoring Browser
  [Roberts, Brant, Johnson ’97]
- “Refactoring: improving the design of existing code”
  [Fowler ’99]

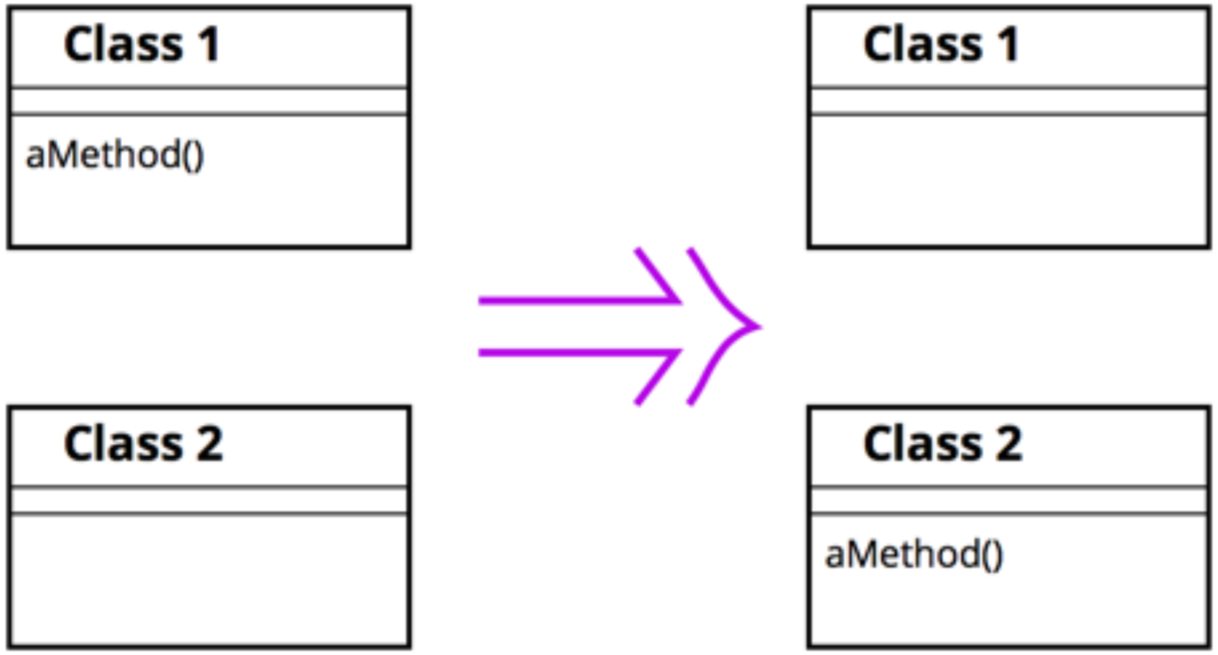
- 30% of changes are refactorings [Soares et al., 2011]
- Extract Method most popular — but performed manually
  [Murphy et al., 2006]

Literature


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
OBJECT TECHNOLOGY SERIES
BOOCH JACOBSON RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

Refactoring: Improving the Design of Existing Code
Martin Fowler with Kent Beck, John Brant, William Opdyke, Don Roberts

Addison Wesley, 1999


Class 1

aMethod()

Class 2

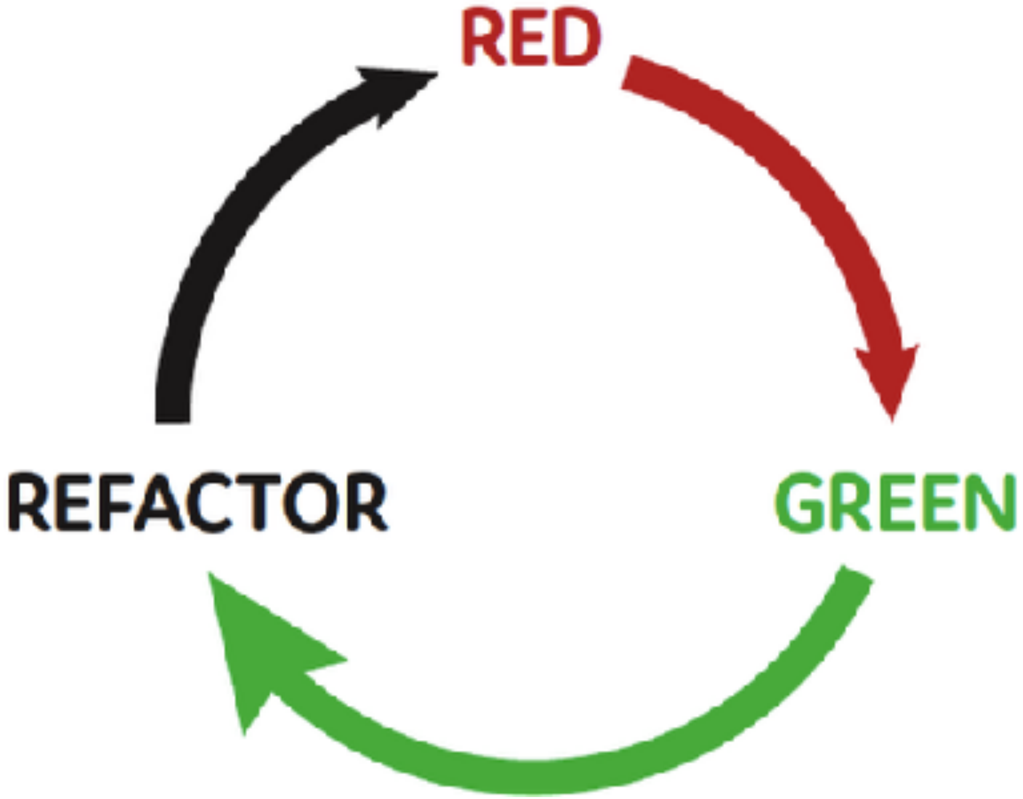
Class 1

Class 2

aMethod()

Adoption of Refactorings

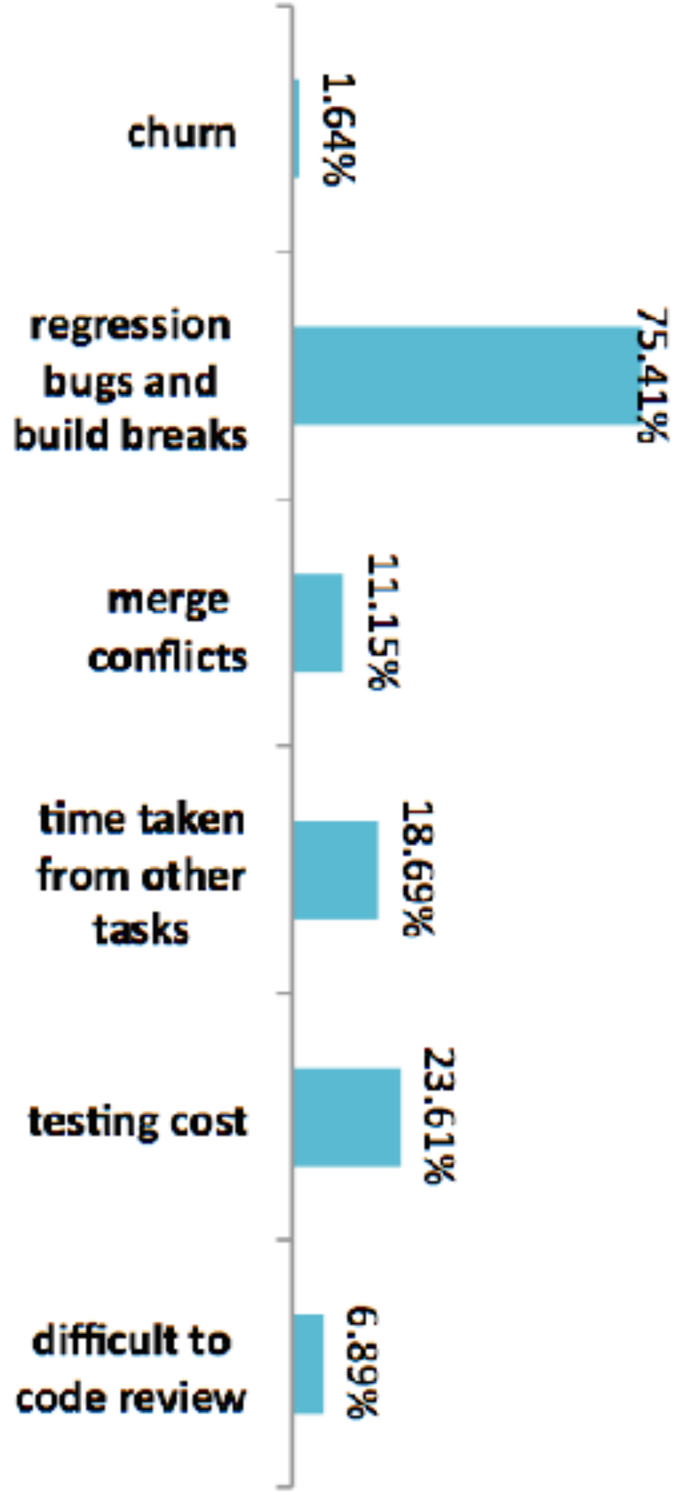
- Agile: fully embraced refactorings
- Developers usually sceptical of automated changes
- Study: developers more confident when they can predict changes
- Problem in OO languages: refactoring touches on multiple contexts



The agile workflow

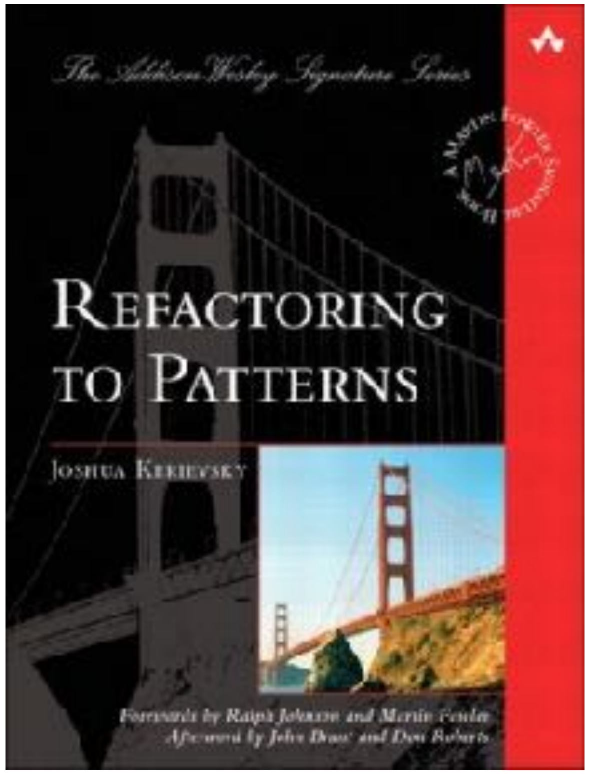
Adoption: Software Engineering Studies

- Kim et al. (FSE, 2012): survey on more than 300 engineers who had used refactoring during Microsoft Windows development
- Tempero et al. (C.ACM, 2017):
    - Survey on 3785 developers in 2009
    - They understand benefits of refactoring, but they see costs and risks as well.



Related Topics: Patterns

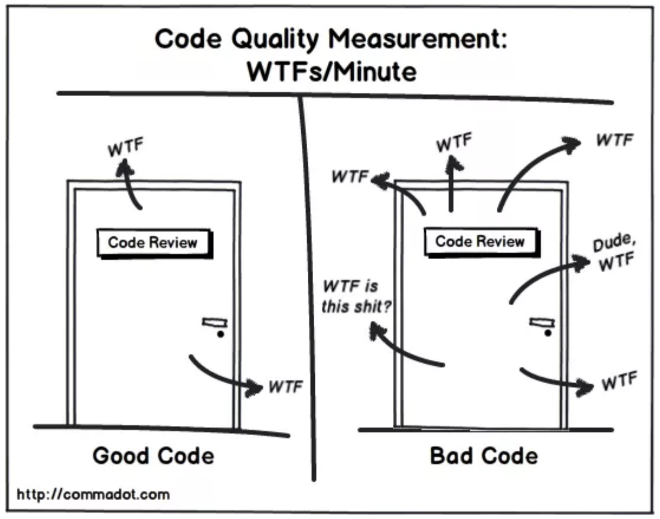
- “Design Patterns: Elements of Reusable Object-Oriented Software” [Gamma, Helm, Johnson, Vlissides, 1994]
- “Refactoring to patterns” [Kerievsky, 2005]


The Addison-Wesley Signature Series
Refactoring to Patterns
Joshua Kerievsky
Forewords by Ralph Johnson and Martin Fowler
Afterword by John Brant and Don Roberts

- “Anti-patterns” and “code smells”: indicators of design deficiencies
- Ignoring exceptions (*AP)*, magic strings (*AP),* repeated code (*CS*), long functions (*CS*)
- Detection partially automated
- Refactoring to more structured solutions

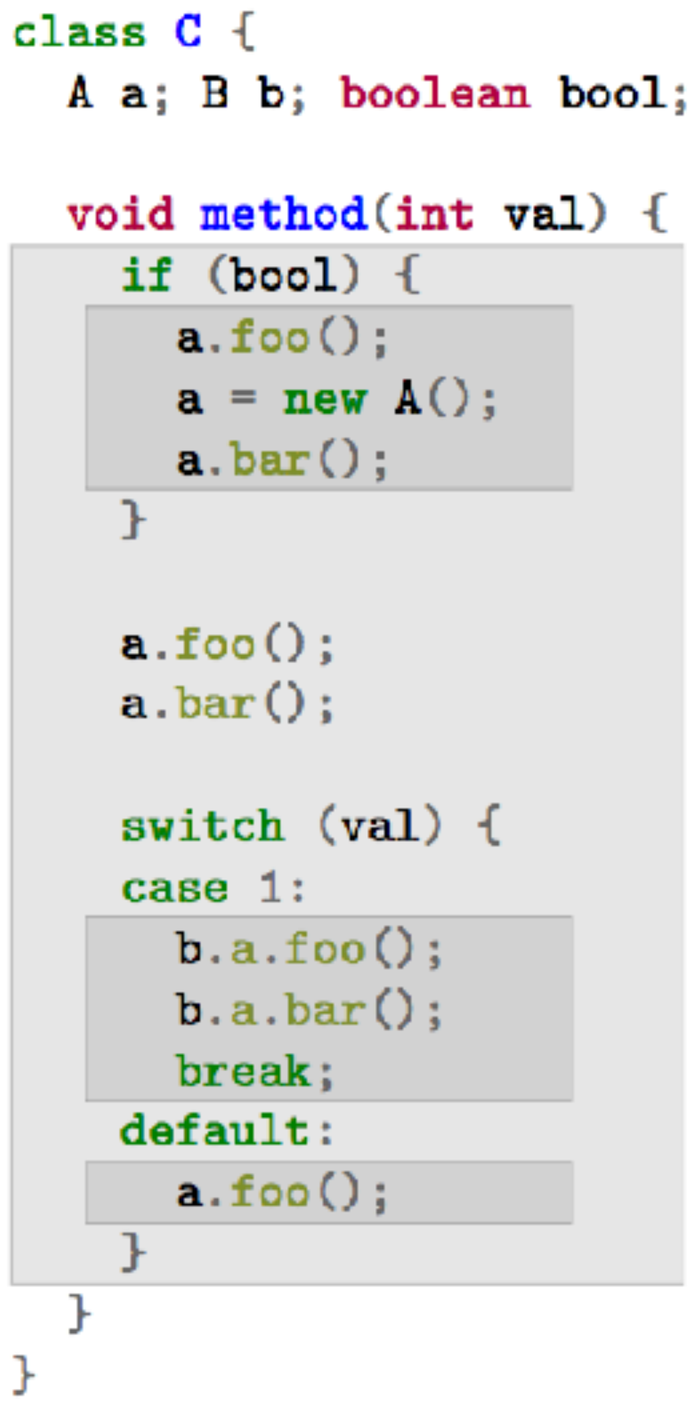
Software Quality Metrics

- How “good” is your code?
- Often subjective, but some guidelines:
    - high cohesion/low coupling between classes
    - long method body
    - class with too many methods
- Refactorings affect those metrics:
    - Extract Method *reduces* length of method and cyclometric complexity…
    - …but obviously increases *number of methods*.



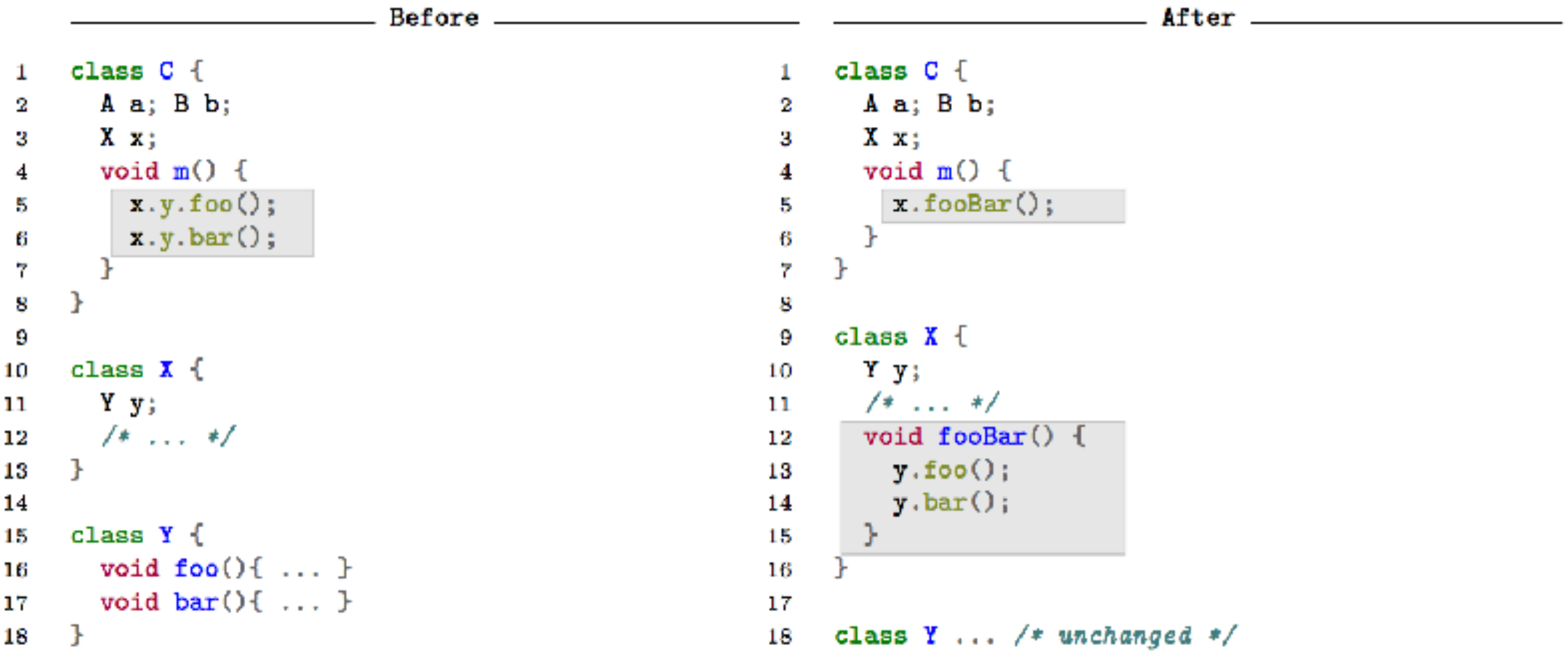
Software Quality Metrics (2)

- Tools like Findbugs, Checkstyle, JDeodorant, SonarQube identify problems
- Developers still need to act on that info
- Problem with automation:
  - large search-space
  - often many (overlapping) possibilities
  - Extract Method ↔ Inline Method “competing” against each other
- Our attempt: Kristensen/Stolz, “Search-based composed refactorings”, NIK 2014



```
class C {
  A a; B b; boolean bool;

  void method(int val) {
    if (bool) {
      a.foo();
      a = new A();
      a.bar();
    }

    a.foo();
    a.bar();

    switch (val) {
    case 1:
      b.a.foo();
      b.a.bar();
      break;
    default:
      a.foo();
    }
  }
}
```

Reducing Coupling



Before

```
class C {
  A a; B b;
  X x;
  void m() {
    x.y.foo();
    x.y.bar();
  }
}

class X {
  Y y;
  /* ... */
}

class Y {
  void foo(){ ... }
  void bar(){ ... }
}
```

After

```
class C {
  A a; B b;
  X x;
  void m() {
    x.fooBar();
  }
}

class X {
  Y y;
  /* ... */
  void fooBar() {
    y.foo();
    y.bar();
  }
}

class Y ... /* unchanged */
```

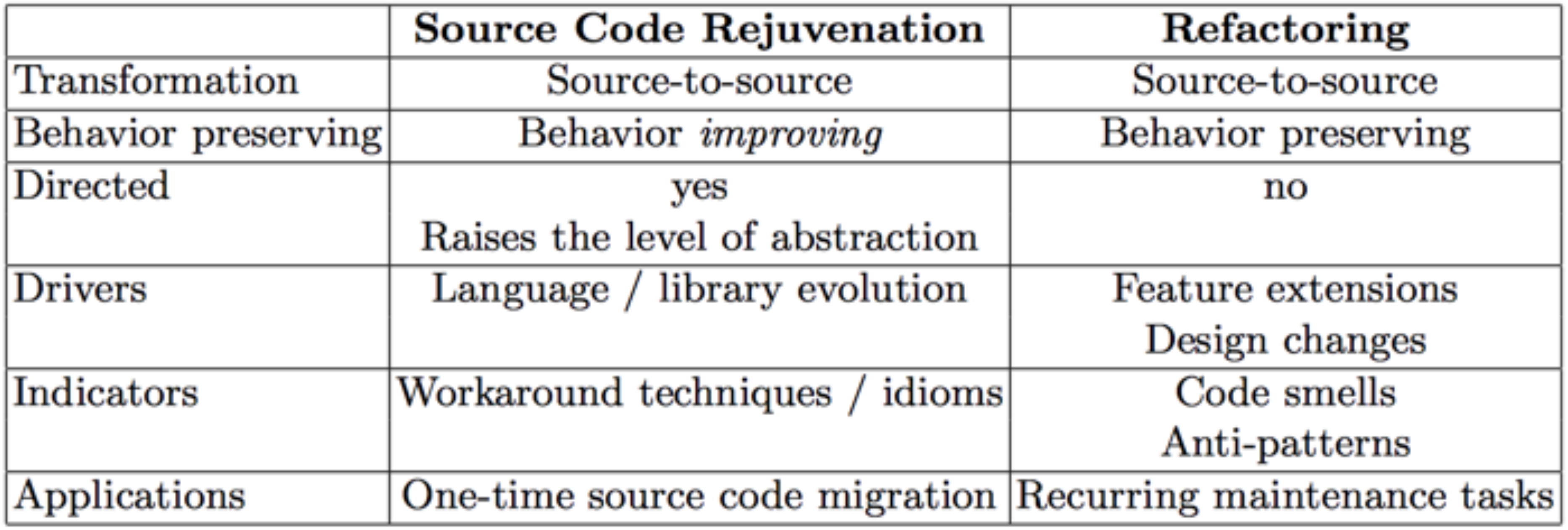
- Coupling Between Object Classes (CBO) of class C improves from 4 to 3…
- …but sometimes introduces additional coupling into the receiving class!

Related Topics: Source Code Rejuvenation

“**Source Code Rejuvenation**”
[Pirkelbauer, Dechev, Stroustrup ’10]

- automated migration of legacy code
- leverages enhanced program language/library facilities
- “*reverse (some forms of) (software) entropy*”
- “*preserves or improves a program’s behavior*”

Source Code Rejuvenation



	Source Code Rejuvenation	**Refactoring**
Transformation	Source-to-source	Source-to-source
Behavior preserving	Behavior *improving*	Behavior preserving
Directed	yes Raises the level of abstraction	no
Drivers	Language / library evolution	Feature extensions Design changes
Indicators	Workaround techniques / idioms	Code smells Anti-patterns
Applications	One-time source code migration	Recurring maintenance tasks

From: Pirkelbauer, Dechev, Stroustrup, SOFSEM 2010

Source Code Rejuvenation

```
vector<int> vec;

// three consecutive push backs

vec.push_back(1);
vec.push_back(2);
vec.push_back(3);
```

Inefficient!

Sizeof() what again?!

```
// copying from an array
int a[] = {1, 2, 3};
vector<int> vec(a,a+sizeof(a)/sizeof(int));
```

Now isn’t that pretty:

```
// rejuvenated source code in C++0x

vector<int> vec = {1, 2, 3};
```

Refactoring in IDEs

- All major IDEs support some form of refactoring
- Here: C, C++, Java
- Special case: command line tools for scripting (Go?)
- Support for scripting languages like Python, JavaScript, …
- Refactoring of UML models
  (semantical overlap with OO-refactoring)

Tool Support for Java

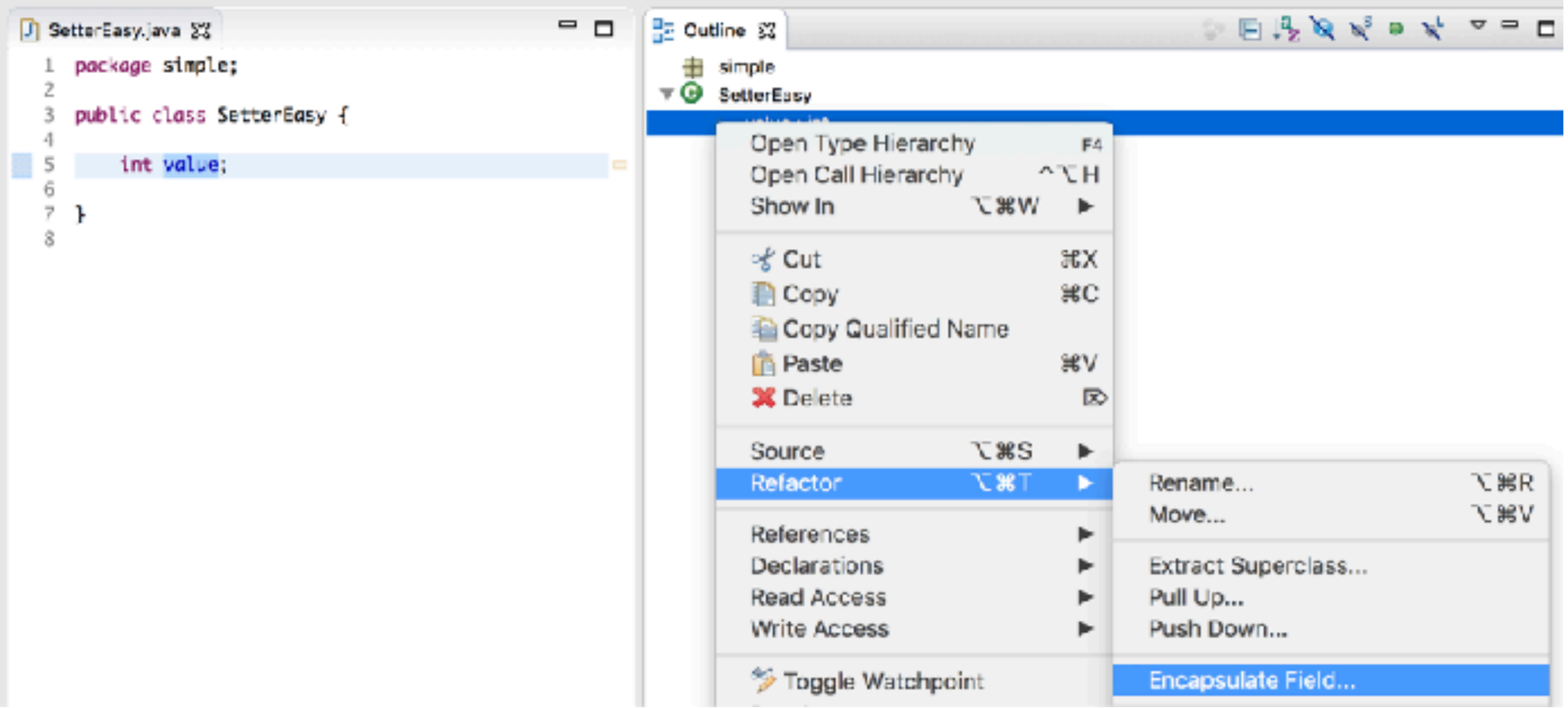
- Common IDEs:
  - Eclipse JDT
  - IntelliJ (Android)
  - NetBeans
- Other object-oriented languages similar:
  - Visual Studio

Refactoring: Common Java Examples

Encapsulate Field: avoid direct field access

1) introduce setter & getter methods;
2) replace all field accesses with calls to new methods;
3) make field private.

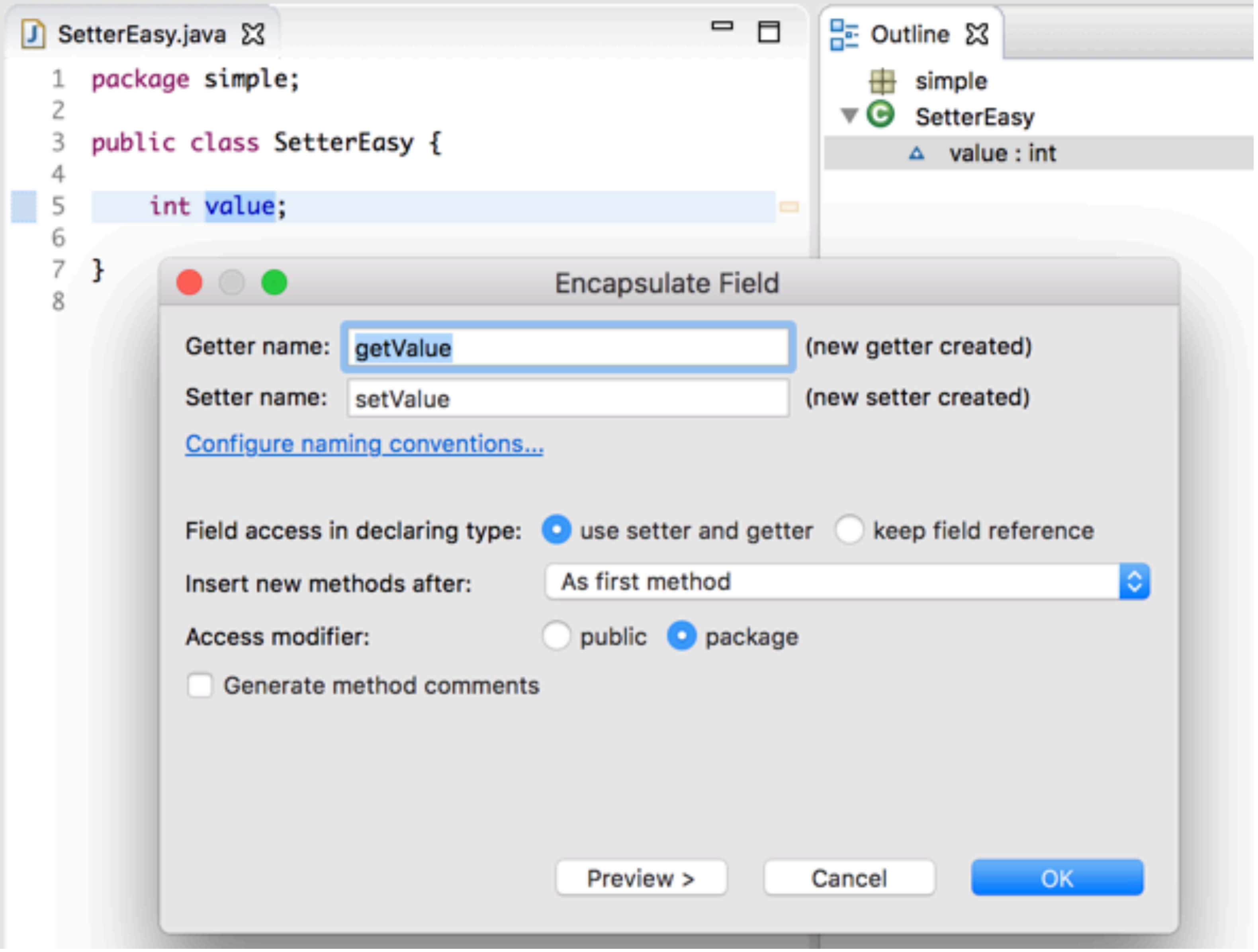
Encapsulate Field



```
package simple;

public class SetterEasy {

    int value;

}
```

Right-click on a field and find the “Refactor” menu.

Encapsulate Field



```
package simple;

public class SetterEasy {

    int value;

}
```

Encapsulate Field

Getter name: getValue (new getter created)

Setter name: setValue (new setter created)

Configure naming conventions...

Field access in declaring type: ● use setter and getter ○ keep field reference

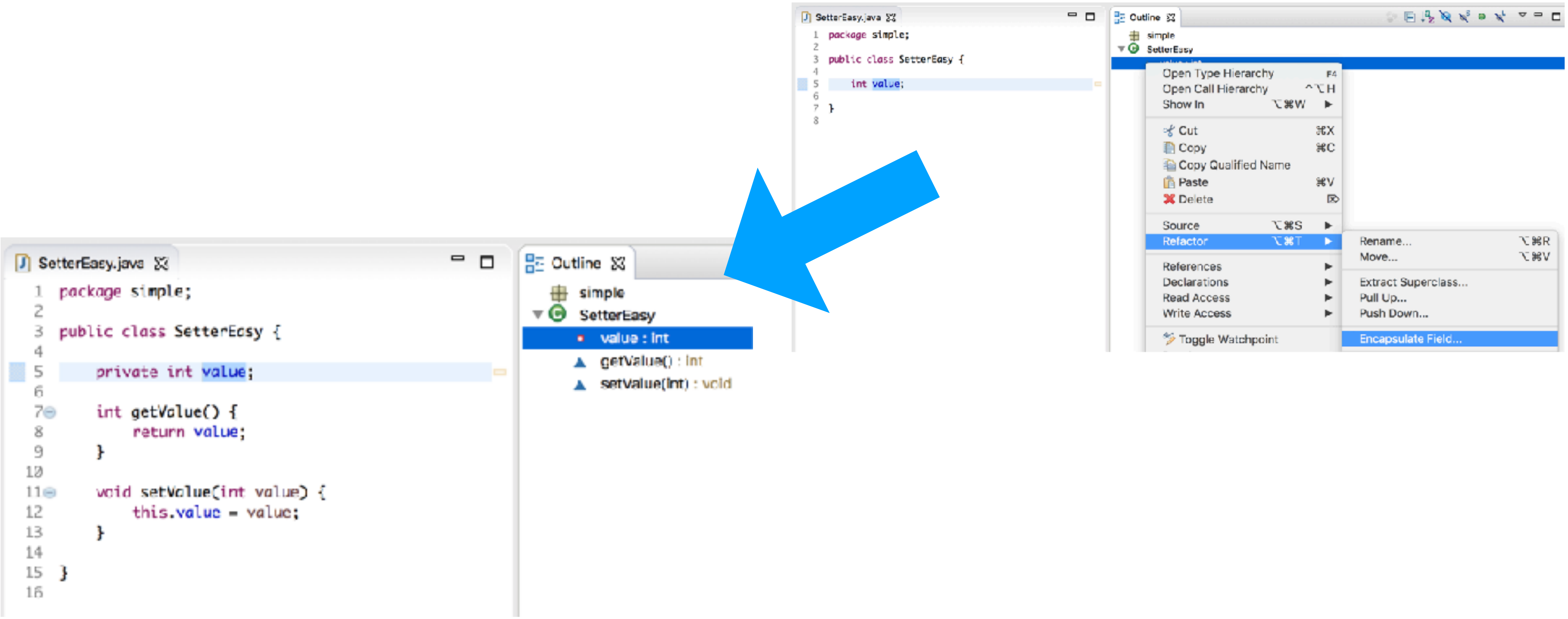
Insert new methods after: As first method

Access modifier: ○ public ● package

□ Generate method comments

IDEs will often have a helpful dialog, because further input is required.

Encapsulate Field

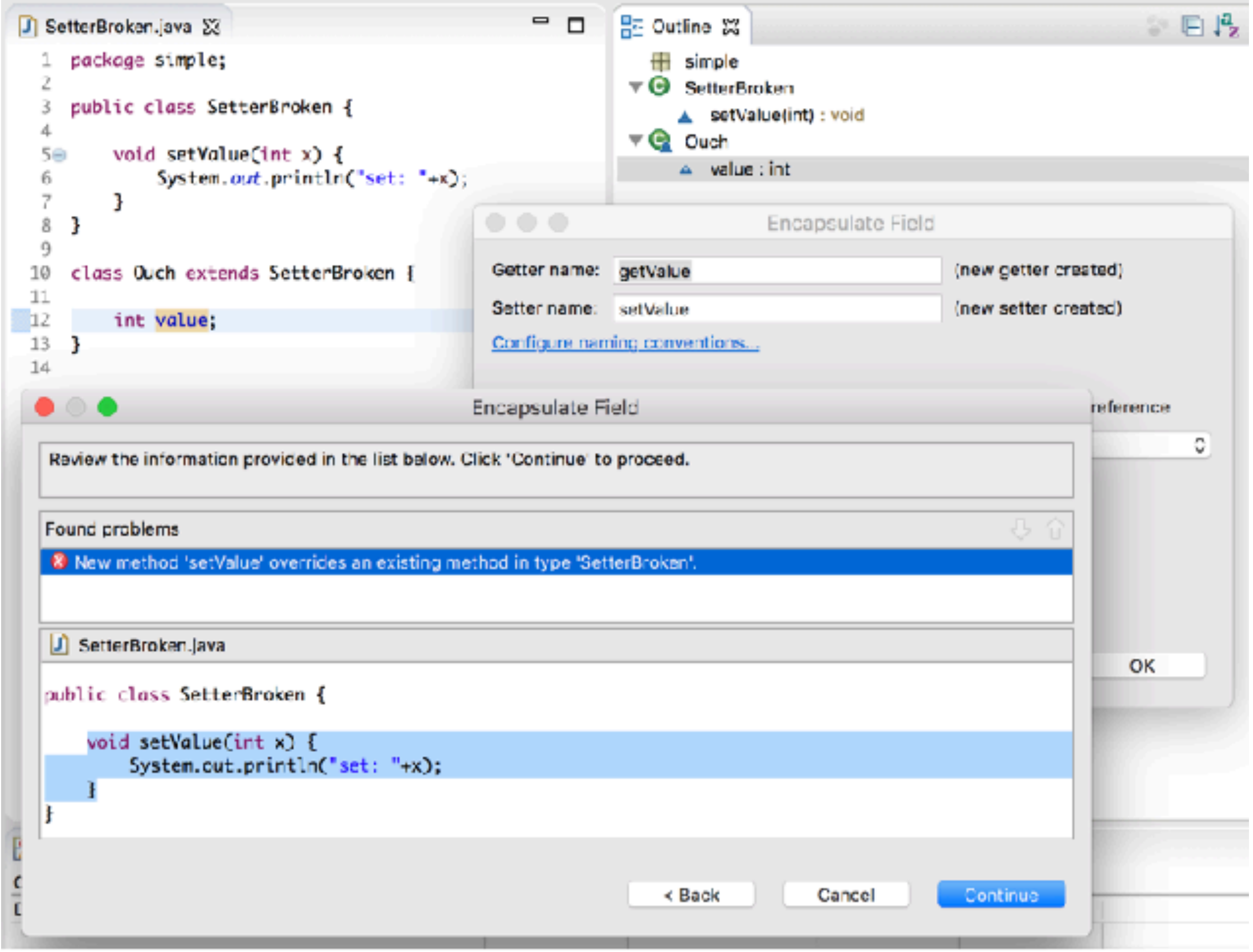


```
package simple;

public class SetterEasy {

	int value;
}
```

```
package simple;

public class SetterEasy {

	private int value;

	int getValue() {
		return value;
	}

	void setValue(int value) {
		this.value = value;
	}

}
```

Enjoy your result!

Encapsulate Field



```
package simple;

public class SetterBroken {

    void setValue(int x) {
        System.out.println("set: "+x);
    }
}

class Ouch extends SetterBroken {

    int value;
}
```

Encapsulate Field

Getter name: getValue (new getter created)

Setter name: setValue (new setter created)

Configure naming conventions...

Encapsulate Field

Review the information provided in the list below. Click 'Continue' to proceed.

Found problems

New method 'setValue' overrides an existing method in type 'SetterBroken'.

SetterBroken.java

```
public class SetterBroken {

    void setValue(int x) {
        System.out.println("set: "+x);
    }
}
```

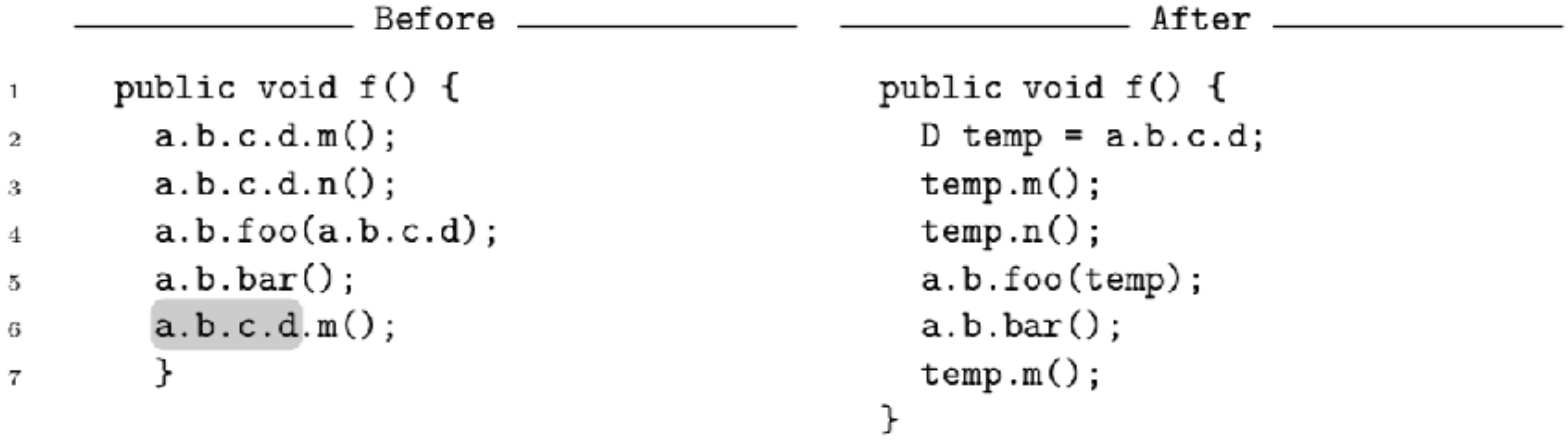
IDEs will even try to be helpful!

Refactoring: Common Java Examples

Encapsulate Field: avoid direct field access
1) introduce setter & getter methods;
2) replace all field accesses with calls to new methods;
3) make field private.

Let’s assume you have to *program* this refactoring.
Can you see what happens if you swap steps 2 & 3?
We will come back later to that.

Refactoring: Extract Local Variable



Before

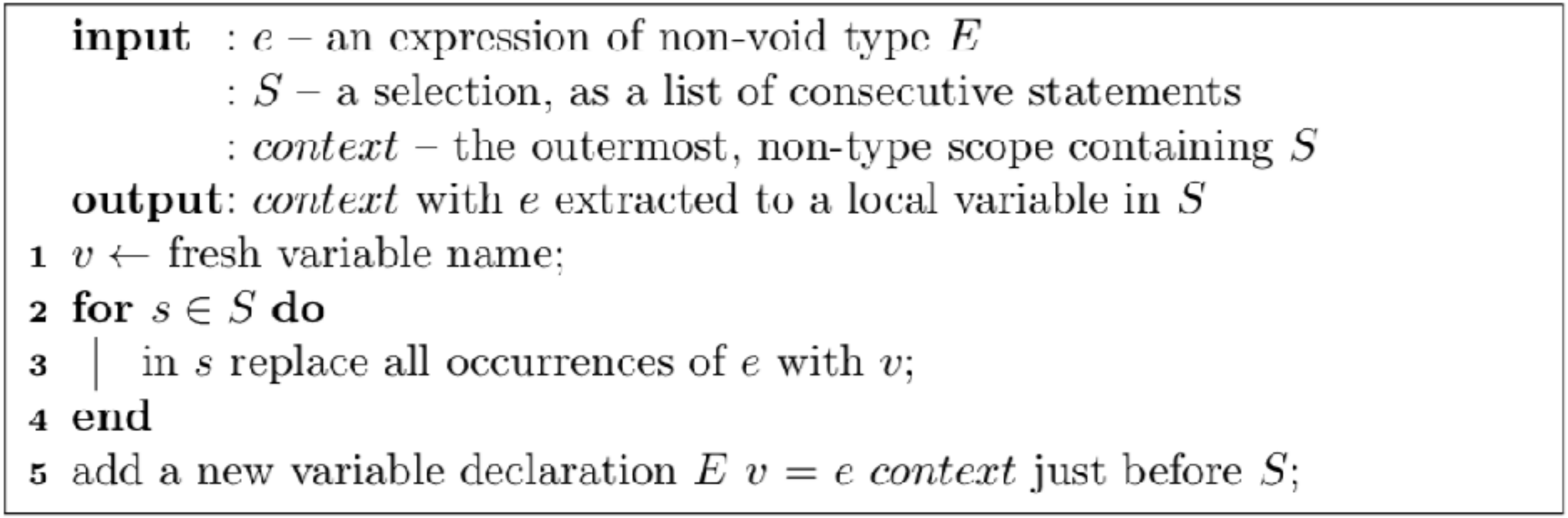
```
public void f() {
  a.b.c.d.m();
  a.b.c.d.n();
  a.b.foo(a.b.c.d);
  a.b.bar();
  a.b.c.d.m();
  }
```

After

```
public void f() {
  D temp = a.b.c.d;
  temp.m();
  temp.n();
  a.b.foo(temp);
  a.b.bar();
  temp.m();
}
```

Compute complex (expensive) expression only once.

Extract Local Variable: Formally



```
input  : $e$ – an expression of non-void type $E$
       : $S$ – a selection, as a list of consecutive statements
       : $context$ – the outermost, non-type scope containing $S$
output : $context$ with $e$ extracted to a local variable in $S$
$v \leftarrow$ fresh variable name;
for $s \in S$ do
    in $s$ replace all occurrences of $e$ with $v$;
end
add a new variable declaration $E\ v = e$ $context$ just before $S$;
```