

# Refactoring

Keith Cassell  
Victoria University of Wellington  
[kcassell@ecs.vuw.ac.nz](mailto:kcassell@ecs.vuw.ac.nz)

# The Only Note You Need to Take<sup>\*</sup>

These slides are at:

[http://homepages.ecs.vuw.ac.nz/~kcassell/  
cassellRefactoringJUG.pdf](http://homepages.ecs.vuw.ac.nz/~kcassell/cassellRefactoringJUG.pdf)

<sup>\*</sup> (Unless I say something brilliant.)

# More Information

- References

- Refactoring : Improving the Design of Existing Code – Fowler, et al

- My refactoring bibliography -

- <http://homepages.ecs.vuw.ac.nz/~kcassell/refactoringPapers.xhtml>

- Web sites

- Refactoring forum -

- <http://tech.groups.yahoo.com/group/refactoring/>

- <https://netfiles.uiuc.edu/dig/RefactoringInfo/>

- <http://www.refactoring.com/>

# Outline

- Introduction to refactoring
- The refactoring process
- State of the practice
- Case study – extract class
- Discussion and/or demos
- Beer

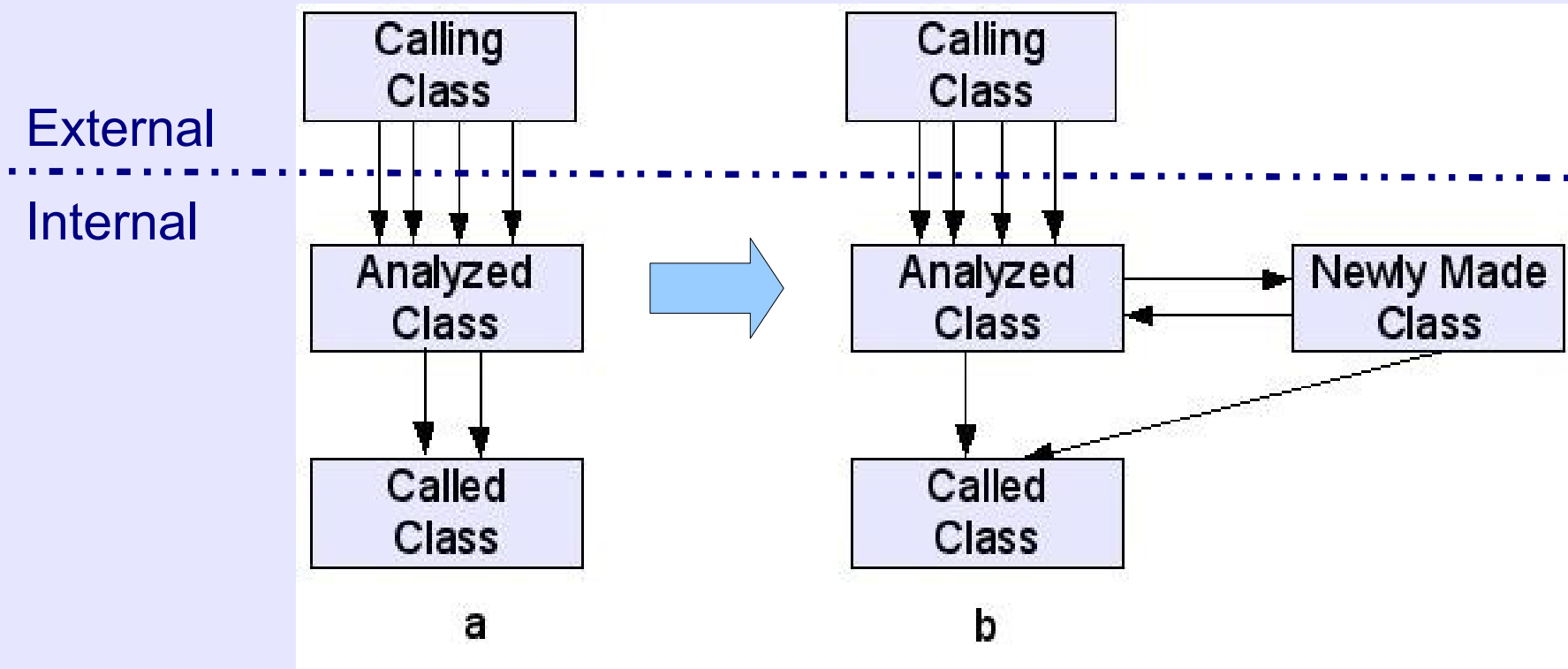
# Definition

Refactoring “... does not alter the external behaviour of the code, yet improves its internal structure”  
(Fowler)



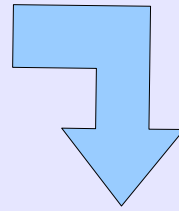
# Definition

Refactoring “... does not alter the external behaviour of the code, yet improves its internal structure”  
(Fowler)





Complex, difficult to understand code



## Refactoring Code to Improve Maintainability



Well-organized, simpler code

# Example Refactorings

- Rename
- Move Field/Method
- Extract Method
- Extract Local Variable
- Extract Interface
- Merge Classes
- Extract Class
- Extract Constant
- Replace Conditional with Polymorphism
- Replace Conditional with Visitor
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance
- Many more....

# Example – Refactoring Description

Refactoring Name: Extract Class (Fowler)

Symptom (a.k.a. “smell”):

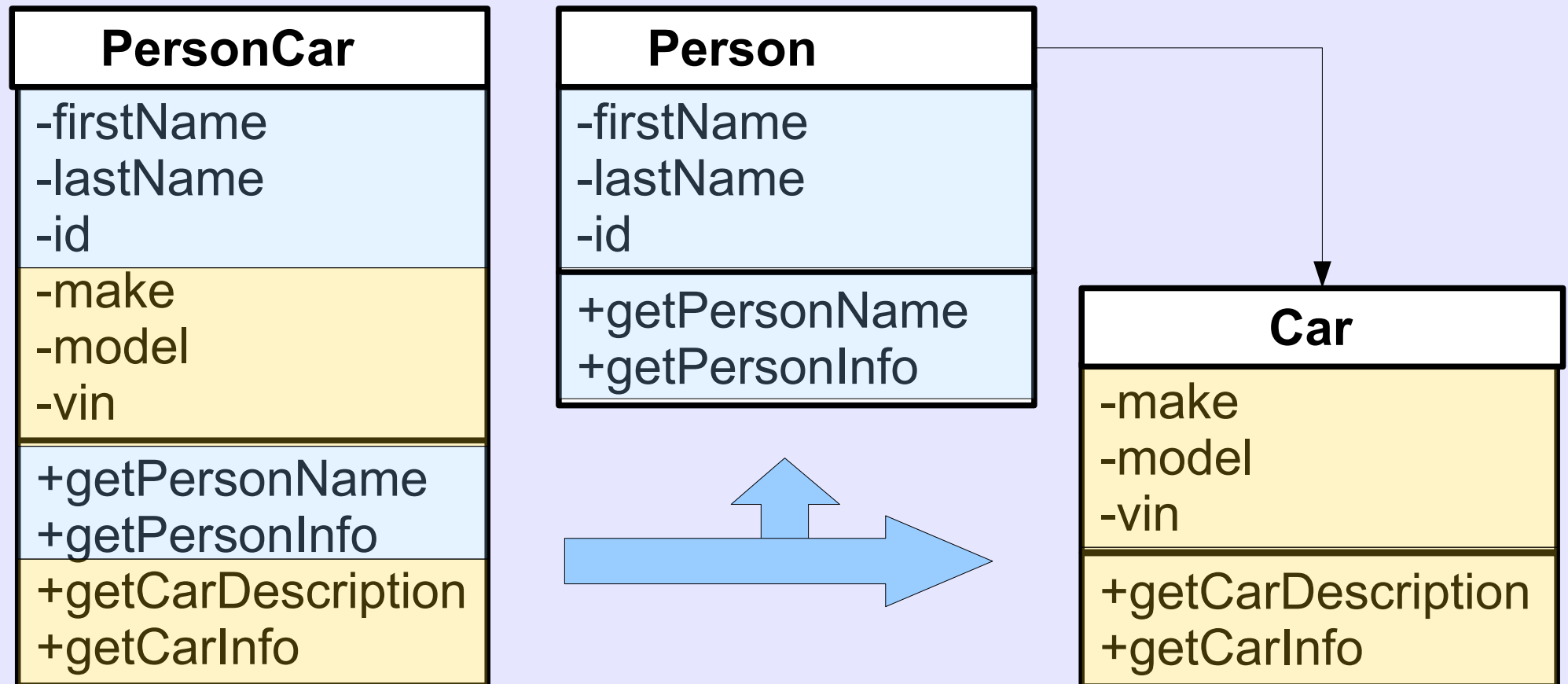
- One class is doing the work of two or more

Cure:

- Create a new class and move the relevant fields and methods from the old class into the new class.

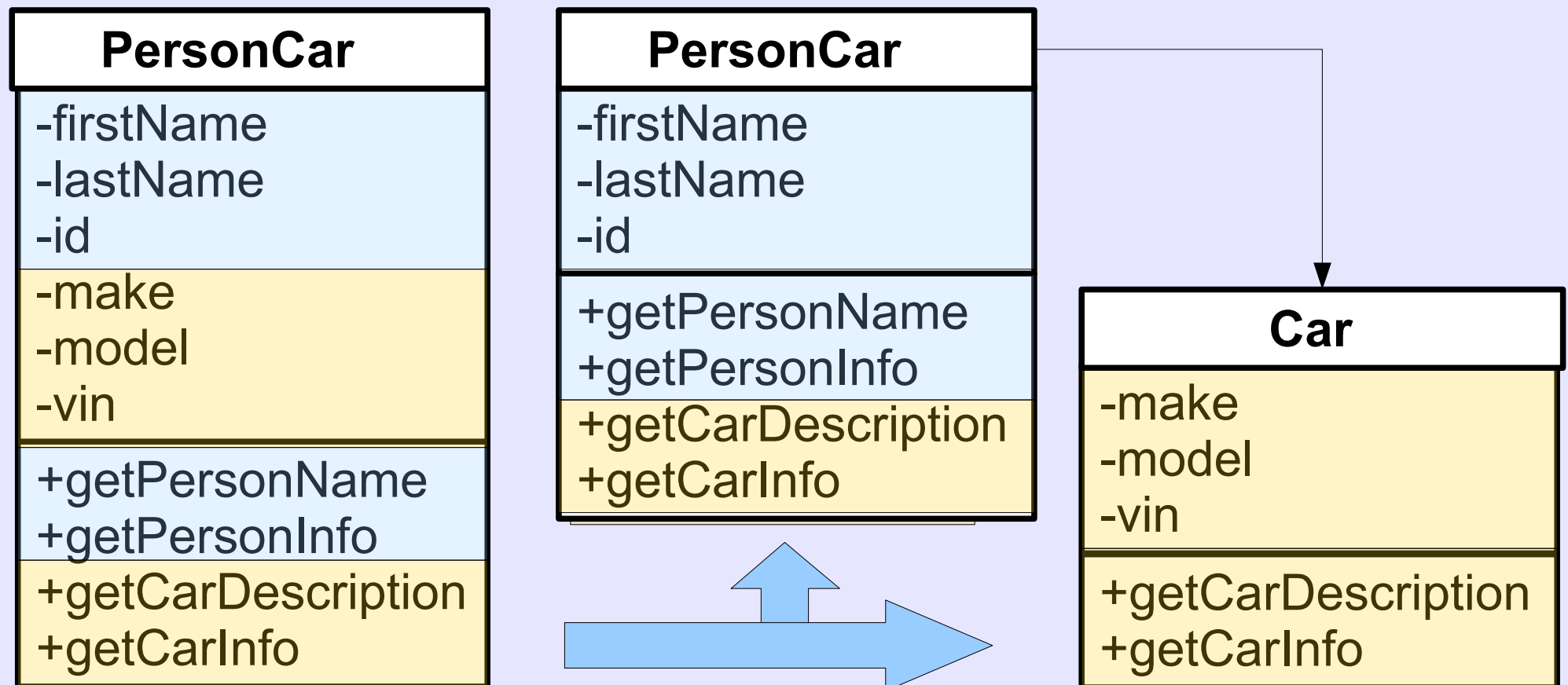
# “Loose” Refactoring

Maintaining behavior when all clients can be identified and modified:



# “Strict” Refactoring

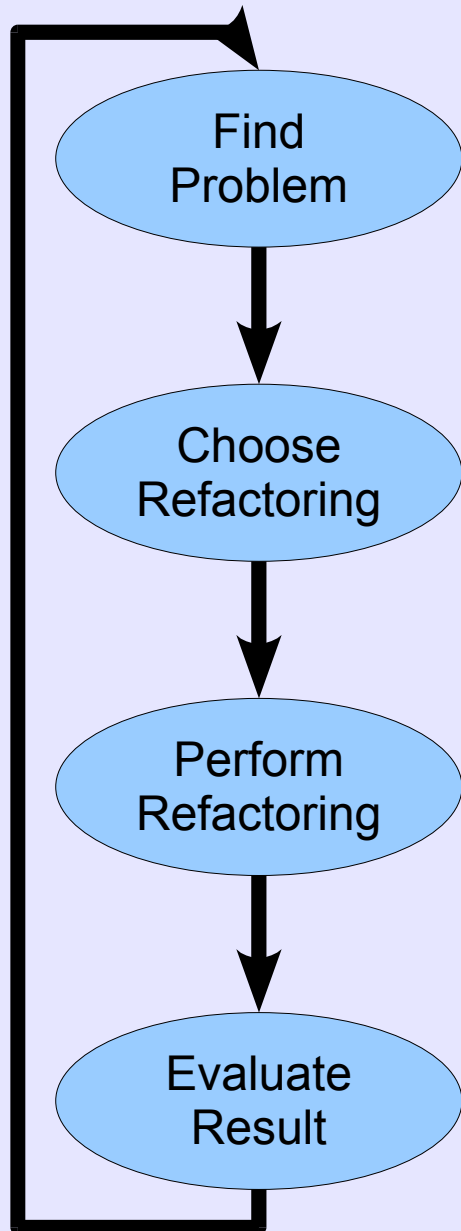
Maintaining behavior (interfaces) for unknown clients imposes many constraints.



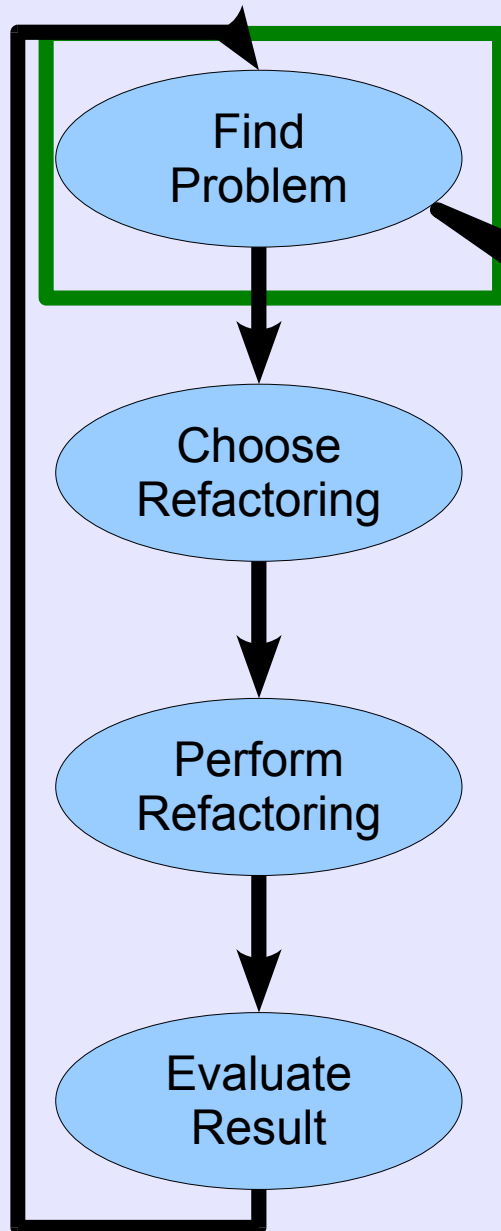
# Outline

- Introduction to refactoring
- **The refactoring process**
- State of the practice
- Case study – extract class
- Discussion and/or demos
- Beer

# The Refactoring Process

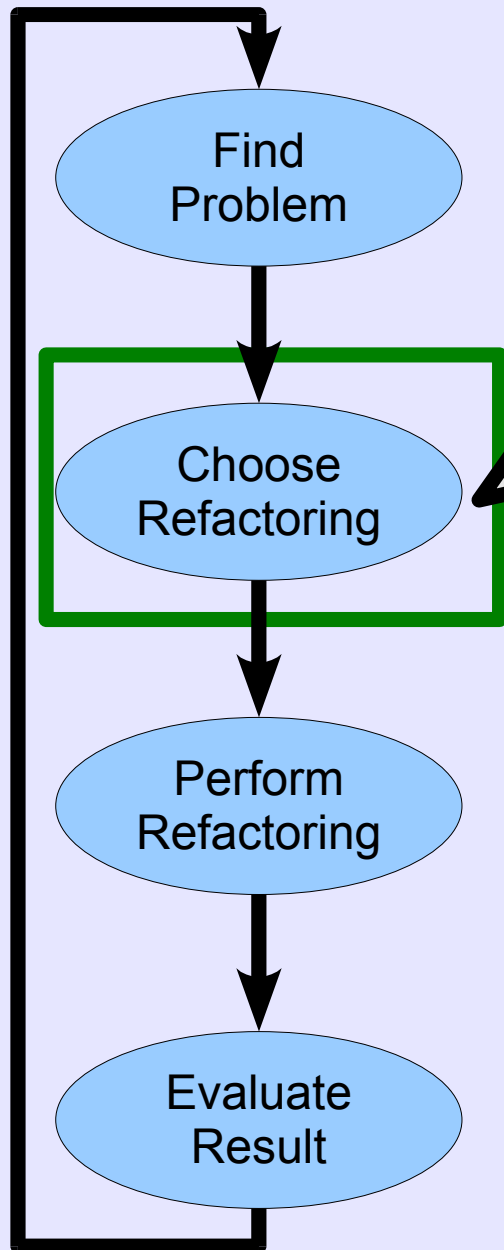


# The Refactoring Process: “Find and Fix”

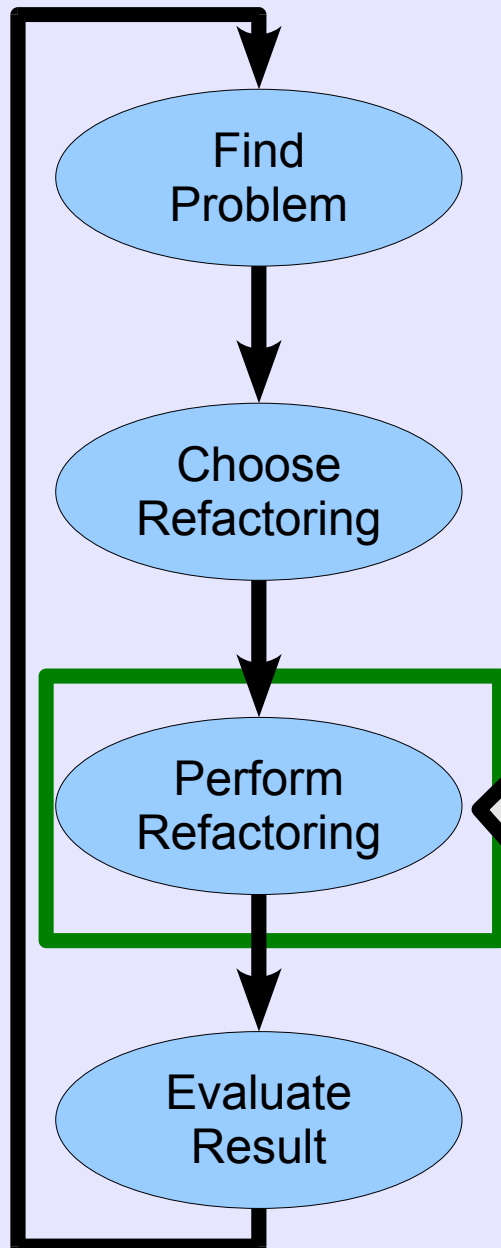


## “Bad Smells”

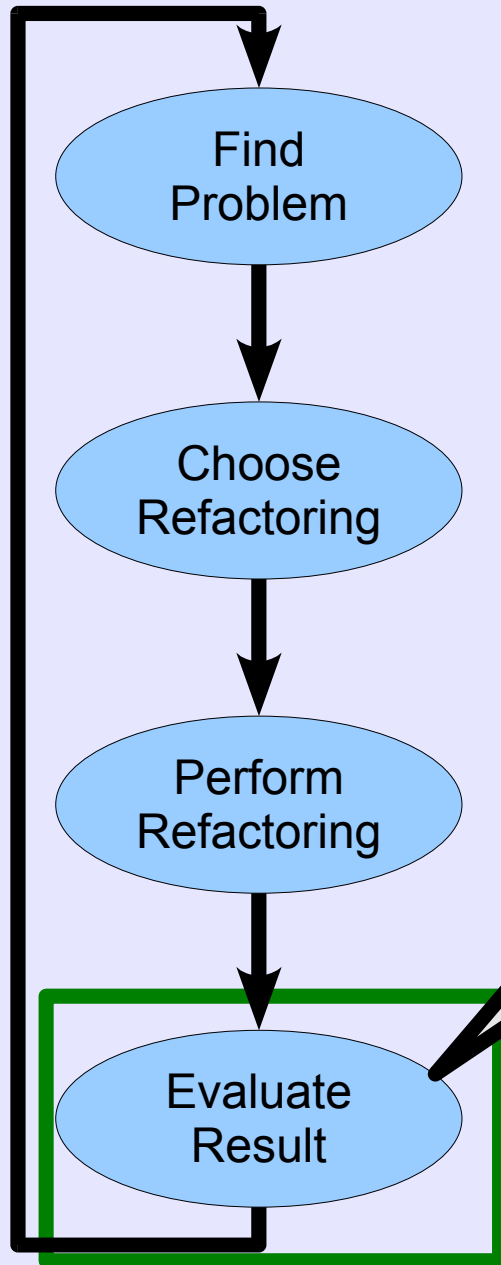
- Dead code
- Duplicate code
- Overly large classes
- Overly complex methods
- “Anti-Patterns”
- “Feature envy” ...



Replace Conditional with Polymorphism  
vs. Replace Conditional with Visitor  
Extract Subclass vs. Create Delegate  
...



- Rename
- Pull up Field
- Extract Method
- Merge Classes
- Split Class
- Replace Conditional with Polymorphism
- Replace Conditional with Visitor
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance



- Human Judgment
- Before and After Measurements
  - Maximum Class Size
  - Maximum Method Size
  - Complexity
  - Cohesiveness
  - etc.
- Test Results

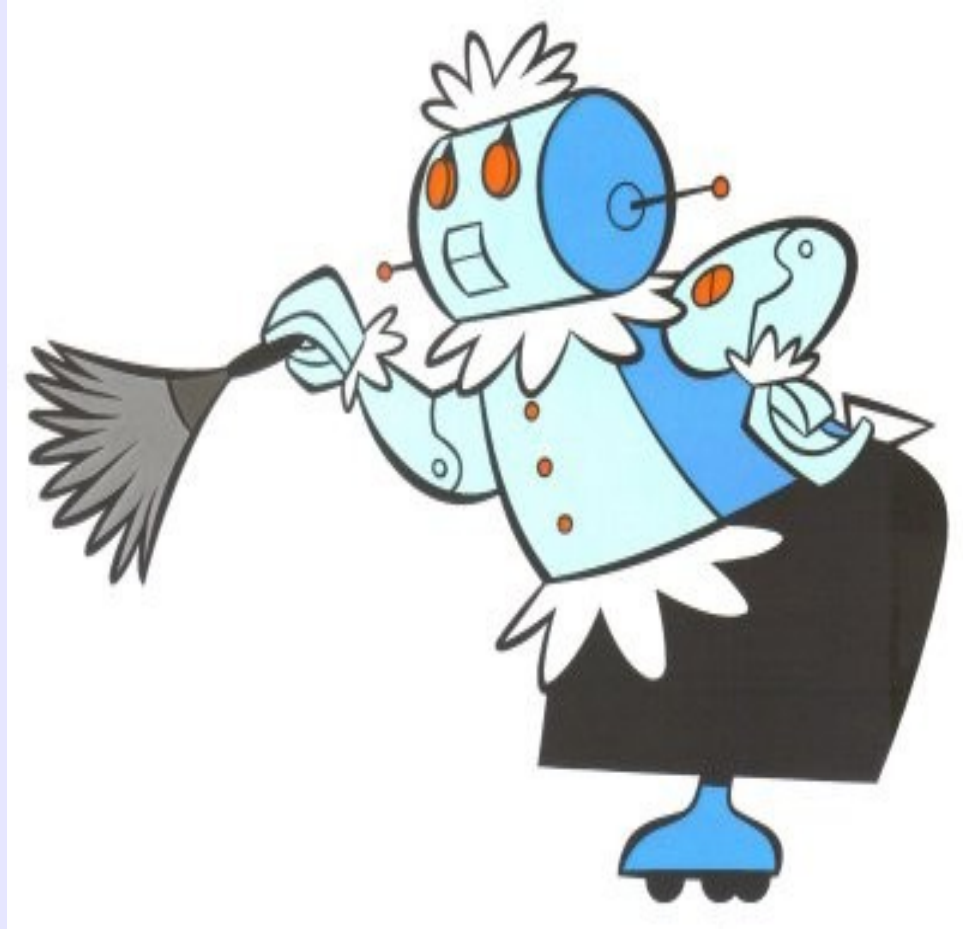
# Manual Refactoring



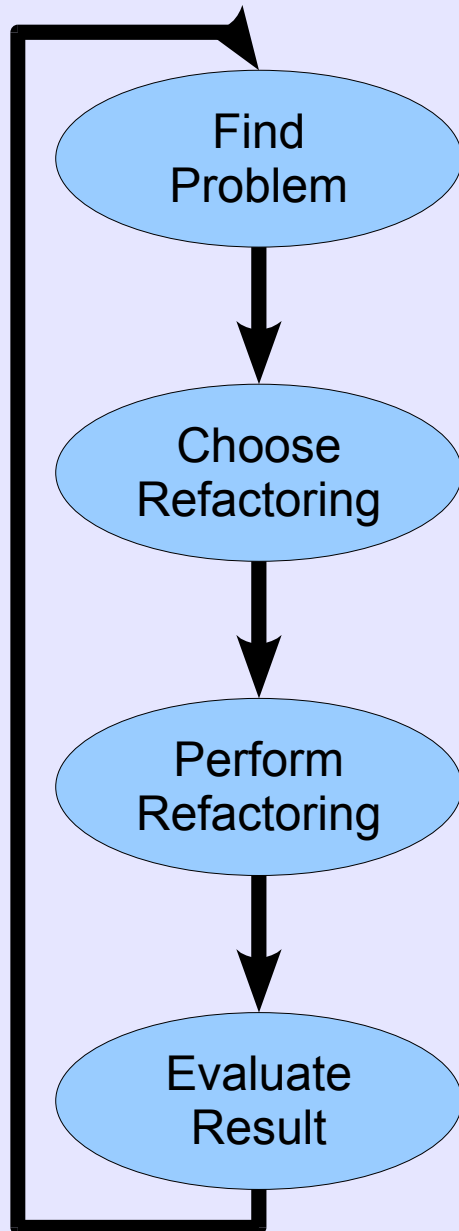
# The Manual Refactoring Process



# Fully Automated Refactoring



# The Automated Refactoring Process



Automated smell detector -  
metrics, anti-patterns, ...

Configurable expert system

Transformation tools

Automated test suites,  
metric evaluations

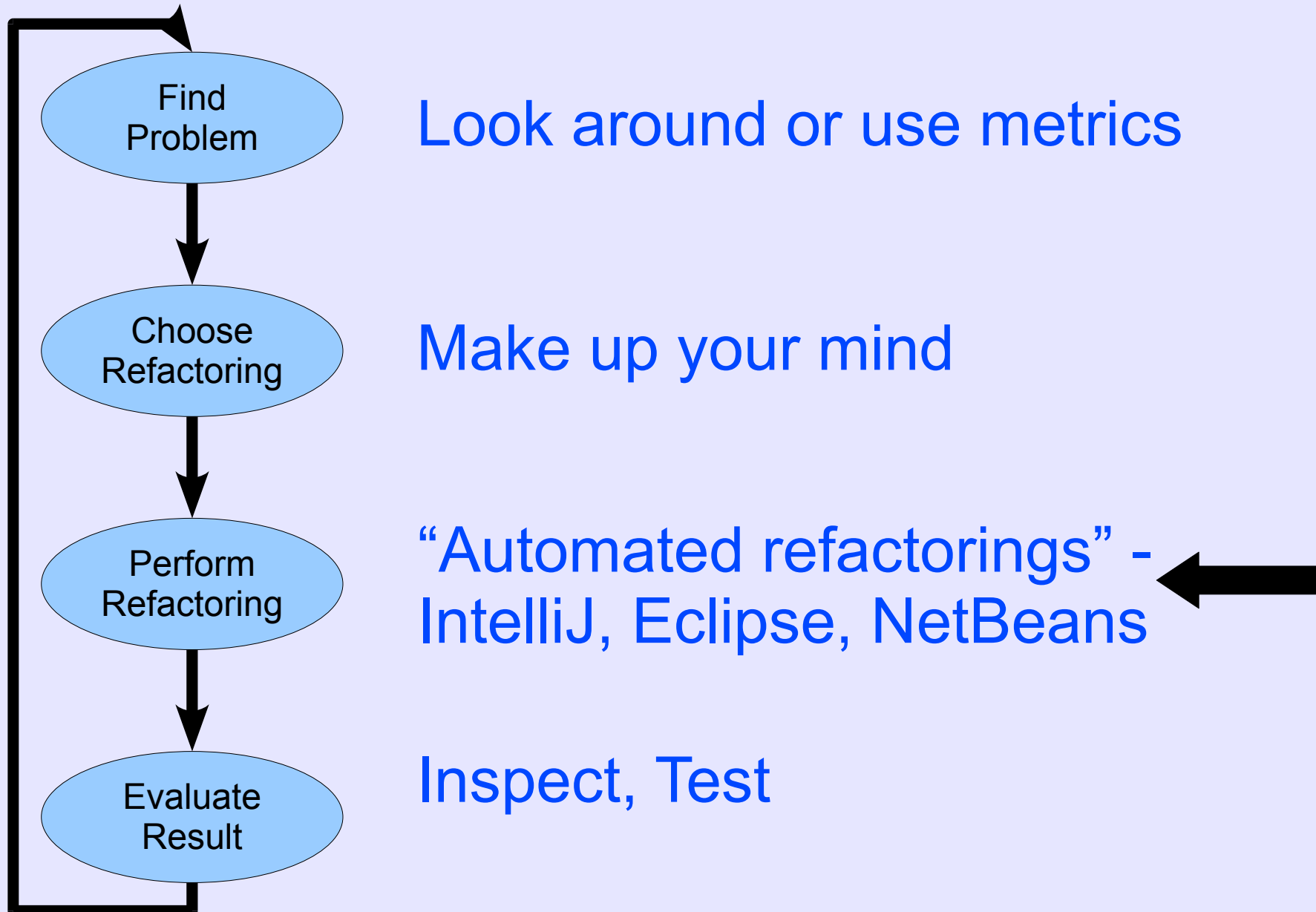
# Outline

- Introduction to refactoring
- The refactoring process
- **State of the practice**
- Case study – extract class
- Discussion and/or demos
- Beer

# State of the Practice



# The State of the Practice



# State of the Practice – Manually identify problem and choose candidate solution

The screenshot shows an IDE window with several tabs: SplitClassDemo.java, PMDTaskMethods.ql, Super1.java, Sub1.java, and Sub2.java. The main editor displays the following Java code:

```
public class Sub1 extends Super1
{
    public int compute(int a) {
        int r = 0;
        r = a < 0 ? -1 : a;
        r = r < 0 ? 0 :
            (a == 0) ? a*a :
            a == 1 ? a++: a--;
        return r;
    }
}
```

A context menu is open over the code block, listing various actions:

- Undo Typing (Ctrl+Z)
- Revert File
- Save (Ctrl+S)
- Open Declaration (F3)
- Open Type Hierarchy (F4)
- Open Call Hierarchy (Ctrl+Alt+H)
- Show in Breadcrumb (Alt+Shift+B)
- Quick Outline (Ctrl+O)
- Quick Type Hierarchy (Ctrl+T)
- Show In (Alt+Shift+W)
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Copy Qualified Name
- Paste (Ctrl+V)
- Quick Fix (Ctrl+1)
- Source (Alt+Shift+S)
- Refactor (Alt+Shift+T)
- Surround With (Alt+Shift+Z)
- Local History
- References
- Declarations
- Run As

The 'Refactor' menu item is expanded, showing a sub-menu with the following options:

- Move... (Alt+Shift+V)
- Change Method Signature... (Alt+Shift+C)
- Extract Method... (Alt+Shift+M)
- Extract Local Variable... (Alt+Shift+L)
- Extract Interface...
- Extract Superclass...

The 'Extract Method...' option is highlighted in the sub-menu. The IDE interface also shows a 'Console' tab, 'Call Hierarchy', and 'Search' buttons at the bottom, and an 'Outline' window on the right showing the class structure.

# Specify Details for Automated Refactoring

The screenshot shows the Eclipse IDE interface. The background window displays a Java class named `Sub1` that extends `Super1`. Inside the `compute` method, a block of code is selected, including variable declarations and conditional assignments. The `Extract Method` dialog box is open in the foreground, allowing the user to configure the new method to be extracted from the selected code.

**Extract Method**

Method name:

Access modifier:  public  protected  default  private

Parameters:

Type	Name
int	a
int	r

Declare thrown runtime exceptions

Generate method comment

Replace all occurrences of statements with method

Method signature preview:  
`protected int computeR(int a, int r)`

Buttons:

# Manually Evaluate Proposed Result

**Extract Method**

Changes to be performed

- Sub1.java - ClassRefactorings/src
  - Organize Imports
  - Sub1
    - compute(int)
    - Substitute statement(s) with call to computeR
    - Create new method 'computeR' from selected statement(s)

Sub1.java

Original Source

```
public class Sub1 extends Super1
{
    public int compute(int a) {
        int r = 0;
        r = a < 0 ? -1 : a;
        r = r < 0 ? 0 :
        (a == 0) ? a*a :
        a == 1 ? a++: a--;
        return r;
    }
}
```

Refactored Source

```
r = computeR(a, r);
return r;
}

protected int computeR(i
{
    r = r < 0 ? 0 :
    (a == 0) ? a*a :
    a == 1 ? a++: a-
return r;
}
```

Preview > OK Cancel

# Refactoring Performed

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project named 'ClassRefactorings' with a source folder 'src' containing files 'Sub1.java', 'Sub11.java', 'Sub2.java', and 'Super1.java'. The main editor window shows the code for 'Sub1.java'. The code defines a class 'Sub1' that extends 'Super1'. It has a public method 'compute' and a protected method 'computerR'. The 'compute' method is currently selected, and a refactoring action is being performed, as indicated by the blue highlight and the 'Writable' status bar.

```
public class Sub1 extends Super1
{
    public int compute(int a) {
        int r = 0;
        r = a < 0 ? -1 : a;
        r = computerR(a, r);
        return r;
    }

    protected int computerR(int a, int r)
    {
        r = r < 0 ? 0 :
            (a == 0) ? a*a :
            a == 1 ? a++ : a--;
        return r;
    }
}
```

# Outline

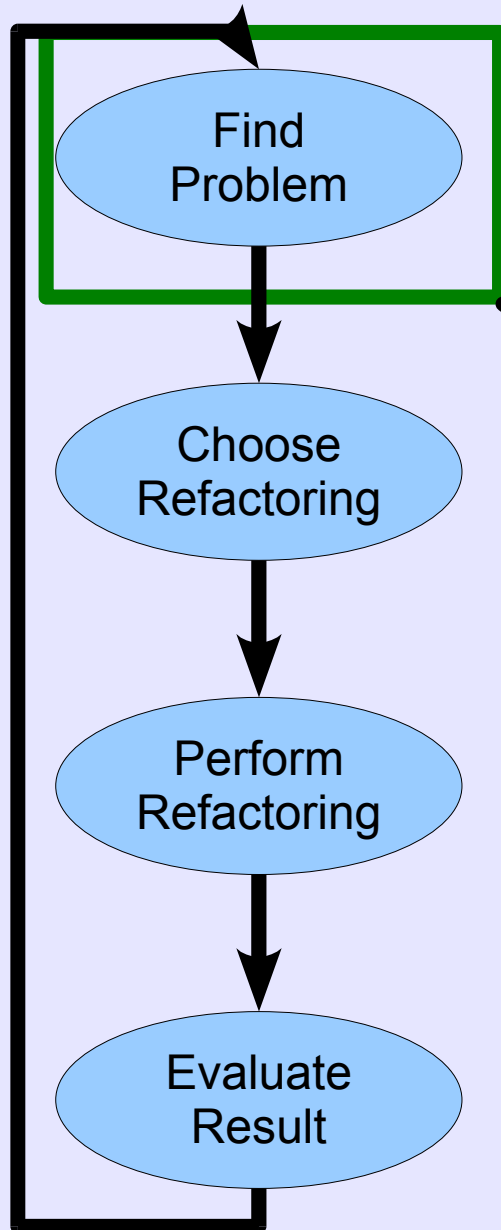
- Introduction to refactoring
- The refactoring process
- State of the practice
- **Case study – extract class**
- Discussion and/or demos
- Beer

# Extract Class - Motivation

“God classes... are the nightmare of maintainers”. - Demeyer

“...a class with too much code is prime breeding ground for duplicated code, chaos, and death” - Fowler

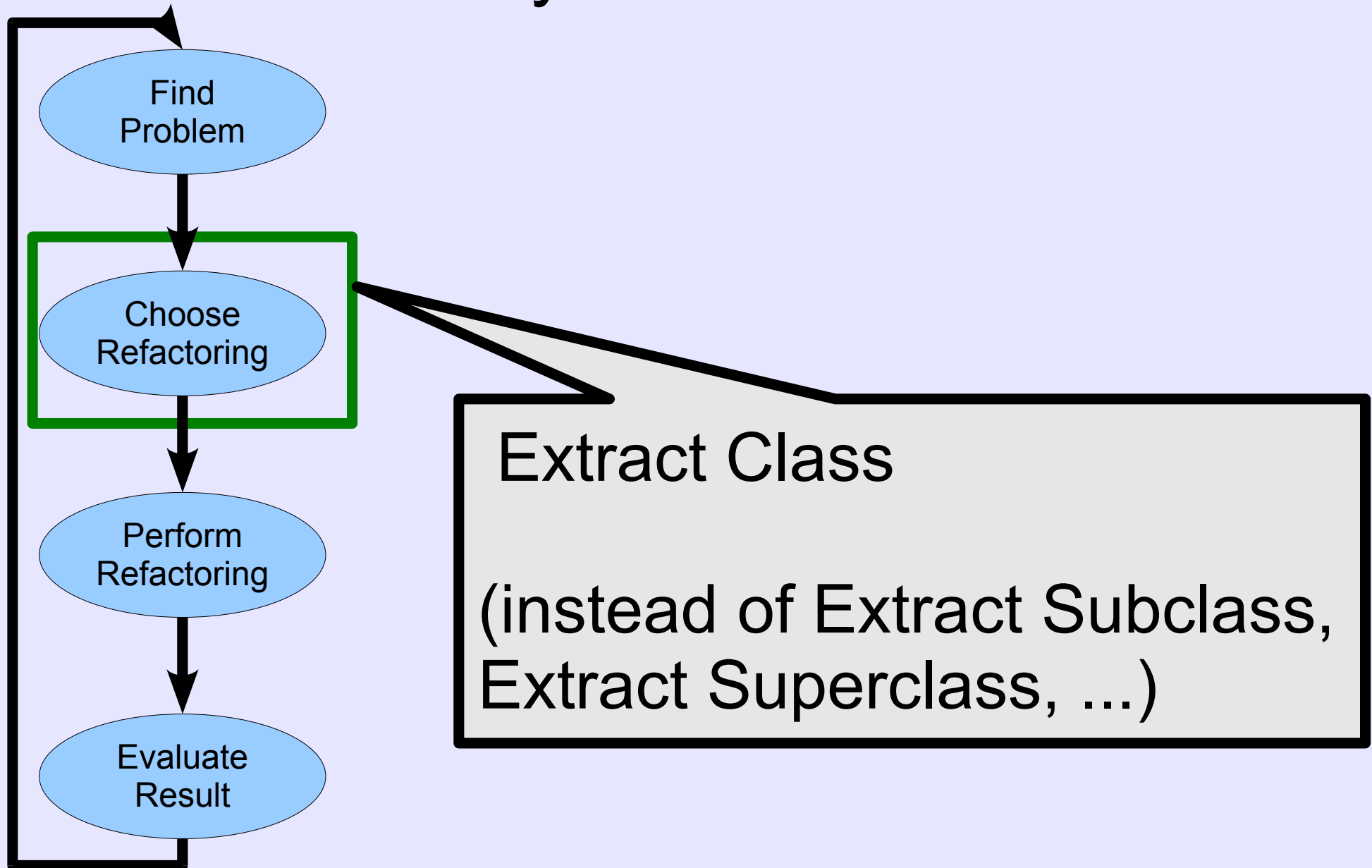
# Case Study – Extract Class



Class Metrics:

# methods > 20 AND  
Cohesion < 0.3

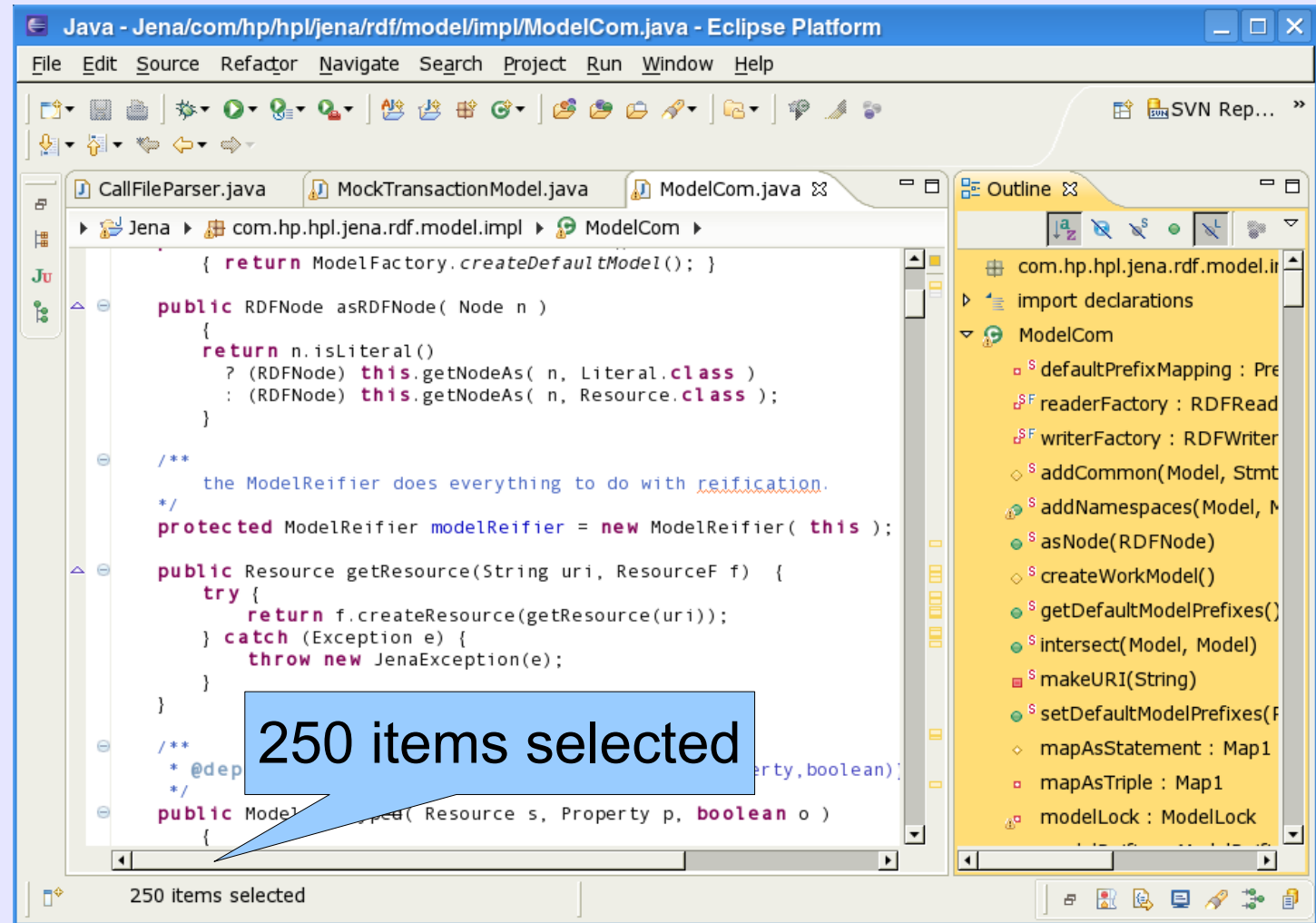
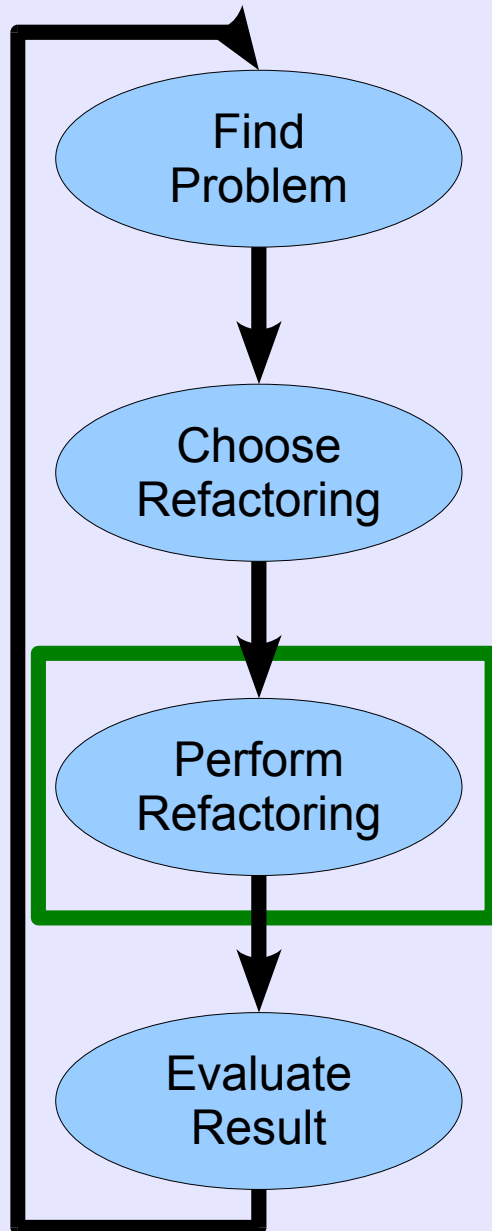
# Case Study – Extract Class



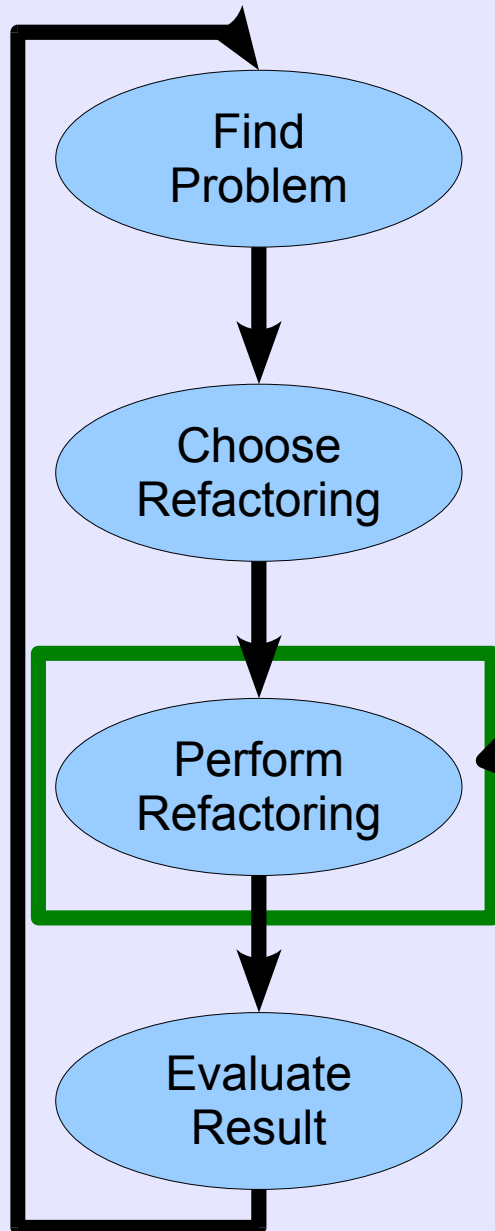
Extract Class

(instead of Extract Subclass,  
Extract Superclass, ...)

# Case Study – Extract Class



# Case Study – Extract Class

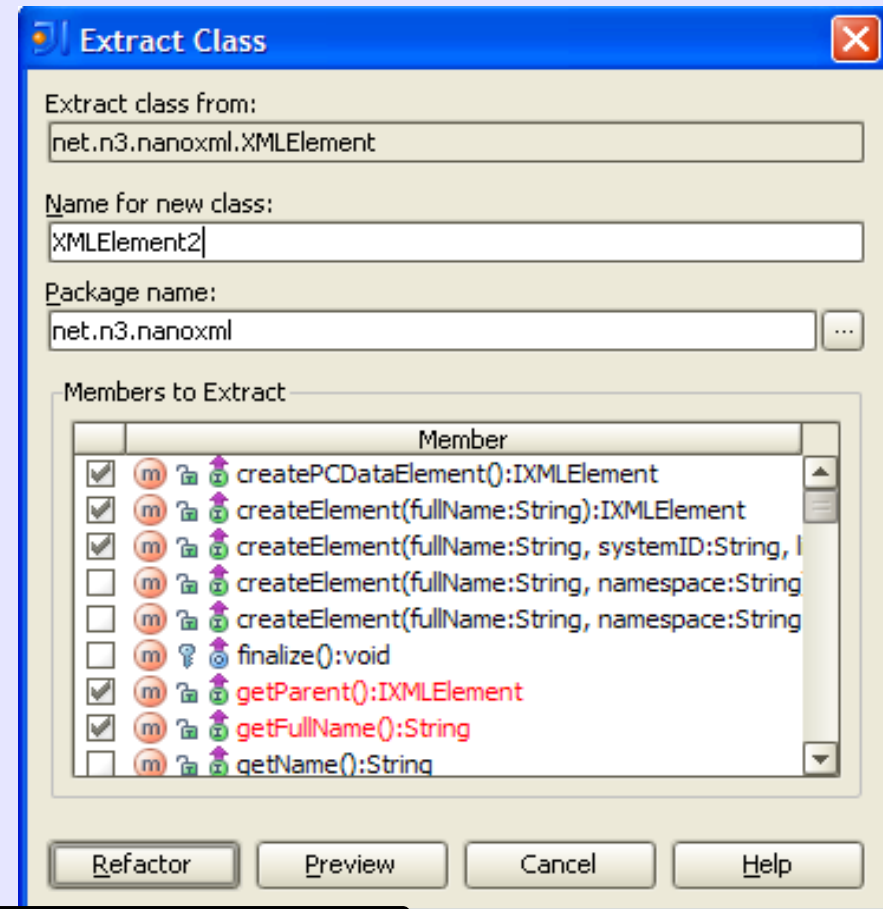
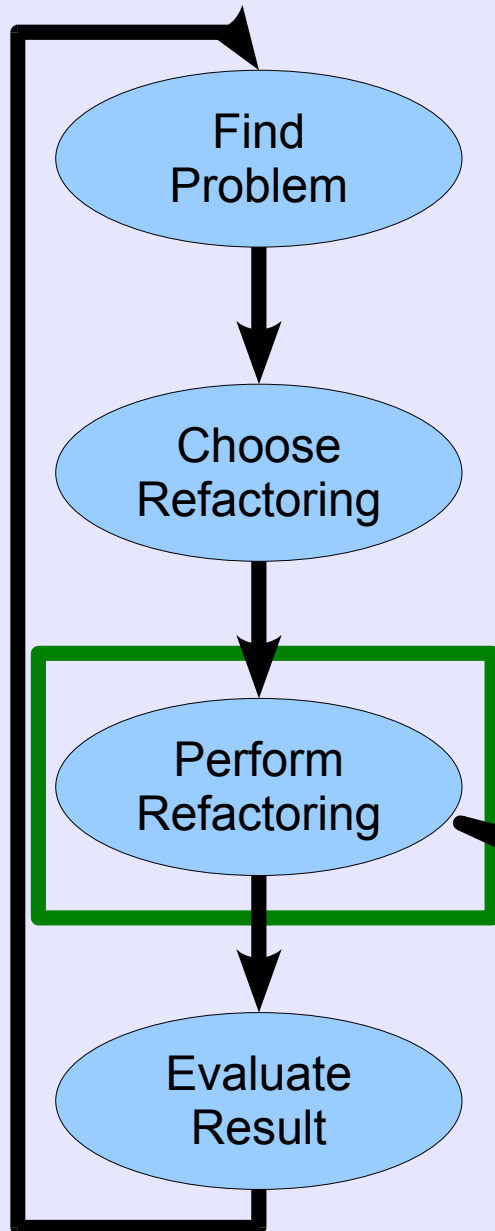


Eclipse's Extract Class is a misnomer.  
It only transfers fields, not methods.

Vote for fixing this bug!

[https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=312347](https://bugs.eclipse.org/bugs/show_bug.cgi?id=312347)

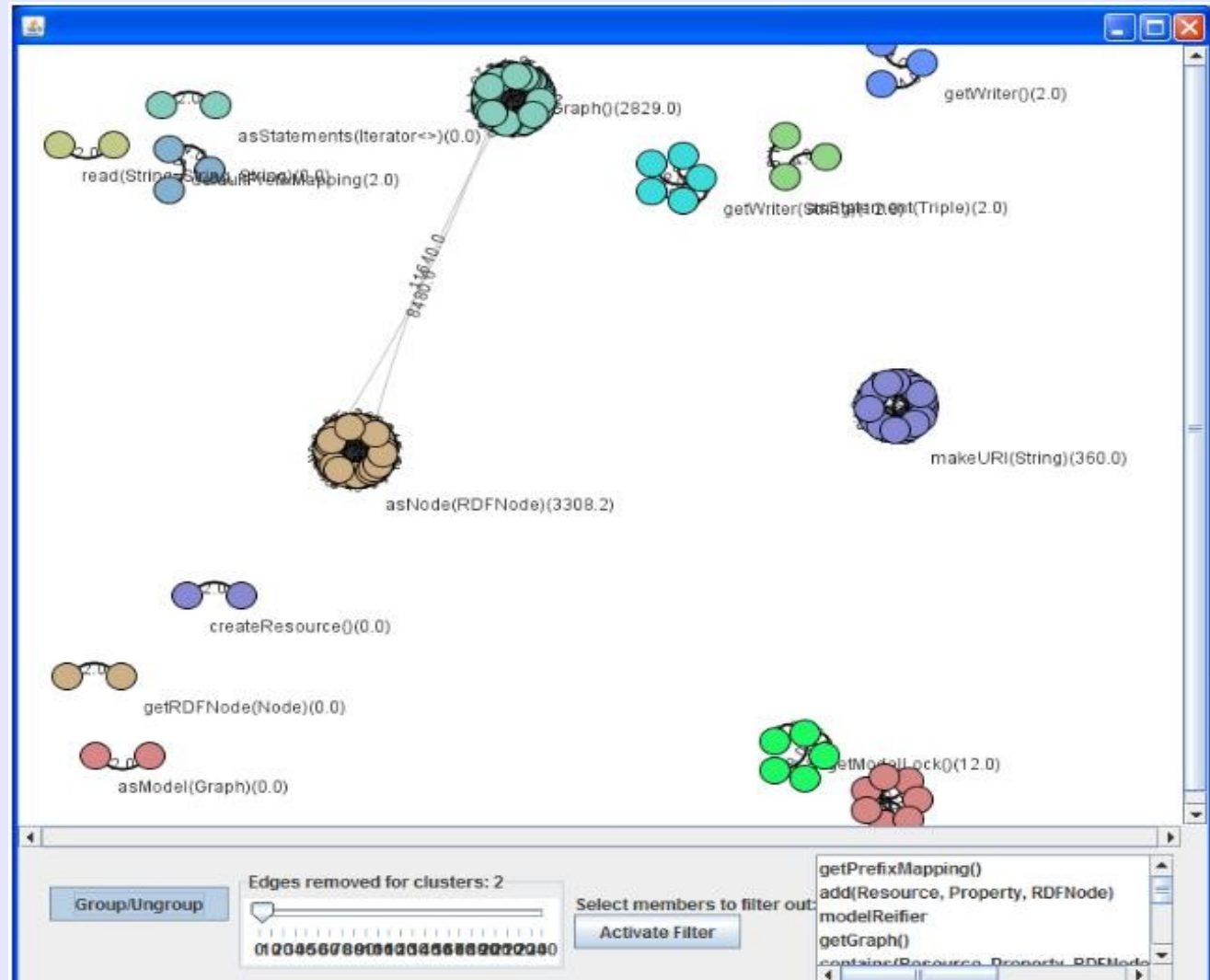
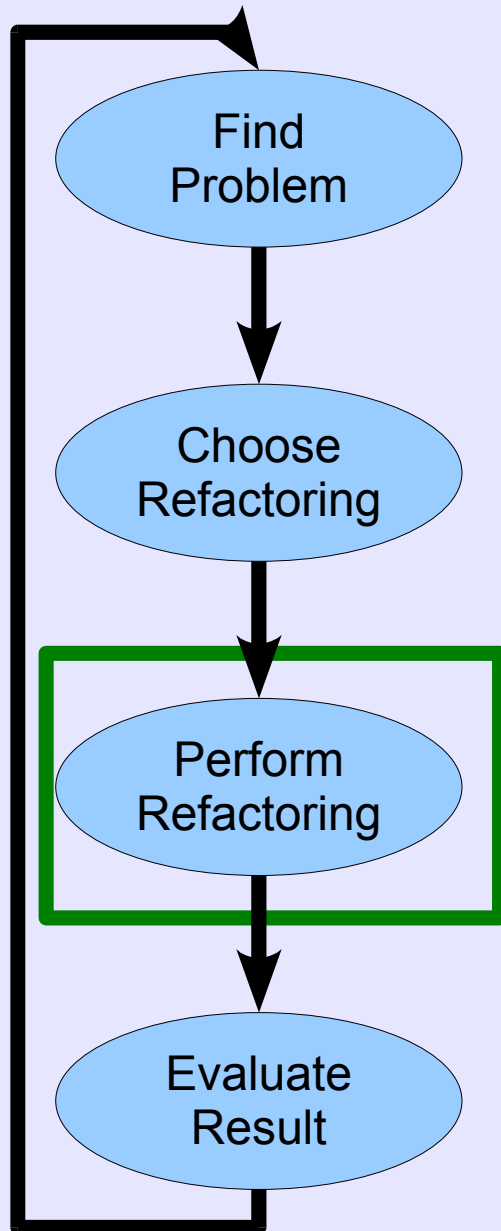
# Case Study – Extract Class



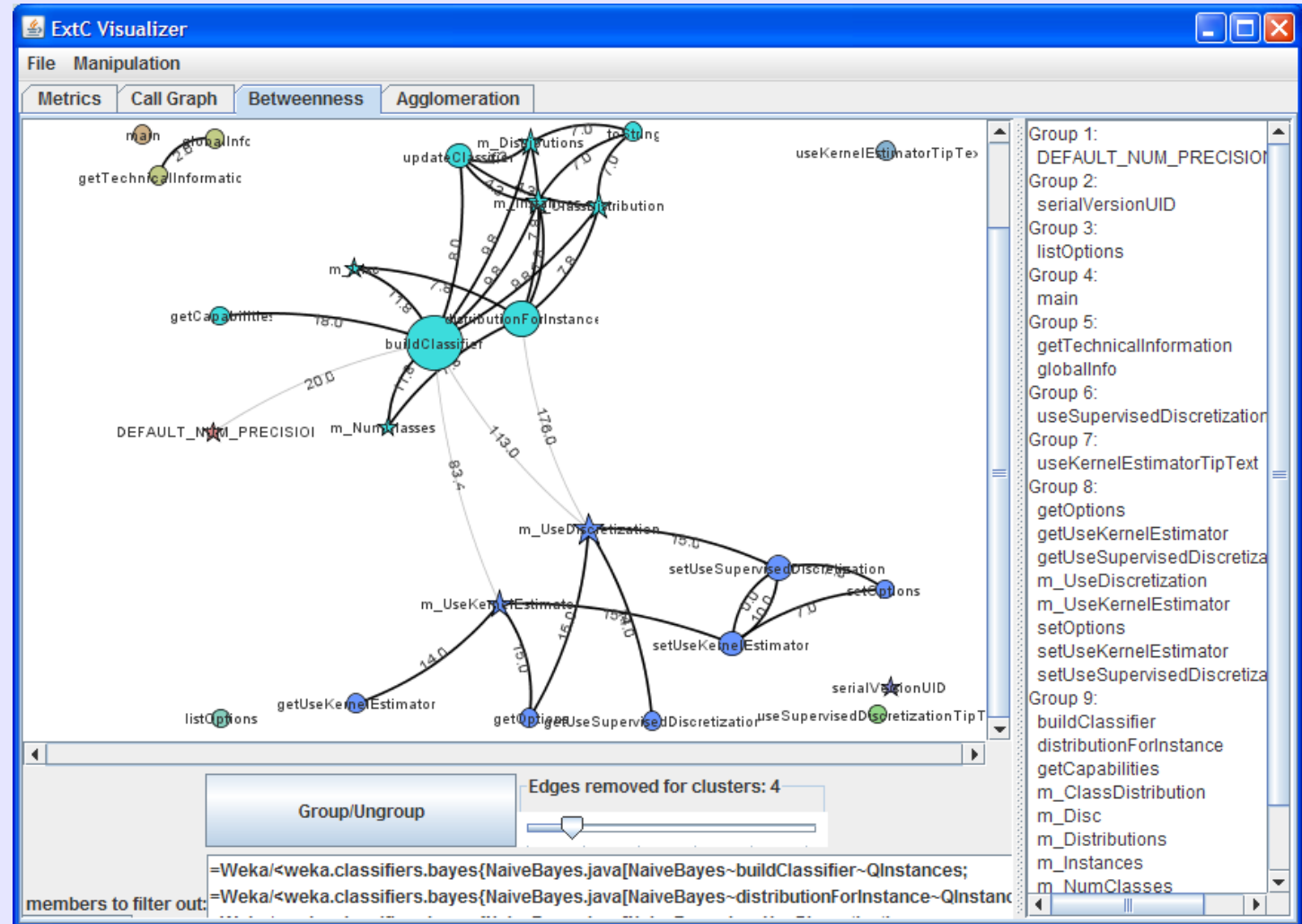
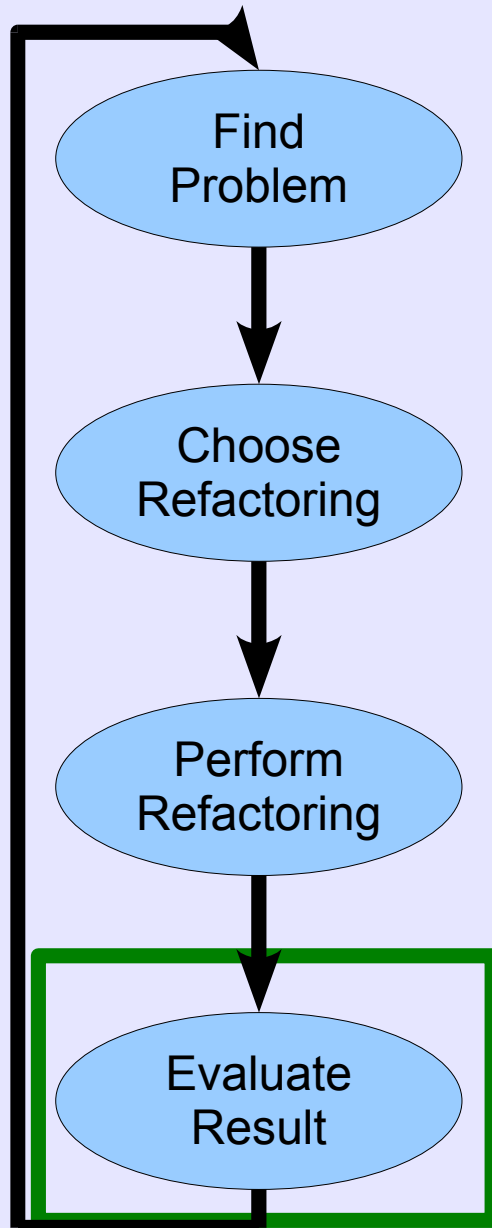
IntelliJ's Extract Class is better, but still buggy.



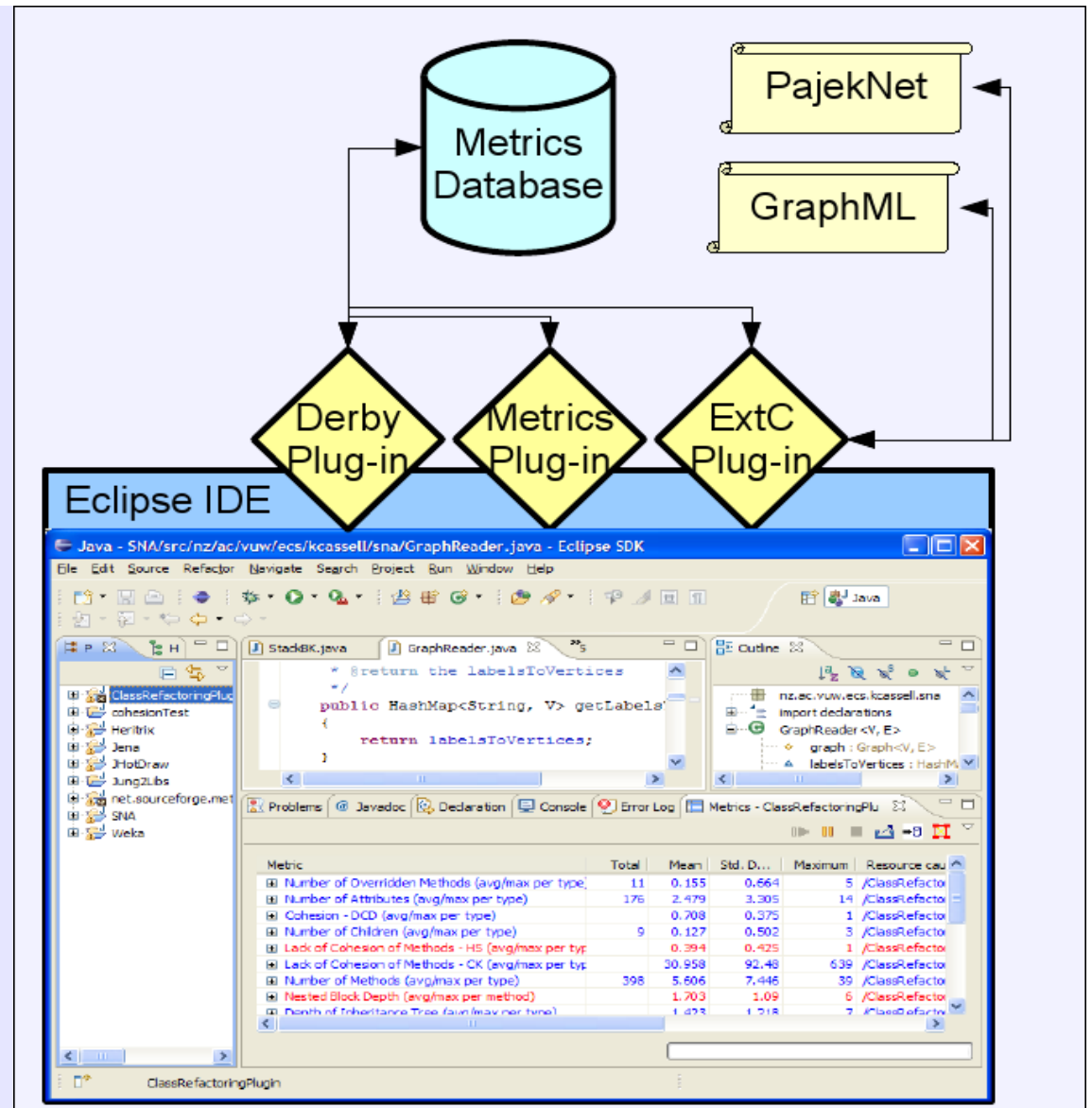
# Case Study – Extract Class



# Case Study – Extract Class



# My Eclipse Class Refactoring Environment



<http://code.google.com/p/ext-c>

<http://sourceforge.net/projects/metrics2/>

[http://db.apache.org/derby/integrate/derby\\_plugin.html](http://db.apache.org/derby/integrate/derby_plugin.html)

# Outline

- Introduction to refactoring
- The refactoring process
- State of the practice
- Case study – extract class
- Discussion and/or demos
- Beer

# Discussion Points

- “Refactor relentlessly” vs. “good enough” vs. being conservative.
- Floss refactoring vs. root canal refactoring
- What would entice you to refactor more?

Questions?

Answers?

Beer?