real-world refactoring

NEAL FORD software architect / meme wrangler

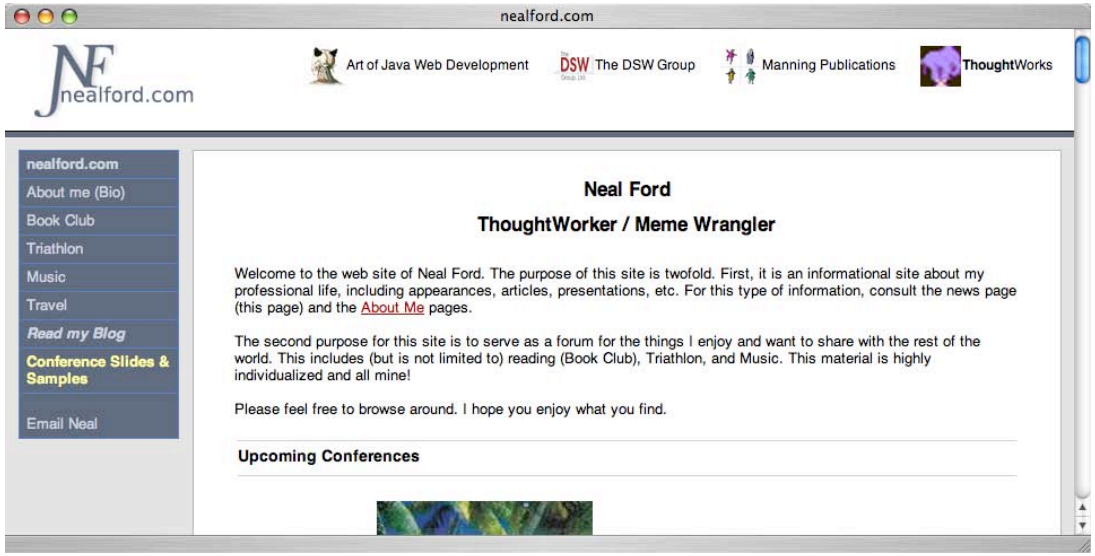
ThoughtWorks

nford@thoughtworks.com
3003 Summit Boulevard, Atlanta, GA 30319
www.nealford.com
www.thoughtworks.com
memeagora.blogspot.com

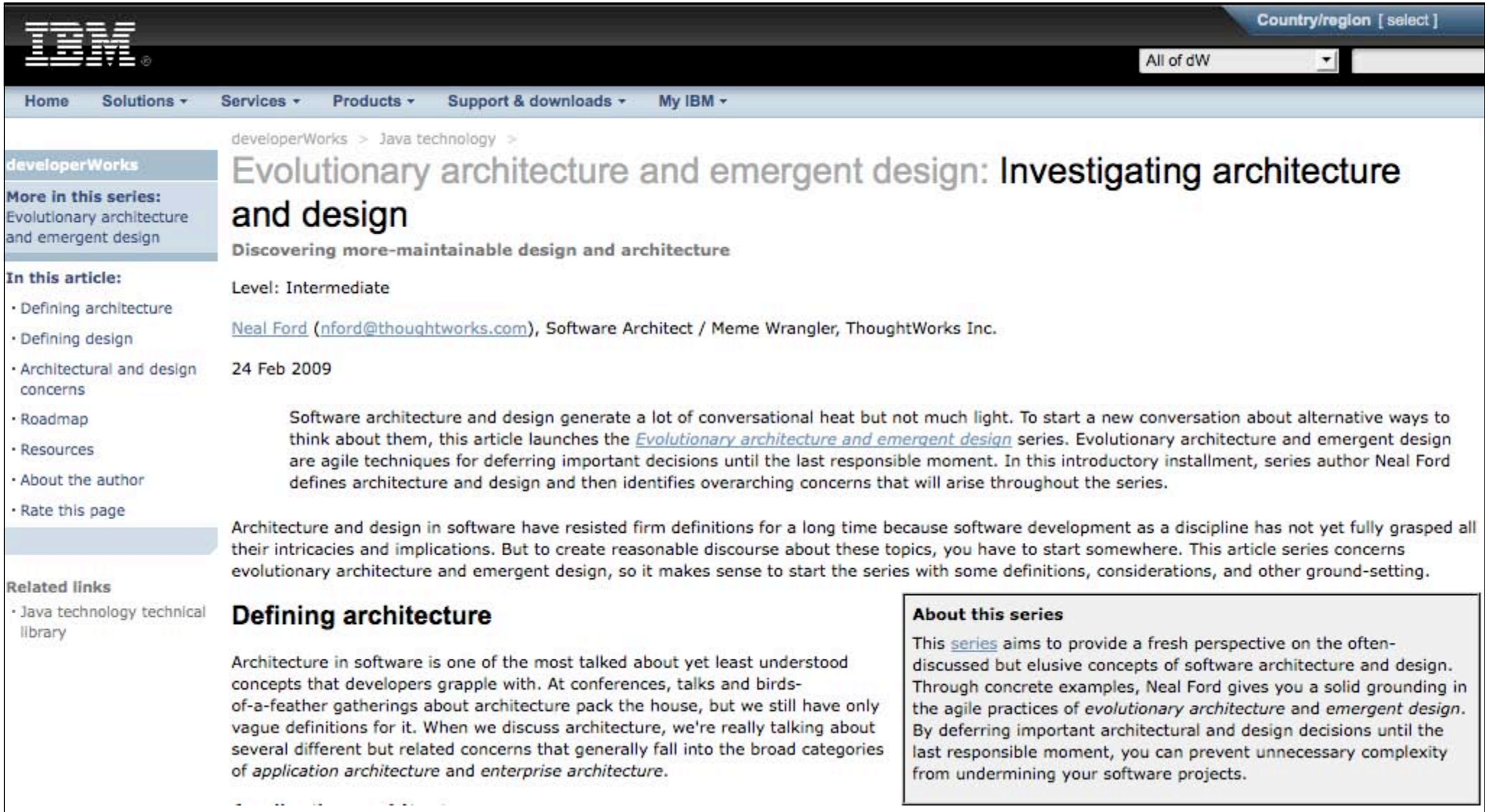
housekeeping

ask questions anytime

download slides from nealford.com



download samples from `github.com/nealford`



IBM

Country/region [select]

All of dW

Home Solutions Services Products Support & downloads My IBM

developerWorks > Java technology >

developerWorks

More in this series:
Evolutionary architecture and emergent design

In this article:

- Defining architecture
- Defining design
- Architectural and design concerns
- Roadmap
- Resources
- About the author
- Rate this page

Related links

- Java technology technical library

Evolutionary architecture and emergent design: Investigating architecture and design

Discovering more-maintainable design and architecture

Level: Intermediate

Neal Ford (nford@thoughtworks.com), Software Architect / Meme Wrangler, ThoughtWorks Inc.

24 Feb 2009

Software architecture and design generate a lot of conversational heat but not much light. To start a new conversation about alternative ways to think about them, this article launches the *Evolutionary architecture and emergent design* series. Evolutionary architecture and emergent design are agile techniques for deferring important decisions until the last responsible moment. In this introductory installment, series author Neal Ford defines architecture and design and then identifies overarching concerns that will arise throughout the series.

Architecture and design in software have resisted firm definitions for a long time because software development as a discipline has not yet fully grasped all their intricacies and implications. But to create reasonable discourse about these topics, you have to start somewhere. This article series concerns evolutionary architecture and emergent design, so it makes sense to start the series with some definitions, considerations, and other ground-setting.

Defining architecture

Architecture in software is one of the most talked about yet least understood concepts that developers grapple with. At conferences, talks and birds-of-a-feather gatherings about architecture pack the house, but we still have only vague definitions for it. When we discuss architecture, we're really talking about several different but related concerns that generally fall into the broad categories of *application architecture* and *enterprise architecture*.

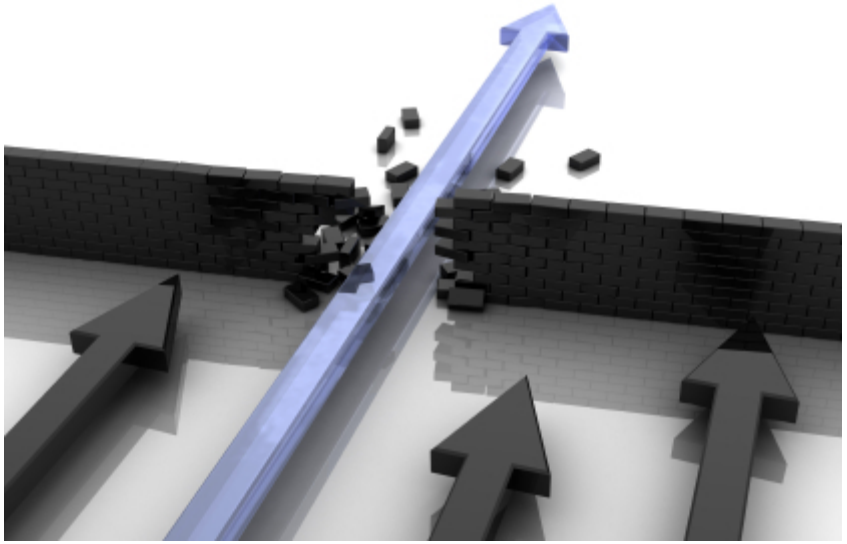
About this series

This series aims to provide a fresh perspective on the often-discussed but elusive concepts of software architecture and design. Through concrete examples, Neal Ford gives you a solid grounding in the agile practices of *evolutionary architecture* and *emergent design*. By deferring important architectural and design decisions until the last responsible moment, you can prevent unnecessary complexity from undermining your software projects.

http://www.ibm.com/developerworks/java/library/j-eaed1/index.html?S_TACT=105AGX02&S_CMP=EDU

http://tinyurl.com/nf-ead

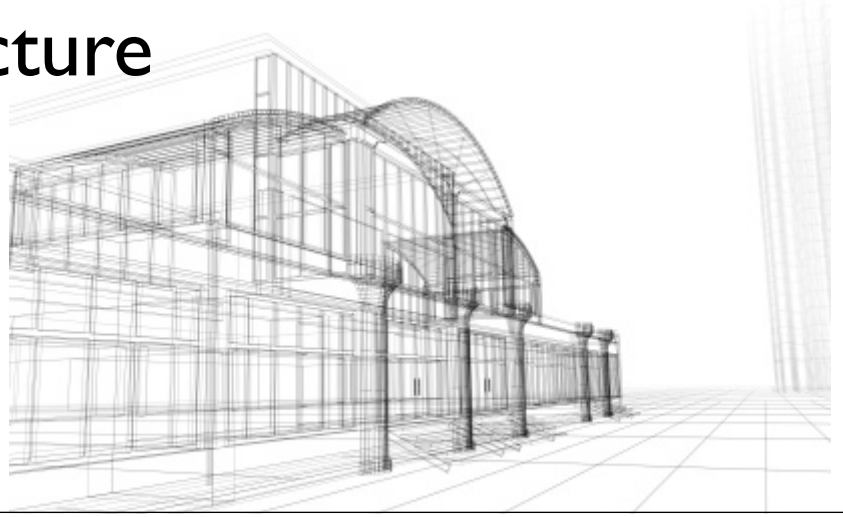
emergent design family



emergent design &
evolutionary architecture

real
world
refactoring

test-driven
design



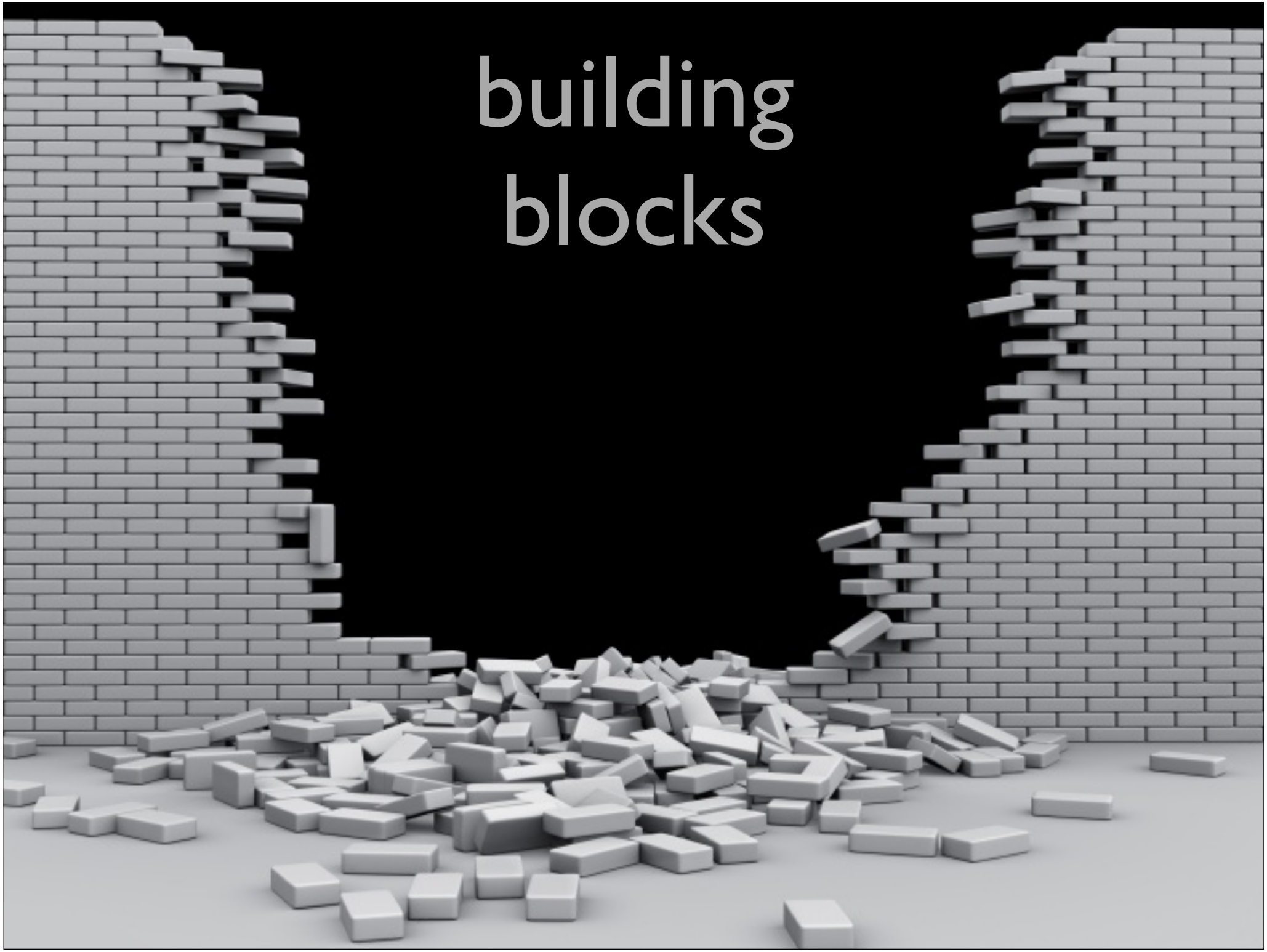
what i cover

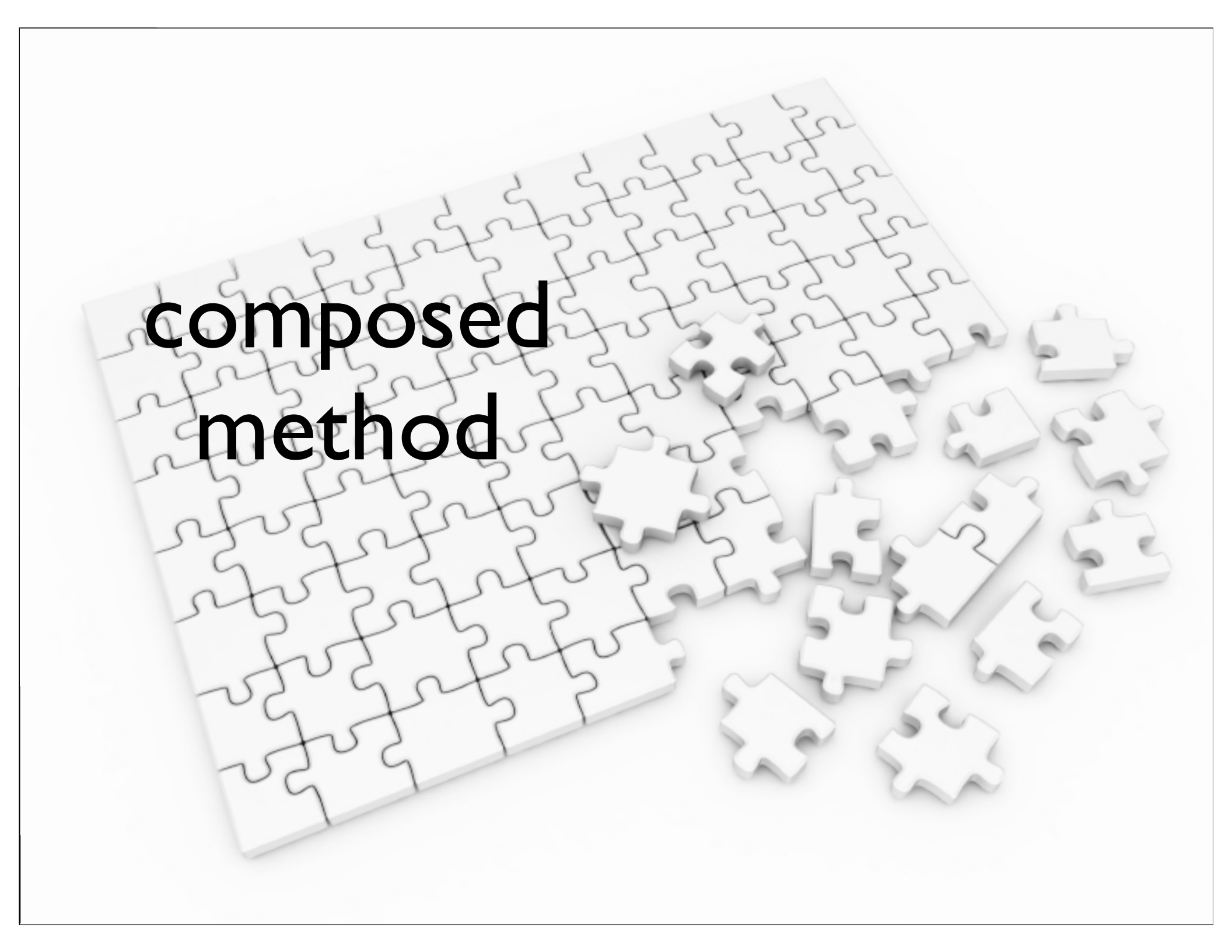
building blocks

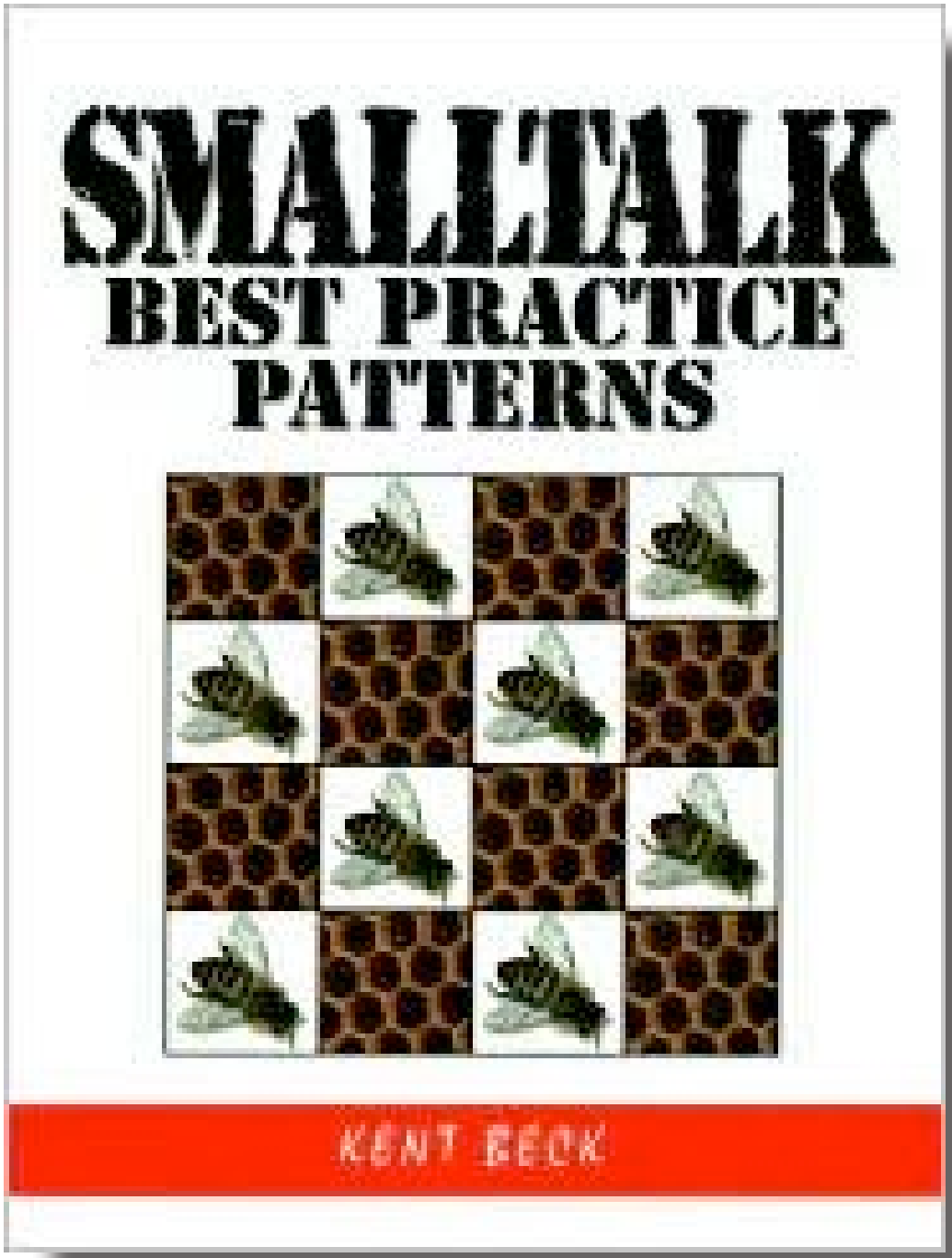
- composed method
- the slap principle

refactoring code, databases, and build files

advice on why & when to refactor


building
blocks


composed
method


SMALLTALK
BEST PRACTICE
PATTERNS
KENT BECK

composed method

Divide your program into methods that perform one identifiable task.

Keep all of the operations in a method at the same level of abstraction.

This will naturally result in programs with many small methods, each a few lines long.

refactoring to composed method

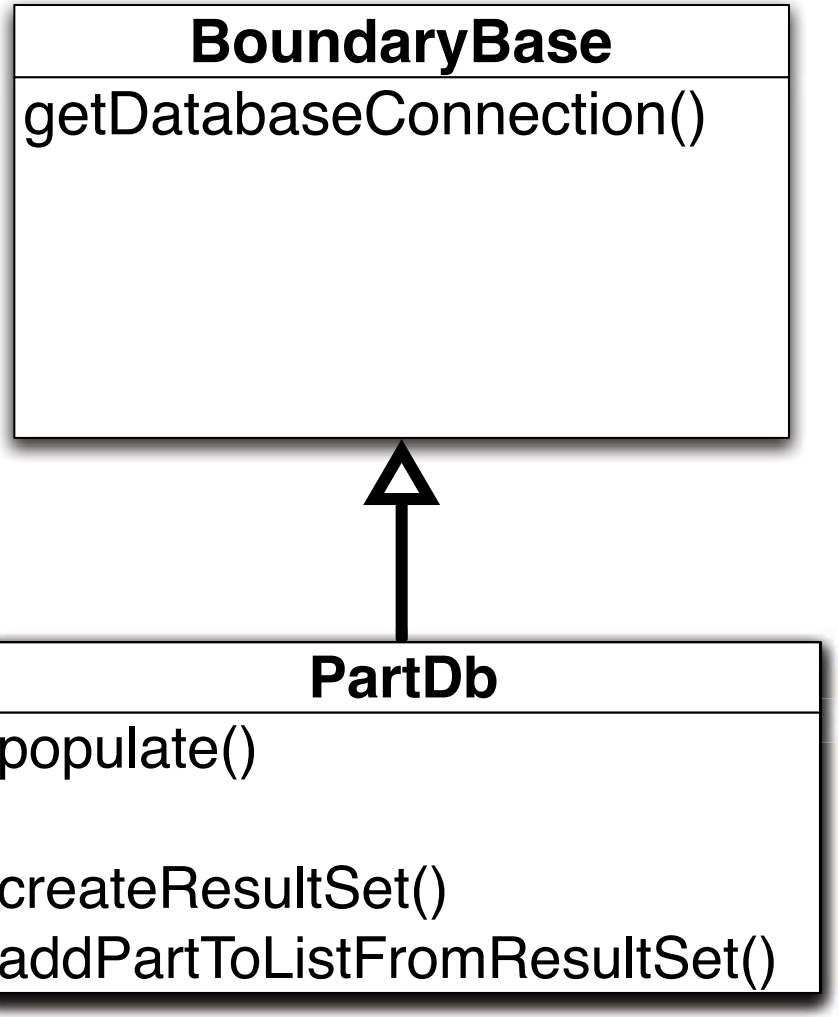
```
public void populate() throws Exception  {
    Connection c = null;
    try {
        Class.forName(DRIVER_CLASS);
        c = DriverManager.getConnection(DB_URL, USER, PASSWORD);
        Statement stmt = c.createStatement();
        ResultSet rs = stmt.executeQuery(SQL_SELECT_PARTS);
        while (rs.next()) {
            Part p = new Part();
            p.setName(rs.getString("name"));
            p.setBrand(rs.getString("brand"));
            p.setRetailPrice(rs.getDouble("retail_price"));
            partList.add(p);
        }
    } finally {
        c.close();
    }
}
```

```
private void addPartToListFromResultSet(ResultSet rs)
        throws SQLException {
    Part p = new Part();
    p.setName(rs.getString("name"));
    p.setBrand(rs.getString("brand"));
    p.setRetailPrice(rs.getDouble("retail_price"));
    partList.add(p);
}
```

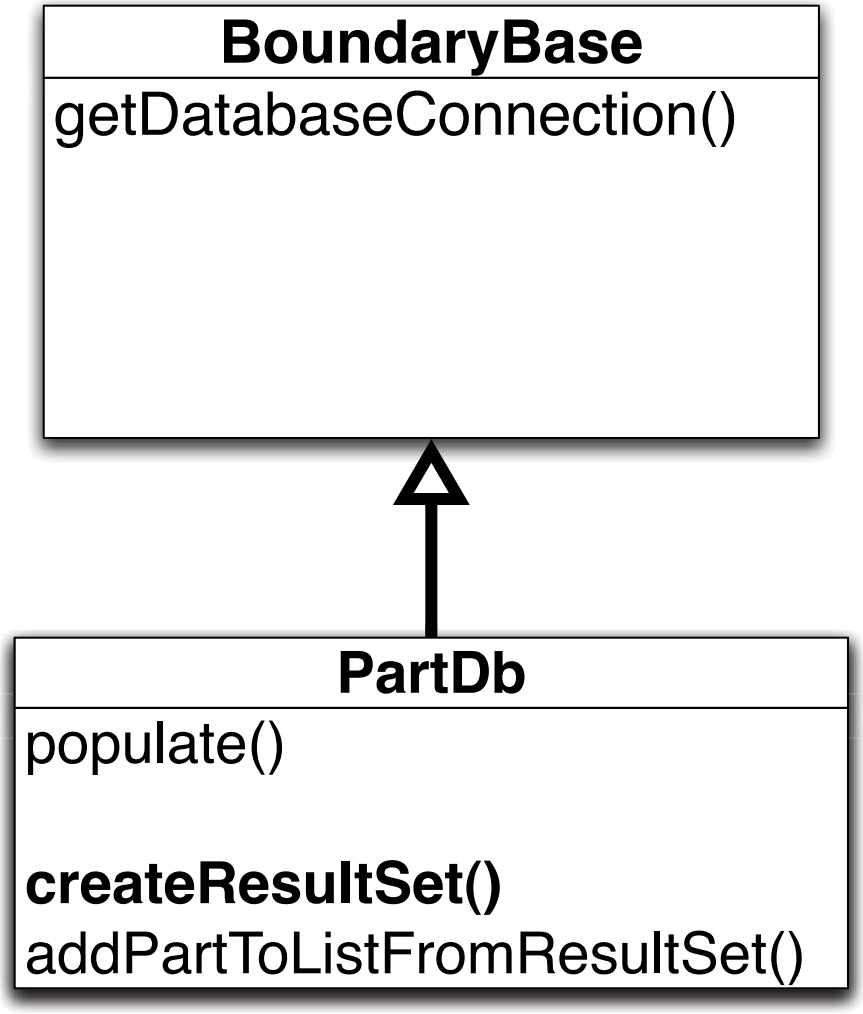
```
public void populate() throws Exception {
    Connection c = null;
    try {
        c = getDatabaseConnection();
        ResultSet rs = createResultSet(c);
        while (rs.next())
            addPartToListFromResultSet(rs);
    } finally {
        c.close();
    }
}
```

```
private ResultSet createResultSet(Connection c)
        throws SQLException {
    return c.createStatement().
            executeQuery(SQL_SELECT_PARTS);
}
```

```
private Connection getDatabaseConnection()
        throws ClassNotFoundException, SQLException {
    Connection c;
    Class.forName(DRIVER_CLASS);
    c = DriverManager.getConnection(DB_URL,
            "webuser", "webpass");
    return c;
}
```




```
private Connection getDatabaseConnection()
        throws ClassNotFoundException, SQLException {
    Connection c;
    Class.forName(DRIVER_CLASS);
    c = DriverManager.getConnection(DB_URL,
            "webuser", "webpass");
    return c;
}
```




```
private ResultSet createResultSet(Connection c)
        throws SQLException {
    return c.createStatement().
            executeQuery(SQL_SELECT_PARTS);
}
```

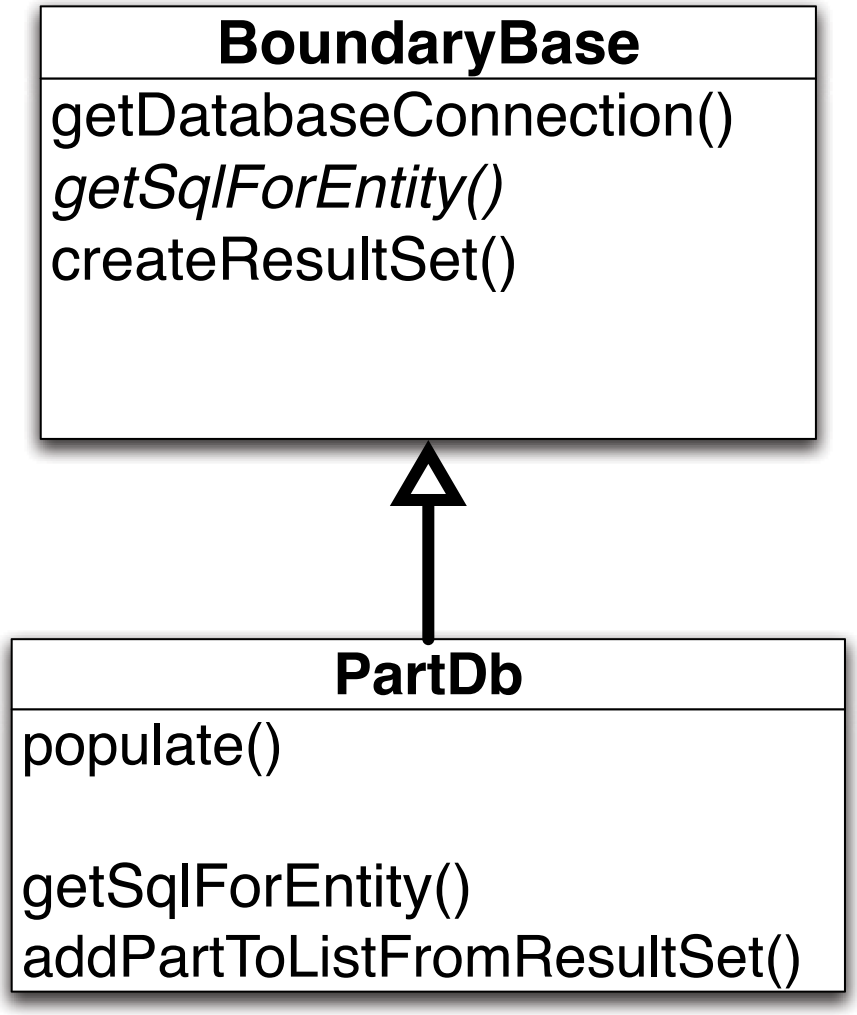
BoundaryBase

```
abstract protected String getSqlForEntity();

protected ResultSet createResultSet(Connection c) throws SQLException {
    Statement stmt = c.createStatement();
    return stmt.executeQuery(getSqlForEntity());
}
```

```
private ResultSet createResultSet(Connection c)
        throws SQLException {
    return c.createStatement().
            executeQuery(SQL_SELECT_PARTS);
}
```

PartDb

```
protected String getSqlForEntity() {
    return SQL_SELECT_PARTS;
}
```

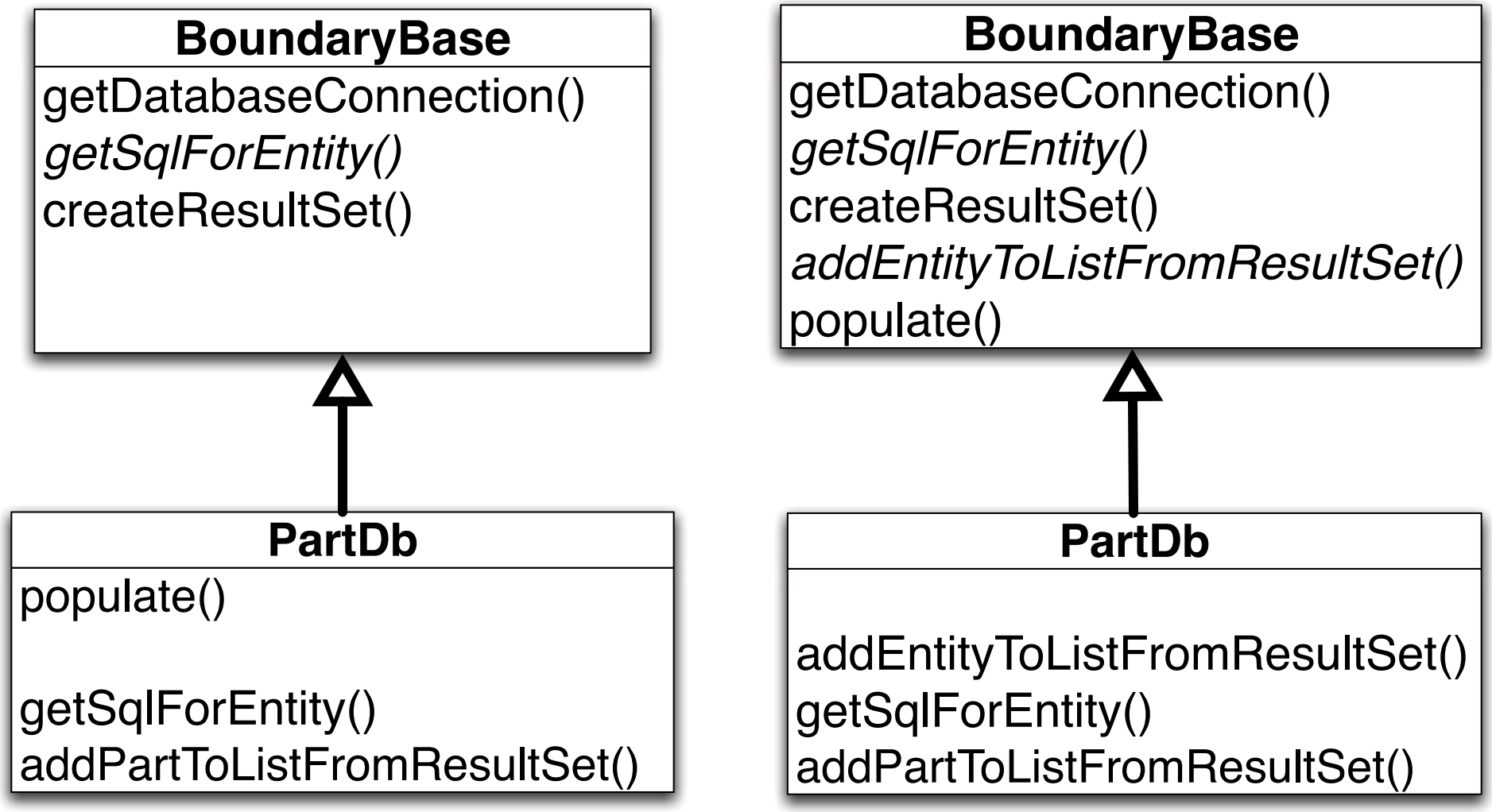

BoundaryBase
getDatabaseConnection()
getSqlForEntity()
createResultSet()
PartDb
populate()
getSqlForEntity()
addPartToListFromResultSet()

old version
(in PartDB)

```
public void populate() throws Exception {
    Connection c = null;
    try {
        c = getDatabaseConnection();
        ResultSet rs = createResultSet(c);
        while (rs.next())
            addPartToListFromResultSet(rs);
    } finally {
        c.close();
    }
}
```

```
abstract protected void addEntityToListFromResultSet(ResultSet rs)
        throws SQLException;

public void populate() throws Exception {
    Connection c = null;
    try {
        c = getDatabaseConnection();
        ResultSet rs = createResultSet(c);
        while (rs.next())
            addEntityToListFromResultSet(rs);
    } finally {
        c.close();
    }
}
```

refactored version
(in BoundaryBase)


BoundaryBase
getDatabaseConnection()
getSqlForEntity()
createResultSet()
PartDb
populate()
getSqlForEntity()
addPartToListFromResultSet()
BoundaryBase
getDatabaseConnection()
getSqlForEntity()
createResultSet()
addEntityToListFromResultSet()
populate()
PartDb
addEntityToListFromResultSet()
getSqlForEntity()
addPartToListFromResultSet()

```
protected Connection getDatabaseConnection() throws ClassNotFoundException,
        SQLException {
    Connection c;
    Class.forName(DRIVER_CLASS);
    c = DriverManager.getConnection(DB_URL, "webuser", "webpass");
    return c;
}

abstract protected String getSqlForEntity();

protected ResultSet createResultSet(Connection c) throws SQLException {
    Statement stmt = c.createStatement();
    return stmt.executeQuery(getSqlForEntity());
}

abstract protected void addEntityToListFromResultSet(ResultSet rs)
        throws SQLException;

public void populate() throws Exception {
    Connection c = null;
    try {
        c = getDatabaseConnection();
        ResultSet rs = createResultSet(c);
        while (rs.next())
            addEntityToListFromResultSet(rs);
    } finally {
        c.close();
    }
}
```

BoundaryBase

PartDb

```
public Part[] getParts() {
    return (Part[]) partList.toArray(TEMPLATE);
}

protected String getSqlForEntity() {
    return SQL_SELECT_PARTS;
}

protected void addEntityToListFromResultSet(ResultSet rs) throws SQLException {
    Part p = new Part();
    p.setName(rs.getString("name"));
    p.setBrand(rs.getString("brand"));
    p.setRetailPrice(rs.getDouble("retail_price"));
    partList.add(p);
}
```

benefits of composed method

shorter methods easier to test

method names become documentation

large number of very cohesive methods

discover reusable assets that you didn't know were there

SLAP

single level of abstraction principle

composed method

Divide your program into methods that perform one identifiable task.

Keep all of the operations in a method at the same level of abstraction.

This will naturally result in programs with many small methods, each a few lines long.

```
public void addOrder(final ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    Connection c = null; PreparedStatement ps = null;
    Statement s = null; ResultSet rs = null;
    boolean transactionState = false;
    try {
        c = dbPool.getConnection();
        s = c.createStatement();
        transactionState = c.getAutoCommit();
        int userKey = getUserKey(userName, c, ps, rs);
        c.setAutoCommit(false);
        addSingleOrder(order, c, ps, userKey);
        int orderKey = getOrderKey(s, rs);
        addLineItems(cart, c, orderKey);
        c.commit();
        order.setOrderKey(orderKey);
    } catch (SQLException sqlx) {
        s = c.createStatement();
        c.rollback();
        throw sqlx;
    } finally {
        try {
            c.setAutoCommit(transactionState);
            dbPool.release(c);
            if (s != null)  s.close();
            if (ps != null) ps.close();
            if (rs != null) rs.close();
        } catch (SQLException ignored) {
        }
    }
}
```

```
public void addOrder(final ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    Connection connection = null; PreparedStatement ps = null;
    Statement statement = null; ResultSet rs = null;
    boolean transactionState = false;
    try {
        connection = dbPool.getConnection();
        statement = connection.createStatement();
        transactionState = setupTransactionStateFor(connection, transactionState);
        addSingleOrder(order, connection, ps, userKeyFor(userName, connection));
        order.setOrderKey(generateOrderKey(statement, rs));
        addLineItems(cart, connection, order.getOrderKey());
        completeTransaction(connection);
    } catch (SQLException sqlx) {
        rollbackTransactionFor(connection);
        throw sqlx;
    } finally {
        cleanUpDatabaseResources(connection, transactionState, statement, ps, rs);
    }
}
```

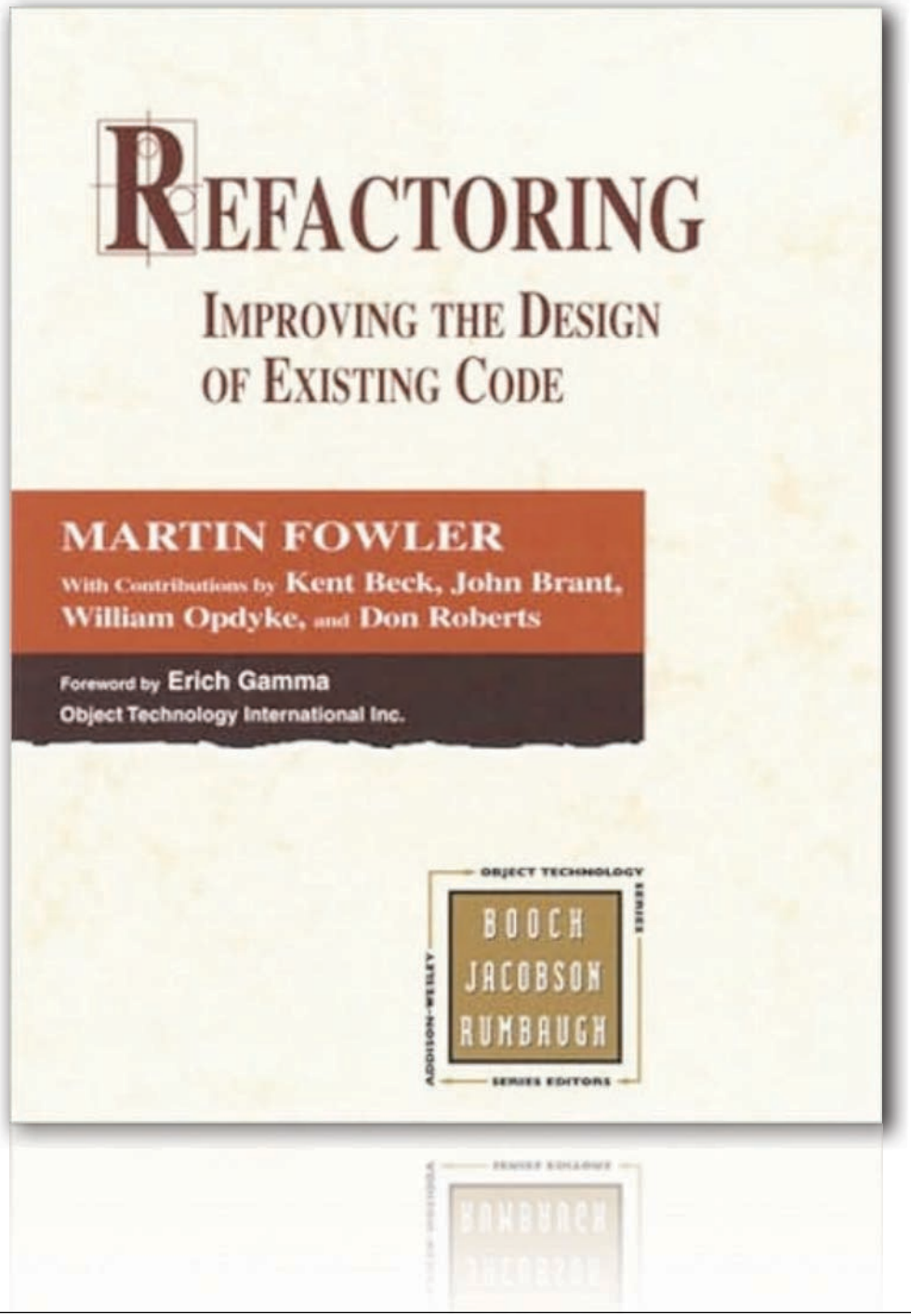
```
public void addOrderFrom(ShoppingCart cart, String userName,
                         Order order) throws SQLException {
    setupDataInfrastructure();
    try {
        add(order, userKeyBasedOn(userName));
        addLineItemsFrom(cart, order.getOrderKey());
    } finally {
        cleanUp();
    }
}
```

```
private void completeTransaction() throws SQLException {
    ((Connection)_db.get("connection")).commit();
}
```

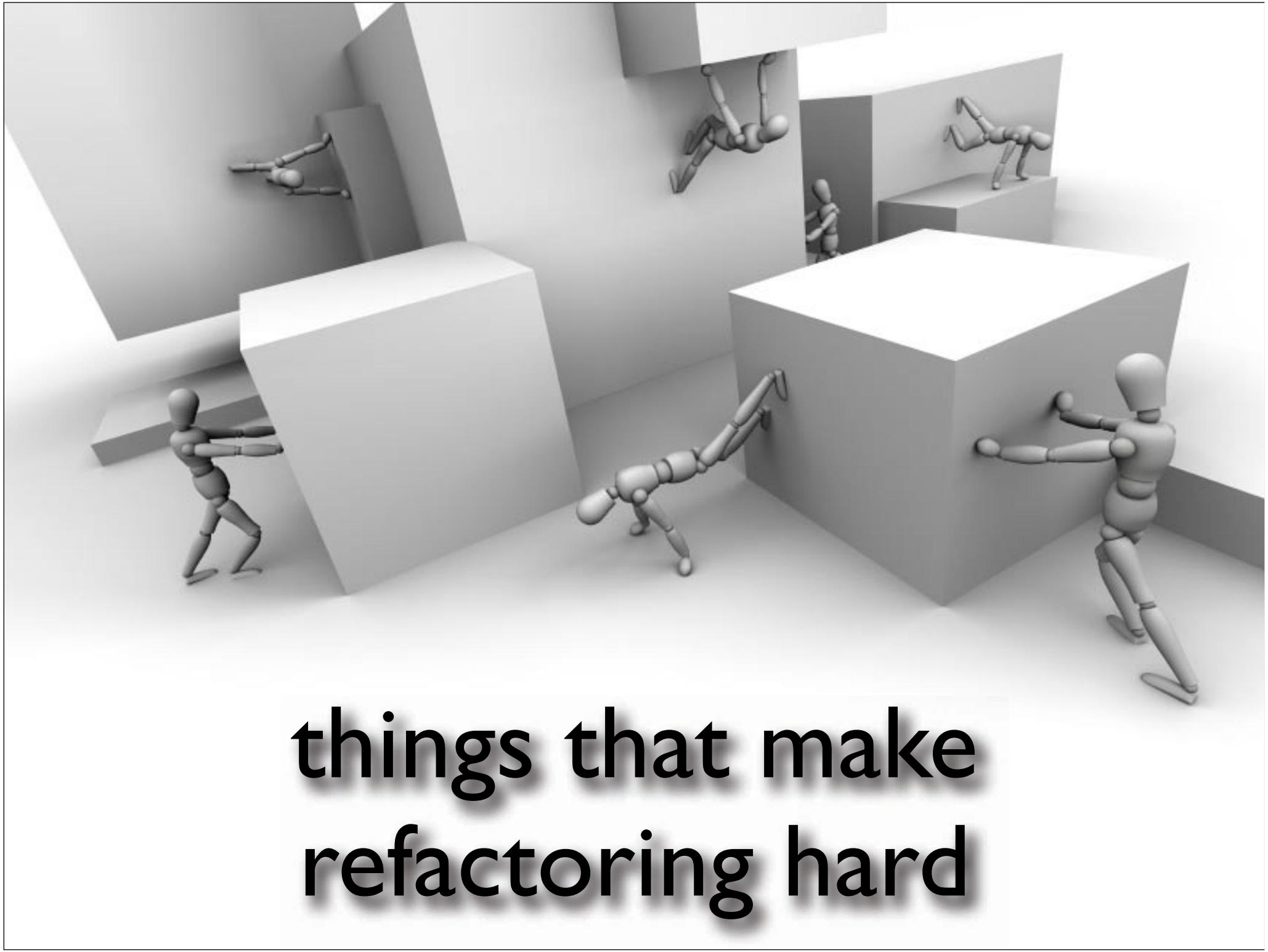
```
public void addOrder(ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    Connection c = null;
    PreparedStatement ps = null;
    Statement s = null;
    ResultSet rs = null;
    boolean transactionState = false;
    try {
        s = c.createStatement();
        transactionState = c.getAutoCommit();
        int userKey = getUserKey(userName, c, ps, rs);
        c.setAutoCommit(false);
        addSingleOrder(order, c, ps, userKey);
        int orderKey = getOrderKey(s, rs);
        addLineItems(cart, c, orderKey);
        c.commit();
        order.setOrderKeyFrom(orderKey);
    } catch (SQLException sqlx) {
        s = c.createStatement();
        c.rollback();
        throw sqlx;
    } finally {
        try {
            c.setAutoCommit(transactior
            dbPool.release(c);
            if (s != null)
                s.close();
            if (ps != null)
                ps.close();
            if (rs != null)
                rs.close();
        } catch (SQLException ignored)
        }
    }
}
```

```
public void addOrderFrom(ShoppingCart cart, String userName,
                         Order order) throws SQLException {
    setupDataInfrastructure();
    try {
        add(order, userKeyBasedOn(userName));
        addLineItemsFrom(cart, order.getOrderKey());
        completeTransaction();
    } catch (SQLException sqlx) {
        rollbackTransaction();
        throw sqlx;
    } finally {
        cleanUp();
    }
}
```

code



Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.



things that make refactoring hard


fat, ugly methods

the problem:

aging code base

no tests

lots of bugs

strong desire to refactor...

...plus gut-wrenching fear

lots of long untested methods

you want to refactor them but...

...you're afraid of what will happen

1. refactor to composed method

2. write tests for the (now) smaller methods

3. write tests when debugging or adding new features

how to I get started writing unit tests?

break long methods into small pieces using composed method

draw a line in the sand: starting next thursday, our code coverage will always rise

every time you find a bug, write a test

write a test for every new feature

you will slowly grow tests around the part of you code that needs it the most

```
public void evaluateExtraParams() {
    super.evaluateExtraParams();

    //Object doubleName = null;

    if (emptyOption != null) {
        addParameter("emptyOption", findValue(emptyOption, Boolean.class));
    }

    if (multiple != null) {
        addParameter("multiple", findValue(multiple, Boolean.class));
    }

    if (size != null) {
        addParameter("size", findString(size));
    }

    if ((headerKey != null) && (headerValue != null)) {
        addParameter("headerKey", findString(headerKey));
        addParameter("headerValue", findString(headerValue));
    }


    if (doubleMultiple != null) {
        addParameter("doubleMultiple", findValue(doubleMultiple, Boolean.class));
    }

    if (doubleSize != null) {
```

from Struts 2.0.11,

`evaluateExtraParms()`, 178 lines

DRY violations

copy & paste

cpd

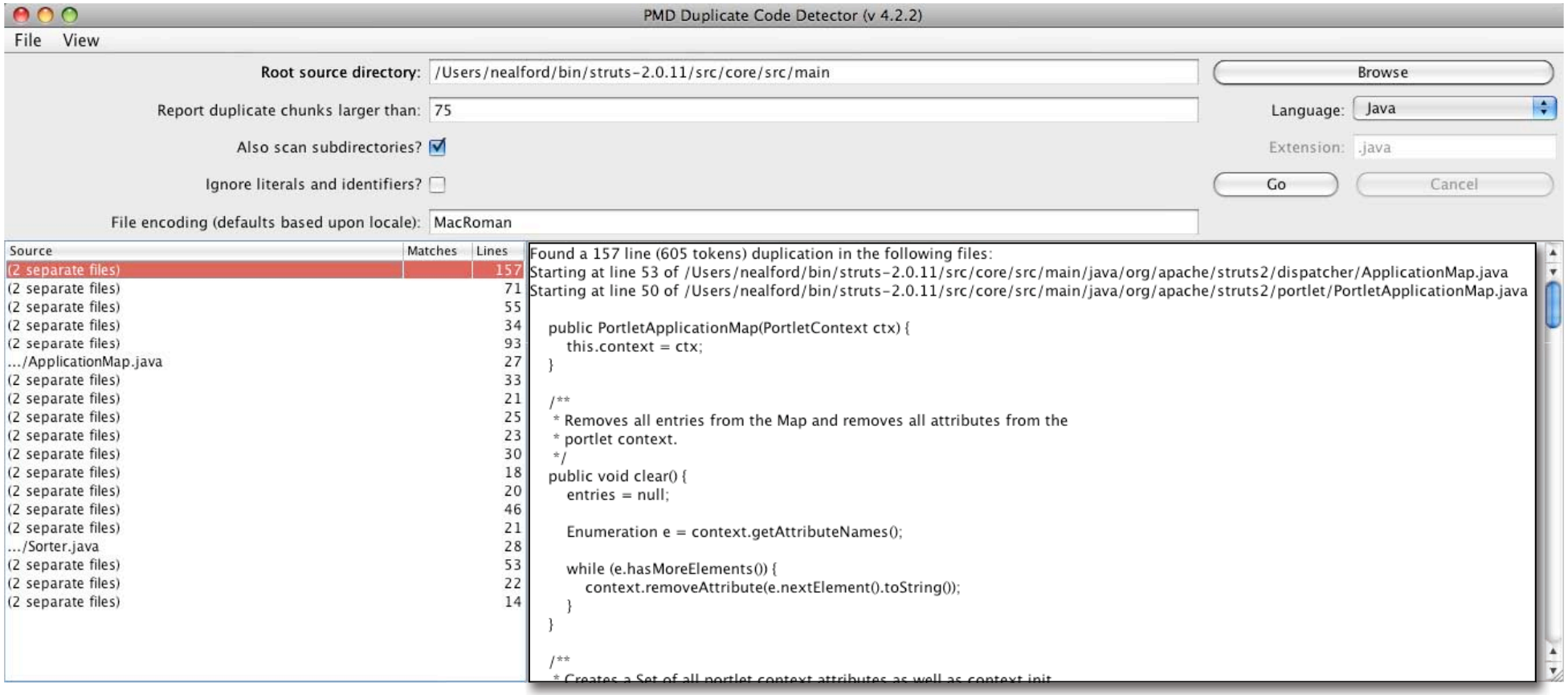
part of the source-code analysis tool **pmd**

configurable window of number of duplicate tokens

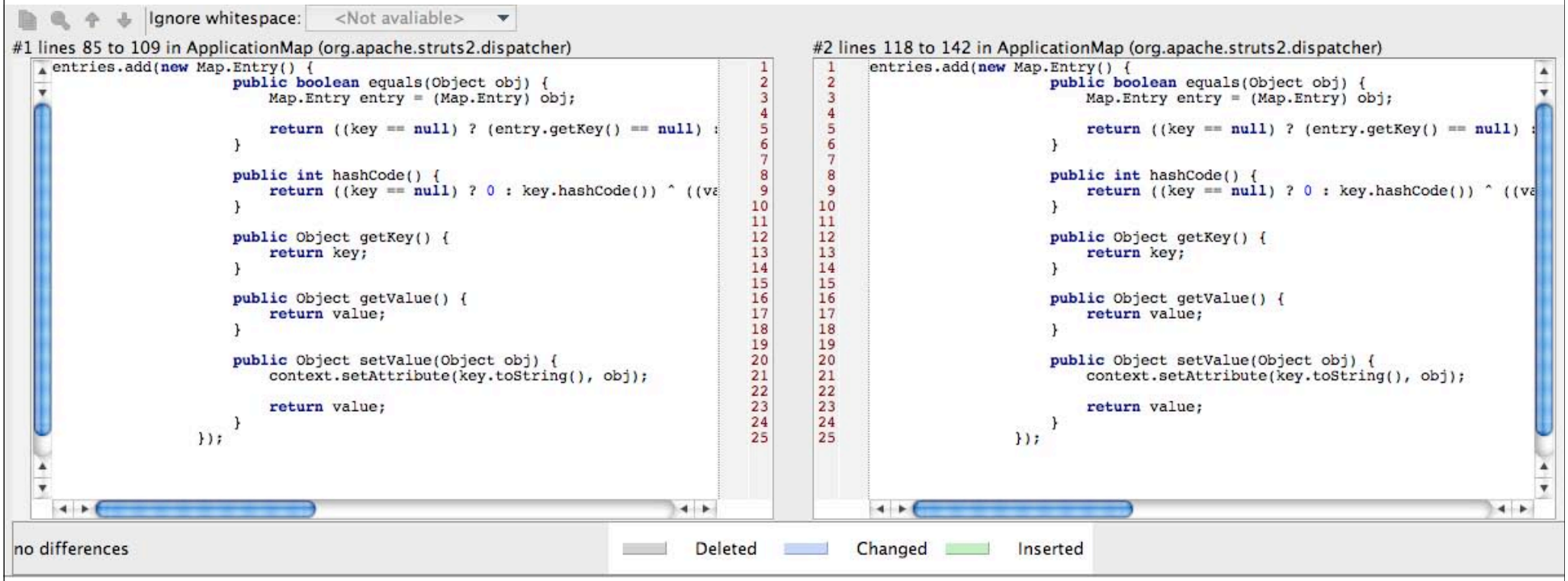
pre-configured with several languages

easy to add new language support

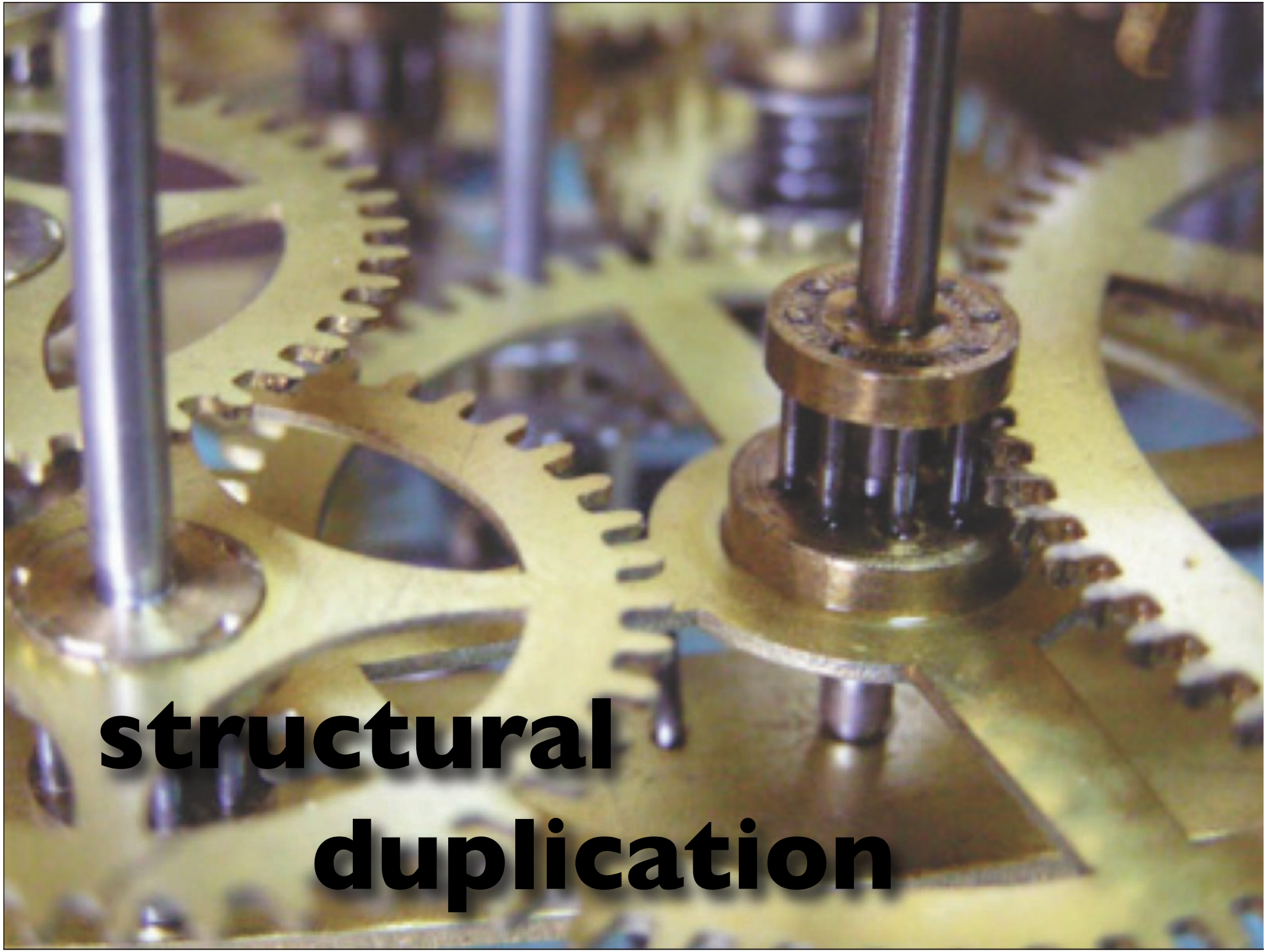
also simian (commercial)


PMD Duplicate Code Detector (v 4.2.2)
File View
Root source directory: /Users/nealford/bin/struts-2.0.11/src/core/src/main
Browse
Report duplicate chunks larger than: 75
Language: Java
Also scan subdirectories?
Extension: .java
Ignore literals and identifiers?
Go
Cancel
File encoding (defaults based upon locale): MacRoman
Source
Matches
Lines
(2 separate files) 157
(2 separate files) 71
(2 separate files) 55
(2 separate files) 34
(2 separate files) 93
.../ApplicationMap.java 27
(2 separate files) 33
(2 separate files) 21
(2 separate files) 25
(2 separate files) 23
(2 separate files) 30
(2 separate files) 18
(2 separate files) 20
(2 separate files) 46
(2 separate files) 21
.../Sorter.java 28
(2 separate files) 53
(2 separate files) 22
(2 separate files) 14
Found a 157 line (605 tokens) duplication in the following files:
Starting at line 53 of /Users/nealford/bin/struts-2.0.11/src/core/src/main/java/org/apache/struts2/dispatcher/ApplicationMap.java
Starting at line 50 of /Users/nealford/bin/struts-2.0.11/src/core/src/main/java/org/apache/struts2/portlet/PortletApplicationMap.java
public PortletApplicationMap(PortletContext ctx) {
this.context = ctx;
}
/**
* Removes all entries from the Map and removes all attributes from the
* portlet context.
*/
public void clear() {
entries = null;
Enumeration e = context.getAttributeNames();
while (e.hasMoreElements()) {
context.removeAttribute(e.nextElement().toString());
}
}
/**
* Creates a Set of all portlet context attributes as well as context init

intellij's duplication view



```
while (enumeration.hasMoreElements()) {
    final String key = enumeration.nextElement().toString();
    final Object value = context.getInitParameter(key);
    entries.add(new Map.Entry() {
        public boolean equals(Object obj) {
            Map.Entry entry = (Map.Entry) obj;

            return ((key == null) ?
                (entry.getKey() == null) :
                key.equals(entry.getKey())) && ((value == null) ?
                    (entry.getValue() == null) :
                    value.equals(entry.getValue()));
        }

        public int hashCode() {
            return ((key == null) ? 0 : key.hashCode()) ^ ((value == null) ? 0 : value.hashCode());
        }

        public Object getKey() {
            return key;
        }

        public Object getValue() {
            return value;
        }

        public Object setValue(Object obj) {
            context.setAttribute(key.toString(), obj);

            return value;
        }
    });
}
```


structural
duplication

given:

```
public class Employee {
    private String name;
    private int salary;
    private int hireYear;

    public Employee(String name, int salary, int hireYear) {
        this.name = name;
        this.salary = salary;
        this.hireYear = hireYear;
    }

    public String getName() { return name; }
    public int getSalary() { return salary;}
    public int getHireYear() { return hireYear; }
}
```

goal:
sort on
any property

comparator mania!

```
public class EmployeeNameComparator implements Comparator<Employee> {
    public int compare(Employee emp1, Employee emp2) {
        return emp1.getName().compareTo(emp2.getName());
    }
}
```

```
public class EmployeeSalyComparator implements Comparator<Employee> {
    public int compare(Employee emp1, Employee emp2) {
        return emp1.getSalary() - emp2.getSalary();
    }
}
```

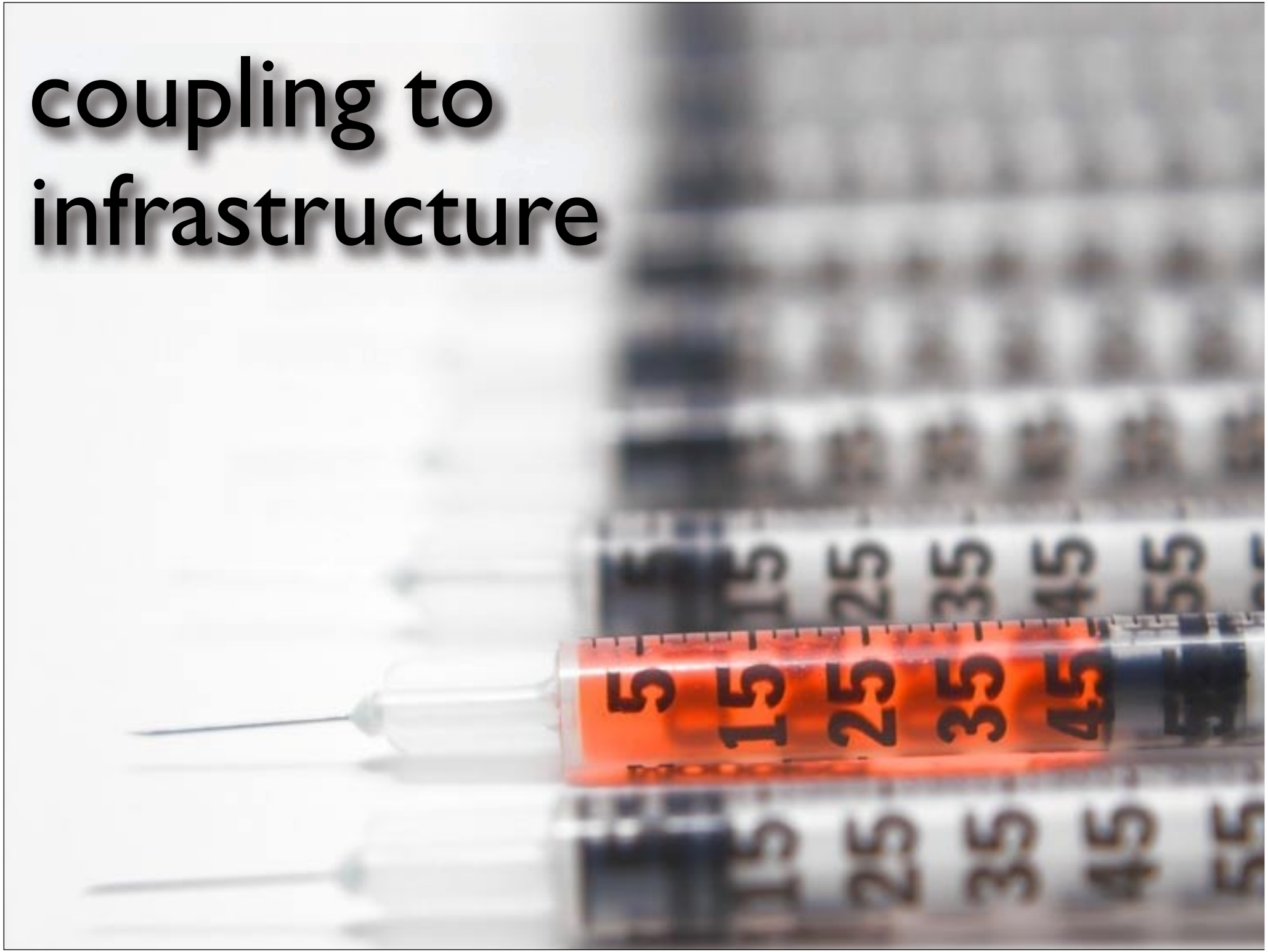
same whitespace, different values

```
public class EmployeeSorter {

    public void sort(List<DryEmployee> employees, String criteria) {
        Collections.sort(employees, getComparatorFor(criteria));
    }

    private Method getSelectionCriteriaMethod(String methodName) {
        Method m;
        methodName = "get" + methodName.substring(0, 1).toUpperCase() +
                methodName.substring(1);
        try {
            m = DryEmployee.class.getMethod(methodName);
        } catch (NoSuchMethodException e) {
            throw new RuntimeException(e.getMessage());
        }
        return m;
    }

    public Comparator<DryEmployee> getComparatorFor(final String field) {
        return new Comparator<DryEmployee>() {
            public int compare(DryEmployee o1, DryEmployee o2) {
                Object field1, field2;
                Method method = getSelectionCriteriaMethod(field);
                try {
                    field1 = method.invoke(o1);
                    field2 = method.invoke(o2);
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
                return ((Comparable) field1).compareTo(field2);
            }
        };
    }
}
```

```
@Before public void setup() {
    _sorter = new EmployeeSorter();
    _list = new ArrayList<DryEmployee>();
    _list.add(new DryEmployee("Homer", 20000, 1975));
    _list.add(new DryEmployee("Smithers", 150000, 1980));
    _list.add(new DryEmployee("Lenny", 100000, 1982));
}

@Test public void name_comparisons() {
    _sorter.sort(_list, "name");
    assertThat(_list.get(0).getName(), is("Homer"));
    assertThat(_list.get(1).getName(), is("Lenny"));
    assertThat(_list.get(2).getName(), is("Smithers"));
}

@Test public void salary_comparisons() {
    _sorter.sort(_list, "salary");
    assertThat(_list.get(0).getSalary(), is(20000));
    assertThat(_list.get(1).getSalary(), is(100000));
    assertThat(_list.get(2).getSalary(), is(150000));
}

@Test public void hireYearComparison() {
    _sorter.sort(_list, "hireYear");
    assertThat(_list.get(0).getHireYear(), is(1975));
    assertThat(_list.get(1).getHireYear(), is(1980));
    assertThat(_list.get(2).getHireYear(), is(1982));
}
```

architectural & design smells

coupling to infrastructure



coupling to infrastructure

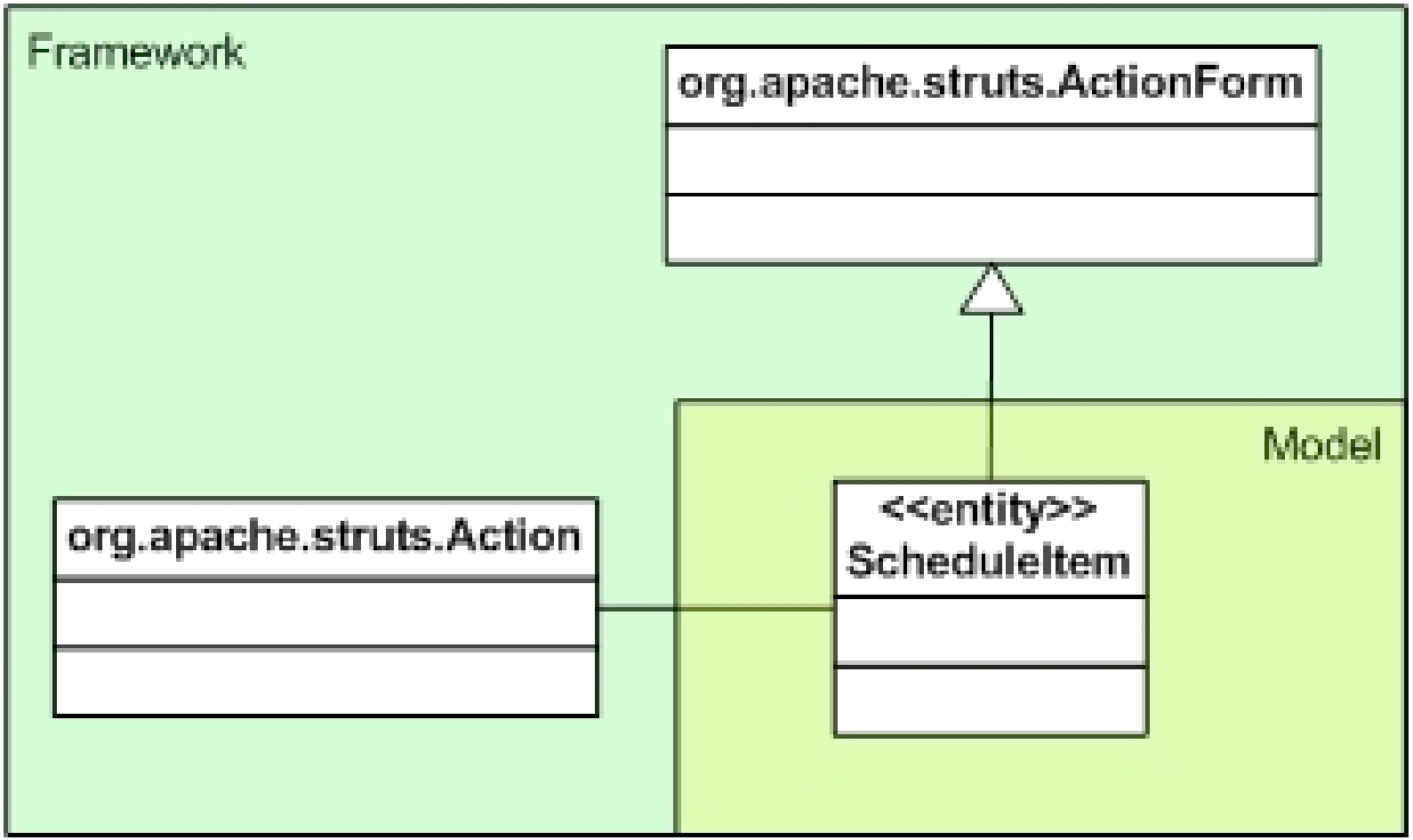
```
import com.giantvendor.seductiveclasses.*
```

pay attention to dependencies

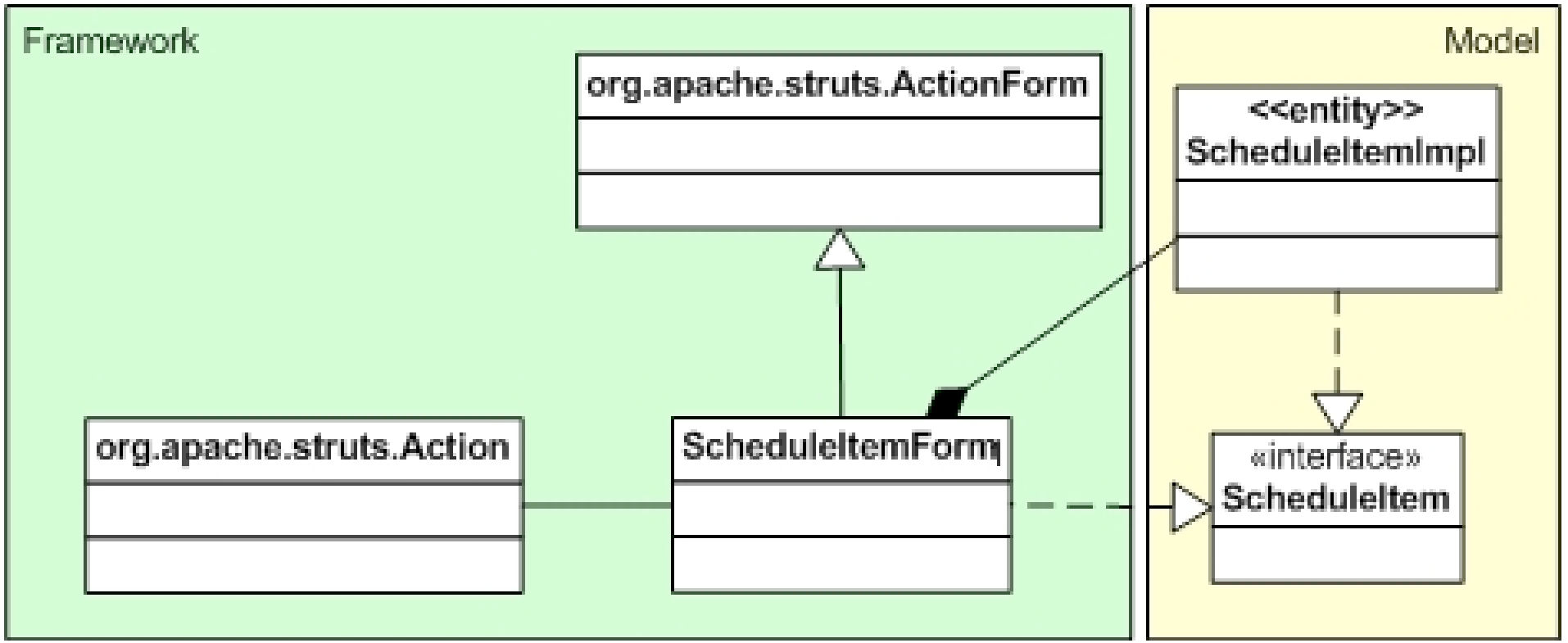
don't extend library/framework classes

prefer composition over inheritance

struts ActionForm

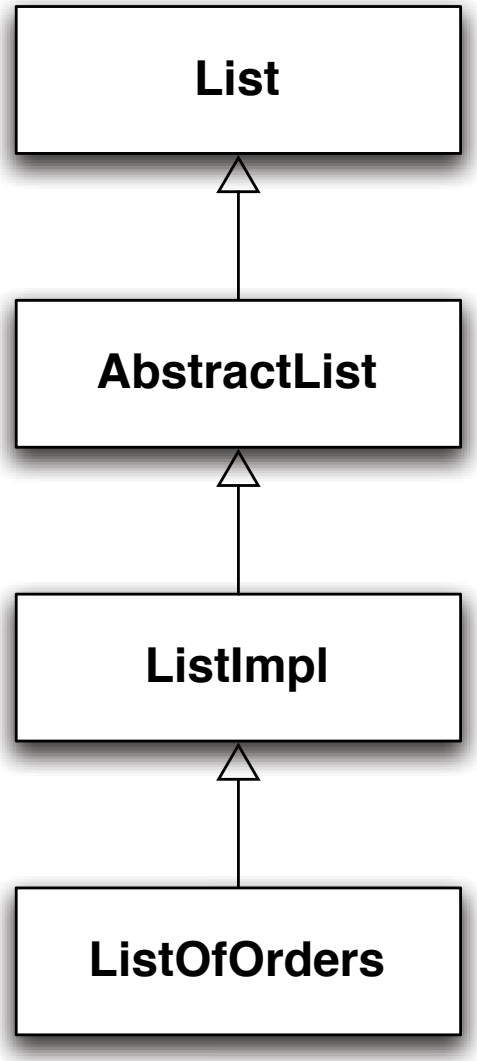


decoupling from struts



list-like inheritance hierarchy

each class has a maximum of 1 subclass

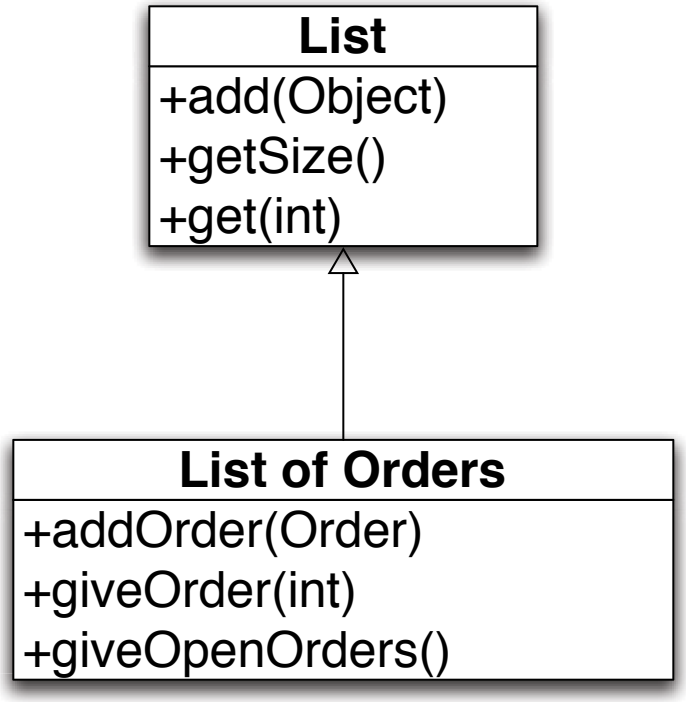


suggests either:

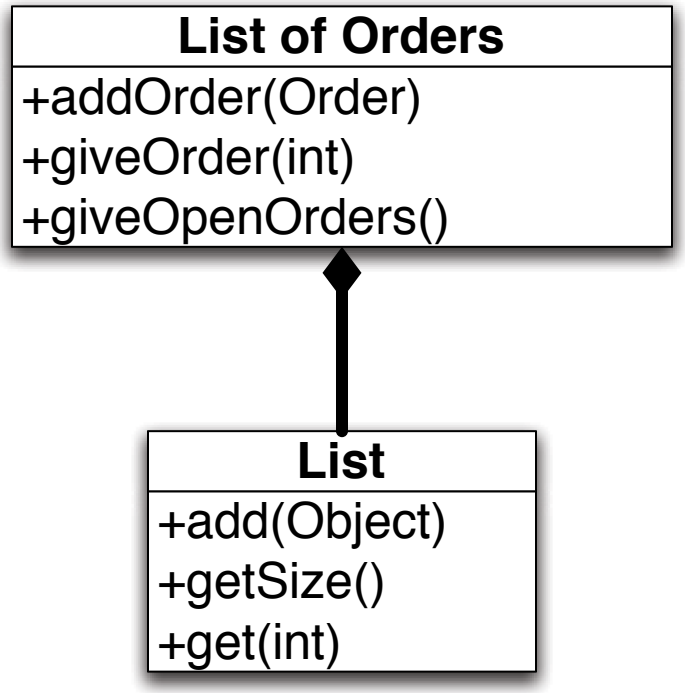
classes that are too big

speculative generalization

sub-classes do not redefine methods

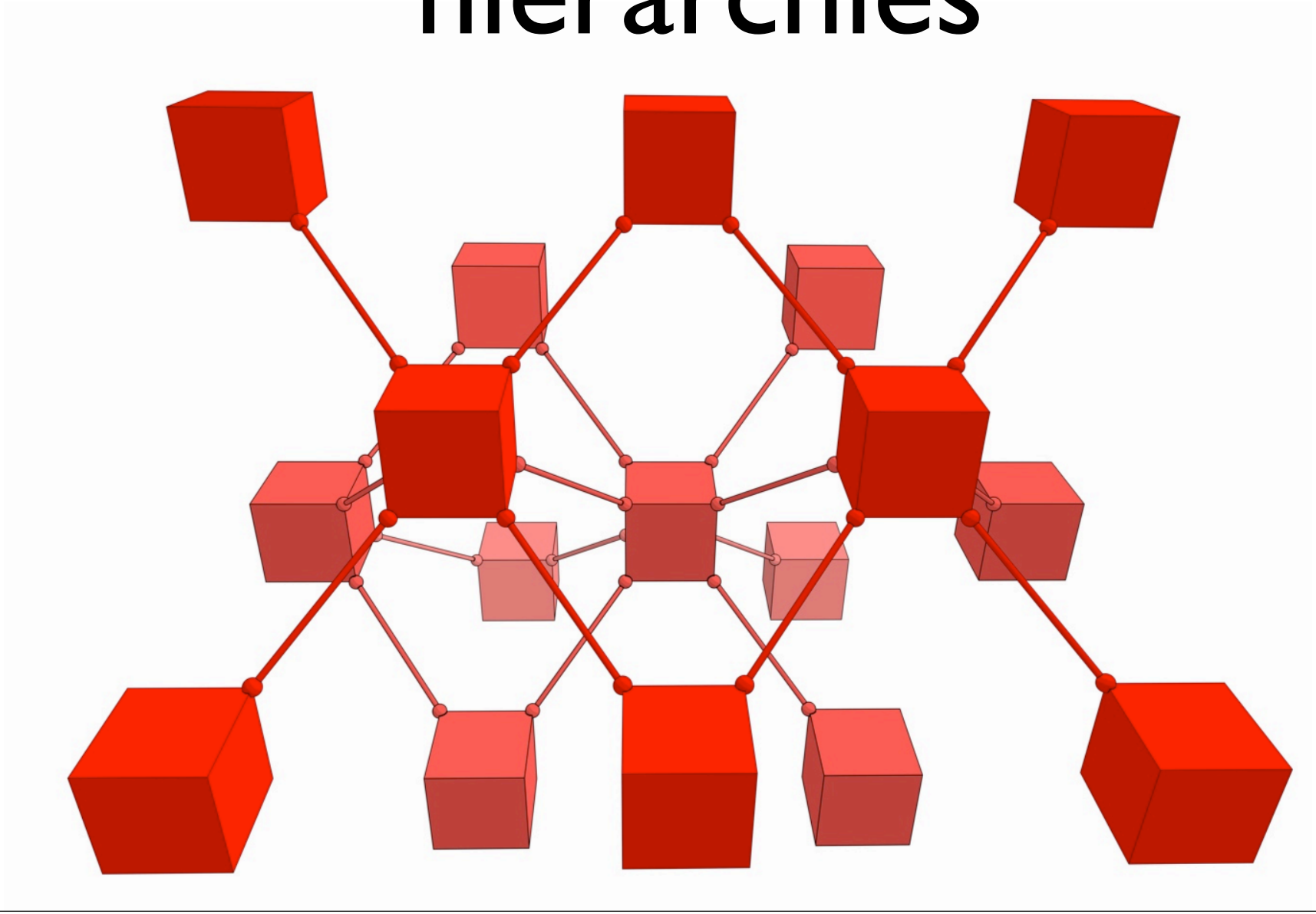


using inheritance to couple to List methods

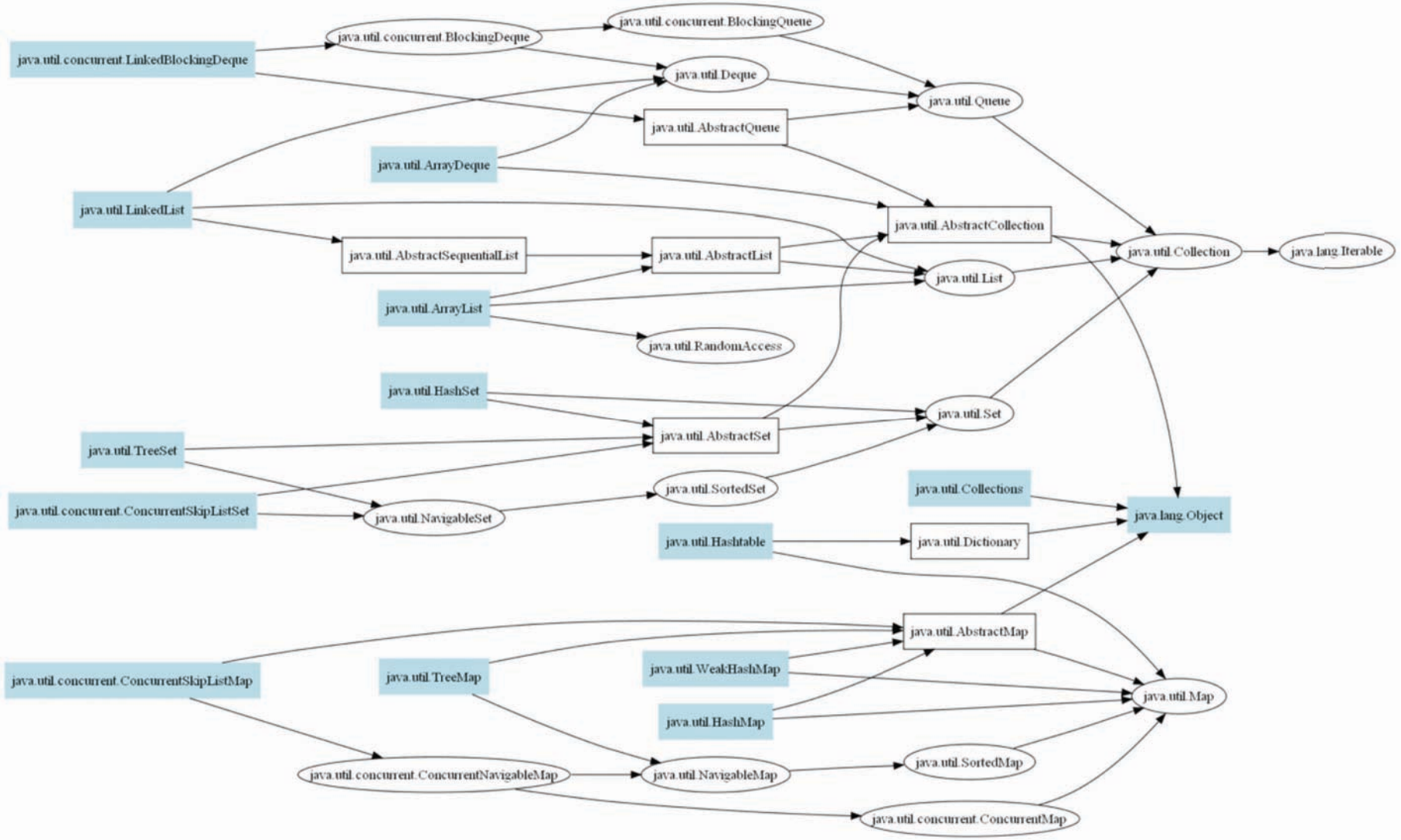


use composition instead

deep inheritance hierarchies



java's collection package



ruby's collections

Array

Set

"humane interface"

Hash

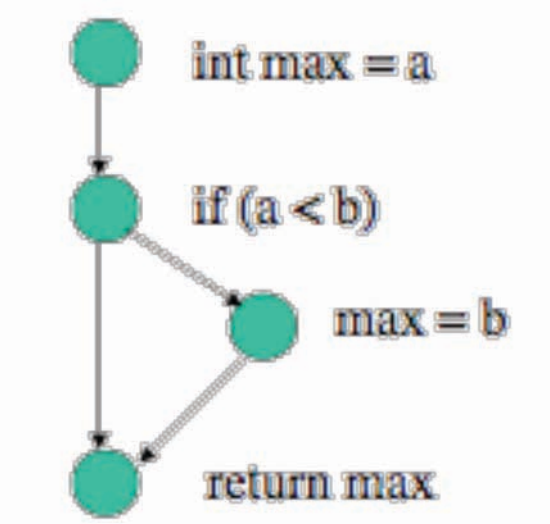
choosing what to refactor

cyclomatic complexity

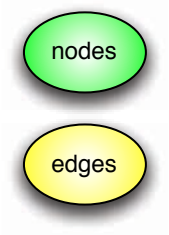
measures complexity of a function

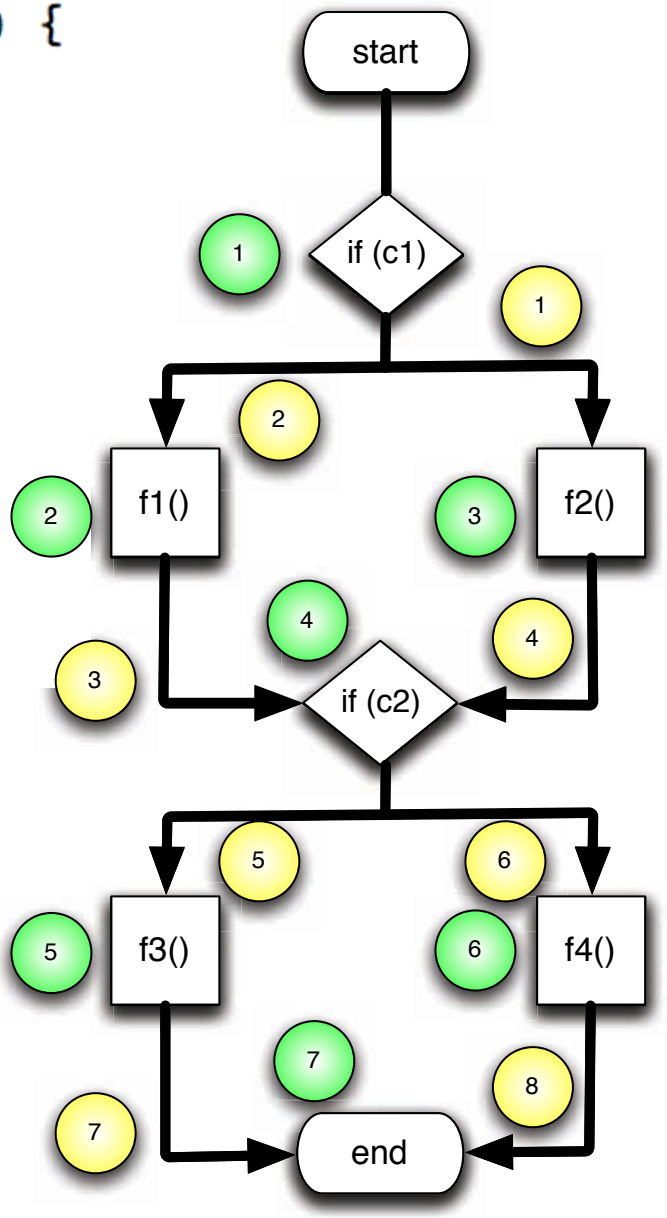
```
V(G)= e - n + 2
V(G) = cyclomatic complexity of G
e= # edges
n= # of nodes
```

```
int max (int a, int b) {
    int max = a;
    if (a < b) {
        max = b;
    }
    return max;
}
```



```
public void doIt() {
    if (c1) {
        f1();
    } else {
        f2();
    }
    if (c2) {
        f3();
    } else {
        f4();
    }
}
```





afferent coupling

∑ of how many classes use this class

incoming calls

determines what is the “hard, crunchy center” of your code base

measure with CKJM or other metrics tools

ckjm

DIT depth of inheritance tree	NOC number of children	NPM number of public methods	**WMC** **weighted** **methods per** **class**
RFC response for class	LCOM lack of cohesion	CBO/CE efferent coupling	CA **afferent** **coupling**

struts 2.x

classname	WMC	Ca
org.apache.struts2.components.Component	28	177
org.apache.struts2.views.freemarker.tags.TagModel	7	47
org.apache.struts2.views.velocity.components.AbstractDirective	8	43
org.apache.struts2.StrutsException	7	23
org.apache.struts2.components.UIBean	53	22
org.apache.struts2.dispatcher.mapper.ActionMapping	13	20
org.apache.struts2.views.jsp.ComponentTagSupport	6	19
org.apache.struts2.dispatcher.Dispatcher	37	19
org.apache.struts2.views.jsp.ui.AbstractUITag	34	18
org.apache.struts2.views.xslt.AdapterFactory	9	16
org.apache.struts2.views.xslt.AdapterNode	10	15
org.apache.struts2.ServletActionContext	11	15
org.apache.struts2.components.table.WebTable	33	12
org.apache.struts2.dispatcher.mapper.ActionMapper	2	11
org.apache.struts2.components.template.TemplateEngine	2	10
org.apache.struts2.components.template.Template	7	10
org.apache.struts2.dispatcher.StrutsResultSupport	13	10
org.apache.struts2.components.Form	24	10
org.apache.struts2.components.ListUIBean	8	9
org.apache.struts2.util.MakeIterator	3	8
org.apache.struts2.StrutsStatics	0	7

UIBean evaluateParams()

```
public void evaluateParams() {
    addParameter("templateDir", getTemplateDir());
    addParameter("theme", getTheme());

    String name = null;

    if (this.key != null) {

       if(this.name == null) {
            this.name = key;
        }

        if(this.label == null) {
            this.label = "%{getText('"+ key +"')}";
        }

    }

    if (this.name != null) {
        name = findString(this.name);
        addParameter("name", name);
    }

    if (label != null) {
        addParameter("label", findString(label));
    }
```

identifying idiomatic patterns

lots of accidental complexity around **`evaluateParameters()`**

what is the purpose of this code?

what would make it cleaner?

how much of this exists?

evaluate.*Params ?

find . -name "*.java" | xargs grep -l "void evaluate.*Params" > pbcopy

./org/apache/struts2/components/AbstractRemoteCallUIBean.java
./org/apache/struts2/components/Anchor.java
./org/apache/struts2/components/Autocompleter.java
./org/apache/struts2/components/Checkbox.java
./org/apache/struts2/components/ComboBox.java
./org/apache/struts2/components/DateTimePicker.java
./org/apache/struts2/components/Div.java
./org/apache/struts2/components/DoubleListUIBean.java
./org/apache/struts2/components/DoubleSelect.java
./org/apache/struts2/components/File.java
./org/apache/struts2/components/Form.java
./org/apache/struts2/components/FormButton.java
./org/apache/struts2/components/Head.java
./org/apache/struts2/components/InputTransferSelect.java
./org/apache/struts2/components/Label.java
./org/apache/struts2/components/ListUIBean.java
./org/apache/struts2/components/OptionTransferSelect.java
./org/apache/struts2/components/Password.java
./org/apache/struts2/components/Reset.java
./org/apache/struts2/components/Select.java
./org/apache/struts2/components/Submit.java
./org/apache/struts2/components/TabbedPanel.java
./org/apache/struts2/components/table/WebTable.java
./org/apache/struts2/components/TextArea.java
./org/apache/struts2/components/TextField.java
./org/apache/struts2/components/Token.java
./org/apache/struts2/components/Tree.java
./org/apache/struts2/components/UIBean.java
./org/apache/struts2/components/UpDownSelect.java

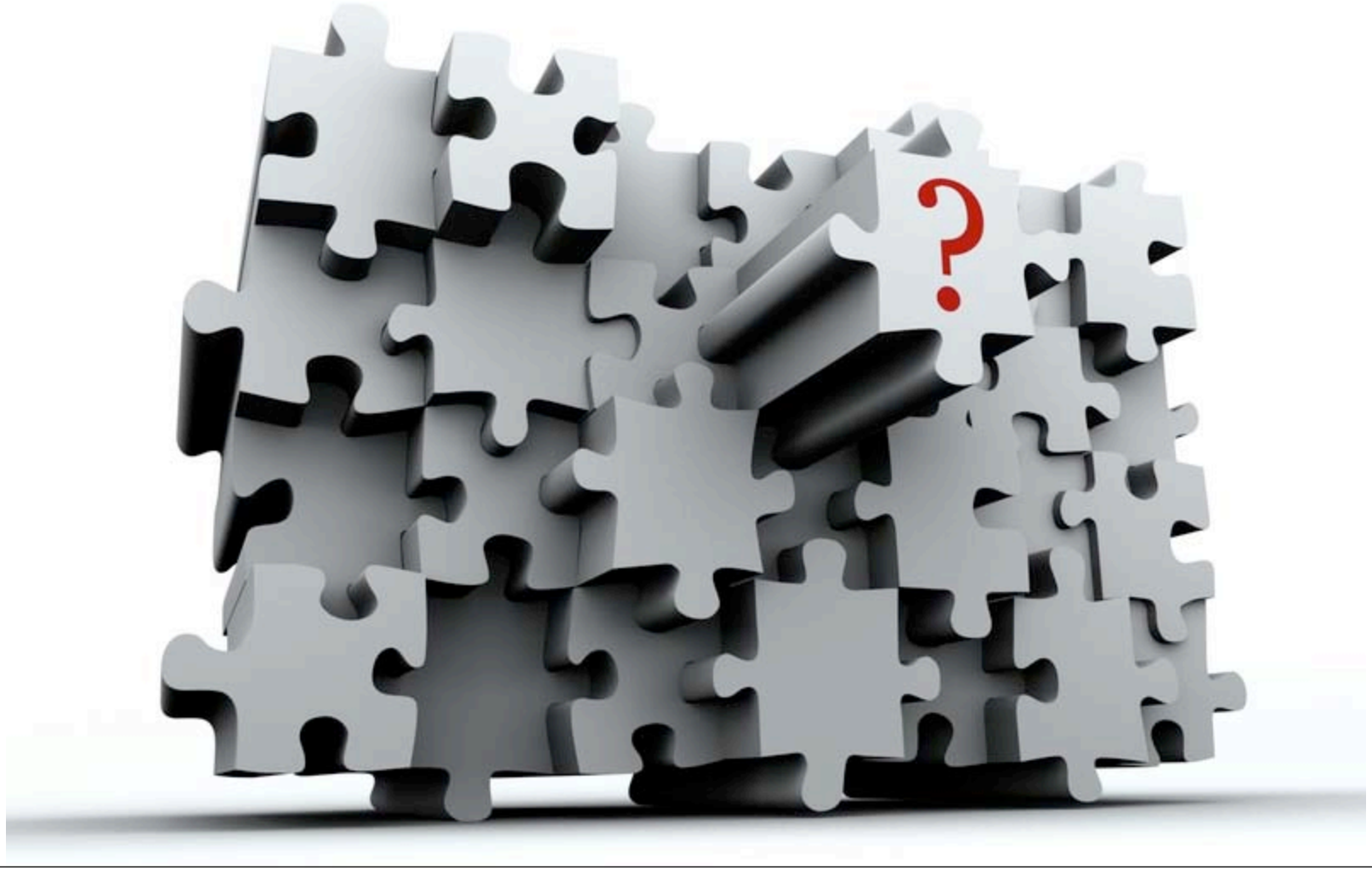
fixing parameters

```
protected void handleDefaultParameters(final String paramName) {
    try {
        Field f = UIBean.class.getField(paramName);
        if (f.get(this) != null)
            addParameter(paramName, findString(paramName));
    } catch (Exception e) {
        throw new RuntimeException(e.getMessage());
    }
}

public void evaluateParams() {
    addParameter("templateDir", getTemplateDir());
    addParameter("theme", getTheme());

    String[] defaultParameters = new String[] {"label", "labelPosition", "requiredPosition",
        "tabindex", "onclick", "ondoubleclick", "onmousedown", "onmouseup", "onmouseover",
        "onmousemove", "onmouseout", "onfocus", "onblur", "onkeypress", "onkeydown",
        "onkeyup", "onselect", "onchange", "accesskey", "cssClass", "cssStyle", "title"};

    for (String s : defaultParameters)
        handleDefaultParameters(s);
```

66 - 9 = 57 lines of obscuring duplication

22 * 3 = 66

intersection of idiomatic & design patterns



unit of work

```
public void addOrderFrom(ShoppingCart cart, String userName,
                         Order order) throws Exception {
    setupDataInfrastructure();
    try {
        add(order, userKeyBasedOn(userName));
        addLineItemsFrom(cart, order.getOrderKey());
        completeTransaction();
    } catch (Exception condition) {
        rollbackTransaction();
        throw condition;
    } finally {
        cleanUp();
    }
}
```

command design pattern

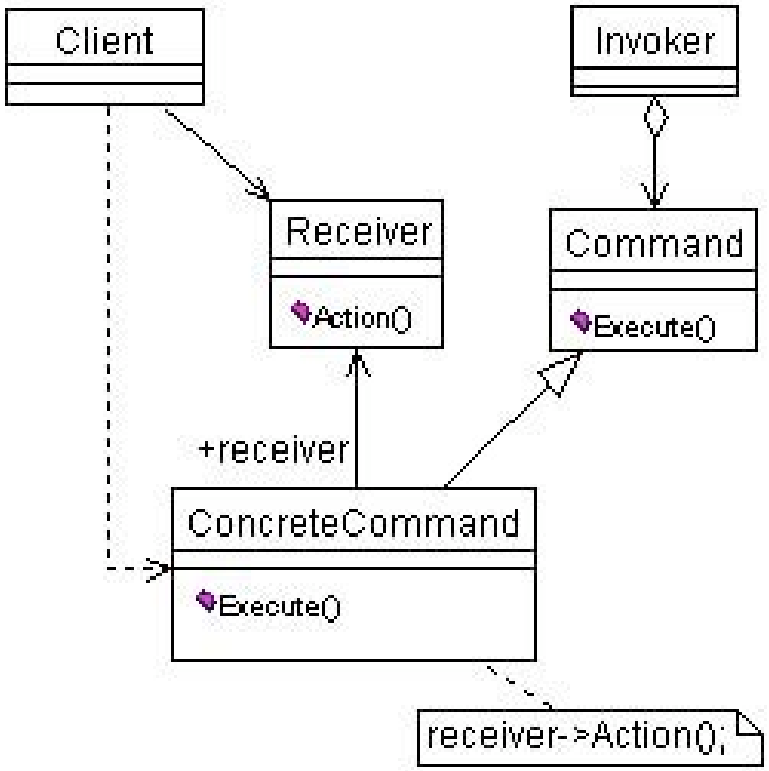
```
public void wrapInTransaction(Command c) throws Exception {
    setupDataInfrastructure();
    try {
        c.execute();
        completeTransaction();
    } catch (Exception condition) {
        rollbackTransaction();
        throw condition;
    } finally {
        cleanUp();
    }
}

public void addOrderFrom(final ShoppingCart cart, final String userName,
                         final Order order) throws Exception {
    wrapInTransaction(new Command() {
        public void execute() {
            add(order, userKeyBasedOn(userName));
            addLineItemsFrom(cart, order.getOrderKey());
        }
    });
}
```

unit of work

common idiomatic pattern

implemented via the command design pattern

use aspects



world's worst framework

every single project depends on it

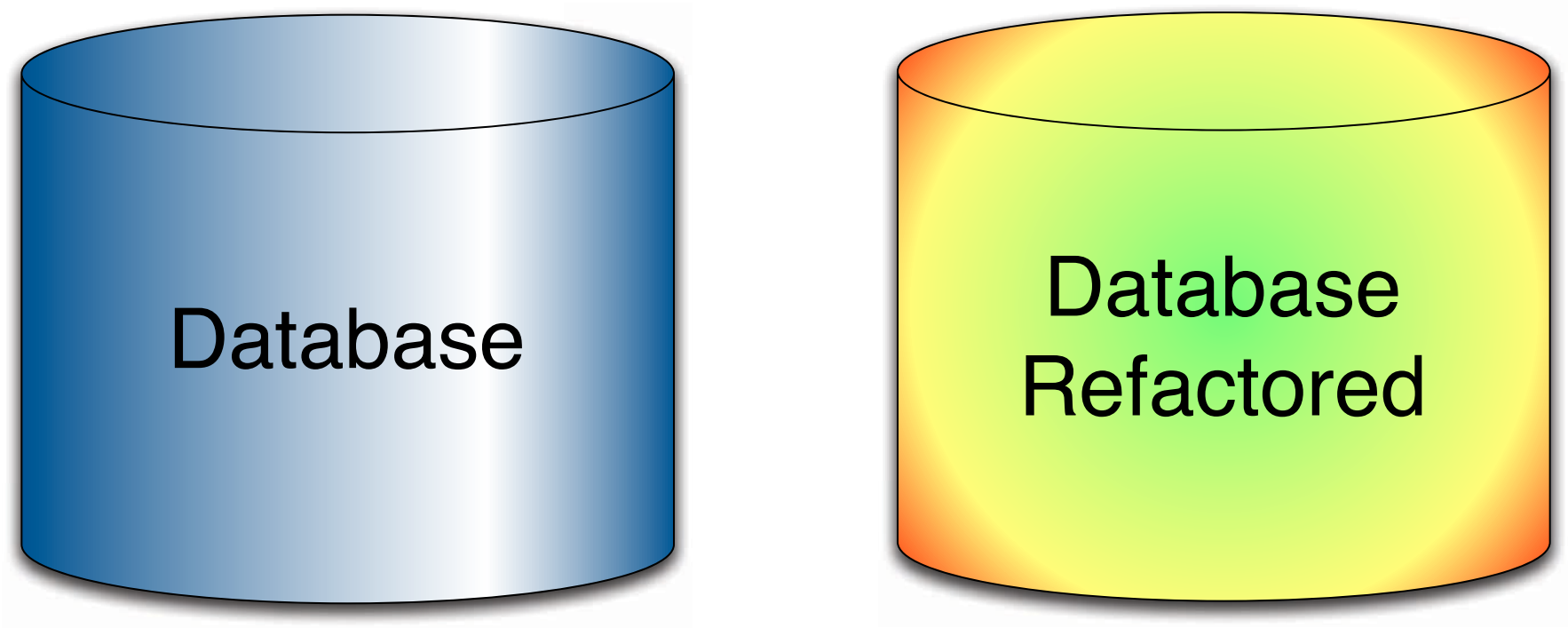
strange configuration

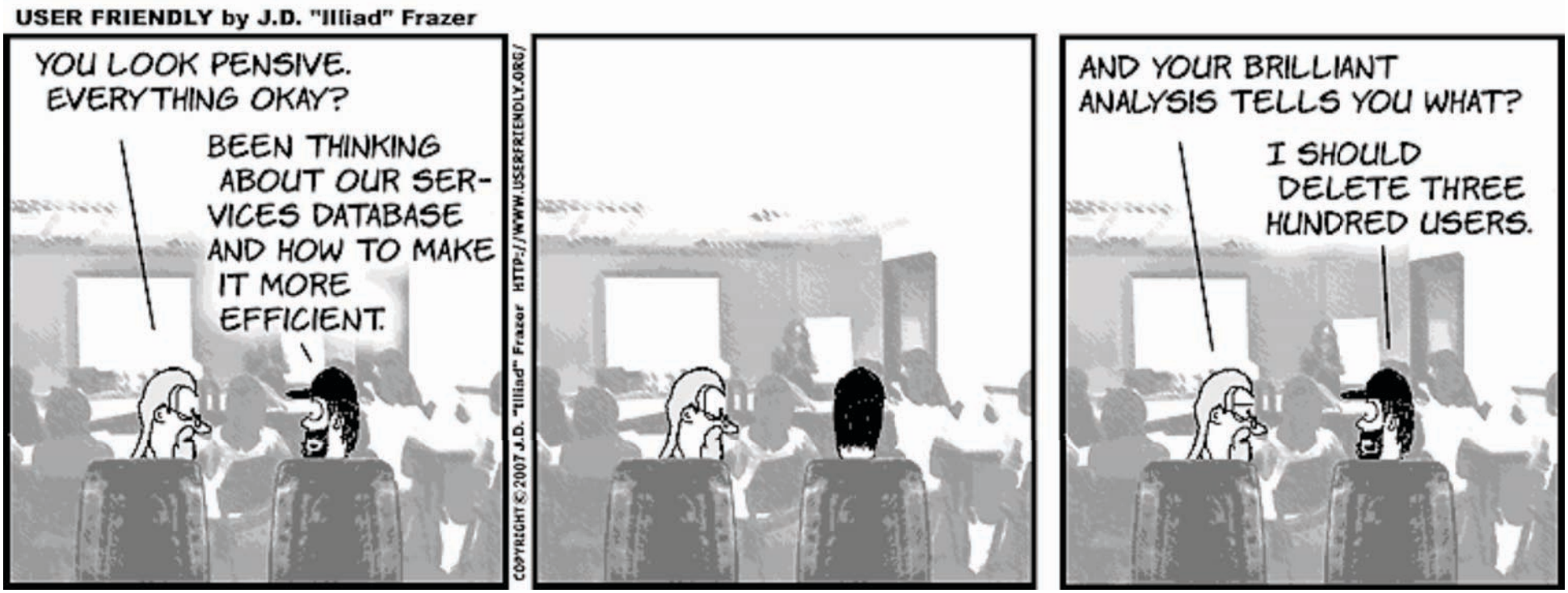
hard to test

no refactoring support

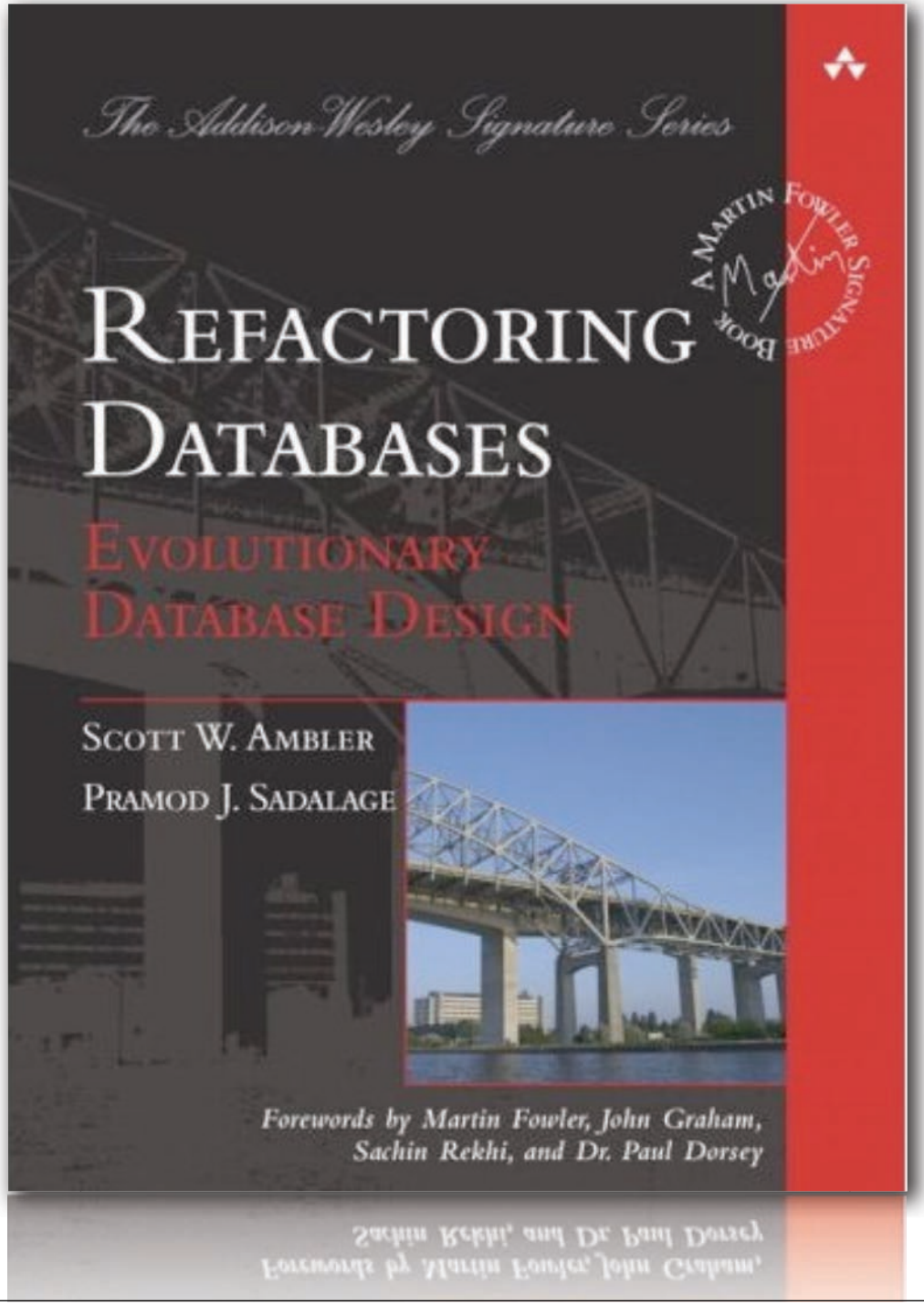
outlives every project

database refactoring





http://ars.userfriendly.org/cartoons/?id=20090304


The Addison Wesley Signature Series
A Martin Fowler Signature Book
Refactoring Databases
Evolutionary Database Design
Scott W. Ambler
Pramod J. Sadalage
Forewords by Martin Fowler, John Graham,
Sachin Rekhi, and Dr. Paul Dorsey

process

1. verify that a database refactoring is appropriate
2. choose the most appropriate database refactoring
3. deprecate the original database schema
4. test before, during, and after
5. modify the database schema

process

6. migrate the source data

7. modify external access program(s)

8. run regression tests

9. version control your work

10. announce the refactoring

dbDeploy

migrations for the rest of the world

open-source tool to help you manage database changes

version database schema changes along with code

builds delta scripts for you

using dbDeploy

someone (dba) creates a sql baseline

developers create delta scripts for each change

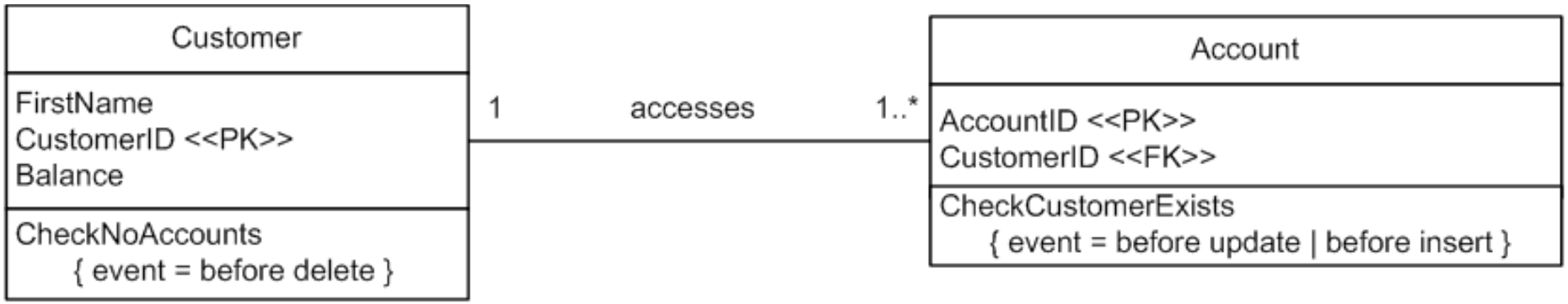
dbDeploy compares delta with patch table

generates change scripts

invoked from ant's sql task

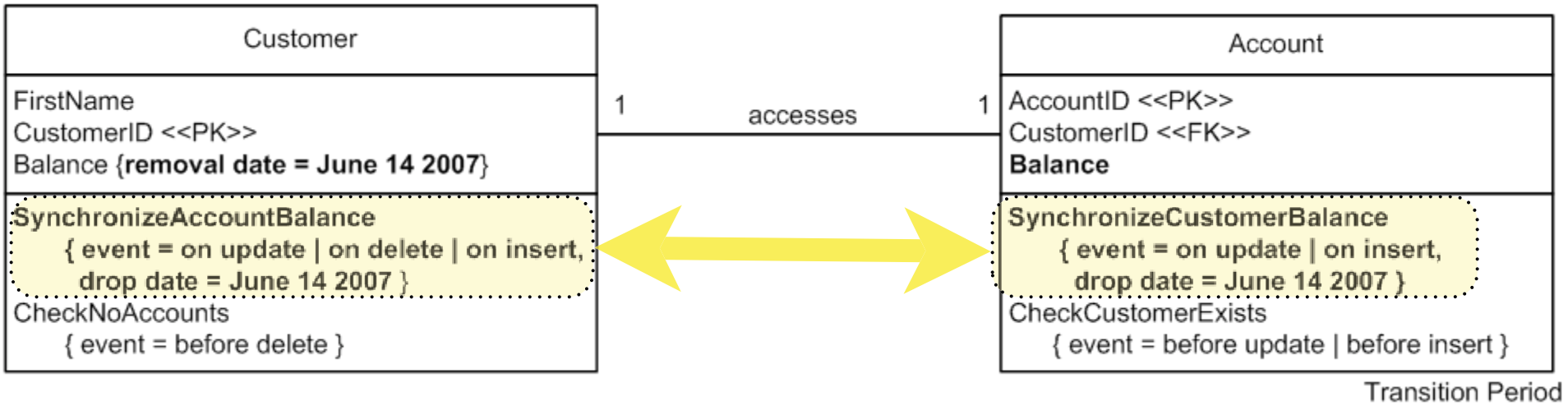
example

move column refactoring

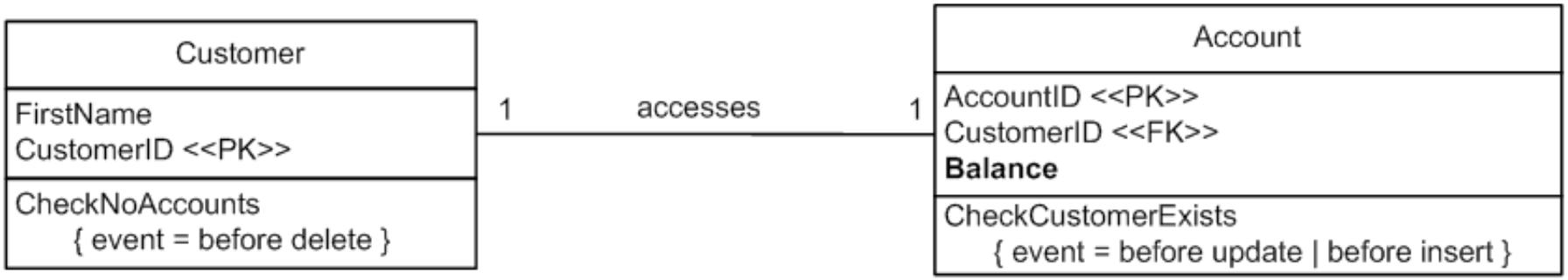


Original Schema

transition period



resulting schema



Resulting Schema

strategies

smaller changes are easier to apply

uniquely identify individual refactorings

implement a large change by many smaller ones

prefer triggers over views or batch synchronization

choose a sufficient deprecation period

strategies

encapsulate data access

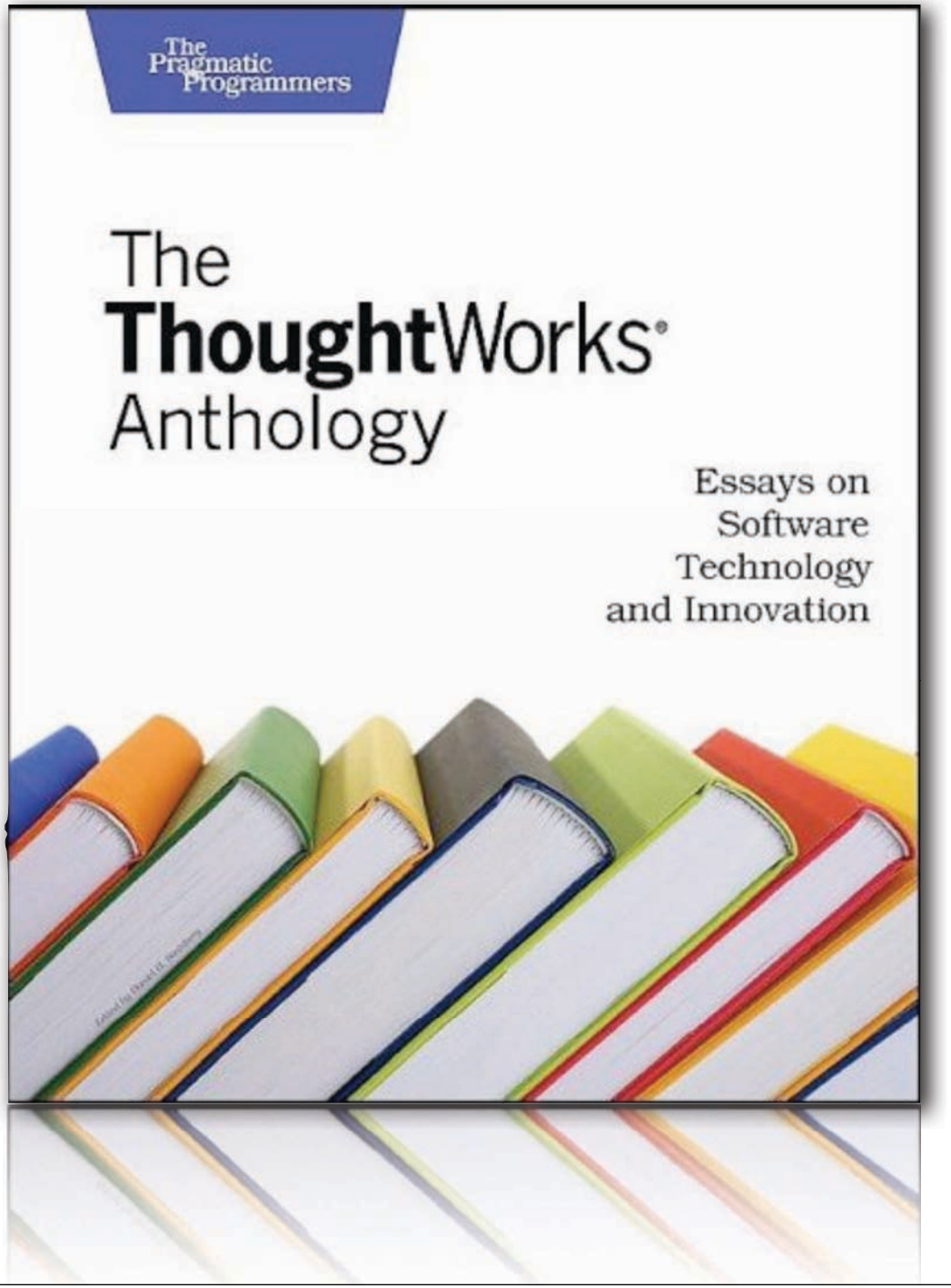
be able to easily set up a database environment

don’t duplicate SQL

put database assets under change control

beware of politics


build files


The Pragmatic Programmers
The ThoughtWorks® Anthology
Essays on Software Technology and Innovation

When we...construct and maintain build files, all those code perfectionist ideals seem to disappear.
► Paul Glover

Chapter 11

Refactoring Ant Build Files

by Julian Simpson, Build Architect

11.1 Introduction

A bad software build system seems to actively resist change. The most innocent change can stop your colleagues from being able to work. Such a reputation won't encourage people to try to improve a build if it eventually gets the job done. This essay will show you how to help ease some of that pain and allow change to take place by describing some short refactorings to apply to your Ant build file. Each refactoring is expressed as a "before" and "after" code example (separated by an arrow to show the direction of transformation) with an explanation. You will be left with some concrete tools that you can apply to your Ant build to make it smaller, more legible, and easier to modify.

What Is Refactoring? And What Is Ant?

Refactoring is the art of making small changes to a system to improve the readability, clarity, or ease of maintenance of a codebase. Refactoring doesn't change functionality; it's used to make the internals of a codebase easier to work on.

Ant is the build tool of choice for many Java-based software projects. It was written at a time when XML seemed to be the answer to any problem in software development. Hence, a project's dependencies are described in an XML file, typically called build.xml. This is the Java equivalent of a Makefile. Ant has been very successful as an open source project, but as projects grow, flex, and become more complicated, it generally becomes hard to maintain.

extract macrodef

Take small blocks of ant code, and pull out into macrodef with appropriate name.

```
<target name="build_and_war_foo.war" >
    <javac srcdir="src/foo" destdir="classes/foo" />
    <copy todir="${classes.dir}" >
        <filterset>
            <filter token="ENV" value="${environment}" />
        </filterset>
        <fileset dir="config" />
    </copy>
    <war destfile="foo.war" >
        <fileset dir="${classes.dir}" />
    </war>
    <move todir="archives" file="foo.war" />
</target>
```

`macrodef`

container task

wraps sequential or parallel tasks

invokable from anywhere in the build file

with attributes

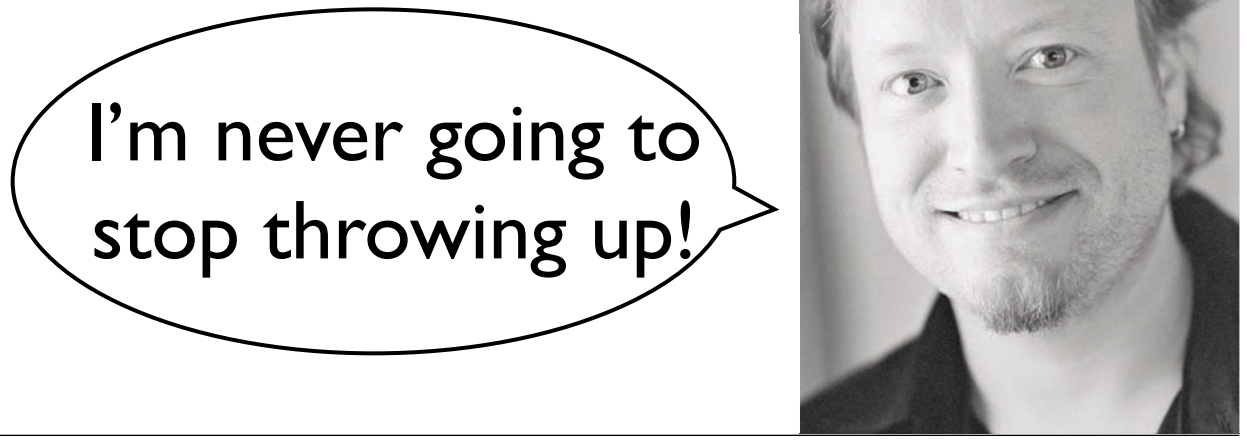
attributes can have default values

```
<macrodef name="build_code" >
    <attribute name="component" />
    <sequential>
        <javac srcdir="src/@{component}" destdir="classes/@{component}" />
        <copy todir="${classes.dir}" >
            <filterset>
                <filter token="ENV" value="${environment}" />
            </filterset>
            <fileset dir="config" />
        </copy>
    </sequential>
</macrodef>
<macrodef name="make_war" >
    <attribute name="component" />
    <sequential>
        <war destfile="@{component}.war" >
            <fileset dir="${classes.dir}" />
        </war>
        <move todir="archives" file="@{component}.war" />
    </sequential>
</macrodef>
<target name="foo.war" >
    <build_code component="foo" />
    <make_war component="foo" />
</target>
```

introduce declaration

use ant's built-in declarative logic to replace if conditions, which can be hard to debug.

```
<target name="deploy" >
    <if>
        <equals arg1="${j2ee.server}" arg2="was" />
        <then>
            <antcall target="was_deploy" />
        </then>
        <else>
            <antcall target="weblogic_deploy" />
        </else>
    </if>
</target>
```



```
<property name="j2ee.server" value="was" />
<import file="${j2ee.server}.build.xml" />
<!-- there is now a an appropriate target named
deploy depending on the version of the app server -->
```

move target to wrapper build file

pull continuous integration targets out of the developer build file; provide some indirection.

separating concerns

mixed concerns obstruct refactoring

logically separate build and continuous integration logic

before:

```
<target name="build" >
    <!-- developer build-->
</target>
<target name="functest" >
    <!-- functional tests-->
</target>
<target name="cruise" depends="update,build,tag" />
<target name="functional_cruise" depends="update,build,functest,tag" />
```

```
<target name="build" >
    <!-- developer build-->
</target>
<target name="functest" >
    <!-- functional tests-->
</target>
```

developer concerns

```
<project name="cruise" default="tag" >
    <target name="tag" depends="build" >
        <!-- code to tag the files you have checked out -->
    </target>
    <target name="build" depends="update" >
        <ant buildfile="build.xml" />
    </target>
    <target name="update" >
        <!-- code to update from your scm system-->
    </target>
</project>
<!-- END ccbuild -->
<project default="update" basedir="."
    xmlns:my="antlib:com.thoughtworks.monkeybook" >
    <target name="update" depends="build" >
        <my:svn_up/>
    </target>
</project>
<!-- END antlibccbuild -->
```

build concerns

enforce internal target

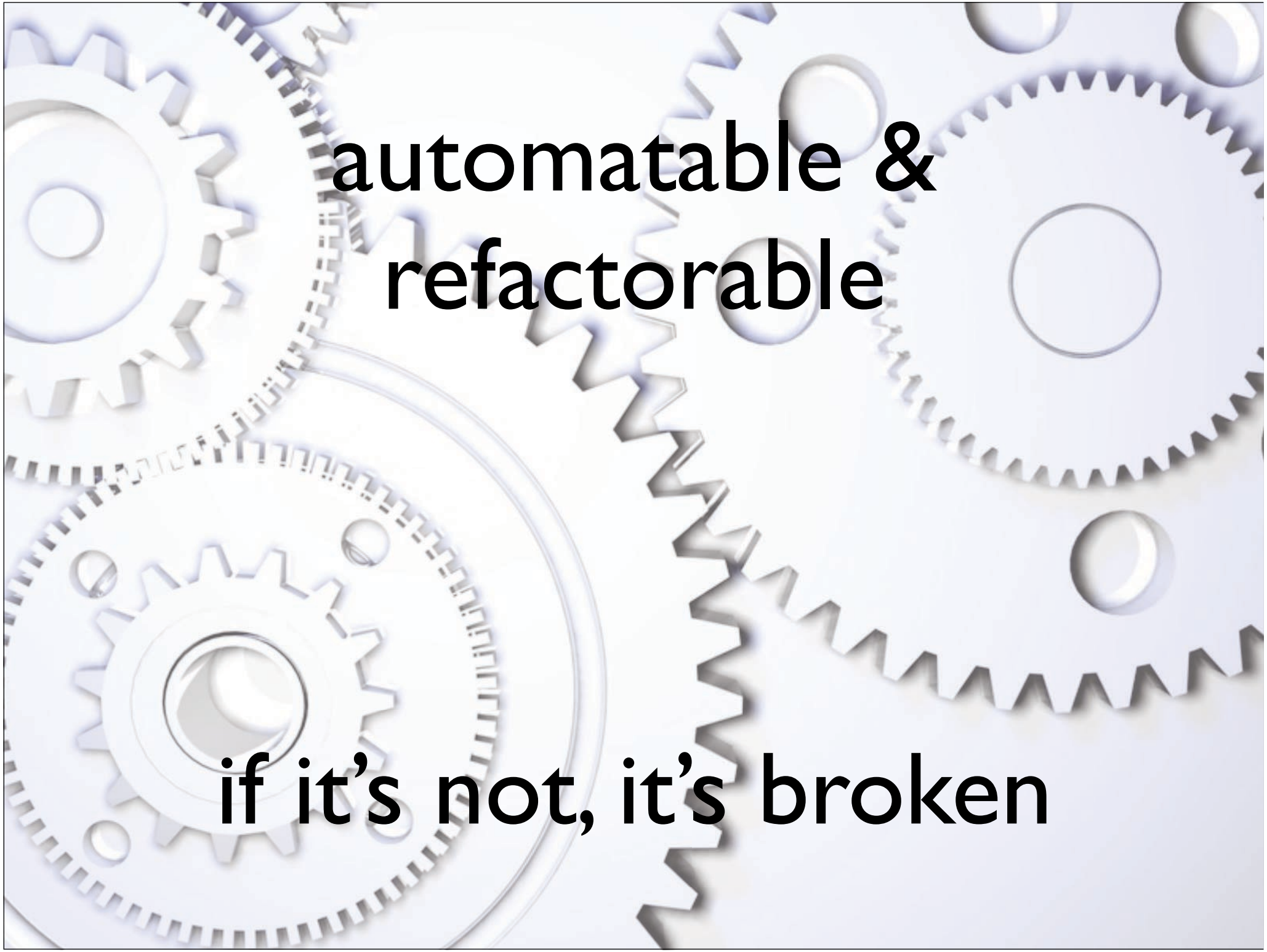
make the first character of internal target names a hyphen so they cannot be called from the command line

Ant encapsulation

no scoping for ant tasks...

...so make you own via conventions

```
<target name="-init" >
    <mkdir dir="build" />
</target>
```


automatable &
refactorable
if it's not, it's broken


everything in your
infrastructure matters

? ' s

please fill out the session evaluations

samples at `github.com/nealford`

NEAL FORD software architect / meme wrangler

ThoughtWorks

nford@thoughtworks.com
3003 Summit Boulevard, Atlanta, GA 30319
www.nealford.com
www.thoughtworks.com
memeagora.blogspot.com