Why Programmers don’t use Refactoring Tools
(and what we can do about it)

Andrew P. Black
joint work with Emerson Murphy-Hill
Outline

- What is Refactoring/Refactoring Tools
  - Refactoring tools are “a good thing”
- Are the tools being used?
  - No
- Why Refactoring Tools are Underused
  - It’s the tools fault
- What’s Wrong with Typical Tools
- How to fix the problem
What is Refactoring?

Changing the structure of code without changing the way that it behaves.

We do it because no one has perfect foresight.
What is Refactoring?

Changing the structure of code without changing the way that it behaves.

We do it because no one has perfect foresight.
What is Refactoring?

Changing the structure of code without changing the way that it behaves.

We do it because no one has perfect foresight.
What is Refactoring?

Changing the structure of code without changing the way that it behaves.

We do it because no one has perfect foresight.
What is Refactoring?

Changing the structure of code without changing the way that it behaves.

We do it because no one has perfect foresight.
What is Refactoring?

Changing the structure of code without changing the way that it behaves.

We do it because no one has perfect foresight.
What is Refactoring?

Changing the structure of code without changing the way that it behaves.

We do it because no one has perfect foresight.
What is Refactoring?

Changing the structure of code without changing the way that it behaves.

We do it because no one has perfect foresight.
Common Refactorings

- Rename
- Insert Superclass
- Push up/down method
- Push up/down field
- Extract Method/Inline Method
- Abstract/Reify field
Common Refactorings

- Rename
- Insert Superclass
- Push up/down method
- Push up/down field
- Extract Method/Inline Method
- Abstract/Reify field

speed := distance / time
Common Refactorings

- Rename
- Insert Superclass
- Push up/down method
- Push up/down field
- Extract Method/Inline Method
- Abstract/Reify field

\[ \text{speed} := \text{distance} / \text{time} \]
Common Refactorings

- Rename
- Insert Superclass
- Push up/down method
- Push up/down field
- Extract Method/Inline Method
- Abstract/Reify field

speed := distance / time  \rightarrow  speed := this.distance() / time
Common Refactorings

- Rename
- Insert Superclass
- Push up/down method
- Push up/down field
- Extract Method/Inline Method
- Abstract/Reify field

\[
\text{speed} := \frac{\text{distance}}{\text{time}}
\]

\[
\text{speed} := \text{this}.\text{distance}() / \text{time}
\]
Common Refactorings

- Rename
- Insert Superclass
- Push up/down method
- Push up/down field
- Extract Method/Inline Method
- Abstract/Reify field

speed := distance / time \rightarrow speed := this.distance() / time
What’s the big deal?

Refactoring is a fancy name for what we used to call “keeping code clean”

- It’s essential for a healthy code base
- Xing and Strouliya (ICSM ‘06) report that up to 70% of code changes can be due to refactoring
- Empirical data show that refactoring does improve code:
  - Kataoka: decreased coupling
  - Benn et al: complexity, size, cohesion, and coupling all improved
  - Kolb et al: maintainability and usability are increased
- ... and many other studies
What’s a Refactoring Tool?

- A tool that automates how you used to refactor by hand.
  - Knows about the semantics of the language
  - Refactors quickly
  - Refactors without introducing new errors

- Refactoring tools are provided by:
  - Eclipse/Java
  - Smalltalk
  - IDEA (from IntelliJ)
  - Visual Studio

- Not refactoring tools:
  - Find/replace
  - SED / AWK
A typical tool interaction

```java
//
// range.start = offset;
// range.length = length;
// widget.setRangeStyleRange(range);

BoxAnnotation ann = (BoxAnnotation) annotation;
ann.setWidget(widget);

String textPaintingOver;
if (length==0)
    textPaintingOver = "";
else
    textPaintingOver = widget.getText(offset, offset+length-1).replace(" ", "");

Rectangle rec = getRectangle(widget, offset, length, ann, textPaintingOver);

if (ann.isSelected()){
    gc.setAlpha(50);
    gc.setBackground(RED);
} else{
    gc.setAlpha(20);
    gc.setBackground(GREEN);
}

gc.fillRectangle(rec);
```
// range.start = offset;
// range.length = length;
// widget.setStyleRange(range);

BoxAnnotation ann = (BoxAnnotation) annotation;
ann.setWidget(widget);

String textPaintingOver;
if (length==0)
textPaintingOver = "";
else
textPaintingOver = widget.getText(offset, offset+length-1).replace(" ", ");

Rectangle rec = getRectangle(widget, offset, length, ann, textPaintingOver);

if (ann.isSelected()){
gc.setAlpha(50);
gc.setBackground(RED);
} else{
gc.setAlpha(20);
gc.setBackground(GREEN);
}
gc.fillRect(rec.r);
range.start = offset;
range.
widget
BoxAnn
ann.set
String
if(len
else
text
else
text
Rectangle rec = getRectangle(widget, offset, length, ann, textPaintingOver);

if(ann.isSelected()){
gc.setAlpha(50);
gc.setBackground(RED);
}else{
gc.setAlpha(20);
gc.setBackground(GREEN);
}
gc.fillRect(rect, gc, rect);
```
//
range.start = offset;
//
range.length = length;
//
widget.setStyleRange(range);

BoxAnnotation ann = (BoxAnnotation) annotation;
ann.setWidget(widget);

String textPaintingOver;
if(length==0)
    textPaintingOver = "";
else
    textPaintingOver = widget.getText(offset, offset+length-1).replace(" ", "");

Rectangle rec = getRectangle(widget, offset, length, ann, textPaintingOver);

if(ann.isSelected()){
    int i = 50;
    gc.setAlpha(i);
    gc.setBackgroundColor(RED);
}else{
    gc.setAlpha(20);
    gc.setBackgroundColor(GREEN);
}
```
Java Tools

Eclipse
JBuilder
Netbeans
IntelliJ IDEA
RefactorIT
X-Develop / CodeGuide
X-Refactory*
JFactor *
JRefactory *
Transmogrify*
JavaRefactor *

* Indicates a “dead” tool
Refactoring Tool under-use

16 Object-Oriented students
  - Only 2 used Refactoring Tools
37 users of Eclipse in PSU lab
  - 2 used Refactoring Tools
112 participants at Agile Open NW 2007:
  - Of 72 programmers, 63 have tools available some of the time
  - They claimed an average of 68% use of the tools
  - Why not 100%?
Refactoring Tool (Under) Usage

- Murphy et al. looked at 41 Programmers
  - Only two refactoring tools used by “most” programmers: Rename and Move
  - Median number of refactoring hotkeys used by programmers is 2; maximum is 5

- Disconnect between refactoring desires and tool use:
  - According to Mantayla, programmers overwhelmingly want to Extract Method
  - But according to Murphy, programmers overwhelmingly perform Rename
Data: Murphy et al.
Vizualization: Murphy-Hill
Agile Open Northwest 2007: When performing a refactoring where a tool is available but you choose not to use it, what usually prevents you?

44 responses. The tool isn’t flexible enough—it doesn’t do quite what I want.

26 responses. I never really learned how to use that particular refactoring tool / I don’t know what tool to use.

24 responses. I can do it faster by hand.

13 responses. I don’t trust the tool to be correct.

7 responses. The tool will probably mutilate my code.

2 responses. My code base is so large that the refactoring tool takes too long.

Other: • Habit. • Menu too big. • Avoid GUIs—Keybindings only! • Prefer to be aware of the changes myself. • Hard to trust the refactored code, even if it applies. • Usually I do multistep refactoring—tools do one step at a time.
Two kinds of Refactoring

Floss Refactoring
- Programmers refactor constantly to maintain healthy software
- Refactoring is interleaved with programming

Root Canal Refactoring
- Programmers refactor in clumps to fix unhealthy software.
- Programming and refactoring are distinct activities.
Floss vs. Root Canal

Floss
- Impromptu: refactor whenever you think the code needs it
- You know exactly what code you’re going to refactor, because you’re working on it
- Supported by Fowler (1999), Parnin+ (2006), Hayashi+ (2006)

Root Canal
- Planned: set aside time for refactoring
- You don’t know what needs to be refactored, but past experience indicates that future changes will be difficult
Why you should Floss rather than waiting for a Root Canal

Your dentist says so:

- “Refactoring is something you do all the time in little bursts. You don’t decide to refactor, you refactor because you want to do something else, and refactoring helps you do that other thing.”

  Martin Fowler, *Refactoring*

- “Avoid the temptation to stop work and refactor for several weeks. Even the most disciplined team inadvertently takes on design debt, so eliminating debt needs to be an ongoing activity. Have your team get used to refactoring as part of their daily work.”

  James Shore, “Design Debt”
Why you should Floss rather than waiting for a Root Canal

Your friends are doing it!

- Weiβgerber and Diehl (MSR 2006) looked at JUnit, ArgoUML and JEdit:
  
  “It turned out that in all three projects, there are no days which only contain refactorings. This is quite surprising, as we would expect that at least in small projects like JUnit there are phases in a project when only refactorings have been done to enhance the program structure.”

- Murphy et al. (Software 2006) observed 41 Eclipse developers:
  
  2672 repository commits
  At most 9 out of 283 iterations were pure refactoring
Why you should Floss rather than waiting for a Root Canal

You are becoming Agile!

- Continuous design and continuous refactoring are key practices for Agile programmers

  “We keep the code simple *at all times*. This minimizes the investment in excess framework and support code. We retain the necessary flexibility through refactoring.”

  Jeffries, Anderson & Hendrickson
  *Extreme Programming Installed*, 2000
Root Canal: Ineffective

- Pizka (2004) describes a root canal refactoring over 5 months; concludes that the time was mostly wasted.
- Bourquin and Keller (2007) describe a root canal refactoring over 7 months
  - few objectively positive results
  - dramatic increase in duplicated code.
Why the Distinction Matters

Claim: a tool built for root canal refactoring will not be very usable for floss refactoring, and to a lesser degree, vice versa.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floss Tool</td>
</tr>
<tr>
<td>• Runs continuously in background.</td>
</tr>
<tr>
<td>• Reports on the code on which programmers are working</td>
</tr>
</tbody>
</table>
So What’s Really Wrong with Refactoring Tools?

Tools don’t always fit with the way programmers want to refactor...

- data from Open Agile Survey:
  94 responses indicated a usability problem, vs
  22 indicated a technical problem

...so programmers snub the tools and refactor by hand.
How *do* Programmers Refactor?
So, what have people been doing about it?

Building better tools!
  – In industry
  – In research groups
Industrial Improvements (Code Identification)

Automated smell detection
- Eclipse’s Test and Performance Tools
- IntelliJ’s code inspections
Industrial Improvements
(Code Identification)

JCosmo

Hayashi and Colleagues

```java
class C2 {
    int foo(int t) {
        int a = F.f(1, F.f(6, 7, t), 3)
        int b = F.f(4, 5, F.f(6, 7, t))
        if (a > 8) a = F.f(1, 2, a);
        return a + b;
    }
}
```
Industrial Improvements (Selection)

```java
if (!ann.isSingleLine()) {
    if (!ann.startsWith(textPaintingOver)) {
        bounds.x = 0;
    }
    if (!ann.isTail()) {
        bounds.width = widget.getBounds().width
    }
}
```
Research Improvements (Selection)

```java
public ASTRegion(WrappedEditor editor, ASTNode node) {
    this.editor = editor;
    this.node = node;

    IDocument document = editor.getDocument();

    try {
        String nodeText = document.get(getOffset(), getLength());
        int splitIndex = nodeText.lastIndexOf('n');

        annotations = new HashMap<BoxAnnotation, Position>();

        if (splitIndex < 0) { // it's a single line node
            annotations.put(new SingleBox(), new Position(getOffset(), getLength()));
        } else {
            annotations.put(new HeadBox(), new Position(getOffset(), splitIndex));
            annotations.put(new TailBox(), new Position(splitIndex, getLength()));
        }
    } catch (BadLocationException e) {
        e.printStackTrace();
    }
}
```
Research Improvements (Selection)
Research Improvements (Selection)

```java
/**
 * @see IAnnotationModelExtension#replaceAnnotations(org.eclipse.jface.text.source.AnnotationModel, Annotation)
 */

public void replaceAnnotations(ISelfDrawingAnnotation[] remove, Map<ISelfDrawingAnnotation, Position> add)

    List<Position> positions = new ArrayList<Position>();

    if(remove!=null)
        for(ISelfDrawingAnnotation r : remove)
            Position p = anns.remove(r);
            if(p!=null)
                positions.add(p);

    if(add!=null){
        anns.putAll(add);
        positions.addAll(add.values());
    }

    fireAnnotationChangedEvent(positions);
*/
```
import java.util.Dictionary;

@ SuppressWarnings("unchecked")
class Banana{

    private int a, b, c, d;
    private double dx, dy;

    public void read(Dictionary dict){
        a = Integer.parseInt(((String)dict.get("a")));
        b = Integer.parseInt(((String)dict.get("b")));
        dx = Double.parseDouble(((String)dict.get("dx")));
        dy = Double.parseDouble(((String)dict.get("dy")));
    }
}
import java.util.Dictionary;

@SuppressWarnings("unchecked")

class Banana{

    private int a, b, c, d;
    private double dx, dy;

    public void read(Dictionary dict){
        a = Integer.parseInt(((String)dict.get("a")));
        b = Integer.parseInt(((String)dict.get("b")));

        dx = Double.parseDouble(((String)dict.get("dx")));
        dy = Double.parseDouble(((String)dict.get("dy")));
    }
}
Research Improvements
(Activation)
Research Improvements (Activation)
Research Improvements (Animal)

```java
class Animal{
}

class Dog extends Animal {
    public void bark() {
        System.out.println("bark!");
    }
}

class Cat extends Animal{
}
```
```java
class Animal{
}

class Dog extends Animal {

    public void bark() {
        System.out.println("bark!");
    }

}

class Cat extends Animal{
}
```
Industrial Improvements (Configuration)

```java
public void print(){
    System.out.println(1);
    printTwo();
}

private void printTwo(){
    System.out.println(2);
}
```
Research Improvements (Configuration)
Research Improvements (Understanding Errors)

```java
boolean areWheelsTrue(){
    Wheel front = bike.getFrontWheel();
    Wheel rear = bike.getRearWheel();
    boolean trueWheels = isWheelTrue(front) && isWheelTrue(rear);
    return trueWheels;
}
```

```java
void goOnVacation(){
    Bike roadBike = getRoadBike();
    Bike mountainBike = getMountainBike();
    loadOnCar(roadBike, mountainBike);
}
```

```java
boolean goForRide(){
    while(!tired(){
        rotatePedals(10);
        if(this.hasCrashed())
            break;
    }
    return SUCCESS;
```
Summary

- Refactoring tools are a *good thing* but only if they are used.
- Programmers don’t use refactoring tools much because they don’t fit with how they work.
- Researchers and industry are attempting to make better refactoring tools! To be successful, we must pay attention to how programmers work.
References


