How are Developers Using Refactoring Tools?

Emerson Murphy-Hill
North Carolina State University
Refactoring Tools are Great!

They do what you’d do anyway, but:

– Faster

– Guaranteed Safety
But...

Do refactoring tools really support the kinds of refactorings that people want to do?

Is the way that refactoring tools work really the way developers refactor?

Do programmers use the features of refactoring tools?

If an answer is no, how can we make tools fit?
A Study of Refactoring

class Foo{
    }

class Bar{
    int a;
    public Bar(int anA){
        this.a = anA;
    }
}

class Bar{
    int a;
    private Bar(int anA){
        this.a = anA;
    }
}

class Bar{
    int a;
    public static Bar create(int anA){
        return new Bar(anA);
    }
}
A Study of Refactoring

class Foo{
}

class Bar{
    int a;
    public Bar(int anA) {
        this.a = anA;
    }
}

class Bar{
    int a;
    private Bar(int anA) {
        this.a = anA;
    }`
    public static Bar create(int anA) {
        return new Bar(anA);
    }
}
A Study of Refactoring

```java
class Foo{
}

class Bar{
    int a;
    public Bar(int anA){
        this.a = anA;
    }
}

class Bar{
    int a;
    private Bar(int anA){
        this.a = anA;
    }
    public static Bar create(int anA){
        return new Bar(anA);
    }
}

...
## A Study of Refactoring

### Study Technique
- Compared refactoring in code to refactoring tool history
- When tools are used, looked at how they were used

### Study Participants
- 2 small development teams
- Around 3 years of development history each
- Eclipse users
- Looked through 40 CVS commits per team
Do refactoring tools really support the kinds of refactorings that people want to do?

Found 287 refactorings
70 have no tool support in Eclipse (24%)

Unsupported refactorings:
– Mostly remove dead code (remove exception, unused method, cast...)
– Significant modifier changing (e.g., public->private)
– Swap statements
– Use List instead of Array
– Replace literal with constant

Conclusion: Refactoring tools largely support that types of refactorings people do.
Is the way that refactoring tools work really the way developers refactor?

**Tactic 1: Floss refactoring**  
Refactoring interspersed with other changes

**Tactic 2: Root-canal refactoring**  
Intense and protracted periods of refactoring
91% of refactoring occurred during floss refactoring.

Conclusion: Many refactoring tools don’t support the dominant refactoring tactic.
Do programmers use the features of refactoring tools?

<table>
<thead>
<tr>
<th>Refactoring Tool</th>
<th>Configuration Option</th>
<th>Default Value</th>
<th>Change Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Local Variable</td>
<td>Declare the</td>
<td></td>
<td>Toolsmiths 5% Mylyn 0%</td>
</tr>
</tbody>
</table>

Conclusion: Refactoring tools’ configuration options are often unchanged.
Findings

Do refactoring tools really support the kinds of refactorings that people want to do? **Yes.**

Is the way that refactoring tools work really the way developers refactor? **No.**

Do programmers use the features of refactoring tools? **No.**

How can we make tools fit?
Some Solutions

http://people.engr.ncsu.edu/ermurph3/pubs.html
“My hands start doing copy-paste without my active control. After a few seconds, I realize that this would have been easier to do with a refactoring tool. But I already started... and continue.”

Tool prototype written by Xi Ge, http://www4.ncsu.edu/~xge/
Conclusion

Emerging data helps us to reflect on how people really refactor

Reflecting allows us to make better tools