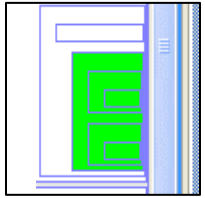
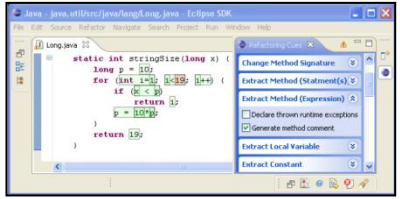
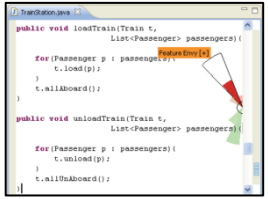
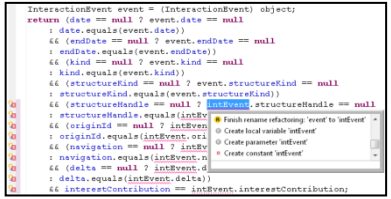
How are Developers Using Refactoring Tools?

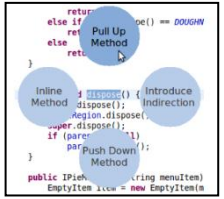
Emerson Murphy-Hill
North Carolina State University

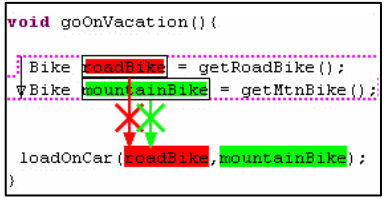












Refactoring Tools are Great!

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{

}
```

```
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);
 }
}
```

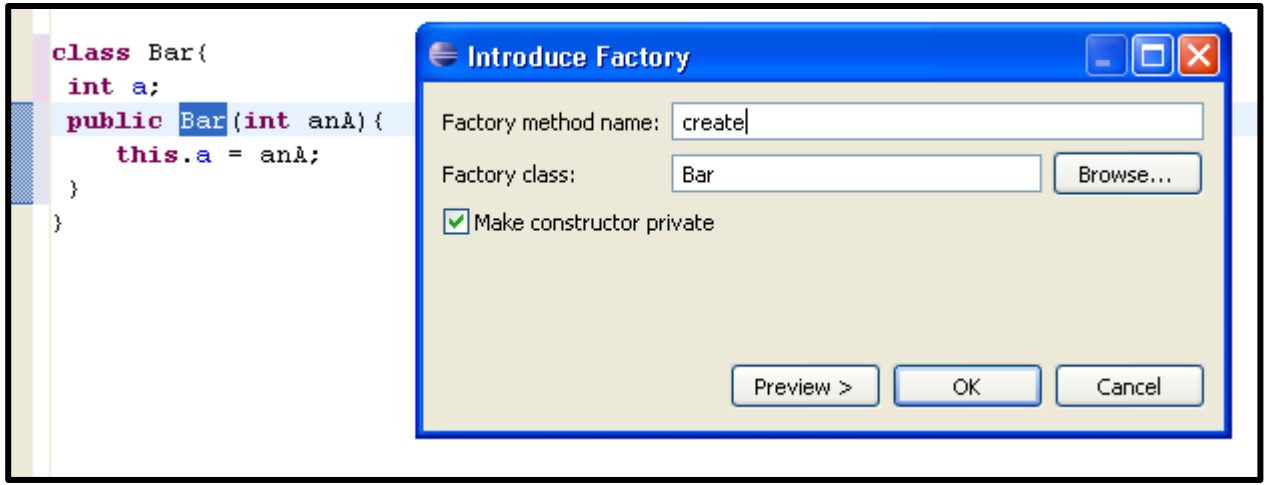
A Study of Refactoring

```
class Foo{

}
```

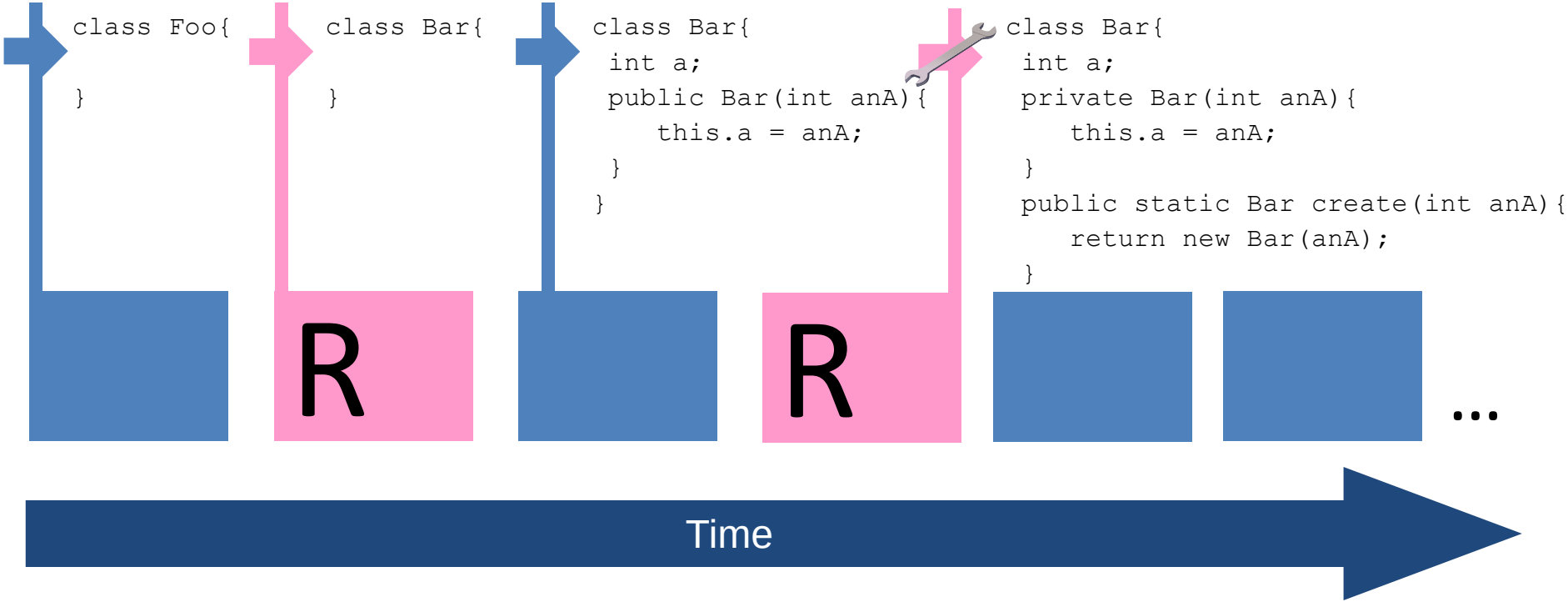
```
class Bar{

}
```

```
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



A Study of Refactoring

Study Technique

- Compared refactorings 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%)

Conclusion: Refactoring tools largely support that types of refactorings people do.

Unsupported refactorings:

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

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

Tactic 1: Floss refactoring

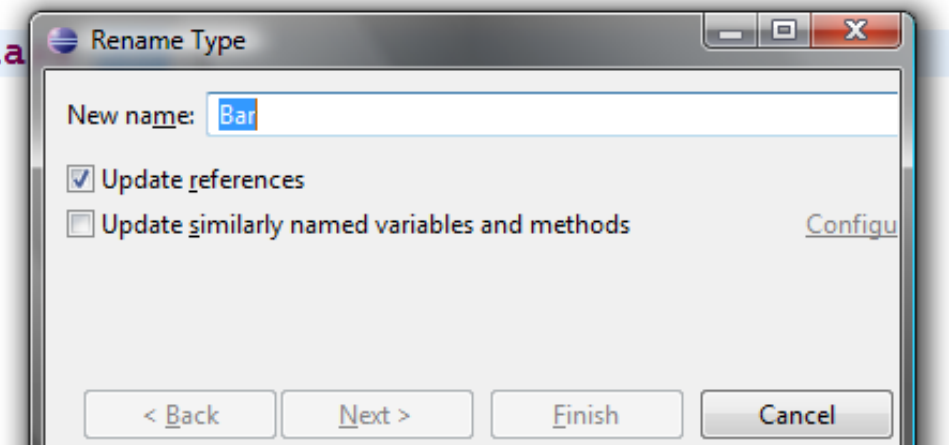
Refactoring interspersed with other changes


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

Enter new name, press **Enter** to refactor

Tactic 2: Root-canal refactoring

Intense and protracted periods of refactoring



91% of refactorings occurred during floss refactoring

Conclusion: Many refactoring tools don't support the dominant refactoring tactic.

Root Canal (Pure) Refactoring

Floss (Mixed) Refactoring

Do programmers **use the features** of refactoring tools?

Refactoring Tool	Configuration Option	Default Value	Change Frequency	
			Toolsmiths	*Mylyn*
Extract Local Variable	Declare the		5%	0%

Conclusion: Refactoring tools' configuration options are often unchanged.

Findings

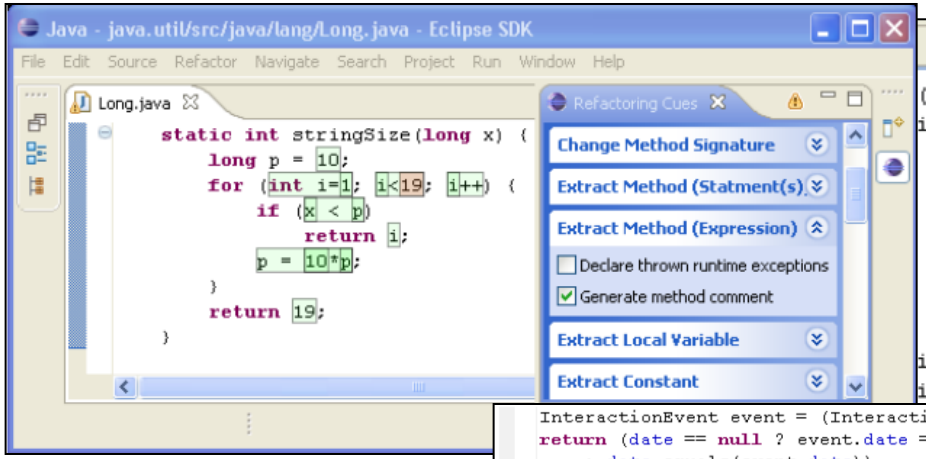
Do refactoring tools really support the kinds of refactorings that people want to do? **Yes.**

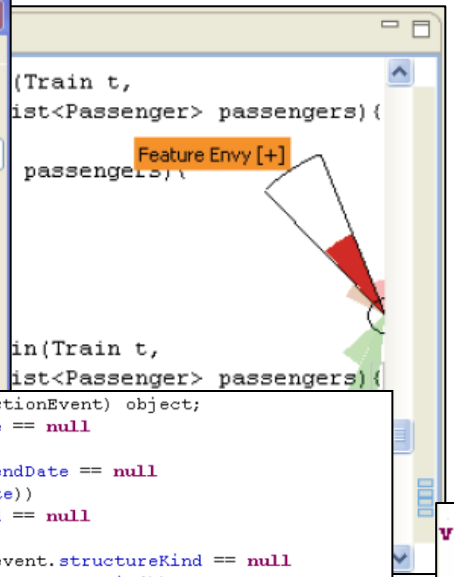
Is the way that refactoring tools work really the way developers refactor? **No.**

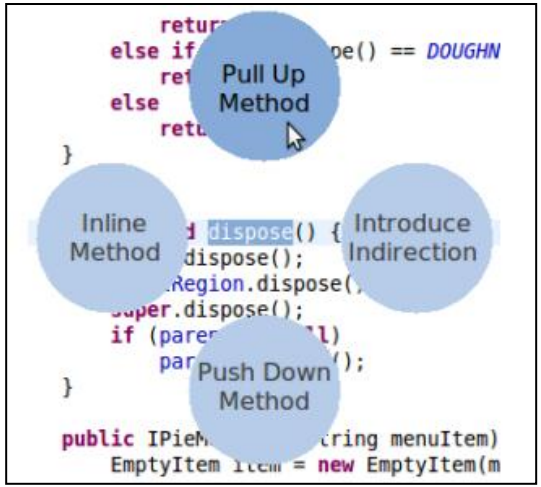
Do programmers use the features of refactoring tools? **No.**

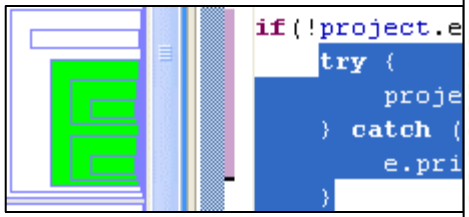
How can we make tools fit?

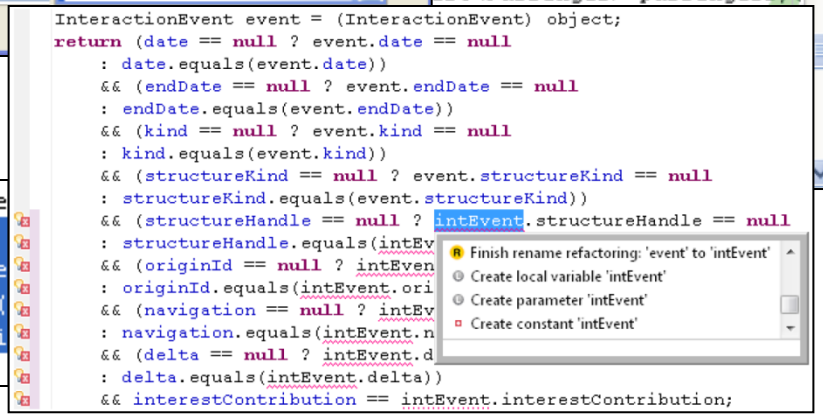
Some Solutions

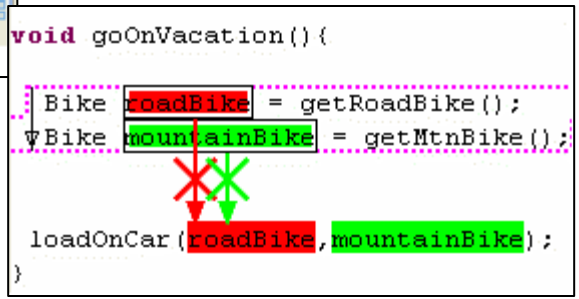












http://people.engr.ncsu.edu/ermurph3/pubs.html

Supporting Manual Refactoring

"My hand
copy-past
active cor
seconds,
would hav
to do with
[tool]. Bu
started…
and conti

```
InteractionEvent event = (InteractionEvent) object;
return (date == null ? event.date == null
    : date.equals(event.date))
    && (endDate == null ? event.endDate == null
    : endDate.equals(event.endDate))
    && (kind == null ? event.kind == null
    : kind.equals(event.kind))
    && (structureKind == null ? event.structureKind == null
    : structureKind.equals(event.structureKind))
    && (structureHandle == null ? intEvent.structureHandle == null
    : structureHandle.equals(intEv
    && (originId == null ? intEven
    : originId.equals(intEvent.ori
    && (navigation == null ? intEv
    : navigation.equals(intEvent.n
    && (delta == null ? intEvent.d
    : delta.equals(intEvent.delta))
    && interestContribution == intEvent.interestContribution;
```

Finish rename refactoring: 'event' to 'intEvent'
Create local variable 'intEvent'
Create parameter 'intEvent'
Create constant 'intEvent'

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