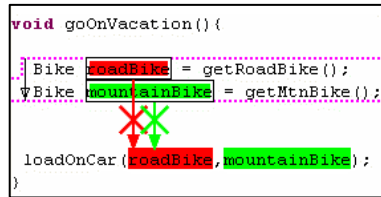
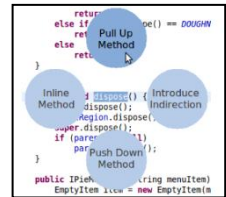
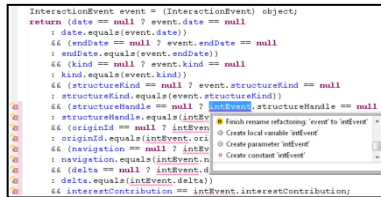
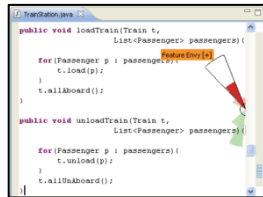
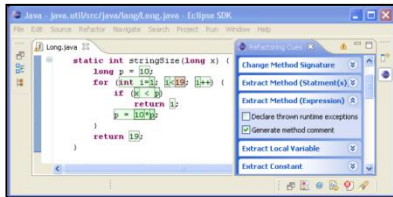
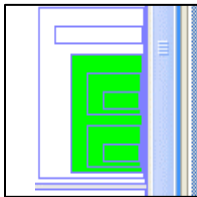


How are Developers Using Refactoring Tools?

Emerson Murphy-Hill
North Carolina State University



Refactoring Tools are Great!

They do what you'd do anyway, but:

- Faster
- Guaranteed Safety

But...

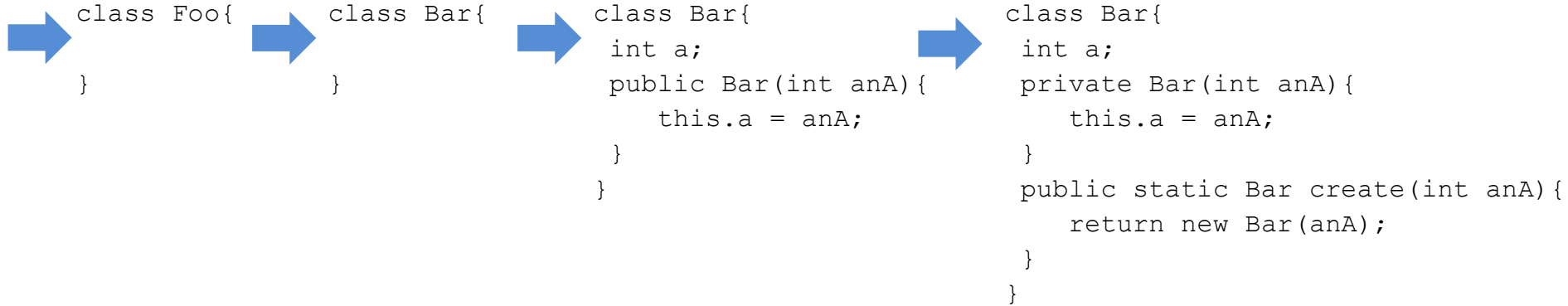
Do refactoring tools really support the **kinds** of refactorings that people want to do?

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



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);
 }
}

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

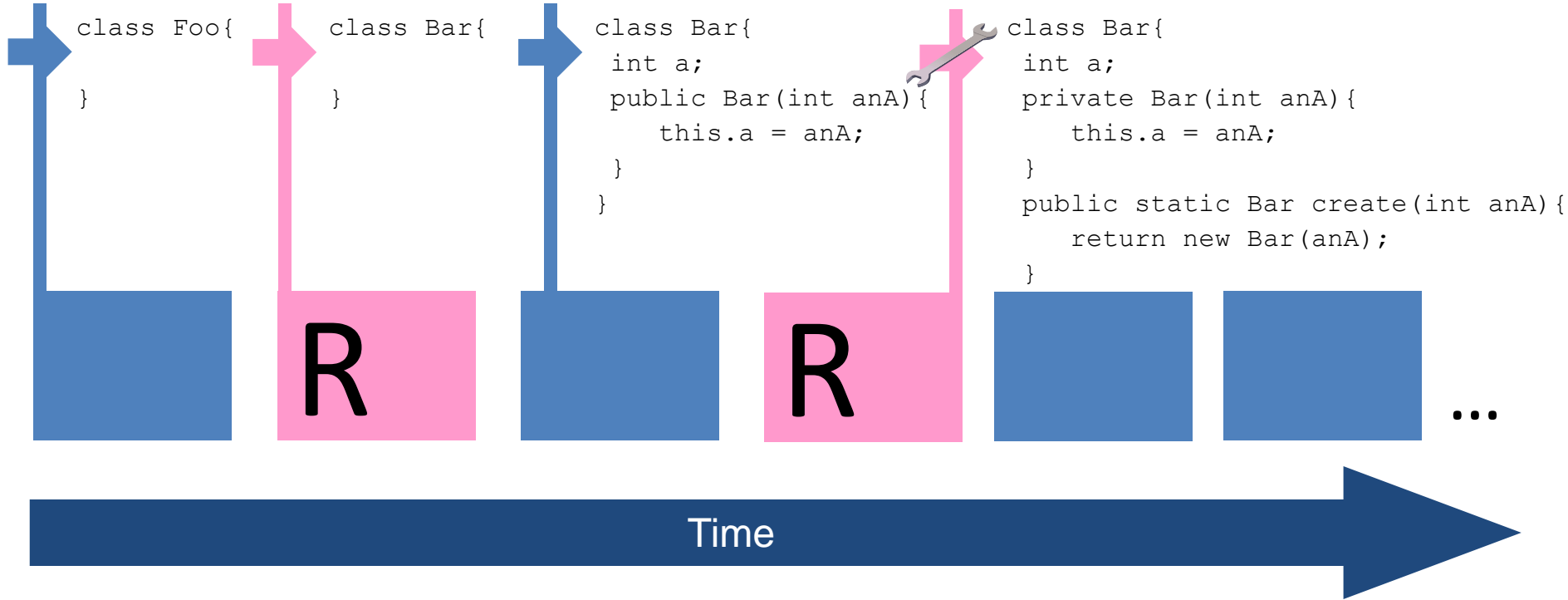
Introduce Factory

Factory method name: create

Factory class: Bar

Make constructor private

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 modifier 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

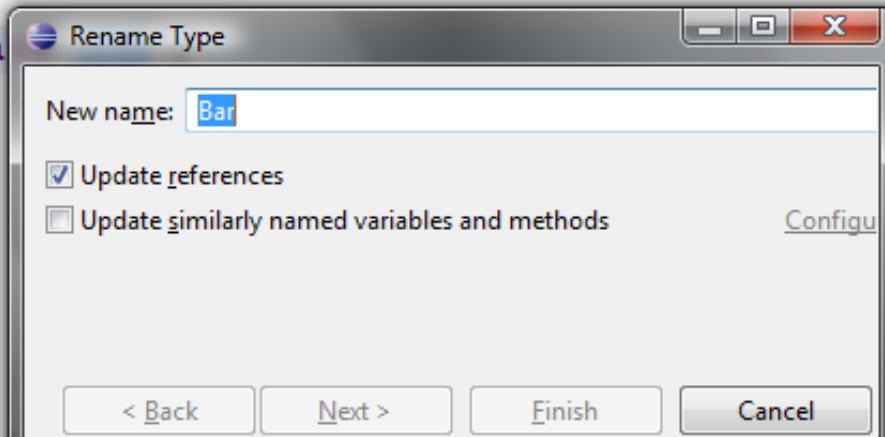
Refactoring interspersed with other changes



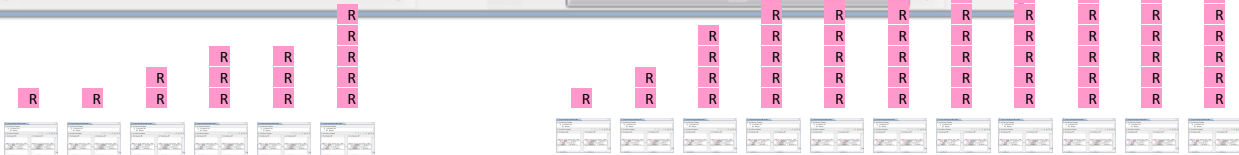
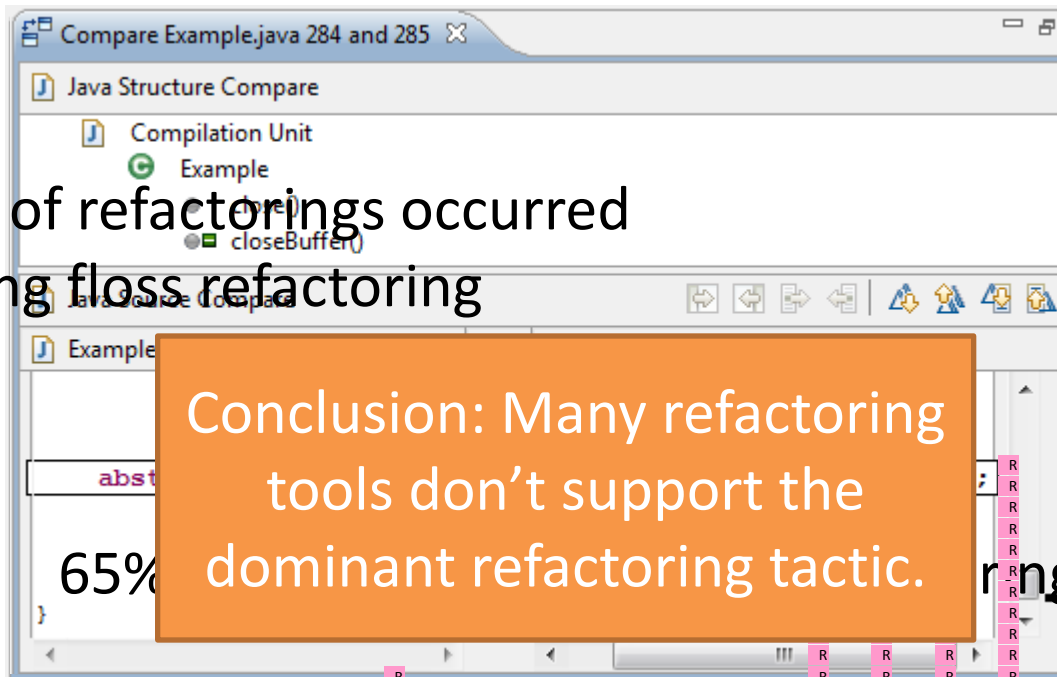
```
class Bar {  
    private Bar(int anA) {  
        this.a = anA;  
    }  
    public static Bar create(int anA)  
        return new Bar(anA);  
}
```

Tactic 2: Root-canal refactoring

Intense and protracted periods of refactoring



91% of refactorings occurred during floss refactoring



Root Canal
(Pure)
Refactoring



Floss
(Mixed)
Refactoring

Do programmers use the features of refactoring tools?

Refactoring Tool	Configuration Option	Default Value	Change Frequency	
			<i>Toolsmiths</i>	<i>Mylyn</i>
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

Java - java.util/src/java/lang/Long.java - Eclipse SDK

```

File Edit Source Refactor Navigate Search Project Run Window Help

Long.java
static int stringSize(long x) {
    long p = 10;
    for (int i=1; i<19; i++) {
        if (x < p)
            return i;
        p = 10*p;
    }
    return 19;
}
    
```

Refactoring Cues

- Change Method Signature
- Extract Method (Statement(s))
- Extract Method (Expression)
- Extract Local Variable
- Extract Constant

Declare thrown runtime exceptions
 Generate method comment

```

(Train t,
List<Passenger> passengers) {
    passengers;
}

in(Train t,
List<Passenger> passengers) {
    
```

Feature Envoy [+]

```

return ...
else if ... == DOUGHN
return ...
else
return ...
}

dispose() {
    Region.dispose();
    super.dispose();
    if (parent != null)
        parent.dispose();
}

public IPieMenu(String menuItem)
    EmptyItem item = new EmptyItem(m
    
```

Pull Up Method

Inline Method

Introduce Indirection

Push Down Method

```

if (!project.e
try {
    proje
} catch (
    e.pri
    
```

```

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
    : 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'

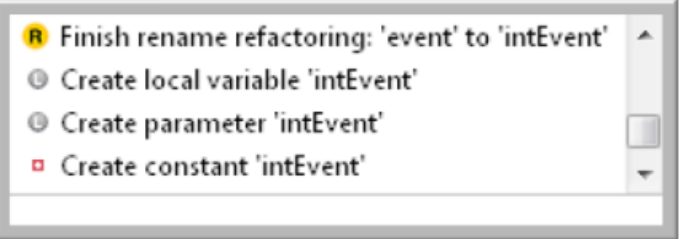
```

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

Supporting Manual Refactoring

“My hand
copy-past
active cor
seconds,
would ha
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(intEvent.structureHandle))  
    && (originId == null ? intEvent.originId == null  
    : originId.equals(intEvent.originId))  
    && (navigation == null ? intEvent.navigation == null  
    : navigation.equals(intEvent.navigation))  
    && (delta == null ? intEvent.delta == null  
    : delta.equals(intEvent.delta))  
    && interestContribution == intEvent.interestContribution;
```



Conclusion

Emerging data helps us to reflect on how people really refactor

Reflecting allows us to make better tools

