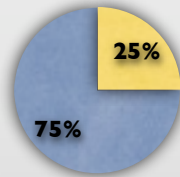


Spyware-ridden software development

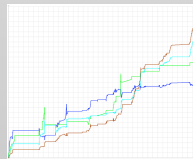
SpyWare is a research prototype built for my Ph.D.



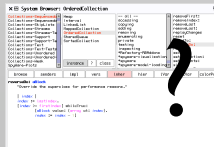
Context



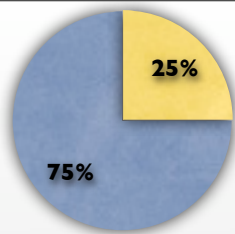
What is SpyWare?



Results & demo

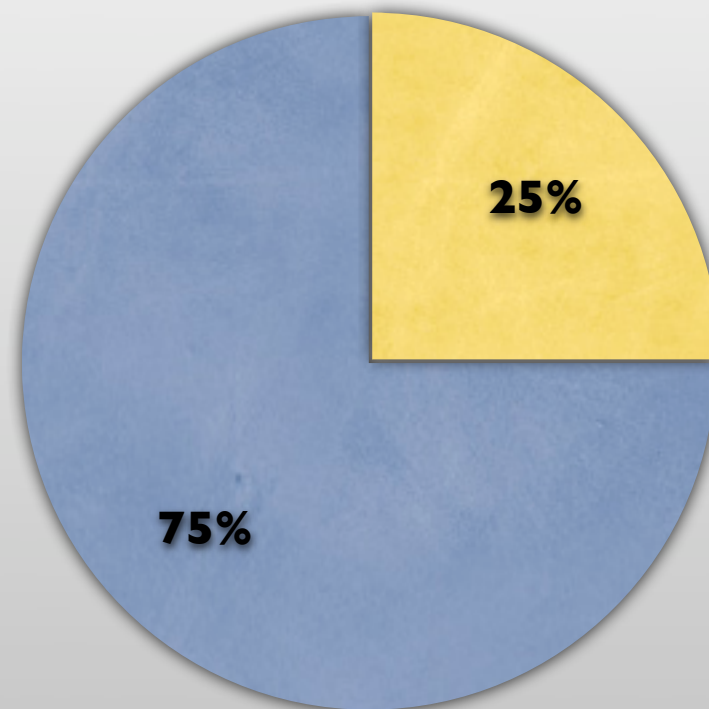


Further possibilities



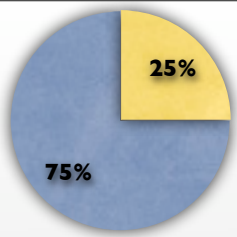
More than 3/4 of the cost of software is maintenance*

“A program that is used in a real-world environment must change, or become progressively less useful in that environment.”
– Lehman’s laws

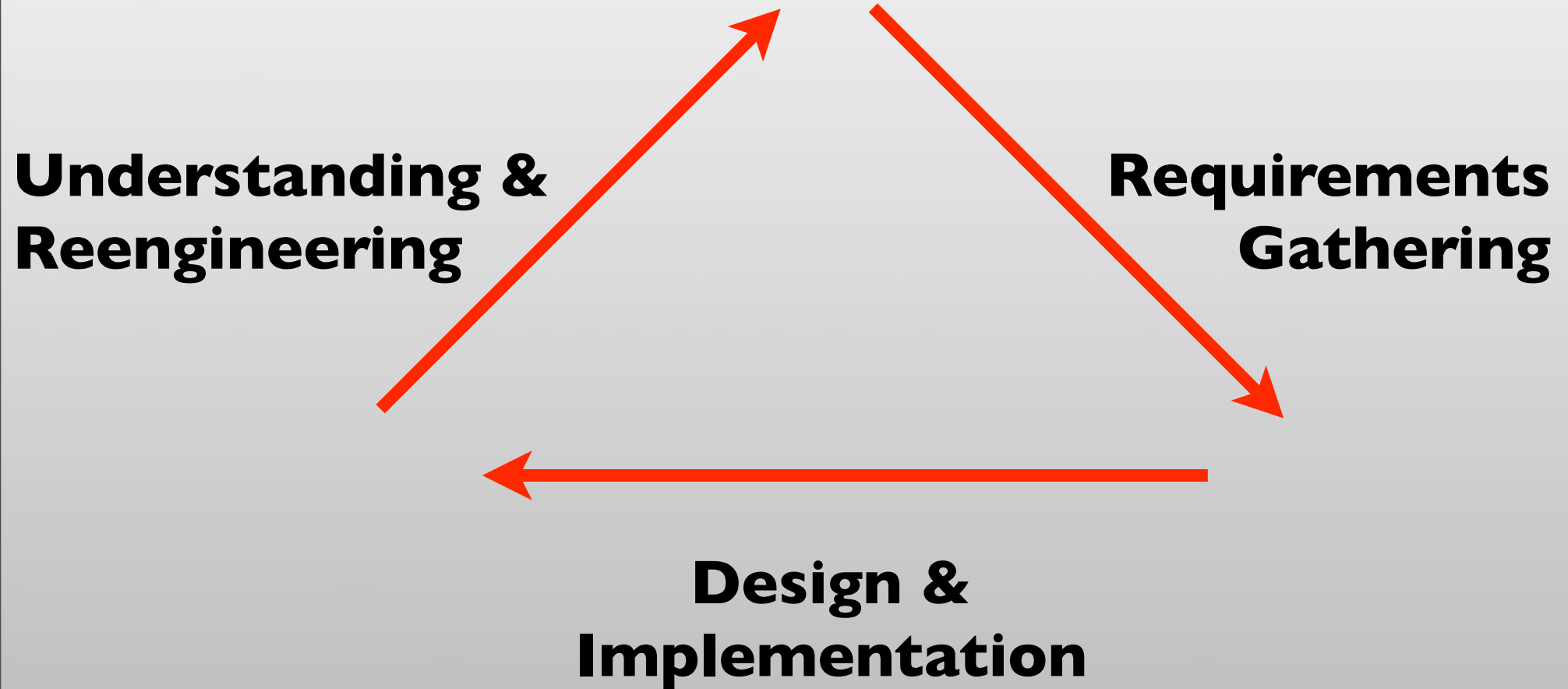


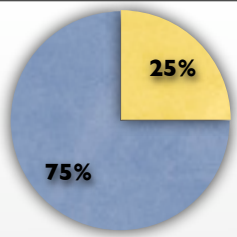
● initial effort
● maintenance

* from: <http://www.cs.jyu.fi/~koskinen/smcosts.htm>



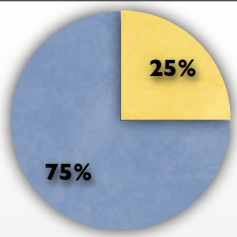
The lifecycle of software



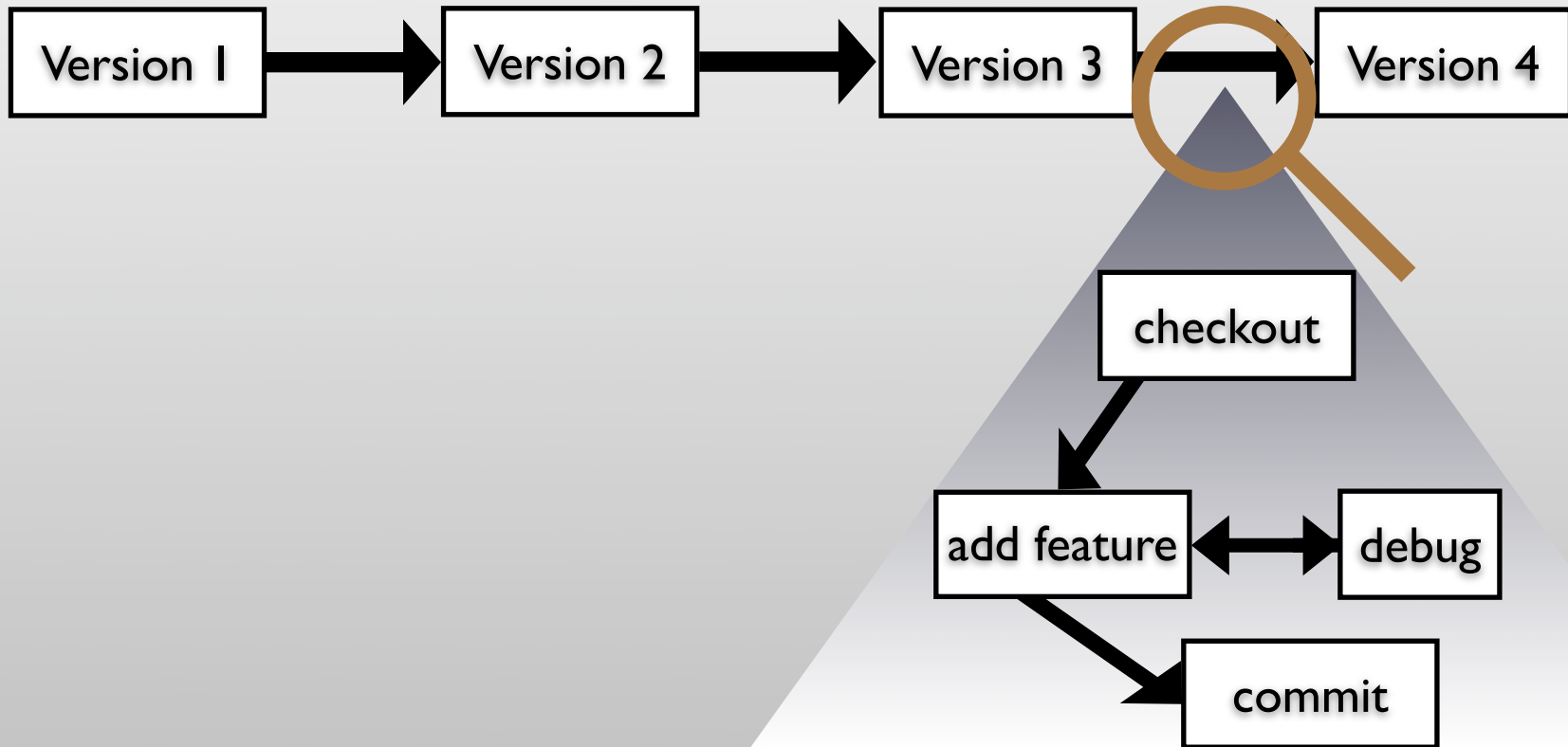


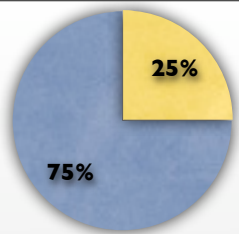
Software development is incremental



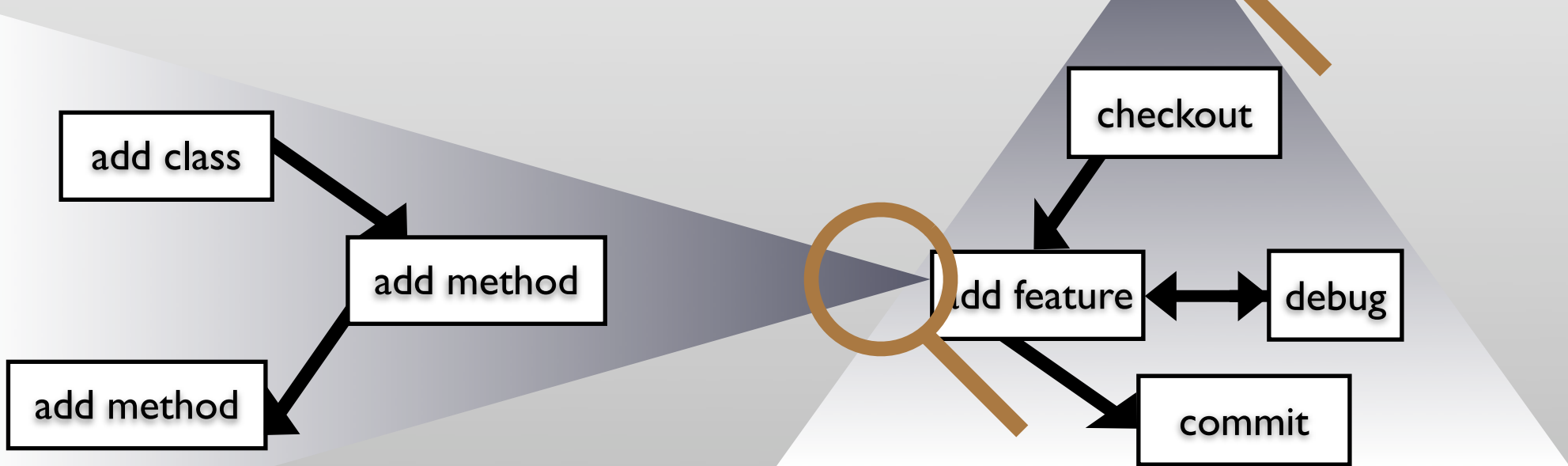
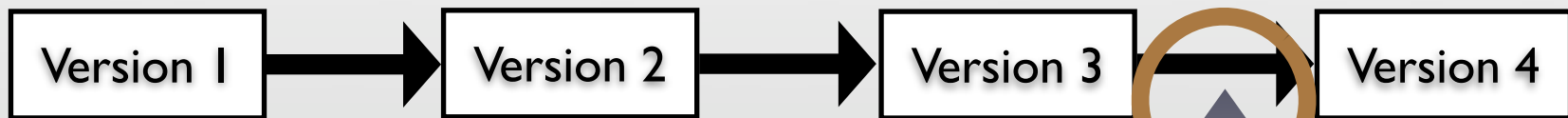


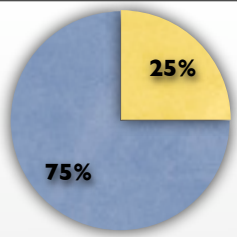
Software development is incremental



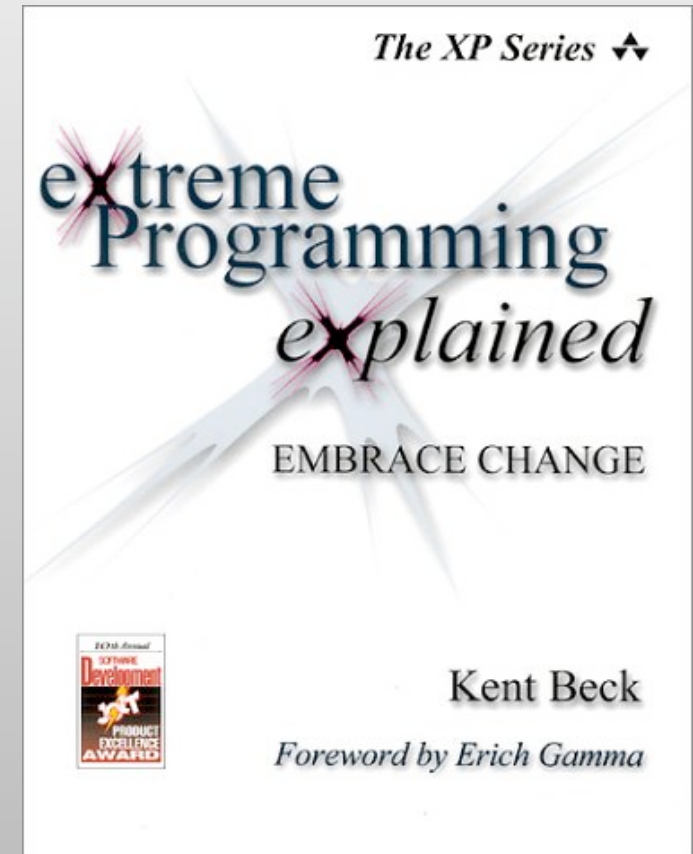
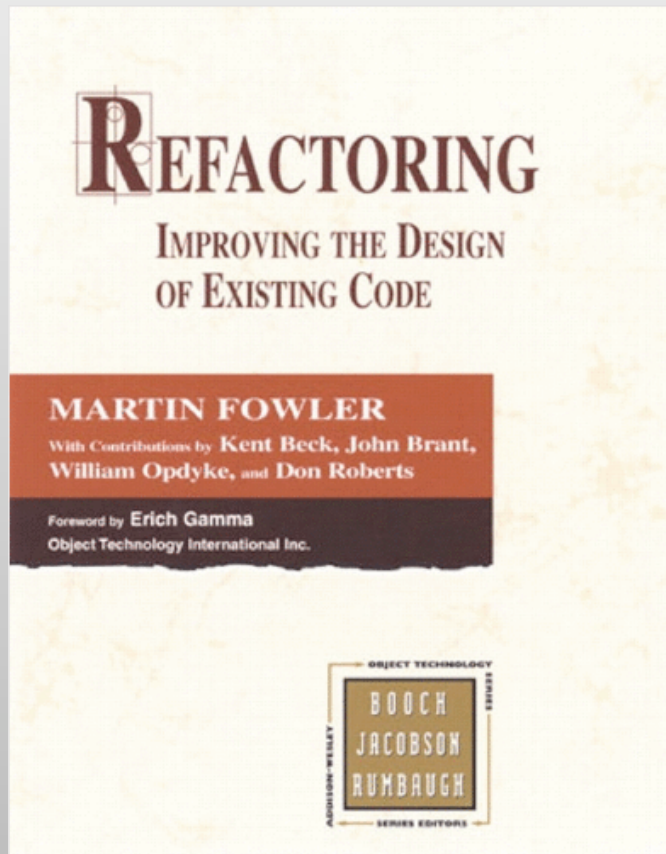


Software development is incremental

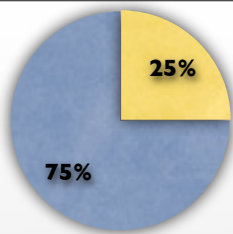




Languages and methodologies target this problem



The holy trinity?



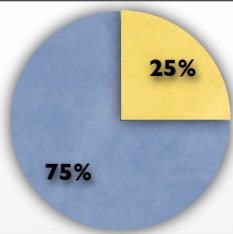
The change rate of the software increases

It is easier to lose reference points

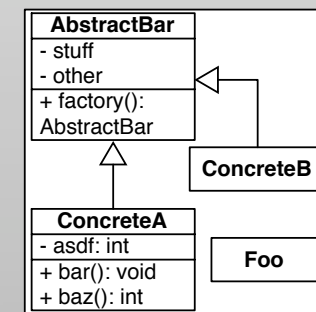
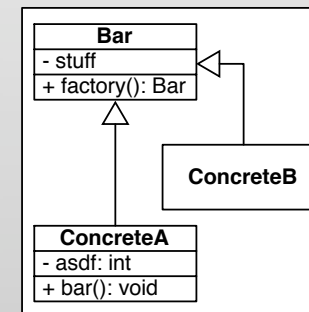
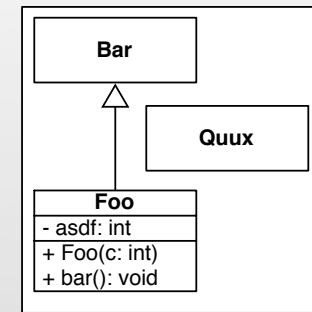
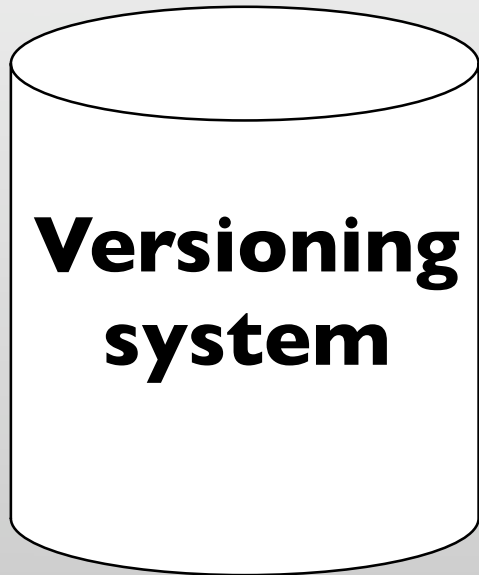
Evolvability and understandability are at odds

XP states that the software is in maintenance 100% of the time ...

Are conventional reengineering tools adapted to agile development?

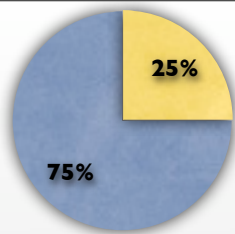


Software evolution analysis helps reengineering

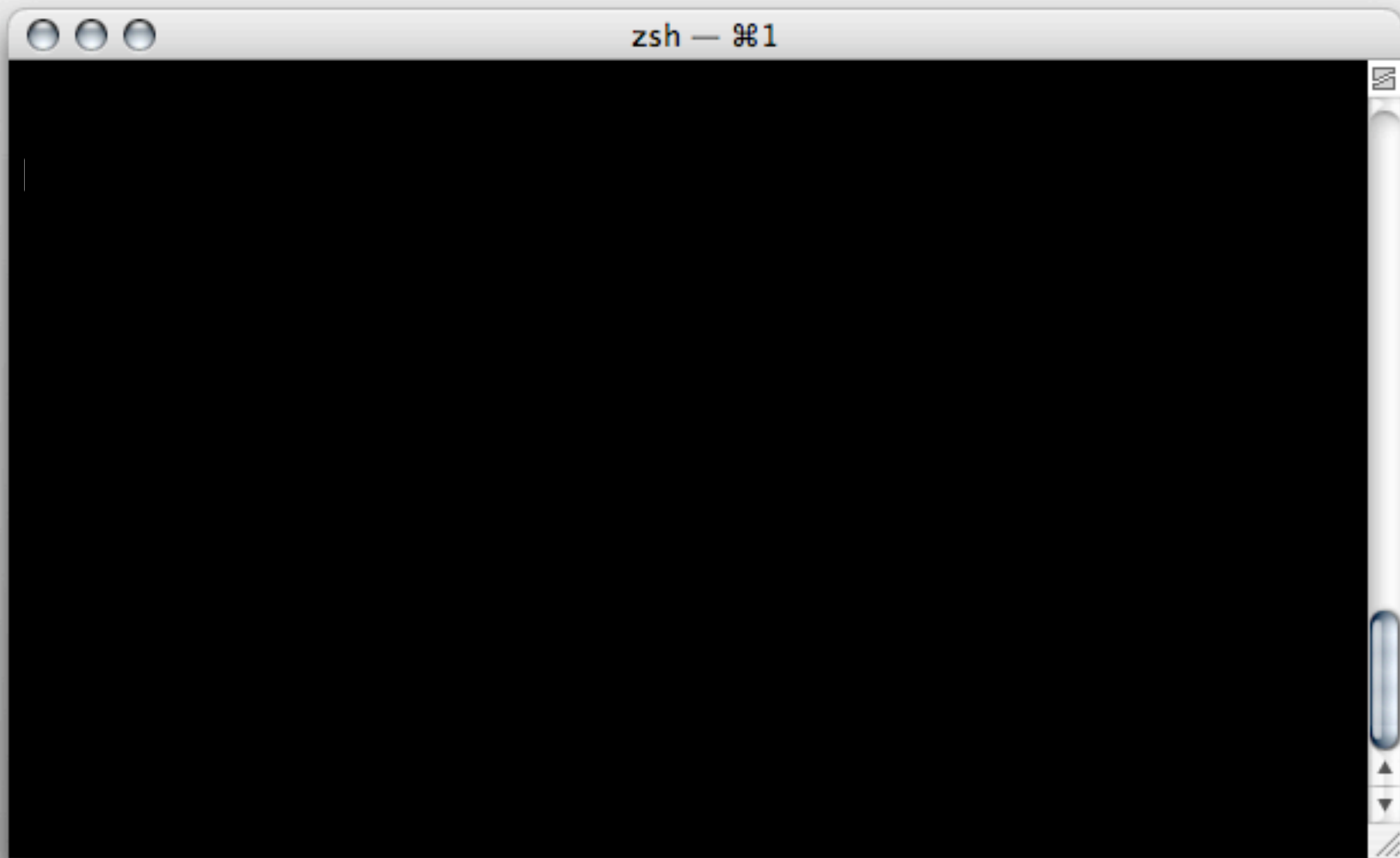


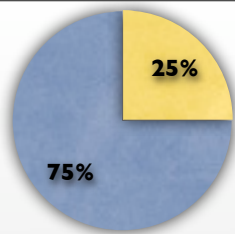
Metrics & trends

History holds useful information



Versioning systems **lose** information

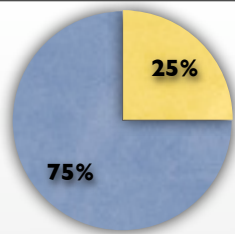




Versioning systems **lose** information

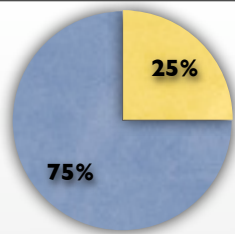
A terminal window titled "zsh — 1" with a black background and green text. The prompt ">" is followed by the command "cv update".

```
zsh — 1  
> cv update
```



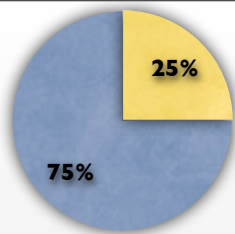
Versioning systems **lose** information

```
zsh — 981  
  
> cvs update  
> vim Foo.cc
```



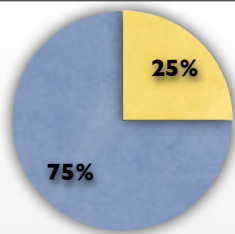
Versioning systems **lose** information

```
zsh — 981  
  
> cvs update  
> vim Foo.cc  
(some work done...)
```



Versioning systems **lose** information

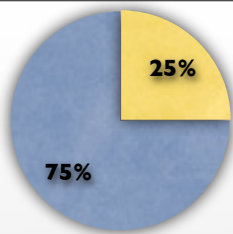
```
zsh — 981  
  
> cvs update  
> vim Foo.cc  
(some work done...)  
> cvs commit
```



Versioning systems **lose** information

```
zsh — 981  
> cvs update  
> vim Foo.cc  
(some work done...)  
> cvs commit
```





Our paranoid programmer saves every 5 minutes

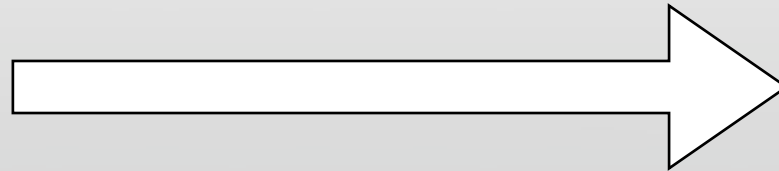
```
class Foo {
  public int x;
  public int y;

  public doFoo() {
    blah.blah(blah);
    z = x + y;
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }

  public quux() {
    return y + 4;
  }

  public asdf() {
    return x * 8 + y;
  }
}

f = new Foo();
f.doFoo();
print f.x + f.y;
```



```
class Foo {
  private int x;
  private int y;

  public getX() { return x; }
  public setX(newX) { x = newX; }

  public getY() { return y; }
  public setY(newY) { y = newY; }

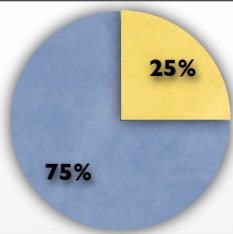
  public baz() {
    blah.blah(blah);
    z = getX() + getY();
    return bar(z);
  }

  public quux() {
    return getY() + 4;
  }

  public asdf() {
    return getX() * 8 + getY();
  }

  private bar(z) {
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }
}

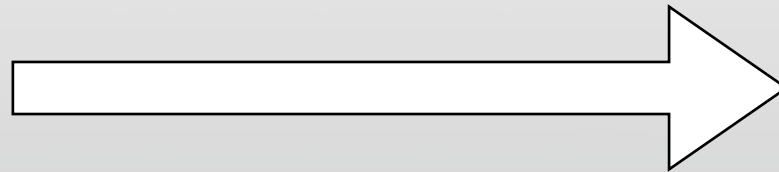
f = new Foo();
f.baz();
print f.getX() + f.getY();
```



Our paranoid programmer saves every 5 minutes

```
class Foo {  
    public int x;  
    public int y;  
  
    public doFoo() {  
        blah.blah(blah);  
        z = x + y;  
        blu = blu * 2;  
        t = blurg(z);  
        bli[t] = blu;  
        return t;  
    }  
  
    public quux() {  
        return y + 4;  
    }  
  
    public asdf() {  
        return x * 8 + y;  
    }  
}
```

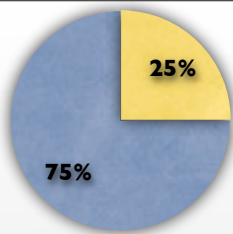
```
f = new Foo();  
f.doFoo();  
print f.x + f.y;
```



I.Extract method

```
class Foo {  
    private int x;  
    private int y;  
  
    public getX() { return x; }  
    public setX(newX) { x = newX; }  
  
    public getY() { return y; }  
    public setY(newY) { y = newY; }  
  
    public baz() {  
        blah.blah(blah);  
        z = getX() + getY();  
        return bar(z);  
    }  
  
    public quux() {  
        return getY() + 4;  
    }  
  
    public asdf() {  
        return getX() * 8 + getY();  
    }  
  
    private bar(z) {  
        blu = blu * 2;  
        t = blurg(z);  
        bli[t] = blu;  
        return t;  
    }  
}
```

```
f = new Foo();  
f.baz();  
print f.getX() + f.getY();
```



Our paranoid programmer saves every 5 minutes

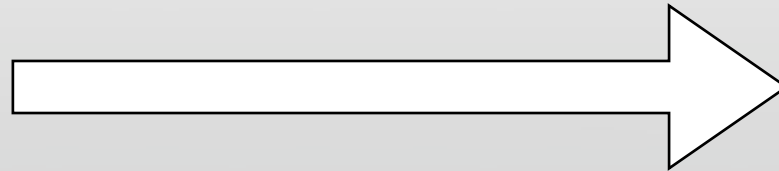
```
class Foo {
  public int x;
  public int y;

  public doFoo() {
    blah.blah(blah);
    z = x + y;
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }

  public quux() {
    return y + 4;
  }

  public asdf() {
    return x * 8 + y;
  }
}

f = new Foo();
f.doFoo();
print f.x + f.y;
```



I.Extract method

```
class Foo {
  private int x;
  private int y;

  public getX() { return x; }
  public setX(newX) { x = newX; }

  public getY() { return y; }
  public setY(newY) { y = newY; }

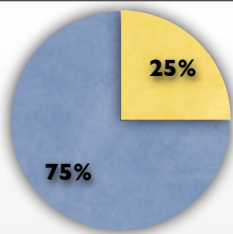
  public baz() {
    blah.blah(blah);
    z = getX() + getY();
    return bar(z);
  }

  public quux() {
    return getY() + 4;
  }

  public asdf() {
    return getX() * 8 + getY();
  }

  private bar(z) {
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }
}

f = new Foo();
f.baz();
print f.getX() + f.getY();
```



Our paranoid programmer saves every 5 minutes

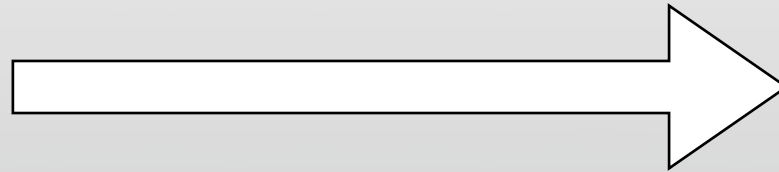
```
class Foo {
  public int x;
  public int y;

  public doFoo() {
    blah.blah(blah);
    z = x + y;
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }

  public quux() {
    return y + 4;
  }

  public asdf() {
    return x * 8 + y;
  }
}

f = new Foo();
f.doFoo();
print f.x + f.y;
```



1. Extract method
2. Rename method

```
class Foo {
  private int x;
  private int y;

  public getX() { return x; }
  public setX(newX) { x = newX; }

  public getY() { return y; }
  public setY(newY) { y = newY; }

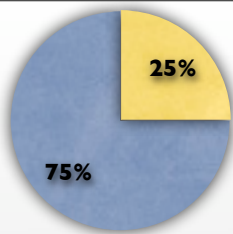
  public baz() {
    blah.blah(blah);
    z = getX() + getY();
    return bar(z);
  }

  public quux() {
    return getY() + 4;
  }

  public asdf() {
    return getX() * 8 + getY();
  }

  private bar(z) {
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }
}

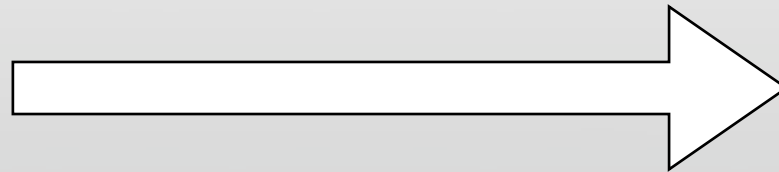
f = new Foo();
f.baz();
print f.getX() + f.getY();
```



Our paranoid programmer saves every 5 minutes

```
class Foo {  
    public int x;  
    public int y;  
  
    public doFoo() {  
        blah.blah(blah);  
        z = x + y;  
        blu = blu * 2;  
        t = blurg(z);  
        bli[t] = blu;  
        return t;  
    }  
  
    public quux() {  
        return y + 4;  
    }  
  
    public asdf() {  
        return x * 8 + y;  
    }  
}
```

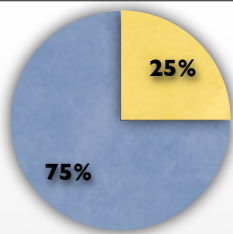
```
f = new Foo();  
f.doFoo();  
print f.x + f.y;
```



1. Extract method
2. Rename method

```
class Foo {  
    private int x;  
    private int y;  
  
    public getX() { return x; }  
    public setX(newX) { x = newX; }  
  
    public getY() { return y; }  
    public setY(newY) { y = newY; }  
  
    public baz() {  
        blah.blah(blah);  
        z = getX() + getY();  
        return bar(z);  
    }  
  
    public quux() {  
        return getY() + 4;  
    }  
  
    public asdf() {  
        return getX() * 8 + getY();  
    }  
  
    private bar(z) {  
        blu = blu * 2;  
        t = blurg(z);  
        bli[t] = blu;  
        return t;  
    }  
}
```

```
f = new Foo();  
f.baz();  
print f.getX() + f.getY();
```



Our paranoid programmer saves every 5 minutes

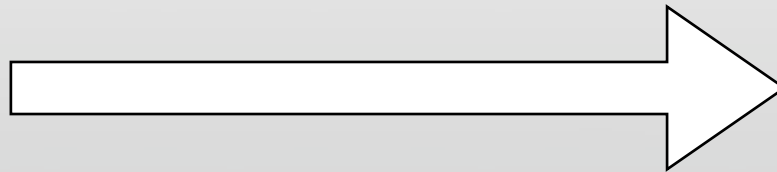
```
class Foo {
    public int x;
    public int y;

    public doFoo() {
        blah.blah(blah);
        z = x + y;
        blu = blu * 2;
        t = blurg(z);
        bli[t] = blu;
        return t;
    }

    public quux() {
        return y + 4;
    }

    public asdf() {
        return x * 8 + y;
    }
}

f = new Foo();
f.doFoo();
print f.x + f.y;
```



1. Extract method
2. Rename method
3. Create accessors

```
class Foo {
    private int x;
    private int y;

    public getX() { return x; }
    public setX(newX) { x = newX; }

    public getY() { return y; }
    public setY(newY) { y = newY; }

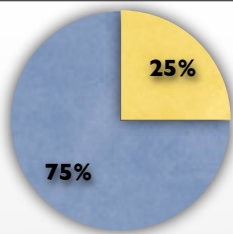
    public baz() {
        blah.blah(blah);
        z = getX() + getY();
        return bar(z);
    }

    public quux() {
        return getY() + 4;
    }

    public asdf() {
        return getX() * 8 + getY();
    }

    private bar(z) {
        blu = blu * 2;
        t = blurg(z);
        bli[t] = blu;
        return t;
    }
}

f = new Foo();
f.baz();
print f.getX() + f.getY();
```



Our paranoid programmer saves every 5 minutes

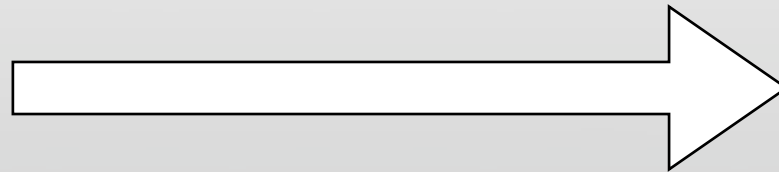
```
class Foo {
  public int x;
  public int y;

  public doFoo() {
    blah.blah(blah);
    z = x + y;
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }

  public quux() {
    return y + 4;
  }

  public asdf() {
    return x * 8 + y;
  }
}
```

```
f = new Foo();
f.doFoo();
print f.x + f.y;
```



1. Extract method
2. Rename method
3. Create accessors

```
class Foo {
  private int x;
  private int y;

  public getX() { return x; }
  public setX(newX) { x = newX; }

  public getY() { return y; }
  public setY(newY) { y = newY; }

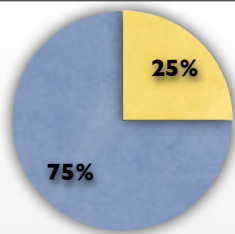
  public baz() {
    blah.blah(blah);
    z = getX() + getY();
    return bar(z);
  }

  public quux() {
    return getY() + 4;
  }

  public asdf() {
    return getX() * 8 + getY();
  }

  private bar(z) {
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }
}
```

```
f = new Foo();
f.baz();
print f.getX() + f.getY();
```



Our paranoid programmer saves every 5 minutes

```
class Foo {
    public int x;
    public int y;

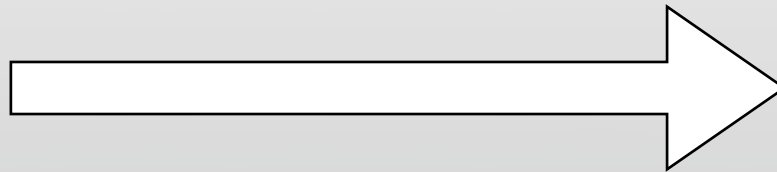
    public doFoo() {
        blah.blah(blah);
        z = x + y;
        blu = blu * 2;
        t = blurg(z);
        bli[t] = blu;
        return t;
    }

    public quux() {
        return y + 4;
    }

    public asdf() {
        return x * 8 + y;
    }
}

f = new Foo();
f.doFoo();
print f.x + f.y;
```

CVS: +18 / -11



1. Extract method
2. Rename method
3. Create accessors

```
class Foo {
    private int x;
    private int y;

    public getX() { return x; }
    public setX(newX) { x = newX; }

    public getY() { return y; }
    public setY(newY) { y = newY; }

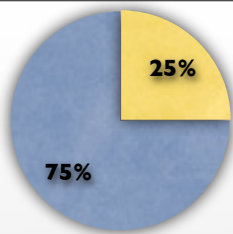
    public baz() {
        blah.blah(blah);
        z = getX() + getY();
        return bar(z);
    }

    public quux() {
        return getY() + 4;
    }

    public asdf() {
        return getX() * 8 + getY();
    }

    private bar(z) {
        blu = blu * 2;
        t = blurg(z);
        bli[t] = blu;
        return t;
    }
}

f = new Foo();
f.baz();
print f.getX() + f.getY();
```

Our paranoid programmer saves every 5 minutes

```
class Foo {
  public int x;
  public int y;

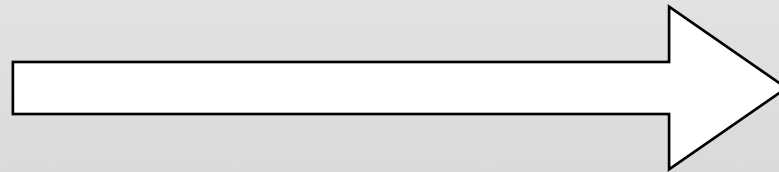
  public doFoo() {
    blah.blah(blah);
    z = x + y;
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }

  public quux() {
    return y + 4;
  }

  public asdf() {
    return x * 8 + y;
  }
}

f = new Foo();
f.doFoo();
print f.x + f.y;
```

CVS: +18 / -11



1. Extract method
2. Rename method
3. Create accessors

```
class Foo {
  private int x;
  private int y;

  public getX() { return x; }
  public setX(newX) { x = newX; }

  public getY() { return y; }
  public setY(newY) { y = newY; }

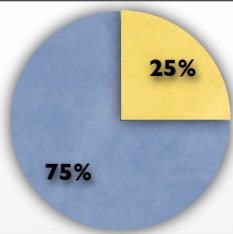
  public baz() {
    blah.blah(blah);
    z = getX() + getY();
    return bar(z);
  }

  public quux() {
    return getY() + 4;
  }

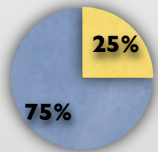
  public asdf() {
    return getX() * 8 + getY();
  }

  private bar(z) {
    blu = blu * 2;
    t = blurg(z);
    bli[t] = blu;
    return t;
  }
}

f = new Foo();
f.baz();
print f.getX() + f.getY();
```



Recapitulation



Software evolution is hard but necessary



Agile practices make it change even faster



Versioning systems are inappropriate sources of information for SE tools

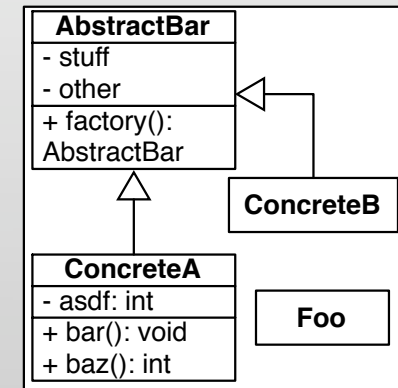
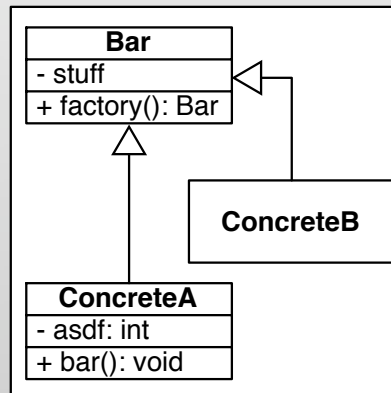


The principles behind SpyWare

- 1. Model changes, not versions**
- 2. Use the IDE, not the code repository**
- 3. Provide tools in the IDE**



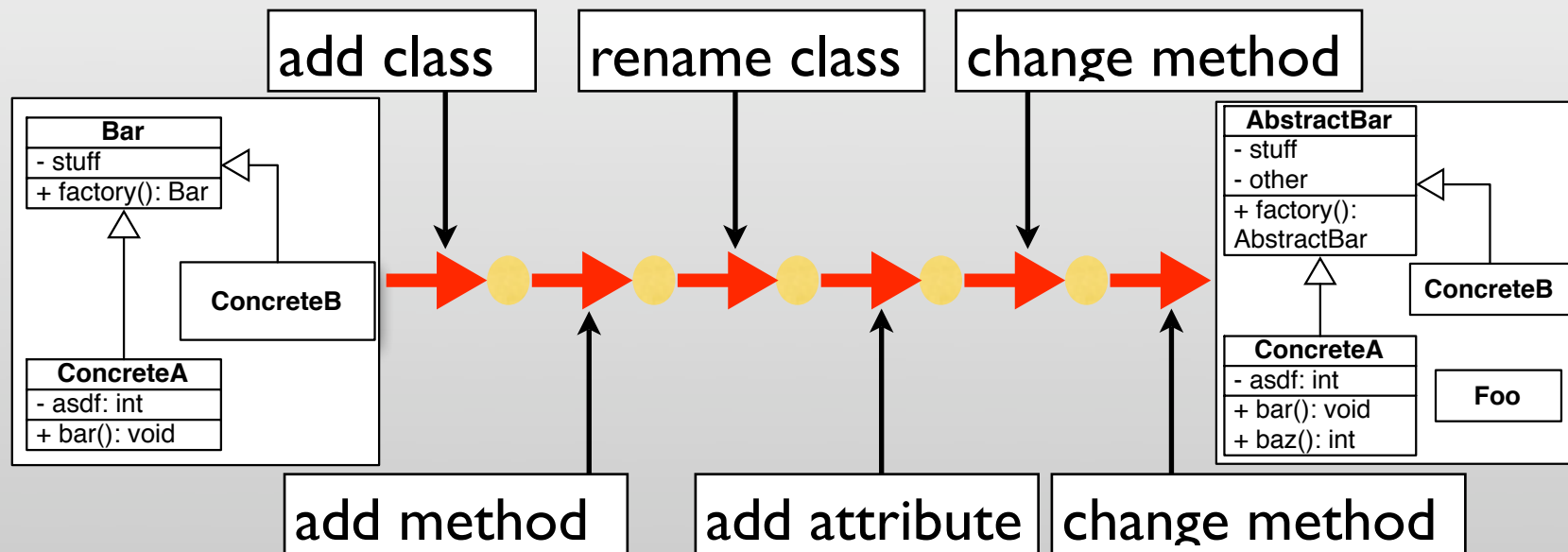
Modelling and capturing change information



Rather than storing versions, we record the semantic actions of the programmer



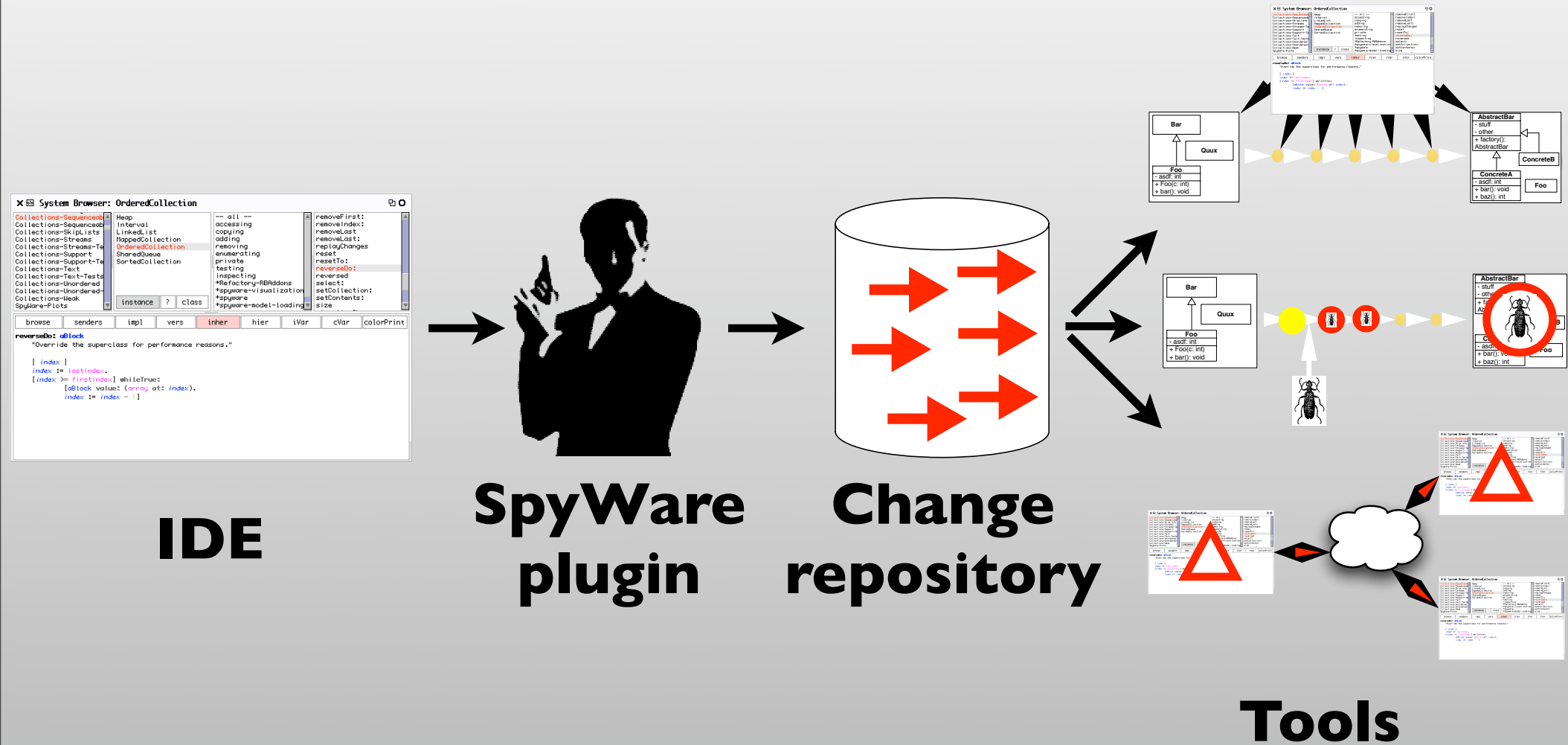
Modelling and capturing change information



Rather than storing versions, we record the semantic actions of the programmer



The general architecture of SpyWare





How does SpyWare work?

System Browser: OrderedCollection

| | | | |
|------------------------|--------------------------|------------------------|-------------------|
| Collections-Sequenceab | Heap | -- all -- | removeFirst: |
| Collections-Sequenceab | Interval | accessing | removeIndex: |
| Collections-SkipLists | LinkedList | copying | removeLast |
| Collections-Streams | MappedCollection | adding | removeLast: |
| Collections-Streams-Te | OrderedCollection | removing | replayChanges |
| Collections-Support | SharedQueue | enumerating | reset |
| Collections-Support-Te | SortedCollection | private | resetTo: |
| Collections-Text | | testing | reverseDo: |
| Collections-Text-Tests | | inspecting | reversed |
| Collections-Unordered | | *Refactory-RBAddons | select: |
| Collections-Unordered | | *spyware-visualization | setCollection: |
| Collections-Weak | | *spyware | setContents: |
| SpyWare-Plots | | *spyware-model-loading | size |

instance ? class

browse senders impl vers **inher** hier iVar cVar colorPrint



How does SpyWare work?

System Browser: OrderedCollection

| | | | |
|------------------------|--------------------------|------------------------|-------------------|
| Collections-Sequenceab | Heap | -- all -- | removeFirst: |
| Collections-Sequenceab | Interval | accessing | removeIndex: |
| Collections-SkipLists | LinkedList | copying | removeLast |
| Collections-Streams | MappedCollection | adding | removeLast: |
| Collections-Streams-Te | OrderedCollection | removing | replayChanges |
| Collections-Support | SharedQueue | enumerating | reset |
| Collections-Support-Te | SortedCollection | private | resetTo: |
| Collections-Text | | testing | reverseDo: |
| Collections-Text-Tests | | inspecting | reversed |
| Collections-Unordered | | *Refactory-RBAddons | select: |
| Collections-Unordered | | *spyware-visualization | setCollection: |
| Collections-Weak | | *spyware | setContents: |
| SpyWare-Plots | | *spyware-model-loading | size |

instance ? class

browse senders impl vers **inher** hier iVar cVar colorPrint

myNewMethod: foo



How does SpyWare work?

System Browser: OrderedCollection

| | | | |
|------------------------|--------------------------|------------------------|-------------------|
| Collections-Sequenceab | Heap | -- all -- | removeFirst: |
| Collections-Sequenceab | Interval | accessing | removeIndex: |
| Collections-SkipLists | LinkedList | copying | removeLast |
| Collections-Streams | MappedCollection | adding | removeLast: |
| Collections-Streams-Te | OrderedCollection | removing | replayChanges |
| Collections-Support | SharedQueue | enumerating | reset |
| Collections-Support-Te | SortedCollection | private | resetTo: |
| Collections-Text | | testing | reverseDo: |
| Collections-Text-Tests | | inspecting | reversed |
| Collections-Unordered | | *Refactory-RBAddons | select: |
| Collections-Unordered | | *spyware-visualization | setCollection: |
| Collections-Weak | | *spyware | setContents: |
| SpyWare-Plots | | *spyware-model-loading | size |

instance ? class

browse senders impl vers **inher** hier iVar cVar colorPrint

```
myNewMethod: foo  
self bar: foo + 1.
```



How does SpyWare work?

System Browser: OrderedCollection

| | | | |
|------------------------|-------------------|------------------------|----------------|
| Collections-Sequenceab | Heap | -- all -- | removeFirst: |
| Collections-Sequenceab | Interval | accessing | removeIndex: |
| Collections-SkipLists | LinkedList | copying | removeLast |
| Collections-Streams | MappedCollection | adding | removeLast: |
| Collections-Streams-Te | OrderedCollection | removing | replayChanges |
| Collections-Support | SharedQueue | enumerating | reset |
| Collections-Support-Te | SortedCollection | private | resetTo: |
| Collections-Text | | testing | reverseDo: |
| Collections-Text-Tests | | inspecting | reversed |
| Collections-Unordered | | *Refactory-RBAddons | select: |
| Collections-Unordered | | *spyware-visualization | setCollection: |
| Collections-Weak | | *spyware | setContents: |
| SpyWare-Plots | | *spyware-model-loading | size |

instance ? class

browse senders impl vers **inher** hier iVar cVar colorPrint

```
myNewMethod: foo
self bar: foo + 1.
self baz: foo - 1.
```



How does SpyWare work?

Method compiled!

The screenshot shows the System Browser interface for the `OrderedCollection` class. The left pane lists various collection classes, with `OrderedCollection` selected. The middle pane shows the class hierarchy, including `Heap`, `Interval`, `LinkedList`, `MappedCollection`, `OrderedCollection`, `SharedQueue`, and `SortedCollection`. The right pane shows the methods of the class, including `removeFirst:`, `removeIndex:`, `removeLast:`, `removeLast:`, `replayChanges`, `reset`, `resetTo:`, `reverseDo:`, `reversed`, `select:`, `setCollection:`, `setContents:`, and `size`. The `reverseDo:` method is highlighted. Below the browser, a red box contains the following code snippet:

```
myNewMethod: foo  
self bar: foo + 1.  
self bar: foo - 1.
```



How does SpyWare work?

System Browser: OrderedCollection

| | | | |
|------------------------|--------------------------|------------------------|-------------------|
| Collections-Sequenceab | Heap | -- all -- | removeFirst: |
| Collections-Sequenceab | Interval | accessing | removeIndex: |
| Collections-SkipLists | LinkedList | copying | removeLast |
| Collections-Streams | MappedCollection | adding | removeLast: |
| Collections-Streams-Te | OrderedCollection | removing | replayChanges |
| Collections-Support | SharedQueue | enumerating | reset |
| Collections-Support-Te | SortedCollection | private | resetTo: |
| Collections-Text | | testing | reverseDo: |
| Collections-Text-Tests | | inspecting | reversed |
| Collections-Unordered | | *Refactory-RBAddons | select: |
| Collections-Unordered | | *spyware-visualization | setCollection: |
| Collections-Weak | | *spyware | setContents: |
| SpyWare-Plots | | *spyware-model-loading | size |

instance ? class

browse senders impl vers **inher** hier iVar cVar colorPrint

```
myNewMethod: foo
self bar: foo + 1.
self baz: foo - 1.
```



The change builder infers change information





The change builder infers change information

Method compiled!





The change builder infers change information

Is it a new one?

Yes!

Who? when?

Romain Robbes
@ 21h32





The change builder infers change information



OK...

create method

add to class

create variable

add to method

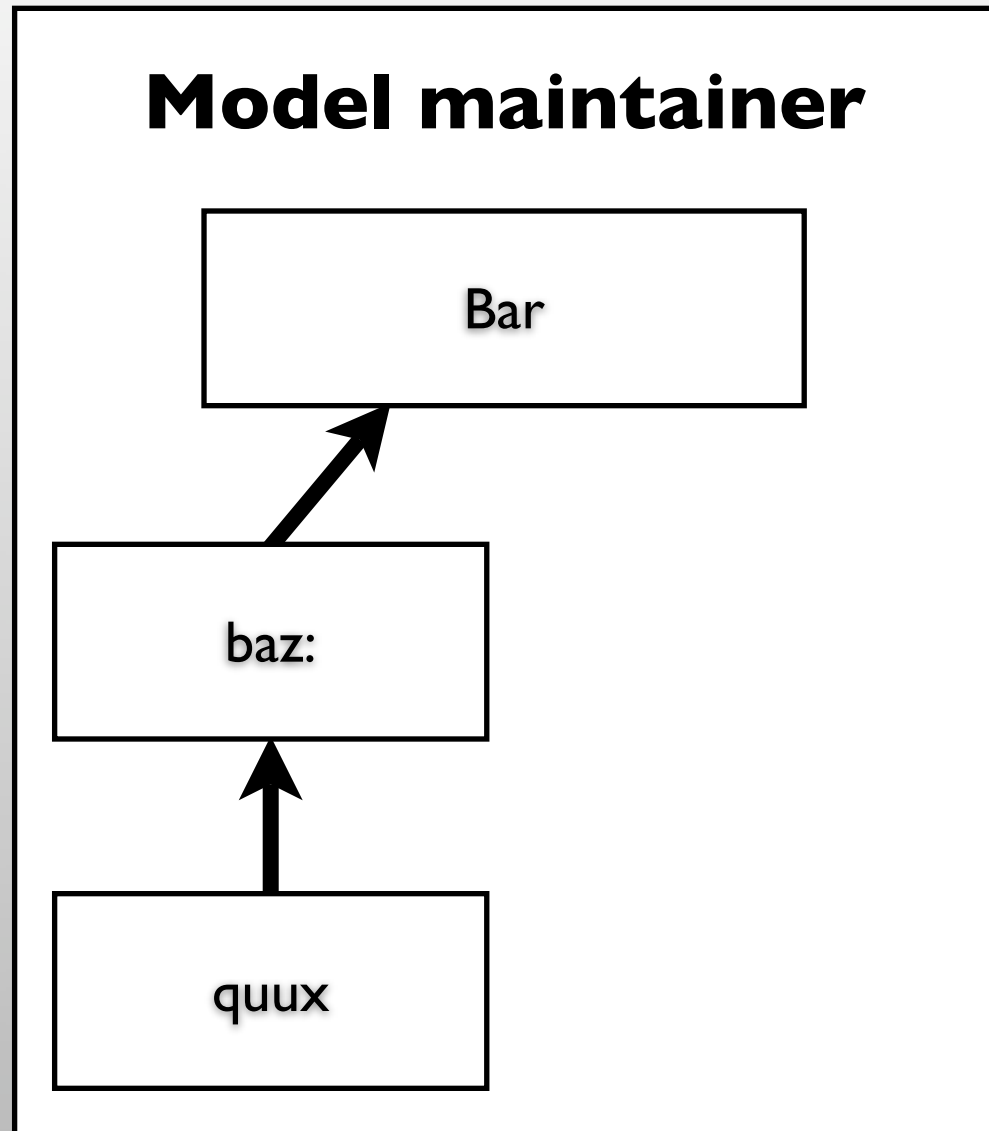


The change builder infers change information





Tools exploit the changes





Tools exploit the changes

create method

add to class

create variable

add to method

Model maintainer

Bar

baz:

quux



Tools exploit the changes

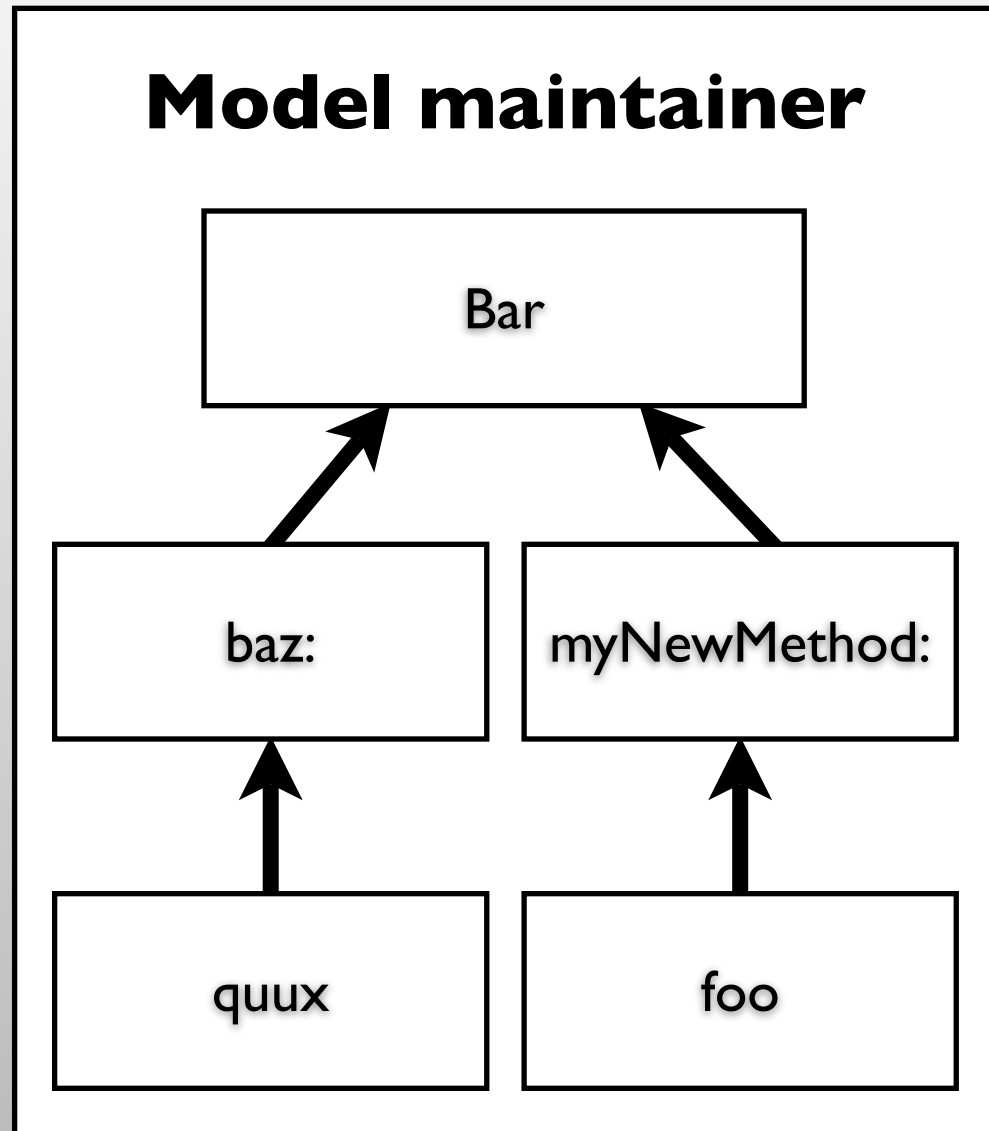
create method

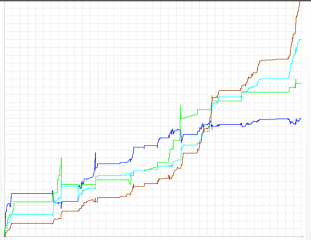
add to class

create variable

add to method

Model maintainer





SpyWare case studies:

Big brother is watching you!



```

x RB: Monitor
CollectionsTests-Arraye Delay
CollectionsTests-Sequen EventSensor
CollectionsTests-Stream EventSensorConstants
CollectionsTests-Text InputSensor
CollectionsTests-Unorde Monitor
Graphics-Display Objec MonitorDelay
Graphics-Primitives Mutex
GraphicsTests-Primitive MutexSet
SpyWare Process
Kernel-Processes instance ? class

-- all --
accessing
initialize-release
signaling-default
signaling-specific
synchronization
waiting-basic
waiting-specific
waiting-timeout
private

checkOwnerProcess
cleanup
critical:
defaultQueue
enter
exit
exitAndWaitInQueue:m
initialize
isOwnerProcess
privateCleanup
    
```

critical: aBlock
*"Critical section.
 Executes aBlock as a critical section. At any time, only one process can be executing code in a critical section.
 NOTE: All the following synchronization operations are only valid inside the critical section of the monitor!"*

| result |

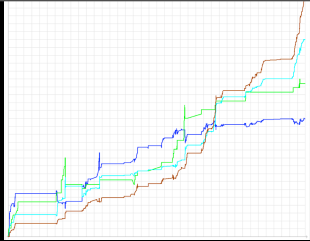
```

[self enter.
result := aBlock value] ensure: [self exit].
    
```

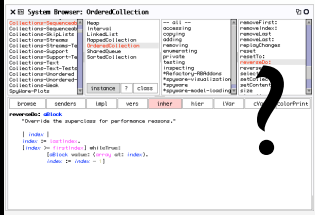
x9



Short SpyWare demo

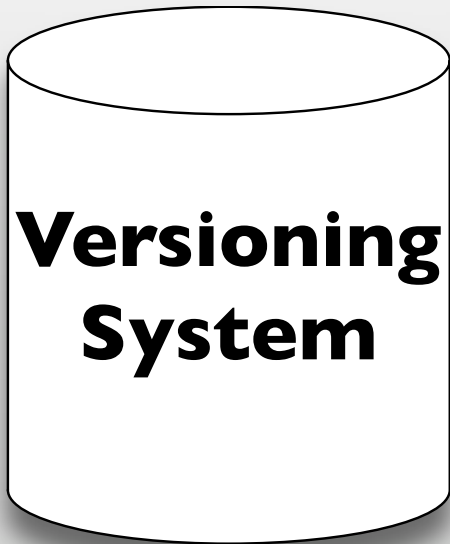
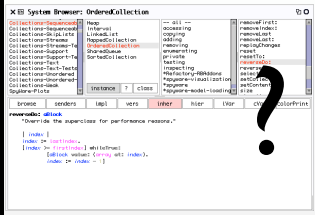


Future possibilities



Several tools can be implemented on top of this mechanism: here are a few.

Fine-grained software evolution analysis



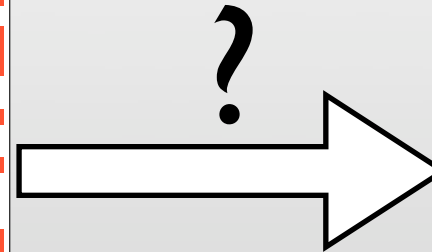
```
class Foo {
    public int x;
    public int y;

    public doFoo() {
        blah.blah(blah);
        z = x + y;
        blu = blu * 2;
        t = blurg(z);
        bli[] = blu;
        return t;
    }

    public quux() {
        return y + 4;
    }

    public asdf() {
        return x * 8 + y;
    }
}

f = new Foo();
f.doFoo();
print f.x + f.y;
```



```
class Foo {
    private int x;
    private int y;

    public getX() { return x; }
    public setX(newX) { x = newX; }

    public getY() { return y; }
    public setY(newY) { y = newY; }

    public bar() {
        blah.blah(blah);
        z = getX() + getY();
        return bar();
    }

    public quux() {
        return getY() + 4;
    }

    public asdf() {
        return getX() * 8 + getY();
    }

    private bar(z) {
        blu = blu * 2;
        t = blurg(z);
        bli[] = blu;
        return t;
    }
}

f = new Foo();
f.bar();
print f.getX() + f.getY();
```

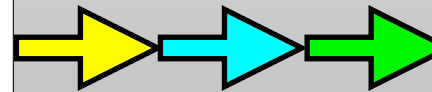
```
class Foo {
    public int x;
    public int y;

    public blah() {
        blah.blah(blah);
        z = x + y;
        blu = blu * 2;
        t = blurg(z);
        bli[] = blu;
        return t;
    }

    public quux() {
        return blu + 4;
    }

    public asdf() {
        return x * 8 + y;
    }
}

f = new Foo();
f.doFoo();
print f.x + f.y;
```



```
class Foo {
    private int x;
    private int y;

    public getX() { return x; }
    public setX(newX) { x = newX; }

    public getY() { return y; }
    public setY(newY) { y = newY; }

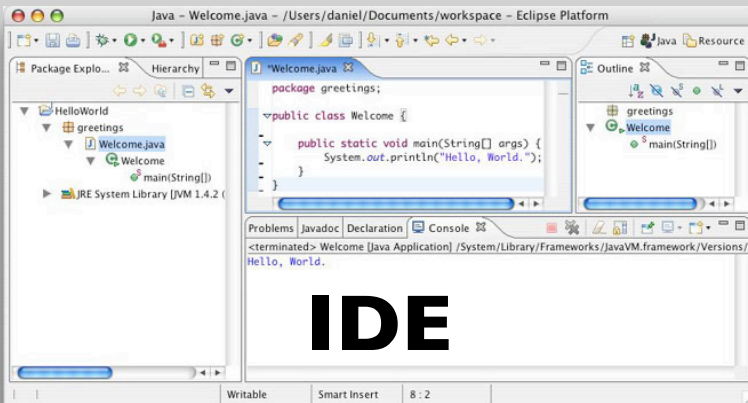
    public bar() {
        blah.blah(blah);
        z = getX() + getY();
        return bar();
    }

    public quux() {
        return getY() + 4;
    }

    public asdf() {
        return getX() * 8 + getY();
    }

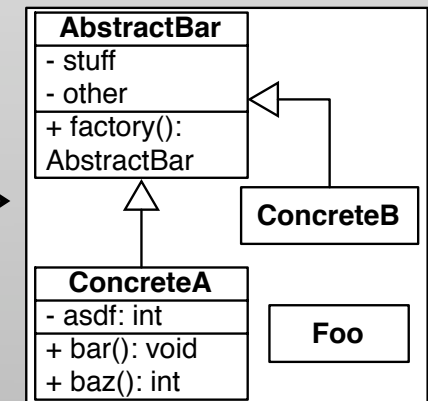
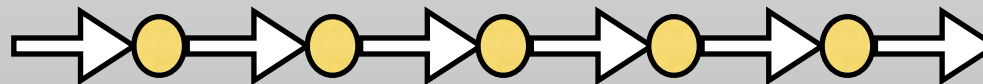
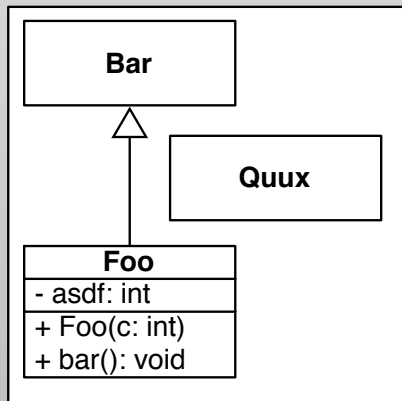
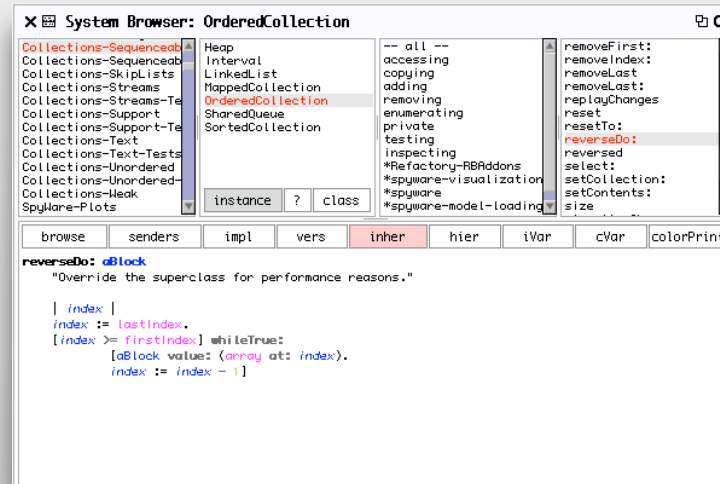
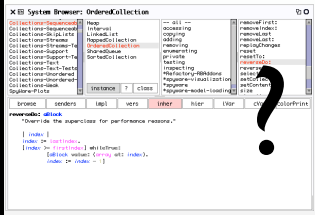
    private bar(z) {
        blu = blu * 2;
        t = blurg(z);
        bli[] = blu;
        return t;
    }
}

f = new Foo();
f.bar();
print f.getX() + f.getY();
```

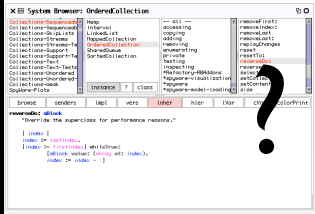


IDE

Query and understand recent changes



Query and understand recent changes

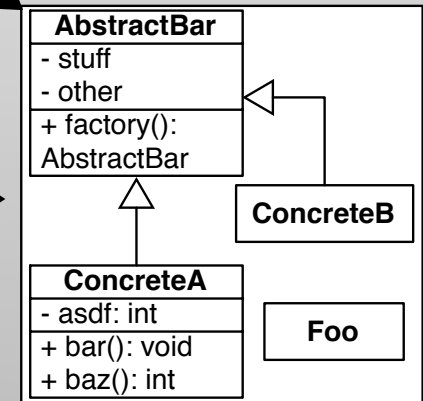
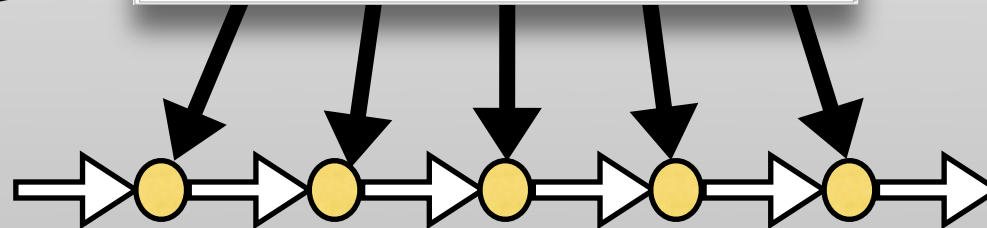
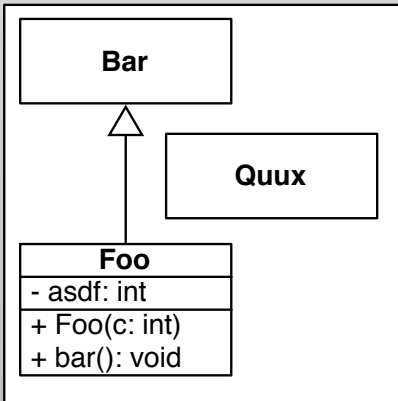


System Browser: OrderedCollection

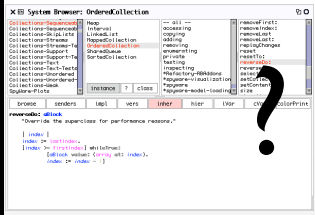
Heap
Interval
LinkedList
MappedCollection
OrderedCollection
SharedQueue
SortedCollection

removeFirst:
removeIndex:
removeLast:
replayChanges:
reset:
resetTo:
reverseDo:
reversed:
select:
setCollection:
setContents:
size

```
reverseDo: aBlock  
"Override the superclass for performance reasons."  
| index |  
index := lastIndex.  
[index >= firstIndex] whileTrue:  
  [aBlock value: (array at: index).  
   index := index - 1]
```



Query and understand recent changes

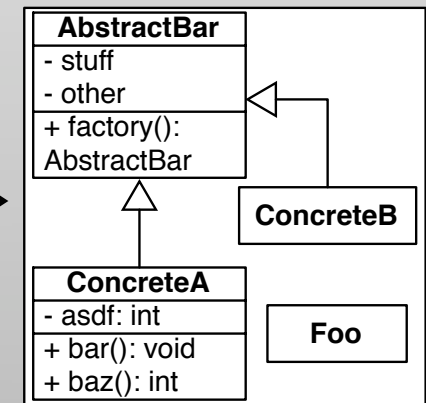
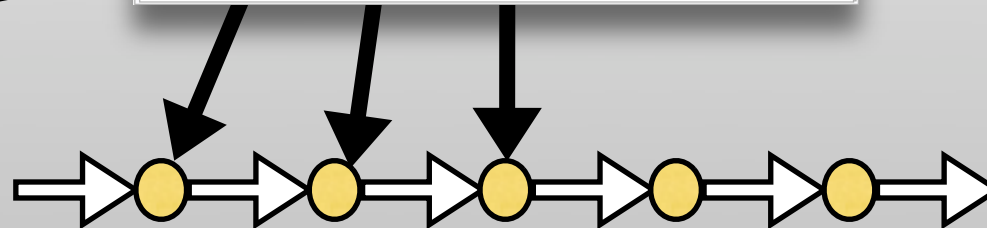
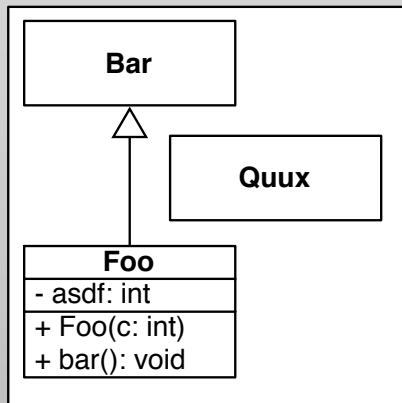


System Browser: OrderedCollection

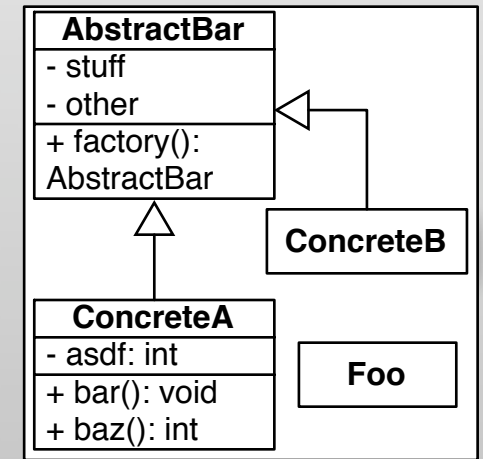
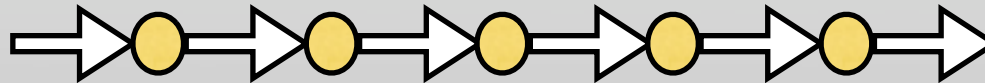
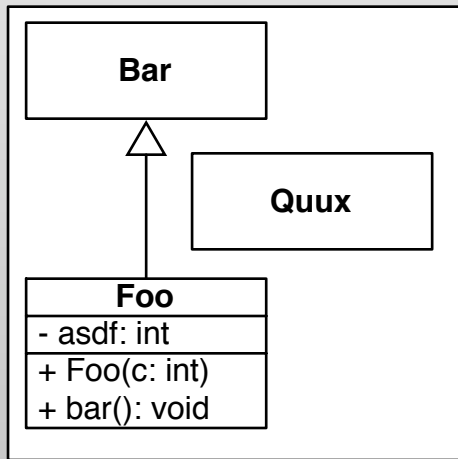
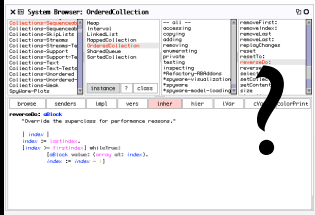
Heap
Interval
LinkedList
MappedCollection
OrderedCollection
SharedQueue
SortedCollection

removeFirst:
removeIndex:
removeLast:
replayChanges:
reset
resetTo:
reverseDo:
reversed:
select:
setCollection:
setContents:
size

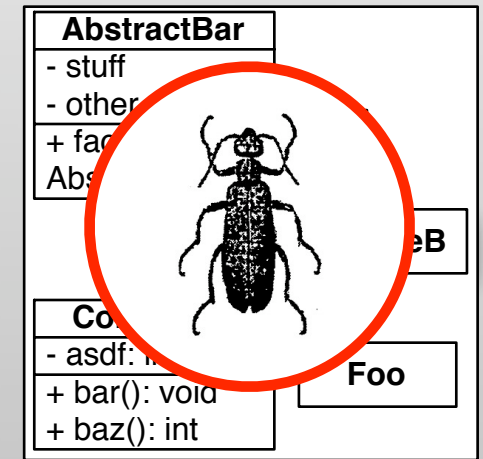
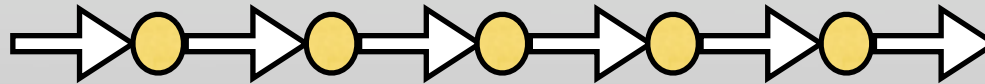
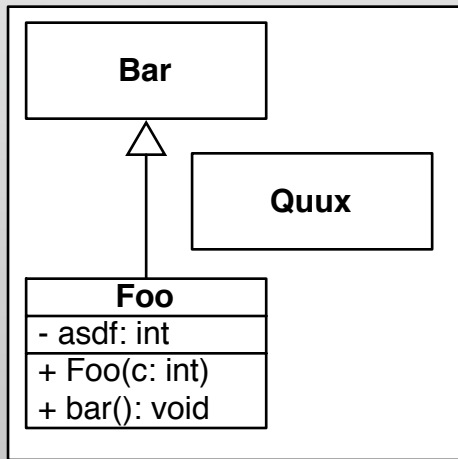
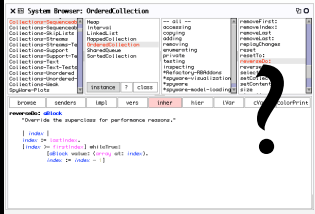
```
reverseDo: aBlock  
"Override the superclass for performance reasons."  
  
| index |  
index := lastIndex.  
[index >= firstIndex] whileTrue:  
  [aBlock value: (array at: index).  
   index := index - 1]
```



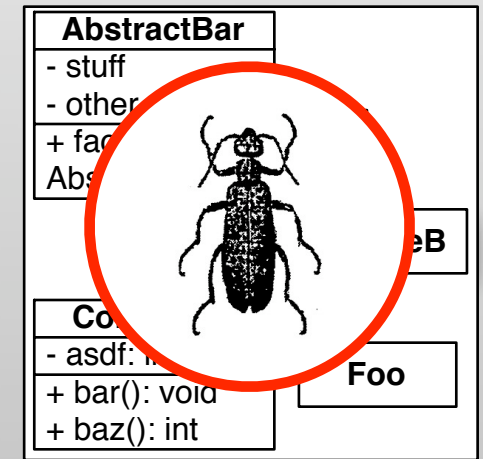
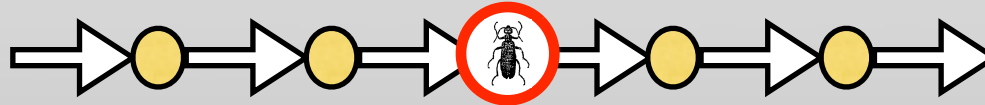
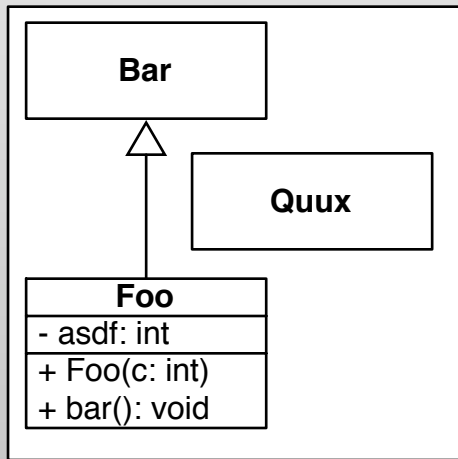
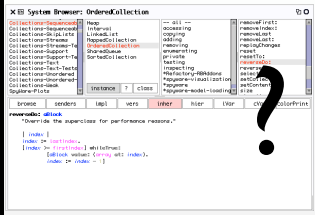
Find precise causes of bugs



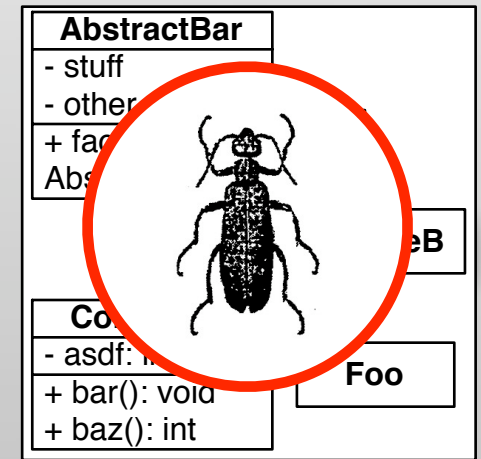
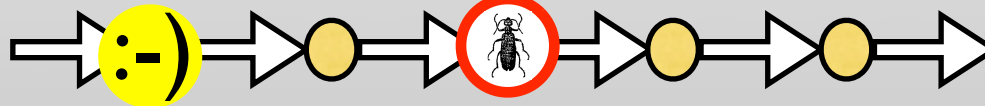
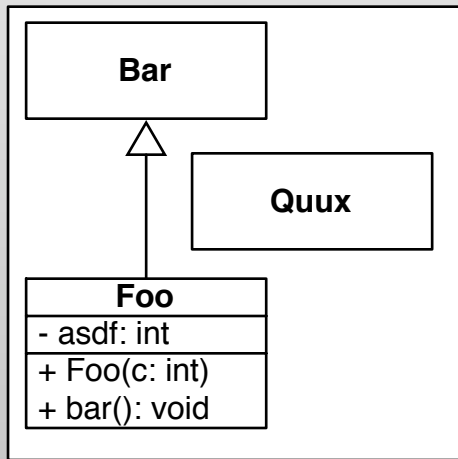
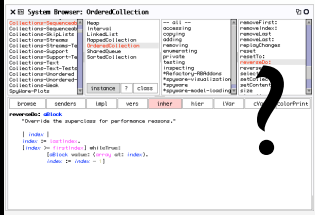
Find precise causes of bugs



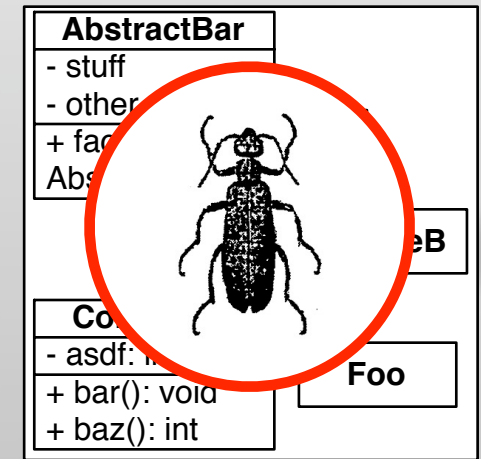
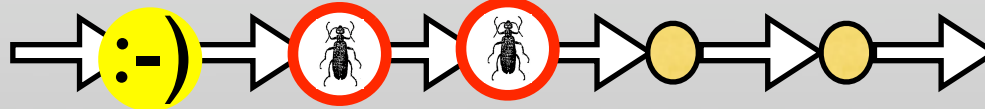
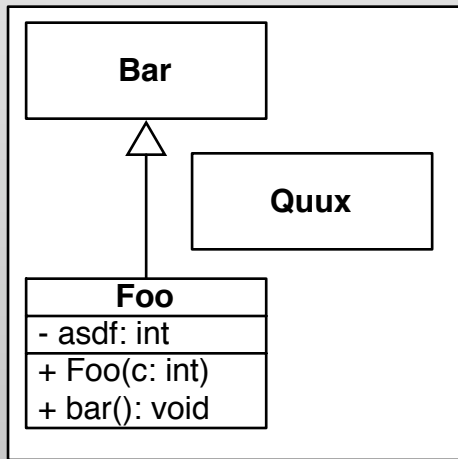
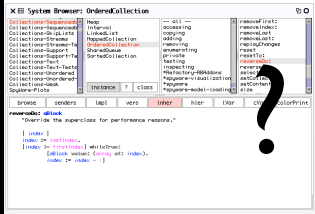
Find precise causes of bugs



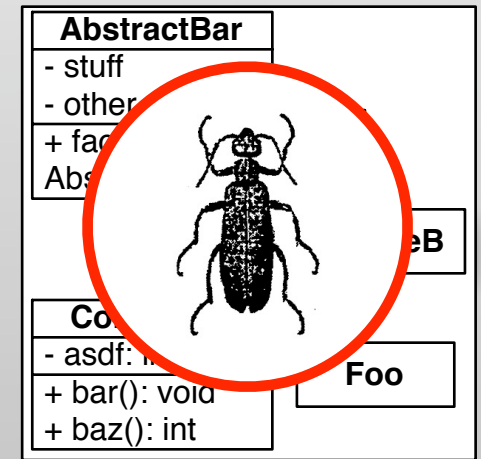
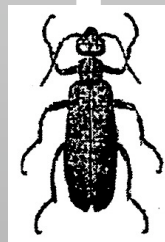
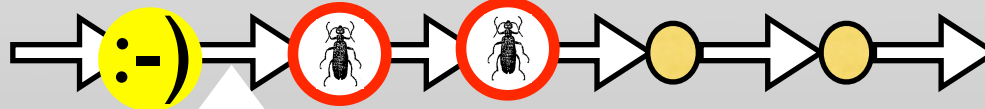
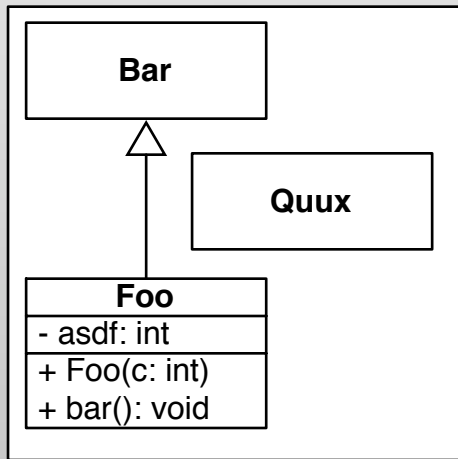
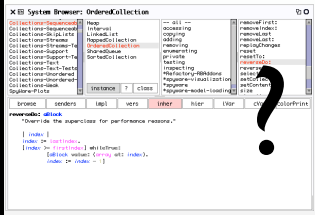
Find precise causes of bugs



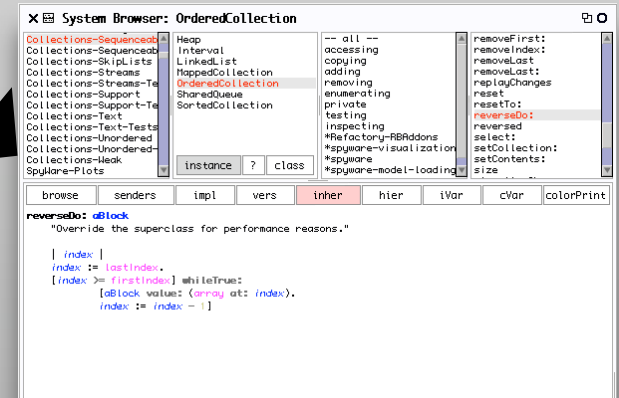
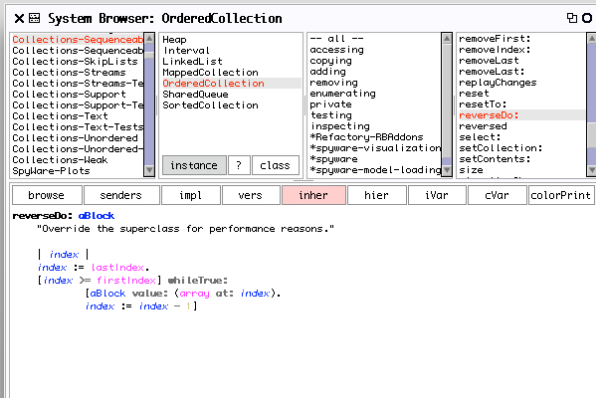
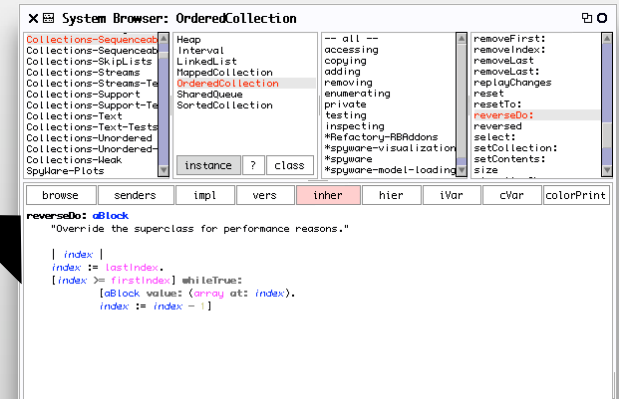
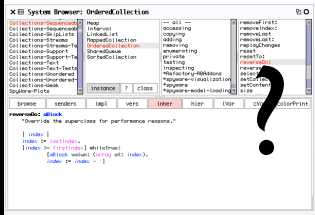
Find precise causes of bugs



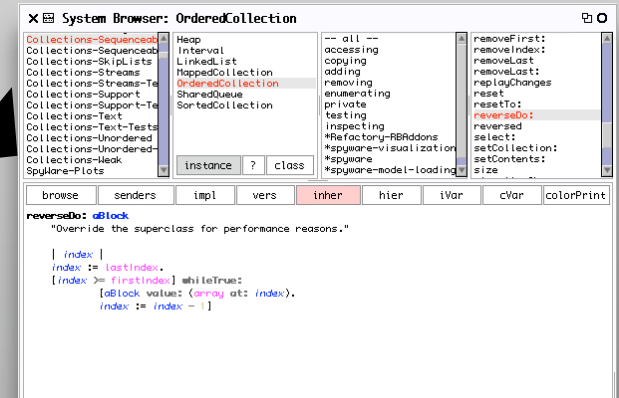
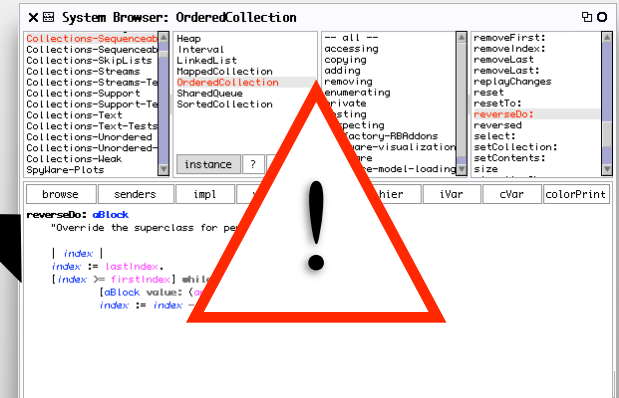
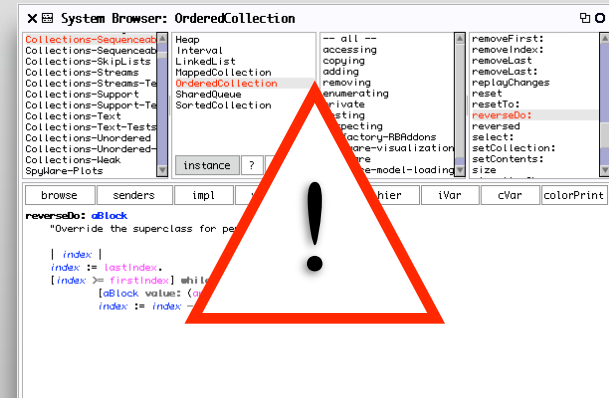
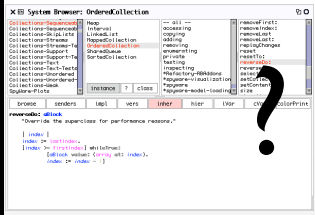
Find precise causes of bugs



Find merge conflicts faster

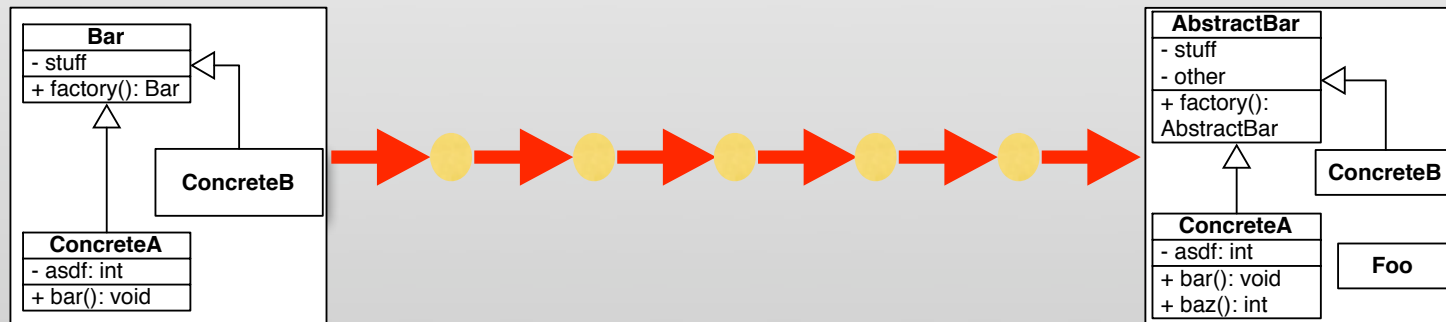


Find merge conflicts faster



Conclusions

SpyWare introduces a model of software changes



- + no information lost
- + accuracy
- + tool support

- performance?
- space?
- validation?

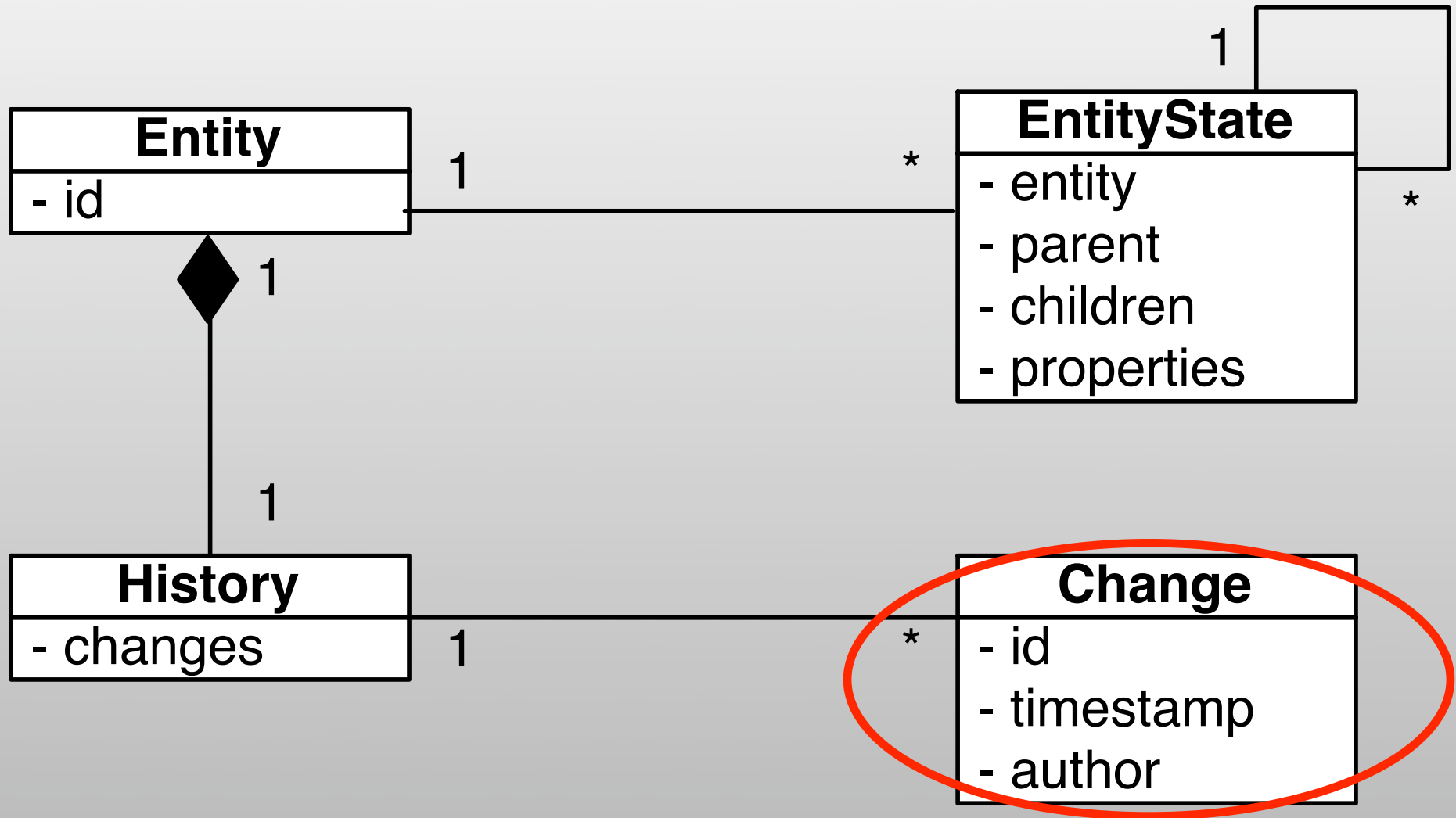


I WANT YOU

FOR SPYWARE NOW

<http://romain.robb.es/spyware>

Our model emphasizes changes over entities



Spyware versus the change log

Ad-hoc format (a bunch of do-its)

Not aware of refactorings

Tied to one image

Data loss because of purges

**Exporting with change sets only keeps
the last version of a method**