Spyware-ridden software development
SpyWare is a research prototype built for my Ph.D.

What is SpyWare?

Results & demo

Further possibilities

Context
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

* from: http://www.cs.jyu.fi/~koskinen/smcosts.htm
The lifecycle of software

- Understanding & Reengineering
- Requirements Gathering
- Design & Implementation

Romain Robbes
Spyware-ridden software development
Software development is incremental

Version 1 → Version 2 → Version 3 → Version 4
Software development is incremental

Version 1 → Version 2 → Version 3 → Version 4

- checkout
- add feature
- debug
- commit

75% 25% 75% 25%
Software development is incremental

- Version 1
- Version 2
- Version 3
- Version 4

- add class
- add method
- commit
- checkout
- debug
- add feature
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

History holds useful information

Versioning system

Metrics & trends

ConcreteA
- asdf: int
+ bar(): void
ConcreteB
- stuff
+ factory(): Bar
AbstractBar
- stuff
- other
+ factory(): AbstractBar
ConcreteA
- asdf: int
+ bar(): void
+ baz(): int
ConcreteB
Bar
- stuff
+ factory(): Bar
ConcreteB
Quux
Foo
- asdf: int
+ Foo(c: int)
+ bar(): void

Bar

Metrics & trends

75% 75% 25% 25%
Versioning systems lose information
Versioning systems lose information.

> cvs update
Versioning systems lose information.

> cvs update
> vim Foo.cc
Versioning systems lose information

> cvs update
> vim Foo.cc
(some work done...)

Romain Robbes
Versioning systems lose information.

> cvs update
> vim Foo.cc
(some work done...)
> cvs commit
Versioning systems lose information.

```bash
> cvs update
> vim Foo.cc
> (some work done...)
> cvs commit
```
Our paranoid programmer
saves every 5 minutes

```java
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;
```

```java
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();
        blu = blu * 2;
        t = blurg(z);
        bli[t] = blu;
        return t;
    }
    public quux() {
        return getY() + 4;
    }
    public asdf() {
        return getX() * 8 + getY();
    }
}

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

Romain Robbes
Spyware-ridden software development
Romain Robbes
Spyware-ridden software development

Our paranoid programmer saves every 5 minutes

1. Extract method
Our paranoid programmer saves every 5 minutes

1. Extract method
Our paranoid programmer saves every 5 minutes

1. Extract method
2. Rename method
Our paranoid programmer saves every 5 minutes

1. Extract method
2. Rename method
Our paranoid programmer saves every 5 minutes

1. Extract method
2. Rename method
3. Create accessors
Our paranoid programmer saves every 5 minutes

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

```java
class Foo {
    private int x;
    private int y;
    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;
```
Our paranoid programmer saves every 5 minutes

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 = x + y;
        blu = blu * 2;
        t = blurg(z);
        bli[t] = blu;
        return bar(z);
    }
    public quux() {
        return y + 4;
    }
    public asdf() {
        return x * 8 + y;
    }
}

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

 CVS: +18 / -11

1. Extract method
2. Rename method
3. Create accessors
Our paranoid programmer saves every 5 minutes

```java
class Foo {
    private int x;
    private int y;
    publicgetX() { return x; }
    public setX(newX) { x = newX; }
    public getY() { return y; }
    public setY(newY) { y = newY; }
    public baz() {
        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.baz();
print f.getX() + f.getY();
```

**CVS:**  \(+18 / -11\)

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

---

Romain Robbes
Spyware-ridden software development
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.
Rather than storing versions, we record the semantic actions of the programmer.
The general architecture of SpyWare

IDE

SpyWare plugin

Change repository

Tools

ConcreteA
- asdf: int
+ bar(): void
+ baz(): int

ConcreteB

AbstractBar
- stuff
- other
+ factory(): 

AbstractBar

Foo

Quux

ConcreteA
- asdf: int
+ bar(): void
+ baz(): int

ConcreteB

AbstractBar
- stuff
- other
+ factory(): 

AbstractBar

Foo

Quux
How does SpyWare work?
How does SpyWare work?

myNewMethod: foo
How does SpyWare work?

```plaintext
myNewMethod: foo
self bar: foo + 1.
```
How does SpyWare work?

```smalltalk
myNewMethod: foo
self bar: foo + 1.
self baz: foo - 1.
```
How does SpyWare work?

Incremental compilation

myNewMethod: foo
self bar: foo + 1.
self baz: foo - 1.
How does SpyWare work?

```smalltalk
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.

- create method
- add to class
- create variable
- add to method

OK...
The change builder infers change information
Tools exploit the changes

Model maintainer

Bar

baz:

quux
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

Bar

baz:

myNewMethod:

quux

foo
SpyWare case studies:

Big brother is watching you!

SpyWare
Short SpyWare demo

SpyWare

Romain Robbes
Spyware-ridden software development
Several tools can be implemented on top of this mechanism: here are a few.
Fine-grained software evolution analysis

Versioning System

IDE
Query and understand recent changes

ConcreteA
- asdf: int
+ bar(): void
+ baz(): int

ConcreteB
AbstractBar
- stuff
- other
+ factory(): AbstractBar

Foo
- asdf: int
+ Foo(c: int)
+ bar(): void

Quux

Bar

Romain Robbes
Spyware-ridden software development
Query and understand recent changes
Query and understand recent changes
Find precise causes of bugs
Find precise causes of bugs
Find precise causes of bugs
Find precise causes of bugs

AbstractBar
- stuff
- other
+ factory():

AbstractBar
- stuff
- other
+ factory():

ConcreteA
- asdf: int
+ bar(): void
+ baz(): int

ConcreteB

Bar

Quux

Foo
- asdf: int
+ Foo(c: int)
+ bar(): void

Quux

Romain Robbes
Spyware-ridden software development
Find precise causes of bugs
Find precise causes of bugs

Romain Robbes

Spyware-ridden software development
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?
FOR SPYWARE NOW

http://romain.robb.es/spyware
Our model emphasizes changes over entities.

**Entity**
- id

**History**
- changes

**EntityState**
- entity
- parent
- children
- properties

**Change**
- id
- timestamp
- author

Romain Robbes
Spyware-ridden software development
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