Software Improvement Group

Static Estimation of Test Coverage

IPA Herfstdagen - Session: Static analysis via code query technologies

Tiago Alves & Joost Visser

November 27th, 2008

Arent Janszoon Ernststraat 595-H
NL-1082 LD Amsterdam
info@sig.nl
www.sig.nl

Software Improvement Group

Characterization

- Based in Amsterdam, The Netherlands
- Spin-off from the Centre of Mathematics and Information Technology (CWI)
- Fact-based IT consulting

Services

- Software Risk Assessment
  - Exhaustive research of software quality and risks
  - Answers specific research questions
  - One time execution
- Software Monitor
  - Automated quality measurements executed frequently (daily / weekly)
  - Information presented in a web-portal
- DocGen
  - Automated generation of technical quality

Our Customers

Software Improvement Group

Financial and Insurance Companies

Government

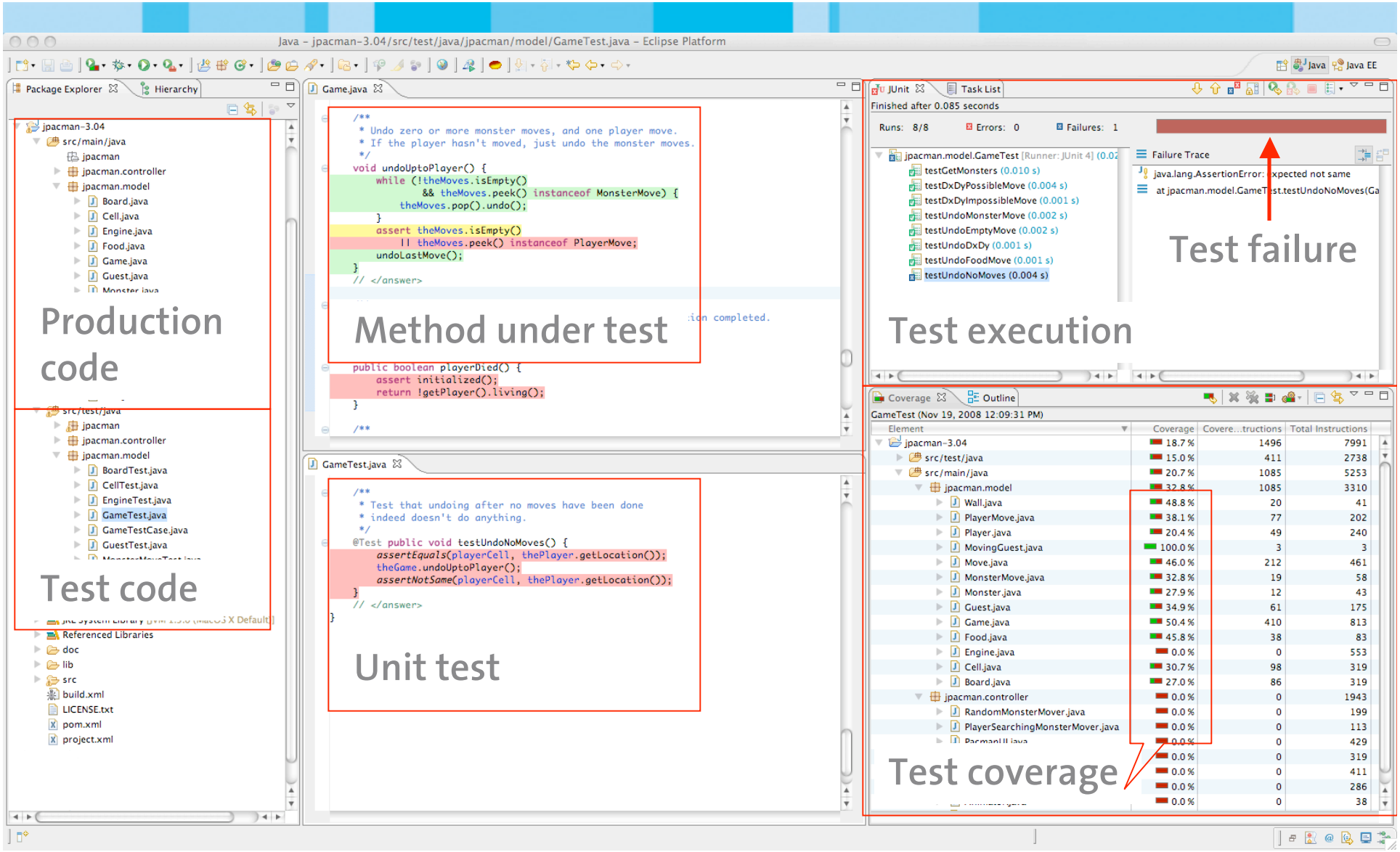
Logistical

IT

Other

Software testing

Software Improvement Group



Measuring test coverage

Software Improvement Group

Pros:

- Indicator for test quality
- Indicator for quality of the software under test
  - Higher coverage => better software quality (in principle)

Cons:

- Full installation required (sources + libraries)
- Instrumentation of source/byte code
  - Problematic in embedded systems
- Execution (Hardware and time constrains)

- Not appropriate to compute in the context of software quality assessment!!

Research Challenge

Software Improvement Group

Motivation

- 13th Testdag, Delft, November 2007
  I. Heitlager, T. Kuipers, J. Visser “Observing unit test maturity in the wild”

Research questions

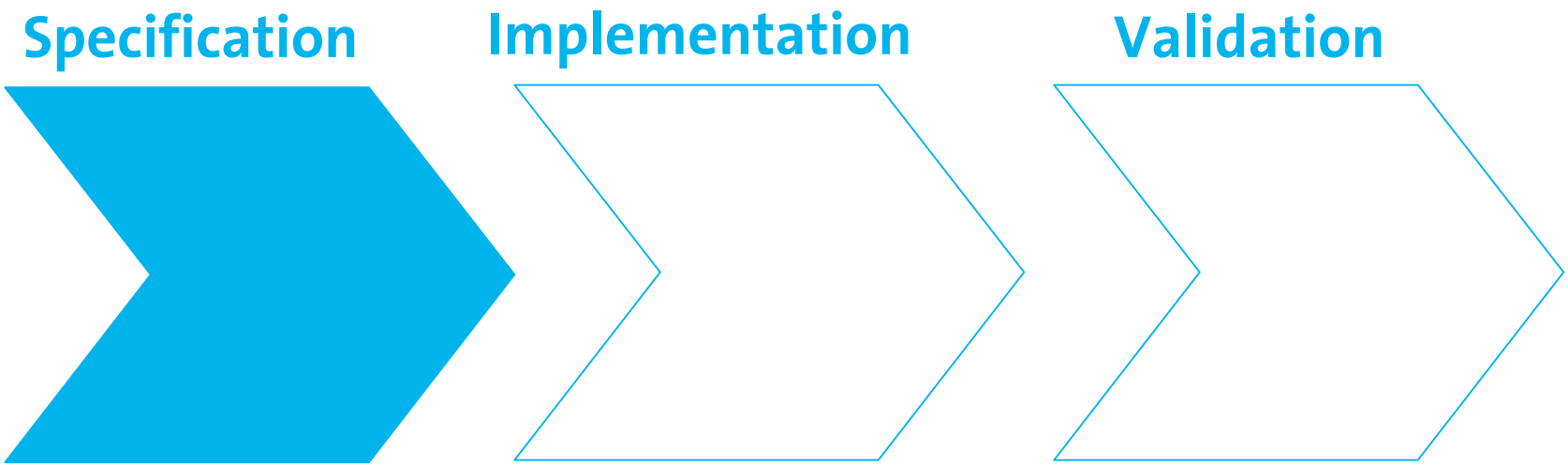
- Is it possible to **estimate** test coverage without running tests?
- What trade-offs can be made between sophistication and accuracy?

Requirements

- Use only static analysis
- Scale to large systems
- Robust against incomplete systems

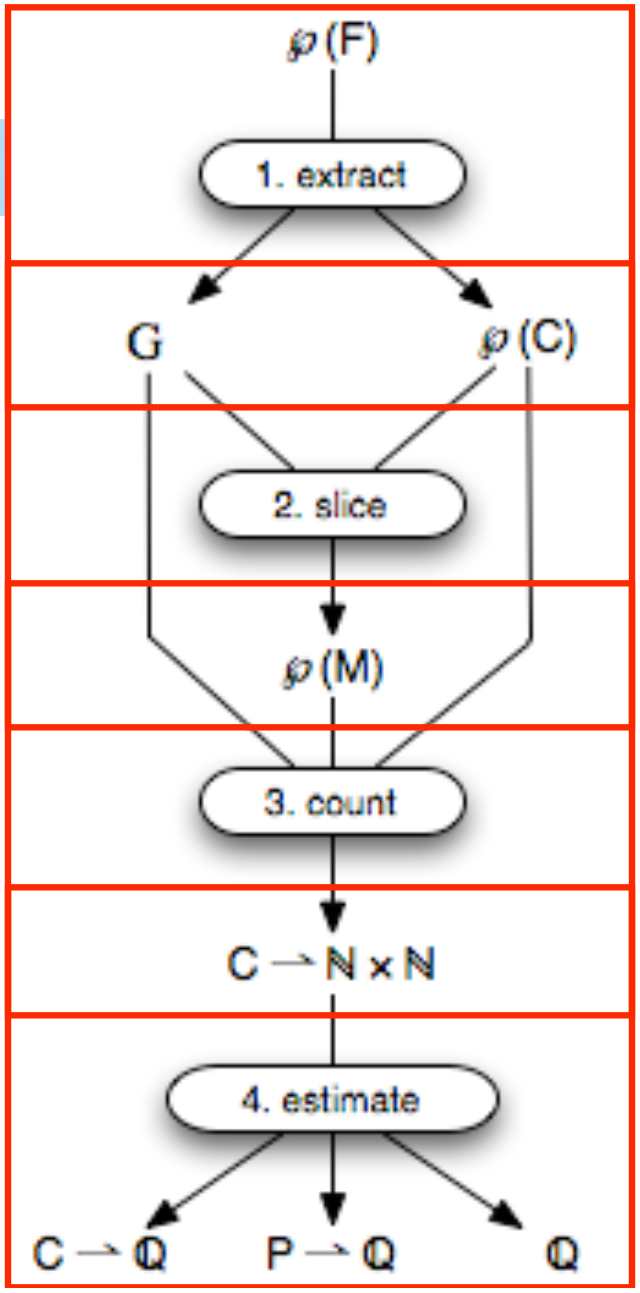
Where do we stand?

Software Improvement Group



- Solution sketch
- Sources of imprecision
- Dealing with imprecision

Solution sketch

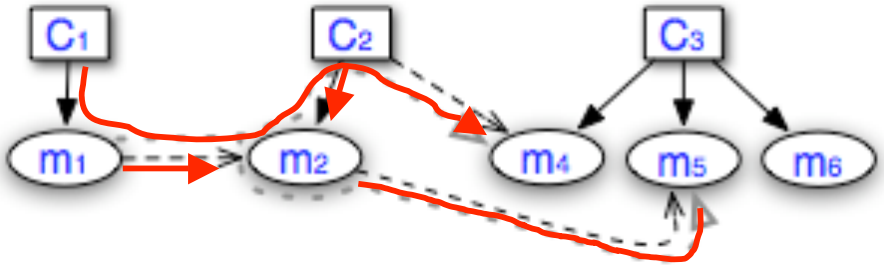


Software Improvement Group

1. **Extract**
   - Extract structural and call information
   - Determine set of test classes
2. **Slice (modified)**
   - Slice graph starting from test methods
   - Set of methods reached from test code
   - Take into account class initializer calls
3. **Count (per class)**
   - Determine number of defined methods
   - Determine number of covered methods
4. **Estimate**
   - Class coverage
   - Package coverage
   - System coverage

Modified slicing specification

Software Improvement Group



$$n \xrightarrow{call} m$$

$$n \xrightarrow{init} m = n \xrightarrow{call} m_i \xleftarrow{def} c \xrightarrow{call} m$$

$$m \xleftarrow{def} c$$

$$n \xrightarrow{invoke} m = n \xrightarrow{call} m \cup n \xrightarrow{init} m$$

$$n \xrightarrow{invoke}{}^{+} m$$

Code coverage formulas

Software Improvement Group

Defined methods: $DM : n_C \rightarrow \mathrm{N}$

Covered methods: $CM : n_C \rightarrow \mathrm{N}$

$$CC(c) = \frac{CM(c)}{DM(c)} \times 100\%$$

$$PC(p) = \frac{\sum_{c \in p} CM(c)}{\sum_{c \in p} DM(c)} \times 100\%$$

$$SC = \frac{\sum_{c \in G} CM(c)}{\sum_{c \in G} DM(c)} \times 100\%$$

What can go wrong? (Sources of imprecision)

Software Improvement Group

Java language

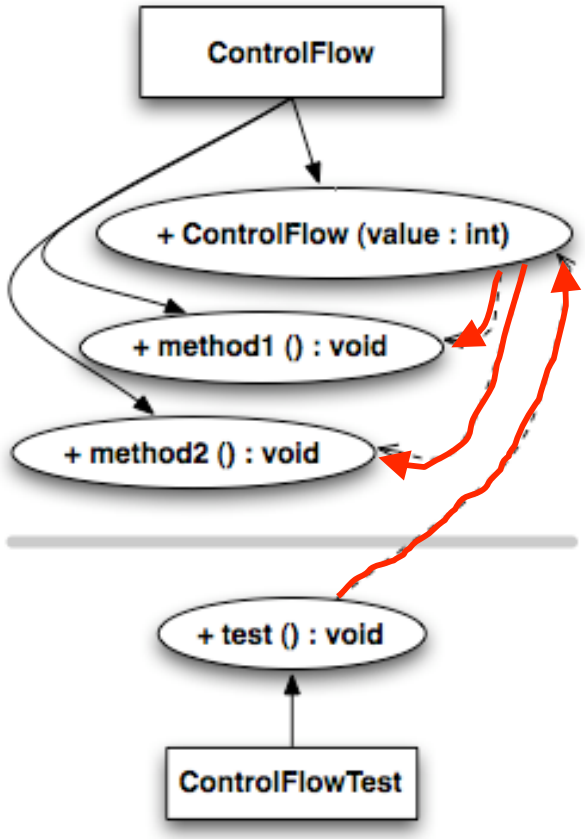
- Control flow
- Dynamic dispatch (inheritance)
- Overloading

General issues

- Frameworks / Libraries call backs
- Identification of test code
- ///CLOVER:OFF flags

Sources of imprecision
Control flow

Software Improvement Group

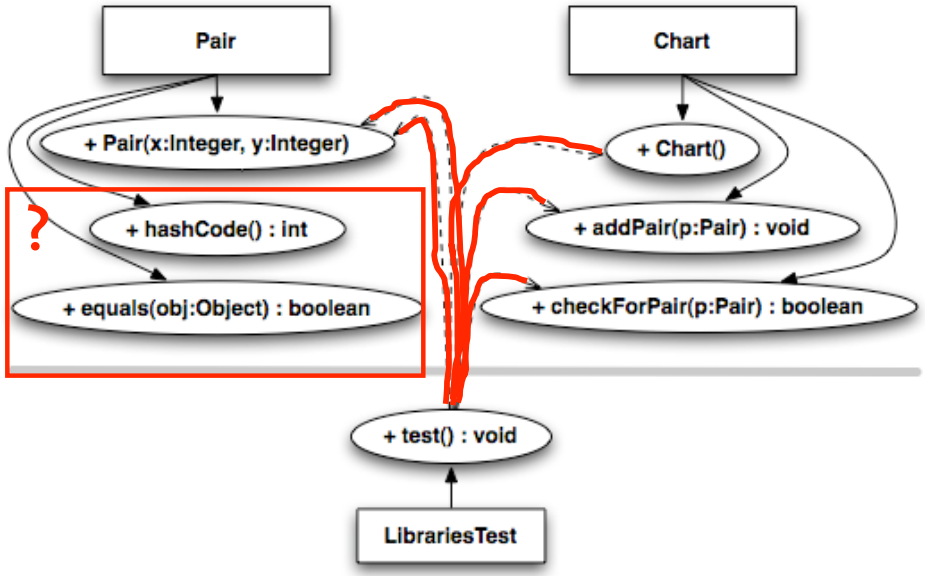



```
class ControlFlow {
 ControlFlow(int value) {
   if (value > 0)
     method1();
   else
     method2();
 }
 void method1() { }
 void method2() { }
}
```

```
import junit.framework.*;
class ControlFlowTest extends TestCase {
  void test() {
    ControlFlow cf =
      new ControlFlow(3);
  }
}
```

Sources of imprecision
Libraries

Software Improvement Group




```
class Pair {
  Integer x; Integer y;

  Pair(Integer x, Integer y) { ... }
  int hashCode() { ... }
  boolean equals(Object obj) { ...}
}
class Chart {
  Set pairs;

  Chart() { pairs = new HashSet(); }
  void addPair(Pair p) { pairs.add(p); }
  void checkForPair(Pair p) { return pairs.contains(p); }
}
```

```
import junit.framework.*;
class LibrariesTest extends TestCase {

  void test() {
    Chart c = new Chart();

    Pair p1 = new Pair(3,5);
    c.addPair(p1);

    Pair p2 = new Pair(3,5);
    c.checkForPair(p2);
} }
```

Dealing with imprecision

Software Improvement Group

Pessimistic approach

- Report only what can be determined to be true
- False negatives
- Estimates lower bound for coverage

Optimistic approach

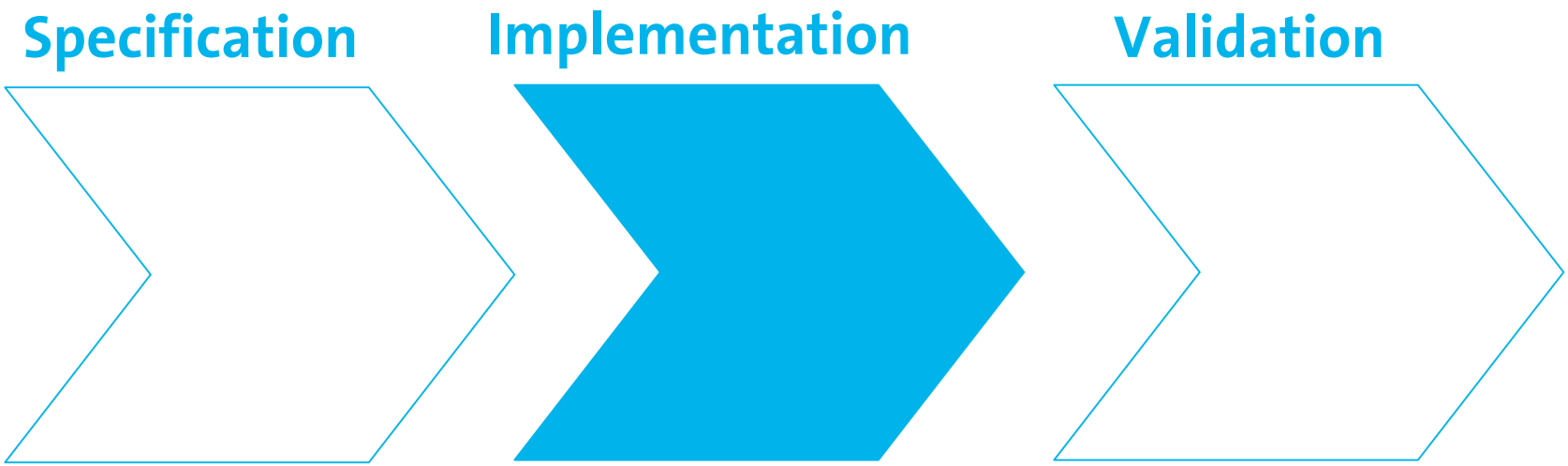
- Report everything that might be true
- False positives
- Estimates upper bound for coverage

Pessimistic vs. Optimistic (software assessment context)

- Pessimistic will always report low coverage
- Optimistic will be sensitive to lack of coverage
- Optimistic will not take into account library calls

Where do we stand?

Software Improvement Group



- Code query technologies
- Definition of abstractions
- Implementation of the method
- Querying the results

Code query technologies

Software Improvement Group

	Style / Paradigm	Type system					Abstraction	Extendability
		String	Integer	Real	Boolea	Other		
RelView	Procedural	-	x	-	-	-	-	-
Grok	Relational	x	x	x	-	-	-	-
Rscript	Relational & Comprehensions	x	x	-	x	Composite	x	x
JRelCal	API	x	x	x	x	Java	x	x
SemmleCode	SQL-like + OO	x	x	x	x	Object	x	x (limited)
Crocopat	Imperative + FO logic	x	x	x	-	-	-	-
GReQL 2	SQL-like + path expr.	x	x	x	x	-	-	-
JTransformer	FO Logic	x	x	x	-	-	-	-

SemmleCode

Software Improvement Group

Commercial product developed by Semmle Ltd. (Oege de Moor et. Al)

Historical overview:

- Started December 2006
- First tutorial July 2007
- Version 1.0 (Beginning 2009 - expected)

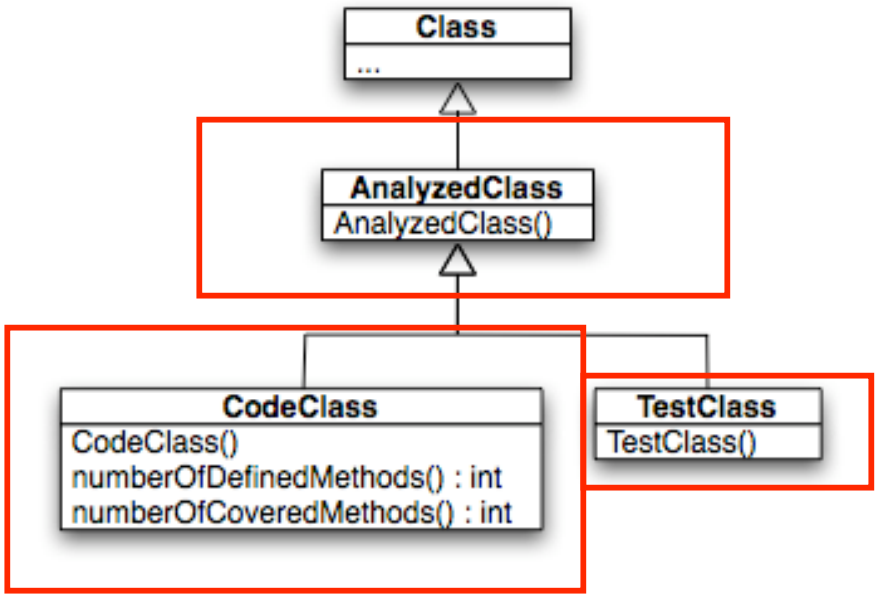
Eclipse plug-in + headless version
Integrated Java + XML extractor

.QL as code query language

- Based on relational calculus + object-oriented model.

Definition of abstractions
Class extension

Software Improvement Group




```
class AnalyzedClass extends Class {
   AnalyzedClass() {
         this.fromSource() and ...
   }
}
class TestClass extends AnalyzedClass {
   TestClass() { isJUnitClassTest(this) }
}
class CodeClass extends AnalyzedClass {
   CodeClass() { not isJUnitClassTest(this) }
   ...
   int numberOfDefinedMethods() {
         result = count( NonAbstractCallable m
                        | this.containsCallable(m))
   }
   int numberOfCoveredMethods() {
         result = count( NonAbstractCallable m
                        | this.containsCallable(m)
                         and  m.isTestCovered())
   }
}
```

Definition of abstractions
Method extension

Software Improvement Group


```
class NonAbstractCallable extends Callable {
  NonAbstractCallable() {
        this.fromSource() and
        not (this.getName() = "<clinit>") and
        not this.hasModifier("abstract") and
        not (this.getLocation().getNumberOfLines() = 0)
  }
  predicate isTestCovered() {
        exists( TestClass tc, Callable tm | tc.contains(tm) and invoke+(tm, this))
  }
}
```

Modified slicing implementation

Software Improvement Group

Binary relational expression

$$n \xrightarrow{call} m \qquad m \xleftarrow{def} c$$

$$n \xrightarrow{init} m = n \xrightarrow{call} m_i \xleftarrow{def} c \xrightarrow{call} m$$

$$n \xrightarrow{invoke} m = n \xrightarrow{call} m \mid n \xrightarrow{init} m$$

$$n \xrightarrow{invoke}{}^{+} m$$

```
predicate invoke(Callable m1, Callable m2) {
  myPolyCall(m1,m2)
  or
  exists(Class c, Callable mi, Callable mj |
    myPolyCall(m1,mi) and
    c.contains(mi) and
    c.contains(mj) and
    mj.getName() = "<clinit>" and
    myPolyCall(mj,m2)
  )
}
```

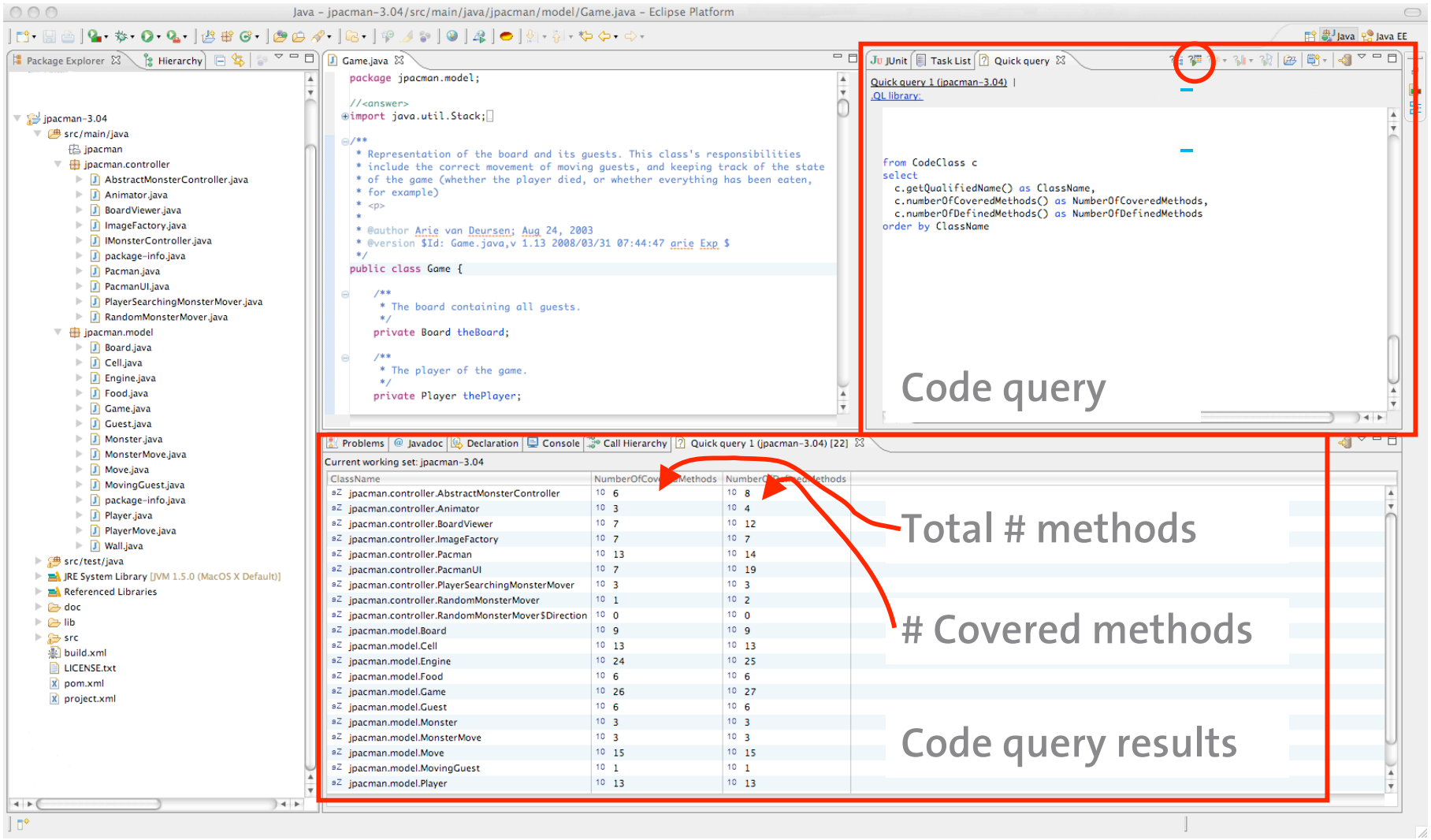
Querying the results
Class-level query

Software Improvement Group


```
from CodeClass c
select
   c.getQualifiedName() as ClassName,
   c.numberOfCoveredMethods() as NumberOfCoveredMethods,
   c.numberOfDefinedMethods() as NumberOfDefinedMethods
order by ClassName
```

Querying the results
Class-level query results for JPacMan

Software Improvement Group



Querying the results
Package-level query

Software Improvement Group


```
from Package p
where p.fromSource()
select
   p as PackageName,
   sum(CodeClass c | p.contains(c) | c.numberOfCoveredMethods())
        as NumberOfCoveredMethods,
   sum(CodeClass c | p.contains(c) | c.numberOfDefinedMethods())
        as NumberOfDefinedMethods
order by PackageName
```

Querying the results
Package-level query results for JPacMan

Software Improvement Group

Pro | Java | Decl | Con | Call | Quic | Quic | Quic

Current working set: jpacman-3.04

PackageName	NumberOfCoveredMethods	NumberOfDefinedMethods
jpacman	0	0
jpacman.controller	47	69
jpacman.model	130	132

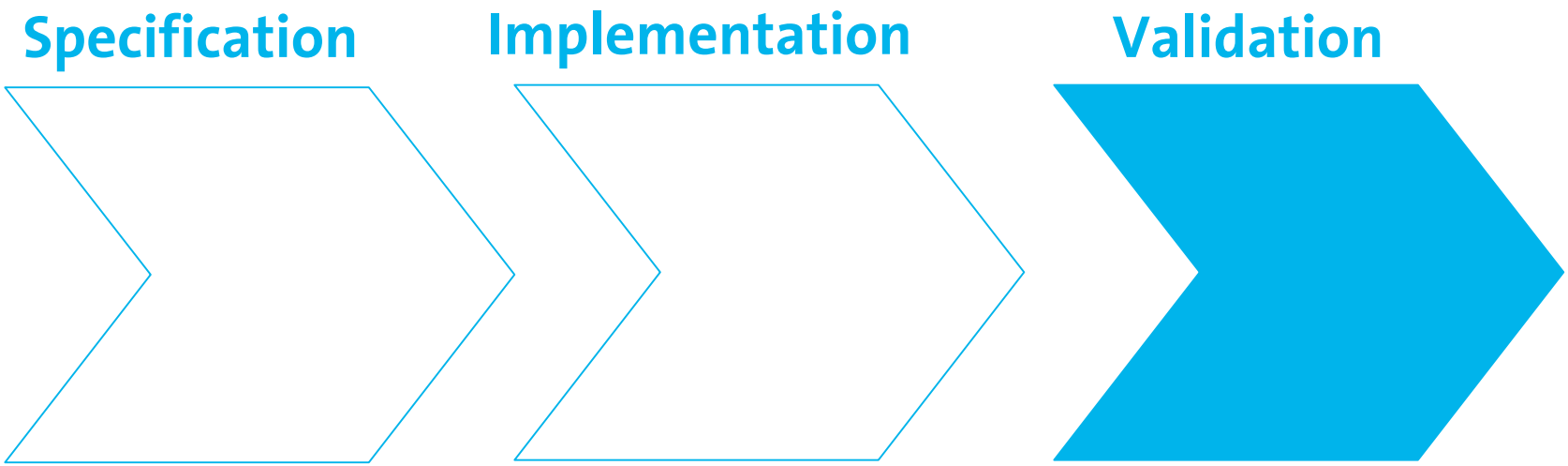
Quick query

Quick query 1 (jpacman-3.04) |
.QL library:

```
from Package p
where p.fromSource()
select
    p as PackageName
  , sum( CodeClass c | p.contains
      as NumberOfCoveredMethods
  , sum( CodeClass c | p.contains
      as NumberOfDefinedMethods
order by PackageName
```

Where do we stand?

Software Improvement Group



- Experimental design
- Data set characterization
- Comparison of results

Experimental design

Software Improvement Group

Data set selection and characterization

- Open-source and proprietary Java systems
- Different application domains
- Different sizes

Execution of experiment

- Clover execution (configuring clover + ant, running tests)
- XML Clover extraction (XSLT transformations for CSV generation)
- SemmleCode execution (text file export + scripts for CSV generation)
- Custom built java tool to read CSV files and generate Excel XLS

Statistical analysis

Software Improvement Group

Distributions

- Histogram of the coverage estimation
- Histogram of the real (clover) coverage

Correlation

- Spearman (rank-correlation)

Estimation different

- Histogram of the differences

Dispersion

- Inter-quartile ranges (dispersion)

Data set characterization

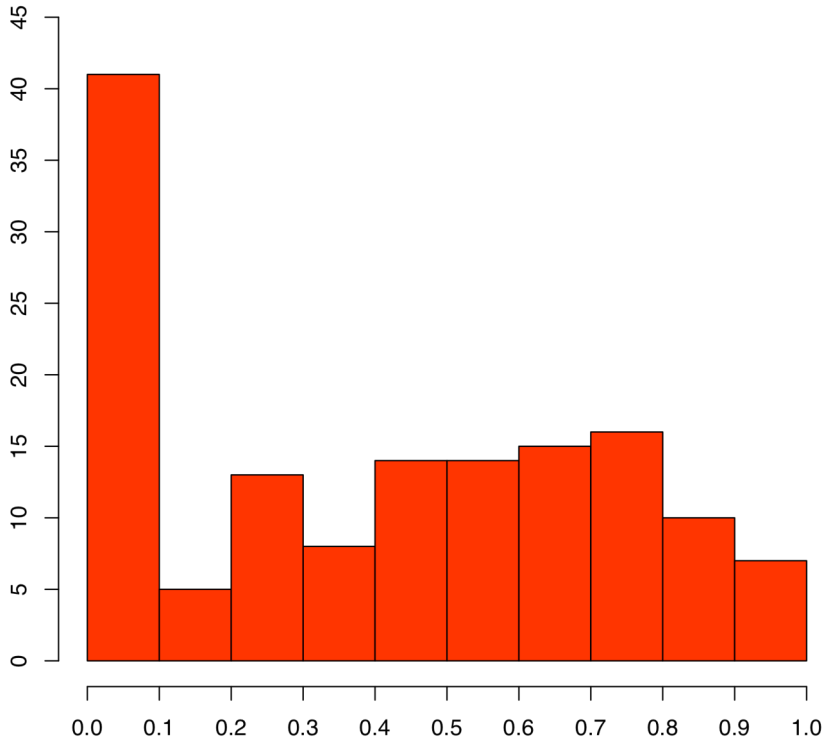
Software Improvement Group

System	Version	Author	LOC	# Packages	# Classes	# Methods
JPacMan	3.0.4	Arie van Deursen	2.5k	3	46	335
Certification	20080731	SIG	3.8k	14	99	413
G System	20080214	C Company	6.4k	17	126	789
Dom4j	1.6.1	MetaStuff	24.3k	14	271	3,606
Utils	1.61	SIG	37.7k	37	506	4,533
JGap	3.3.3	Klaus Meffert	42.9k	27	451	4,995
Collections	3.2.1	Apache	55.4k	12	714	6,974
PMD	5.0b6340	Xavier Le Vourch	62.8k	110	894	6,856
R System	20080214	C Company	82.3k	66	976	11,095
JFreeChart	1.0.10	JFree	127.7k	60	875	10,680
DocGen	r40981	SIG	127.7k	112	1,786	14,909
Analysis	1.39	SIG	267.5k	284	3,199	22,315

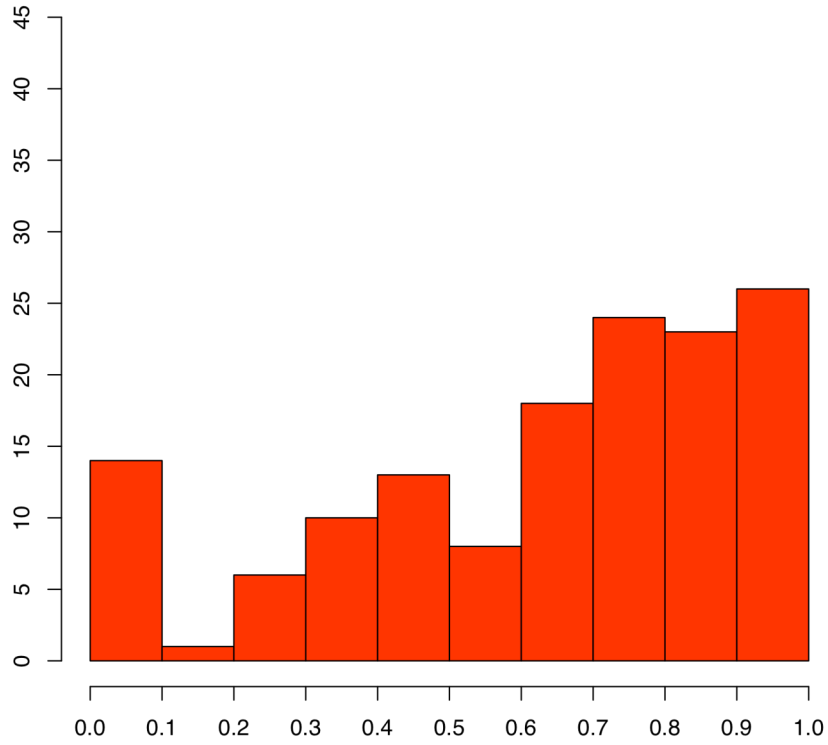
Dom4j: detailed statistical analysis
Class coverage distributions comparison

Software Improvement Group

Clover

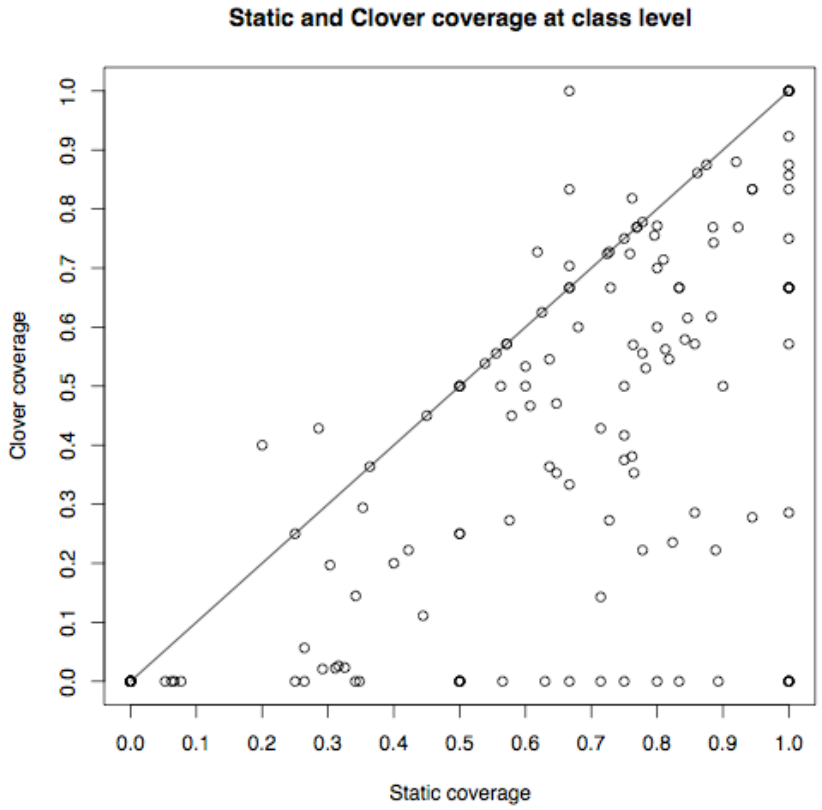


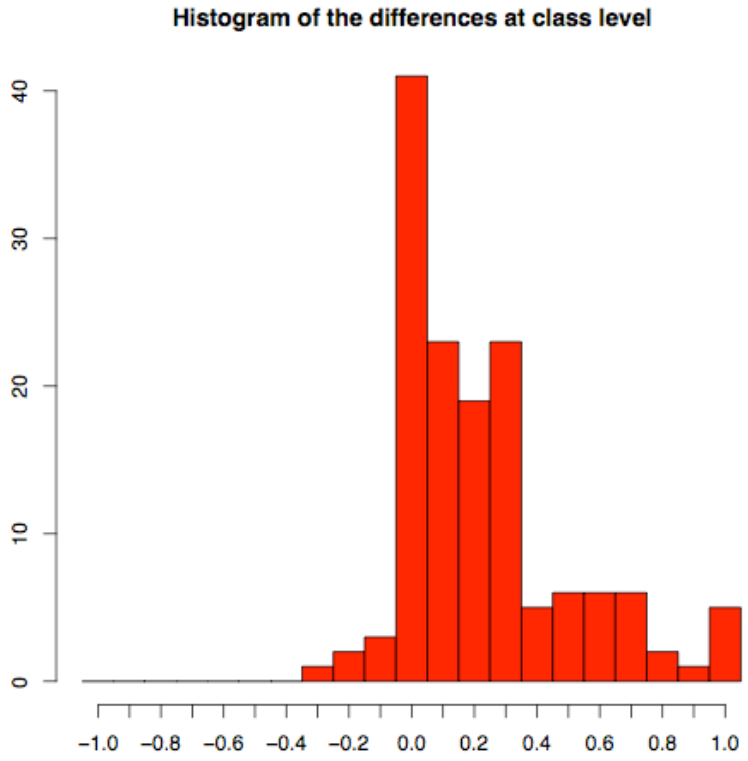
Static



Dom4j: detailed statistical analysis
Class coverage comparison + differences

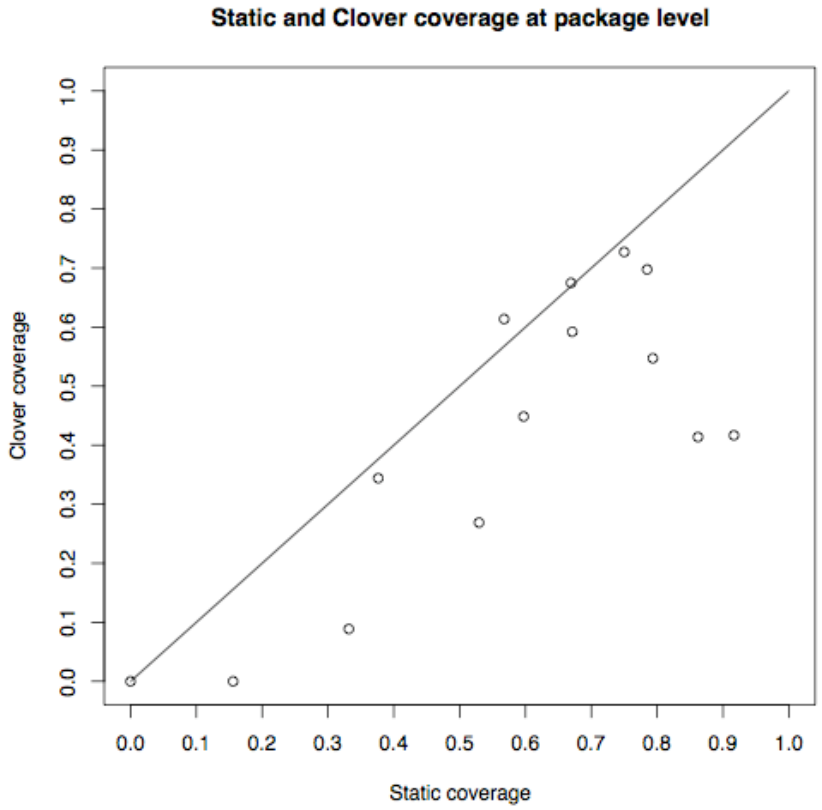
Software Improvement Group

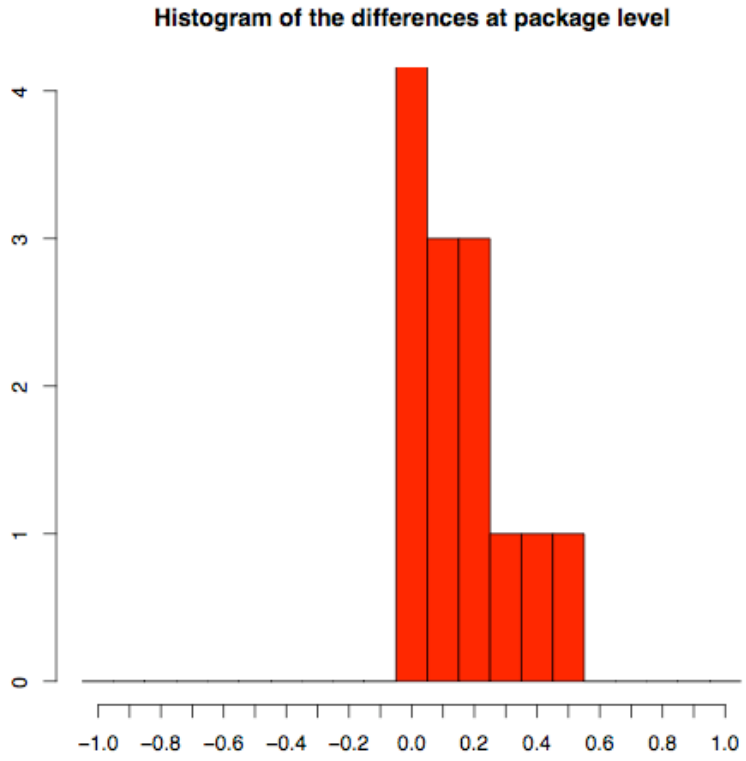




Dom4j: detailed statistical analysis
Package coverage comparison + differences

Software Improvement Group





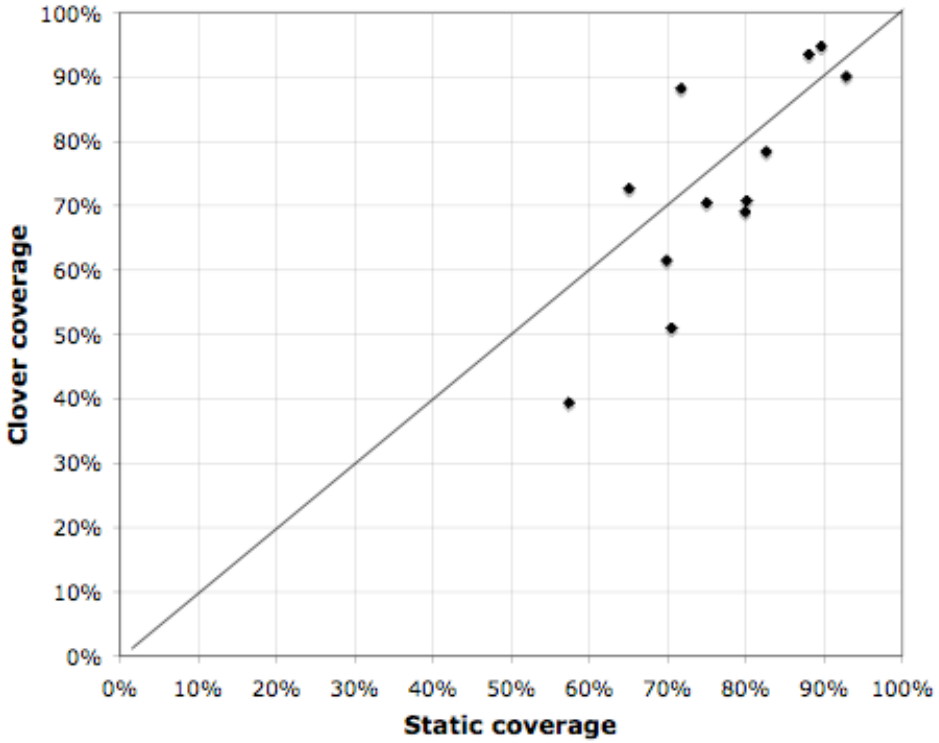
Statistical analysis
(Class and package coverage comparison)

Software Improvement Group

System name	Spearman		Median		Interquartile range	
	Class	Package	Class	Package	Class	Package
JPacMan	0.467*	1	0	-0.130	0.037	-
Certification	0.368**	0.520	0	0	0	0.015
G System	0.774**	0.694**	0	0	0	0.045
Dom4j	0.584**	0.620**	0.167	0.118	0.333	0.220
Utils	0.825**	0.778**	0	0.014	0	0.100
JGap	0.733**	0.786**	0	0	0.433	0.125
Collections	0.549**	0.776**	0	0.049	0.027	0.062
PMD	0.638**	0.655**	0	0.058	0.097	0.166
R System	0.727**	0.723**	0	-0.079	0.043	0.162
JFreeChart	0.632**	0.694**	0	0.048	0.175	0.172
DocGen	0.397**	0.459**	0	0.100	0.400	0.386
Analysis	0.391**	0.486**	0	-0.016	0.333	0.316

Statistical analysis
(System-level coverage comparison)

Software Improvement Group

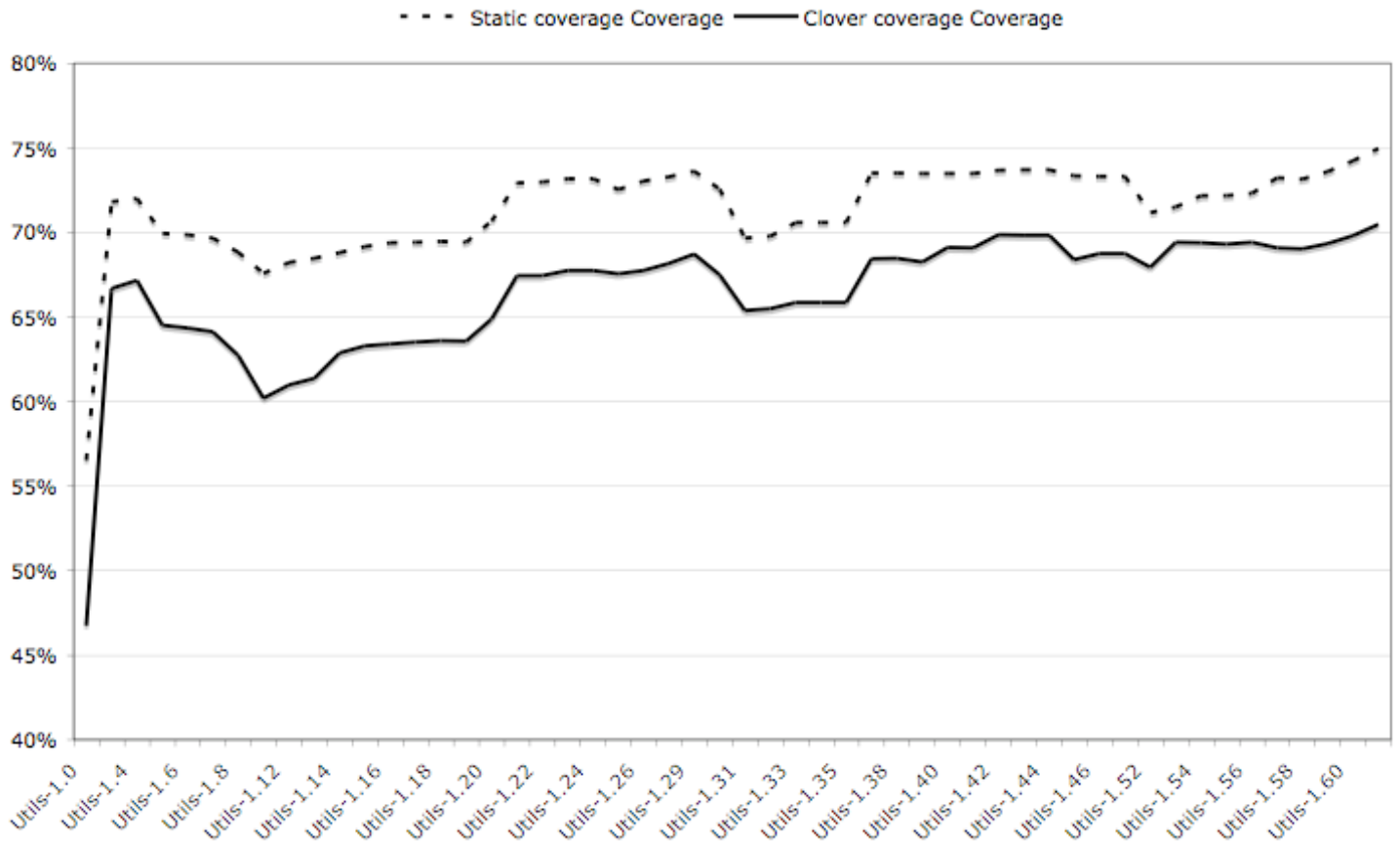


Correlation: 0.769

System	Static	Clover	Diff
JPacMan	88.06%	93.53%	-5.47%
Certification	92.82%	90.09%	2.73%
G System	89.61%	94.81%	-5.19%
Dom4j	57.40%	39.37%	18.03%
Utils	74.95%	70.47%	4.48%
JGap	70.51%	50.99%	19.52%
Collections	82.62%	78.39%	4.23%
PMD	80.10%	70.76%	9.34%
R System	65.10%	72.65%	-7.55%
JFreeChart	69.88%	61.55%	8.33%
DocGen	79.92%	69.08%	10.84%
Analysis	71.74%	88.23%	-16.49%

Utils revisions

Software Improvement Group



Conclusion

Is it possible to determine test coverage without running tests?

- Yes we can!!!
- Spearman: high correlation between static and clover coverage
- In general static coverage identifies the same values as Clover

What trade-offs can be made between sophistication and accuracy?

- Average absolute difference: 9%
- Class and Package coverage needs further improvement

Implementation

- SemmleCode: 92 LOC
- Java SIG Analysis: 256 LOC

Thank you!

Questions?

Tiago Alves
t.alves@sig.nl

Joost Visser
j.visser@sig.nl