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

Financial and Insurance Companies

Government

Logistical

IT

Other

KLM

Getronics PinkRoccade

ENECO

ergie

ING

Leng en Zekerheid

VROM

RDW

DHL

Centric

PricewaterhouseCoopers

Rabobank

InterBank

ZwitserLeven

POLITIE

EUROMAX

TERMINAL

EXACT

Software

IBM

Gasunie

Friesland Bank

delta Lloyd

Allianz

LeasePlan

ProRail

CHESS

LOGICA

CMG

kadaster
Software testing

Production code

Test code

Method under test

Unit test

Test execution

Test coverage

Test failure

Software Improvement Group
Measuring test coverage

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!!
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?

- Solution sketch
- Sources of imprecision
- Dealing with imprecision
Solution sketch

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

\[
\begin{align*}
\text{call} & \quad n \rightarrow m \\
\text{def} & \quad m \leftarrow c \\
\text{init} & \quad \text{init} \\
\text{call} & \quad \text{call} \\
\text{def} & \quad \text{def} \\
\text{call} & \quad \text{call} \\
\text{init} & \quad \text{init} \\
\text{invoke} & \quad \text{invoke} \\
\text{call} & \quad \text{call} \\
\text{init} & \quad \text{init} \\
\text{invoke} & \quad \text{invoke} \\
\end{align*}
\]
Code coverage formulas

Defined methods: \( DM : n_c \rightarrow N \)

Covered methods: \( CM : n_c \rightarrow N \)

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

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

\[
SC = \frac{\sum_{c \in G} CM(c)}{\sum_{c \in G} DM(c)} \times 100\%
\]
What can go wrong? (Sources of imprecision)

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

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

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. static;

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

**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?

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

<table>
<thead>
<tr>
<th>Style/Paradigm</th>
<th>Style/Paradigm</th>
<th>Type system</th>
<th>Type system</th>
<th>Type system</th>
<th>Type system</th>
<th>Abstraction</th>
<th>Extendability</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReView</td>
<td>Procedural</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Grok</td>
<td>Relational</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Rscript</td>
<td>Relational &amp; Comprehensions</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>Composite</td>
<td>x</td>
</tr>
<tr>
<td>JRelCal</td>
<td>API</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>java</td>
<td>x</td>
</tr>
<tr>
<td>SemmleCode</td>
<td>SQL-like + OO</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Object</td>
<td>x (limited)</td>
</tr>
<tr>
<td>Crocopat</td>
<td>Imperative + FO logic</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GReQL2</td>
<td>SQL-like + path expr.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>JTransformer</td>
<td>FO Logic</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
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

```java
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

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

Binary relational expression

\[
\begin{align*}
\text{call} & : n \rightarrow m \\
\text{init} & : m = n \rightarrow m_i \overset{\text{def}}{\rightarrow} c \\
\text{invoke} & : n \rightarrow m = n \rightarrow m \mid n \rightarrow m_i \overset{\text{def}}{\rightarrow} c \\
\text{invoke} & : n \rightarrow + m \\
\end{align*}
\]

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

```
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

Code query

Total # methods
# Covered methods
Code query results
Querying the results
Package-level query

```sql
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

<table>
<thead>
<tr>
<th>PackageName</th>
<th>NumberOfCoveredMethods</th>
<th>NumberOfDefinedMethods</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpacman</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>jpacman.controller</td>
<td>10</td>
<td>47</td>
</tr>
<tr>
<td>jpacman.model</td>
<td>10</td>
<td>130</td>
</tr>
</tbody>
</table>

```
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?

- Experimental design
- Data set characterization
- Comparison of results

IPA Herfstdagen, Nunspeet, November 24-28, 2008
Static Estimation of Test Coverage
Tiago Alves
Experimental design

**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

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

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

Clover

Static

IPA Herfstdagen, Nunspeet, November 24-28, 2008
Static Estimation of Test Coverage
Tiago Alves
Dom4j: detailed statistical analysis
Class coverage comparison + differences

Static and Clover coverage at class level

Histogram of the differences at class level

IPA Herfstdagen, Nunspeet, November 24-28, 2008
Static Estimation of Test Coverage
Tiago Alves
Dom4j: detailed statistical analysis
Package coverage comparison + differences

IPA Herfstdagen, Nunspeet, November 24-28, 2008
Static Estimation of Test Coverage
Tiago Alves
### Statistical analysis
(Class and package coverage comparison)

<table>
<thead>
<tr>
<th>System name</th>
<th>Spearman</th>
<th>Median</th>
<th>Interquartile range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class</td>
<td>Package</td>
<td>Class</td>
</tr>
<tr>
<td>JPacMan</td>
<td>0.467*</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Certification</td>
<td>0.368**</td>
<td>0.520</td>
<td>0</td>
</tr>
<tr>
<td>G System</td>
<td>0.774**</td>
<td>0.694**</td>
<td>0</td>
</tr>
<tr>
<td>Dom4j</td>
<td>0.584**</td>
<td>0.620**</td>
<td>0.167</td>
</tr>
<tr>
<td>Utils</td>
<td>0.825**</td>
<td>0.778**</td>
<td>0</td>
</tr>
<tr>
<td>JGap</td>
<td>0.733**</td>
<td>0.786**</td>
<td>0</td>
</tr>
<tr>
<td>Collections</td>
<td>0.549**</td>
<td>0.776**</td>
<td>0</td>
</tr>
<tr>
<td>PMD</td>
<td>0.638**</td>
<td>0.655**</td>
<td>0</td>
</tr>
<tr>
<td>R System</td>
<td>0.727**</td>
<td>0.723**</td>
<td>0</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>0.632**</td>
<td>0.694**</td>
<td>0</td>
</tr>
<tr>
<td>DocGen</td>
<td>0.397**</td>
<td>0.459**</td>
<td>0</td>
</tr>
<tr>
<td>Analysis</td>
<td>0.391**</td>
<td>0.486**</td>
<td>0</td>
</tr>
</tbody>
</table>
Statistical analysis
(System-level coverage comparison)

Correlation: 0.769

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

IPA Herfstdagen, Nunspeet, November 24-28, 2008
Static Estimation of Test Coverage
Tiago Alves
Static estimation of test coverage

Tiago Alves

IPA Herfstdagen, Nunspeet, November 24-28, 2008
Static Estimation of Test Coverage
Tiago Alves
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