Software Improvement Group

Static Estimation of Test Coverage

Tiago Alves & Joost Visser

May 29, 2008

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

Introduction

Software Improvement Group

Background

- 2004 - Computer Science and Systems Engineering, University of Minho, Braga
- 2006 - MSc in Informatics
  - Grammar engineering (ISO VDM-SL Grammar in SDF)
  - SdfMetz: Metrication of syntax formalisms (SDF, DMS, Antlr, and Bison)
  - VooDooM: model transformation and code generation (VDM-SL -> SQL)
  - Teaching compiler course 3rd year students
- 2006 - European Space Operations Center (ESA), Damstadt, Germany
  - Team member responsible for managing development of a prototype system
  - Requirements specification for new software system
  - Acceptance test of a communication system.
- 2007 - PhD at University of Minho and Software Improvement Group
  - 2LT Extensions: Contraint-aware transformations
  - Static estimation of test coverage

Measuring testing 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:

- Tied with software development process
  - Full installation required (sources + libraries)
- Instrumentation of source/byte code (problematic in embedded systems)
- Execution (Hardware or time constraints)

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

Research Challenge

Software Improvement Group

13th Testdag, Delft, November 2007

- I. Heitlager, T. Kuipers, J. Visser “Observing unit test maturity in the wild”

Research questions:

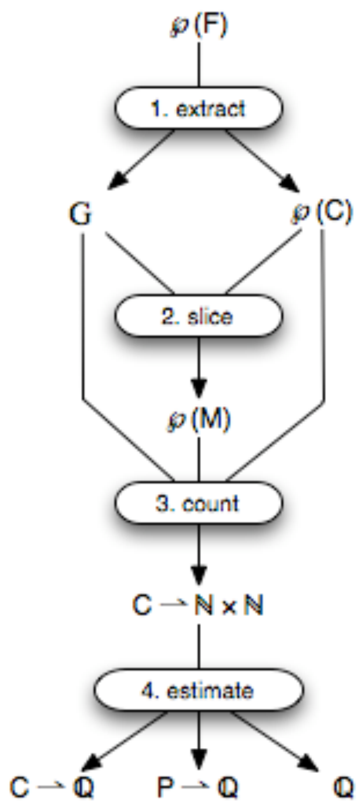
- Is it possible to determine test coverage without running tests?
- What trade-offs can be made between sophistication and accuracy?

Requirements

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

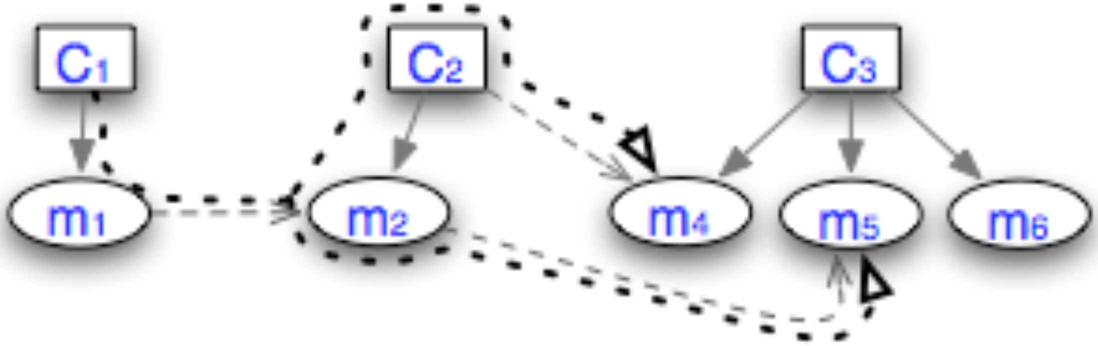
Solution sketch

Software Improvement Group



1. **Extract**
   - Extract structural and call information
   - Determine set of test classes
2. **Slice (modified)**
   - Slice graph starting from the 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



Binary relational expression

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

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

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

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

SemmleCode implementation

```
predicate isTestCovered() {
    exists(TestClass tc, Callable tm
            | tc.contains(tm) and
              tm.polyCalls+(this))
    or
    exists(Callable m |m.hasName("<clinit>")
                          and m.polyCalls+(this))
  }
```

What can go wrong?
(Sources of imprecision)

Software Improvement Group

Java language

- Control flow
  - Conditional statements (if-then, if-then-else)
  - Switch statements (switch, case)
  - Looping statements (for, while, do-while)
  - Branching statements (break, continue, return)
- Dynamic dispatching
  - Inheritance
- Overloading

General issues

- Frameworks / Libraries call backs
- Identification of test code
  - Test code is recognized by determining JUnit dependencies
- ///CLOVER:OFF flags

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

Experimental design

Software Improvement Group

Data set selection and characterization

- Open-source and proprietary Java systems
- Available clover report (XML or HTML)

Execution of experiment

- SemmleCode execution (text file export + scripts for CSV conversion)
- XML Clover extraction (XSLT transformations to CSV conversion)
- HTML Clover extraction (grep, sed, awk, wc scripts to CSV conversion)
- Custom built java tool to read CSV files and XLS creation

Statistical analysis

- Histograms (distribution)
- Scatter charts (correlation)
- Spearman (correlation)
- Inter-quartile ranges (dispersion)

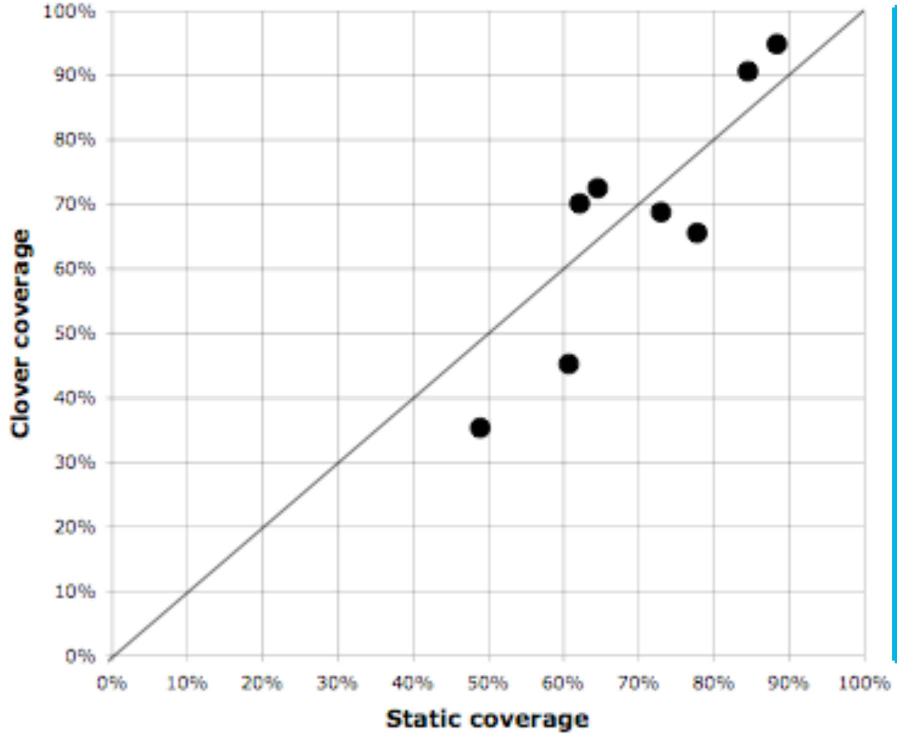
Data set characterization

Software Improvement Group

System name	LOC	#Packages	#Classes	#Methods
Pacman	2987	2	20	181
G System	6265	15	53	385
Utils	23604	35	260	2571
Dom4j	42863	14	144	2481
PMD	51219	40	455	3398
Architect	58477	17	220	2781
DepFinder	73861	12	261	4686
R System	79776	62	600	5620

Statistical analysis (System coverage comparison)

Software Improvement Group



System	Static	Clover	Diff
Pacman	84.53%	90.61%	-6.08%
G System	88.37%	94.81%	-6.44%
Utils	72.95%	68.73%	4.22%
Dom4j	60.69%	45.20%	15.49%
PMD	77.77%	65.50%	12.27%
Architect	48.98%	35.30%	13.68%
DepFinder	62.14%	70.08%	-7.94%
R System	64.54%	72.46%	-7.92%

Statistical Analysis
(Class and package coverage comparison)

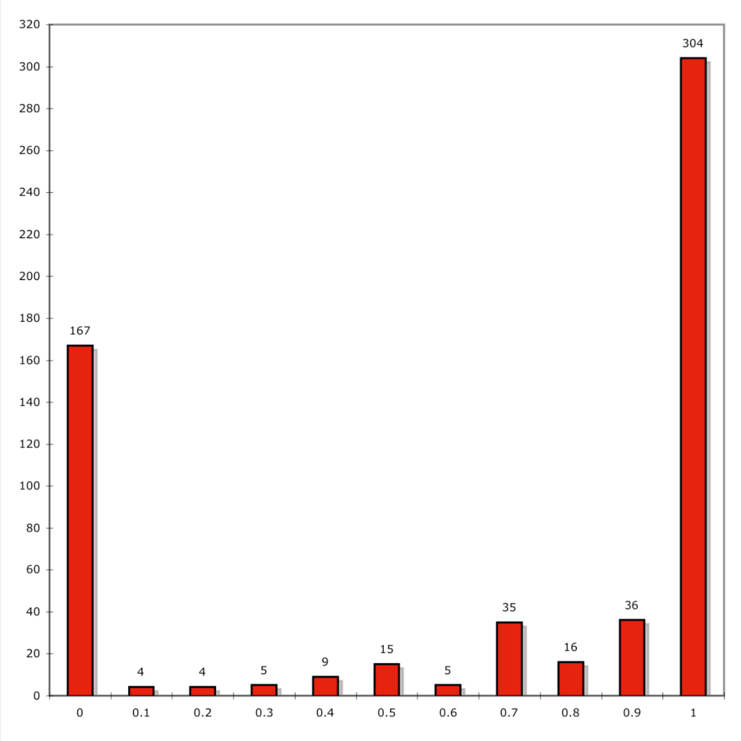
Software Improvement Group

System name	Spearman		Median		Inter-quartile range	
	Class	Package	Class	Package	Class	Package
Pacman	0.275	1	0	-0.086	0.093	-
G System	0.777**	0.694**	0	0	0	0.046
Utils	0.737**	0.825**	0	0.01	0.038	0.109
Dom4j	0.557**	0.625*	0.17	0.099	0.373	0.243
PMD	0.702**	0.693**	0	0.066	0.128	0.189
Architect	0.504**	0.5*	0	0.064	0.28	0.197
DepFinder	0.659**	0.396	0	-0.004	0.13	0.132
R System	0.752**	0.652**	0	-0.1	0.01	0.186

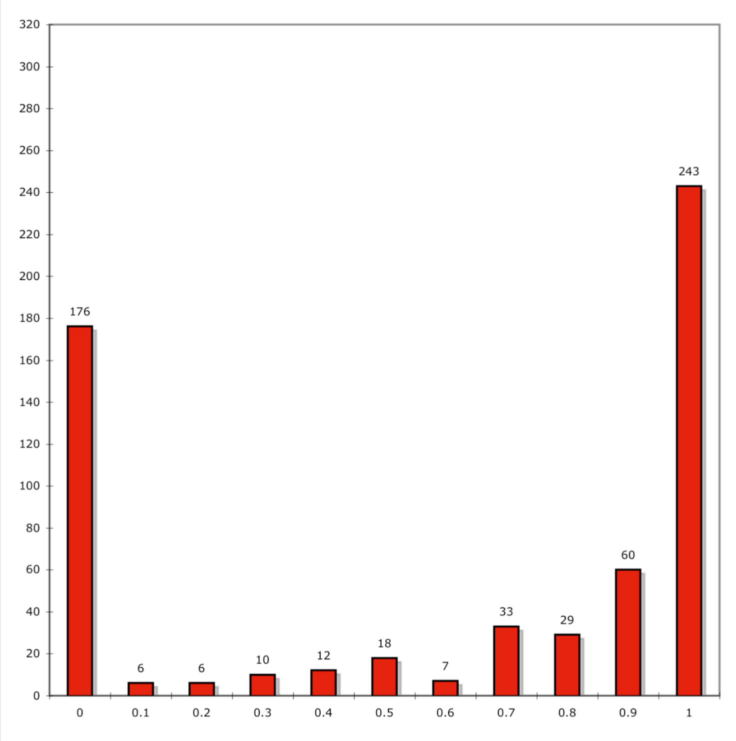
R System: detailed statistical analysis (Class coverage histograms comparison)

Software Improvement Group

Clover

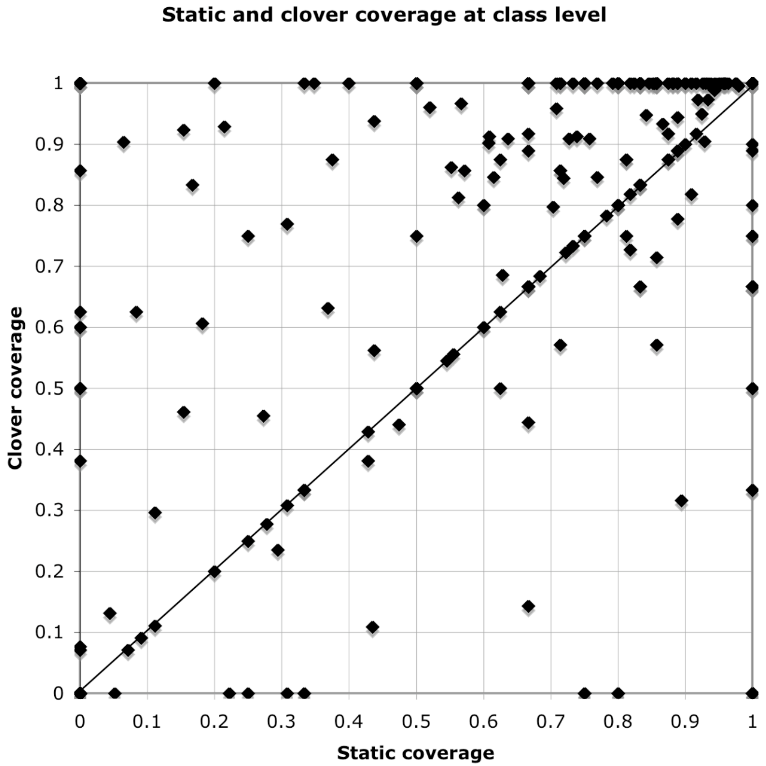


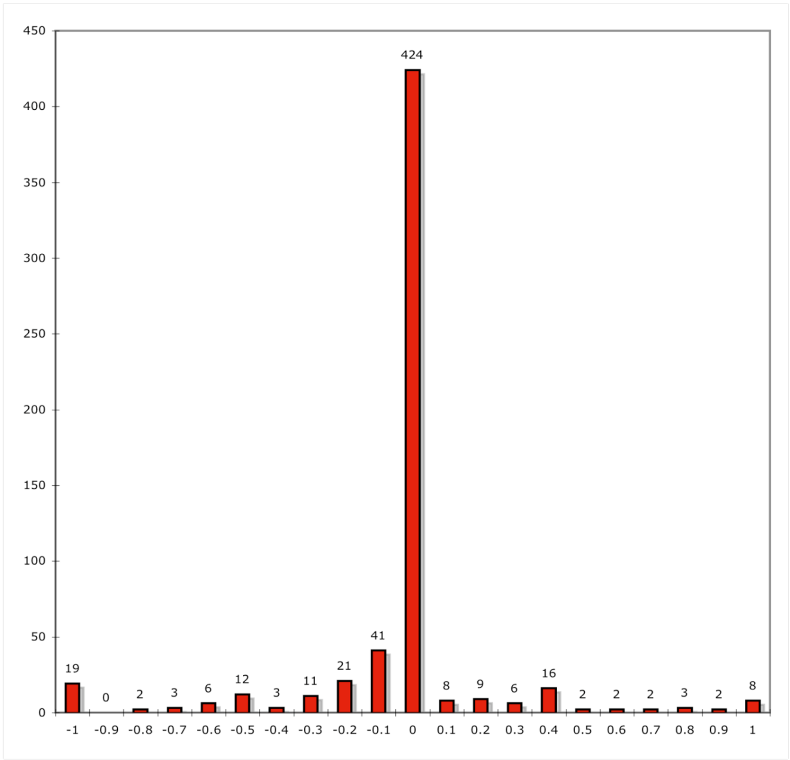
Static



Static estimation of test coverage

R System: detailed statistical analysis (Class coverage comparison + differences)

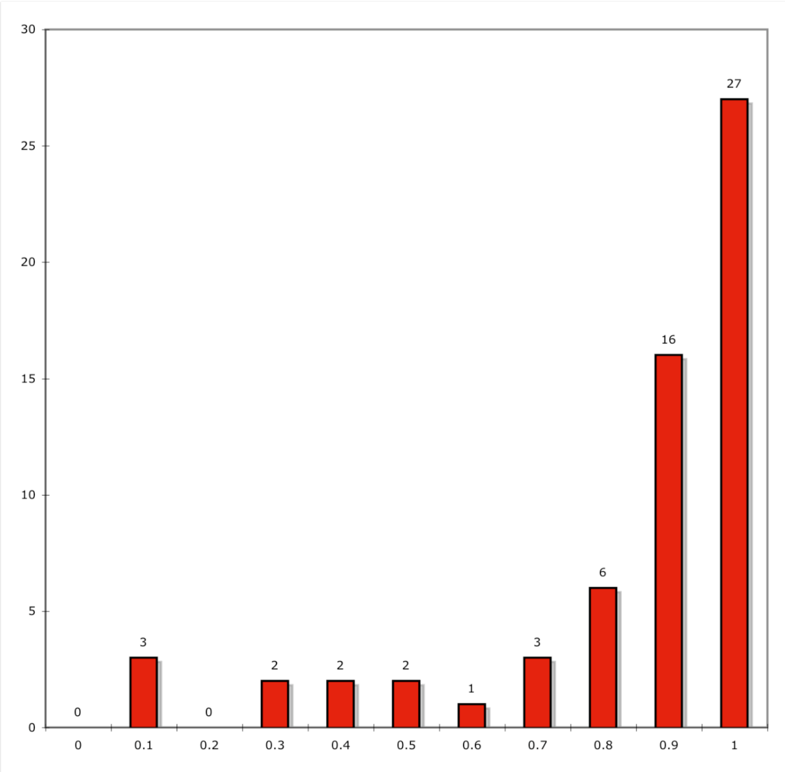




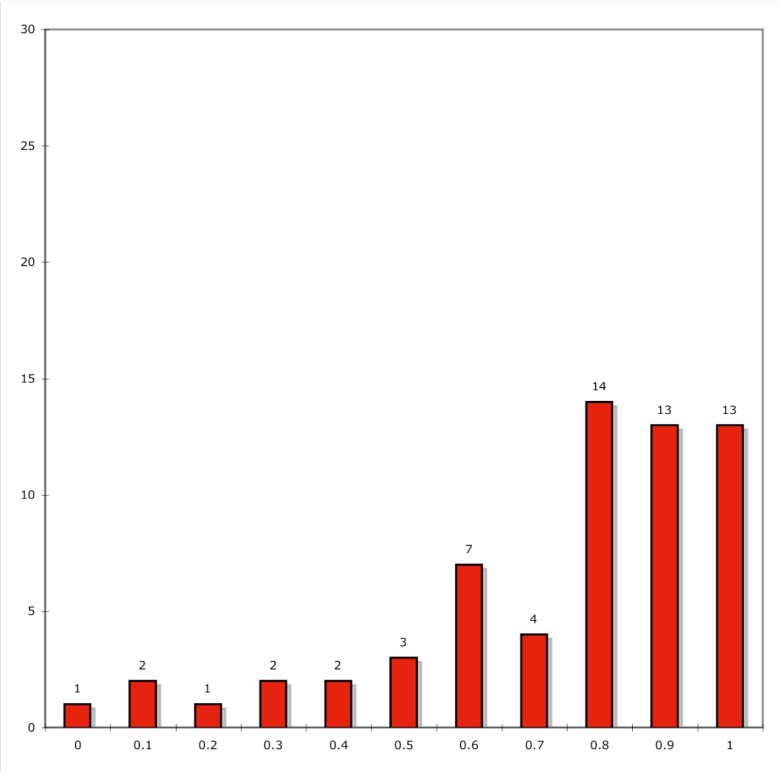
R System: detailed statistical analysis (Package coverage histograms comparison)

Software Improvement Group

Clover

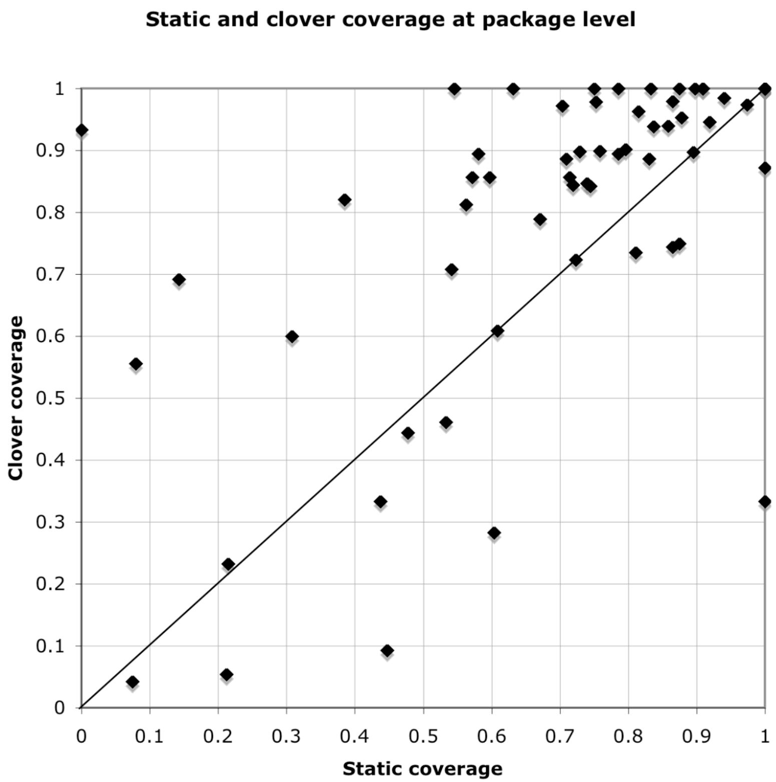


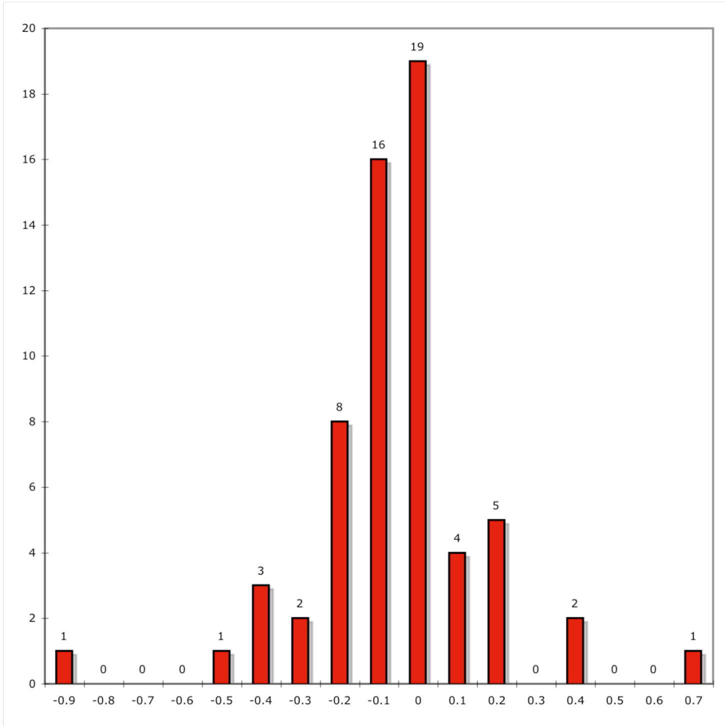
Static



Static estimation of test coverage

R System: detailed statistical analysis (Package coverage comparison + differences)





Static estimation of test coverage

Conclusion

Is it possible to determine test coverage without running tests?

- Yes!!!
- 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 for system coverage: 9%
- Class and Package coverage needs further improvement

Implementation

- SemmleCode: 92 LOC = 76 LOC (extensions) + 16 LOC (3 Queries)
- SIG Monitor: 265 LOC = 136 + 56 + 22 + 23 + 14 + 15 (6 classes)

Future work

Software Improvement Group

Implementation:

- Add analysis to production at SIG (done)
- Add tests (in progress)

Research:

- Use LOC as a weight for better estimation of coverage
- Compute static levels of testing
  - T. Kanstrén. *Towards a deeper understanding of test coverage*
- Investigate the use of McCabe + #Tests + #asserts + Test(LOC) / Code(LOC)

Software Improvement Group

Questions?