Software Improvement Group

Comparative Study of Code Query Technologies

Tiago Alves

March 25th, 2009

Software Improvement Group
Arent Janszoon Ernststraat 595-H
NL-1082 LD Amsterdam
www.sig.nl

Software Improvement Group

Company

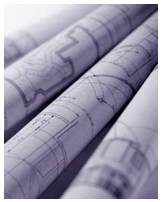
- Spin-off from CWI in 2000, self-owned, independent
- Management consultancy grounded in source code analysis
- Innovative, strong academic background, award-winning, profitable

Services

- Software Risk Assessments (snapshot) and Software Monitoring (continuous)
- Toolset enables to analyze source code in an automated manner
- Experienced staff transforms analysis data into recommendations
- We analyze over 50 systems annually
- Focus on technical quality, primarily maintainability / evolvability

Services

Software Improvement Group



DocGen

- Automated generation of technical documentation

Assessments

- In-depth investigation of software quality and risks

Monitoring

- Continuous measurement and decision support

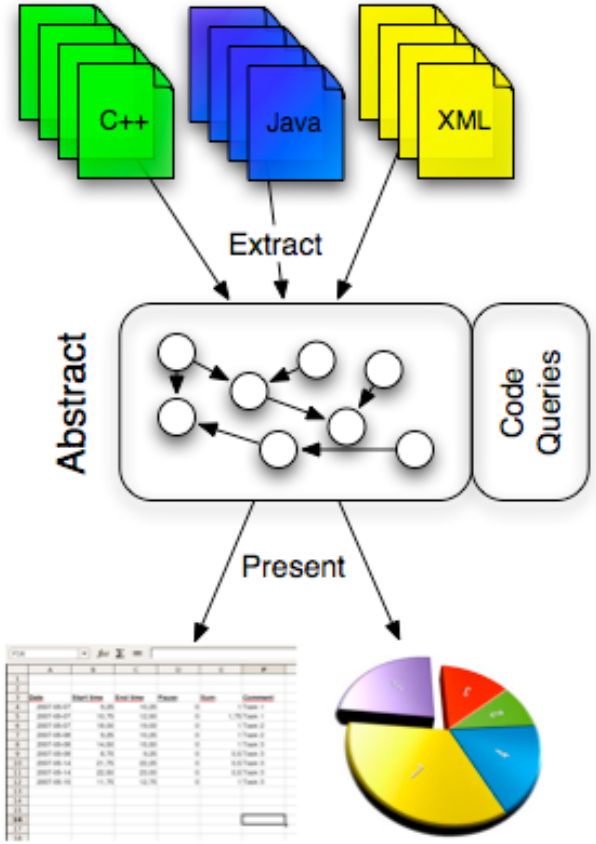
Certification

- Five levels of technical quality

Who is using our services?

Software Improvement Group

Background



- **Extract**
  - Facts and relations and represent them an intermediary structure
- **Abstract/Enrich**
  - Add new facts and relations
- **Present**
  - Visualize or extract information

Motivation

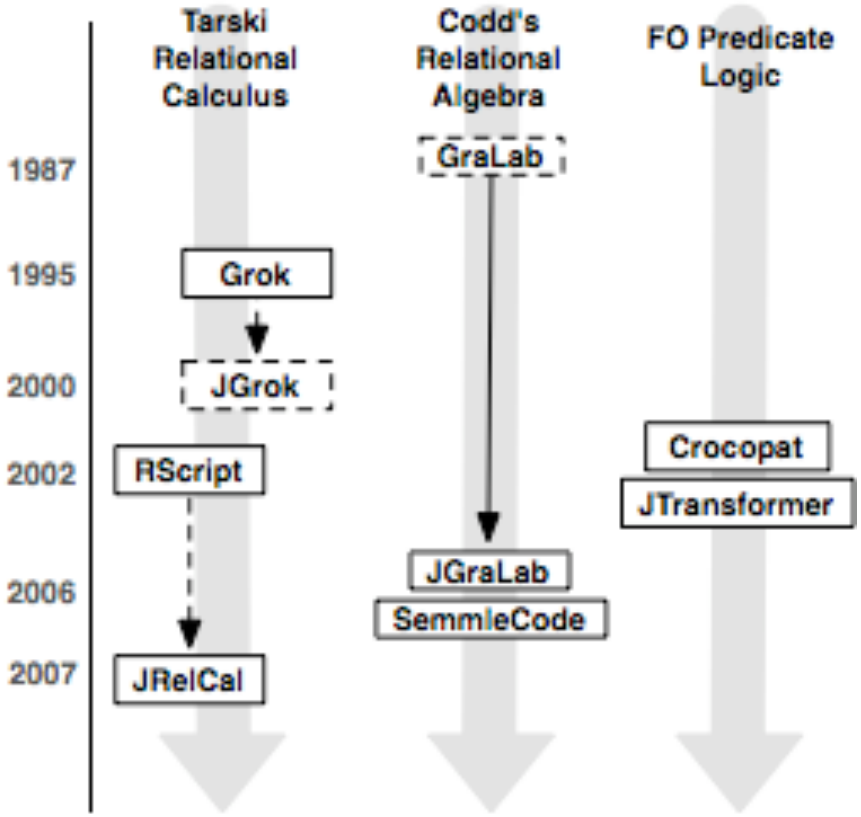
Software Improvement Group

- **Current implementation of SIG tooling**
  - Extraction for several languages
    - Graph as intermediary data structure
  - Abstraction implemented in Java
    - Use of visitors
  - Presentation
    - Tables and Charts
- **Problems**
  - Implementation verbose and imperative
  - Reuse among analysis difficult
  - Error prone

- **Use of code query technologies to improve SIG productivity**
  - Replace current imperative implementation for a more declarative one

Code Query Technologies - Timeline

Software Improvement Group



- **Grok, JGrok**
  - Ric Holt, Canada
  - Implemented in Turing
- **Rscript**
  - Paul Klint, Netherlands
  - Implemented in ASF+SDF
- **JRelCal**
  - Tijs van der Storm, Netherlands
  - Implemented in Java
- **GraLab, JGraLab**
  - Jürgen Ebert, Germany
  - Implemented in Java
- **SemmleCode**
  - Oege de Moor, UK
  - Implemented in Java
- **CrocoPat**
  - Dirk Beyer, Germany
  - Implemented in C
- **JTransformer**
  - Günter Kniesel, Germany
  - Implemented in Java

Comparison

Software Improvement Group

- **Code query example**
  - Experience the language and tool
- **Language criteria**
  - Overview of the language features
- **Tool criteria**
  - Overview of the tool features

Language criteria

- **Style/Paradigm**
  - Compare implementation conciseness
- **Types**
  - Support for Integers, Chars, Strings, ...
- **Parametrization**
  - Behavior depends on a parameter value
- **Polymorphism**
  - Abstract over the fact types
- **Modularity**
  - Reuse of queries to construct other queries
- **Libraries**
  - Support of libraries of queries

Tool criteria

- **Output formats**
  - Text, preformatted text, tables, charts, others?
- **Interactive interface**
  - Command line interface (CLI), Graphical user interface (GUI), Eclipse plug-in
- **API support**
  - Invocations of the functionality from a host program
- **Interchange format**
  - To store facts from the extraction and results of abstraction
- **Extraction support**
  - None, Java, C/C++, XML, others?
- **Licensing**
  - Free, Open-source, Proprietary

Scenarios

- **Interactive use**
  - The tool is used directly by the software analyst (exploratory)
  - The user specifies and executes the queries, and extracts results

- **Tool integration**
  - The tool is used by a programmer as a component to build other tools
  - Reimplementation of existent functionality

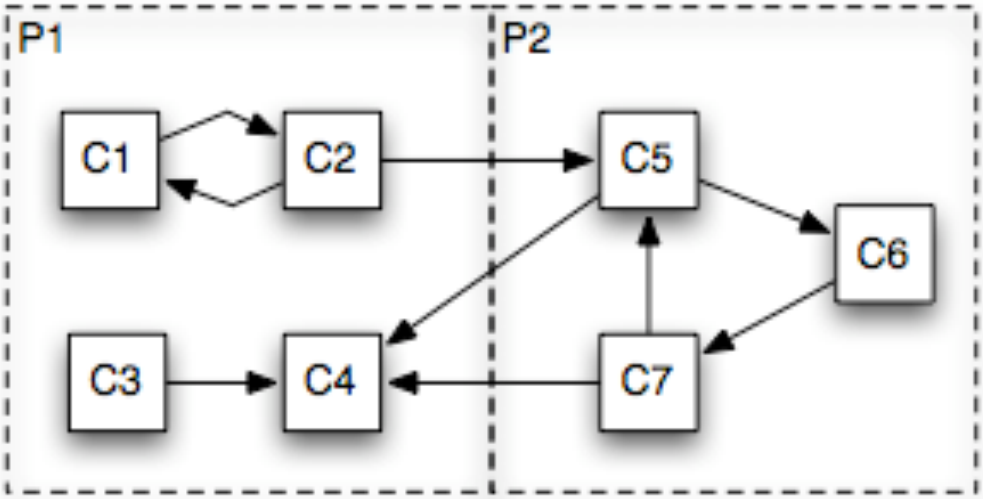
Criteria vs. Scenarios

Software Improvement Group

Criteria vs. Scenarios		Interactive use	Tool integration
Language	Style/Paradigm	Important	Important
	Types	Important	Relevant
	Parametrization	Important	Relevant
	Polymorphism	Important	Relevant
	Modules	Important	Relevant
	Libraries	Important	Relevant
Tool	Output formats	Important	Important
	Interactive use	Important	Not important
	API support	Not important	Important
	Interchange format	Important	Important
	Extraction support	Important	Relevant
	Licensing	Relevant	Important

Package instability

Software Improvement Group



Afferent Coupling

Ca = # of classes outside the package that depend upon classes within the package

Ca = { (P1, C5), (P1, C7), (P2, C2) }

Efferent Coupling

Ce = # of classes inside the package that depend upon classes outside the package

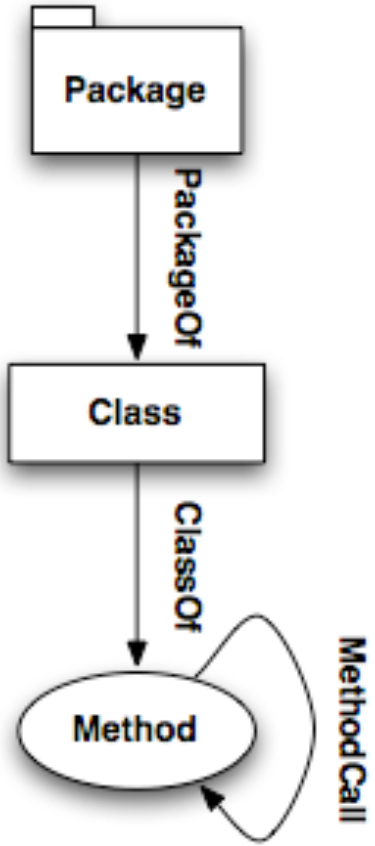
Ce = { (P1, C2), (P2, C5), (P2,C7) }

Package Instability I = Ce / (Ca + Ce)

I = { (P1, 1 / (2 + 1)) , (P2, 2 / (1 + 2)) } = { (P1, 0.33), (P2, 0.67) }

Input Relations

Software Improvement Group



Relations

PackageOf : Package x Class

ClassOf : Class x Method

MethodCall : Method x Method

Defined relations

ClassDepInterPackg : Class x Class

AffCoupling : Package x Class
EffCoupling : Package x Class

AfferentCoupling : Package x N
EfferentCoupling : Package x N

PackageInstability : Package x N

Lifting implementation example

Software Improvement Group

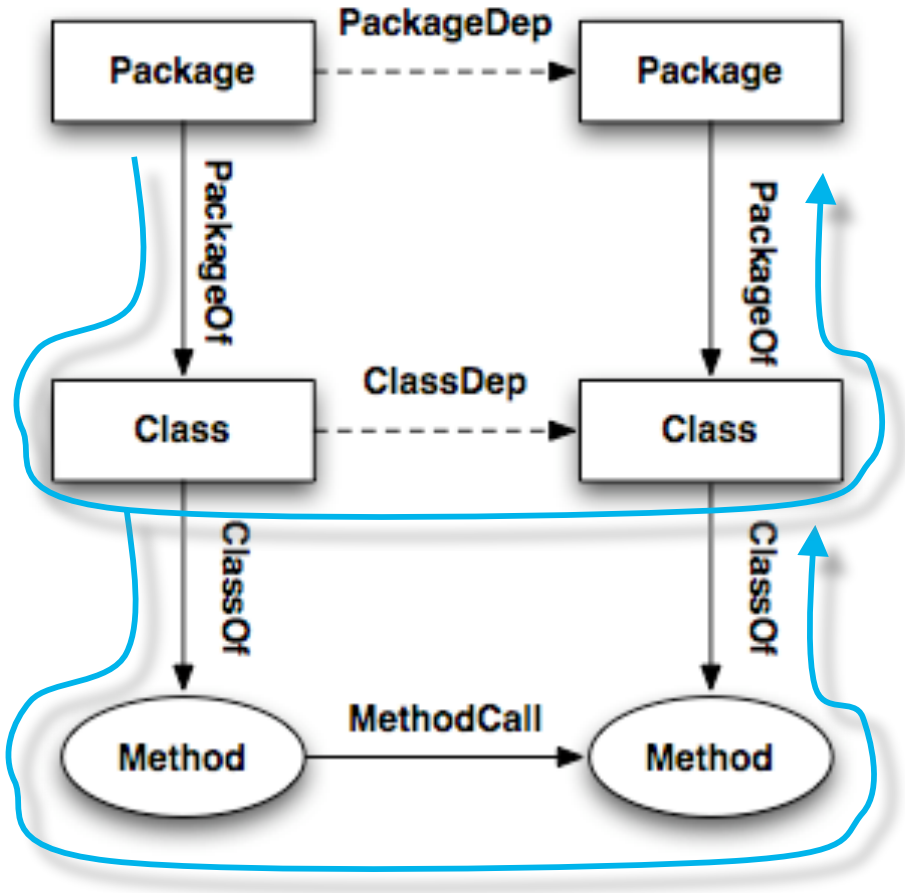
Input relations

ClassOf : Class x Method
MethodCall : Method x Method

Compute lifting

ClassDep : Class x Class
= ClassOf o MethodCall o inv (ClassOf)

PacakgeDep : Package x Package
= PackageOf o ClassDep o inv (PackageOf)



Grok

Software Improvement Group


```
PackageDep := PackageOf o ClassDep o (inv PackageOf)

PackgDepInterPackg := PackageDep - (id dom PackageOf)

ClassFowardRel := (inv PackageOf) o PackgDepInterPackg o PackageOf

ClassDepInterPackg := ClassForwardRel ^ ClassDep

AffCoupling := PackageOf o (inv ClassDepInterPackg)

AfferentCoupling := indegree (AffCoupling)
```

Rscript

Software Improvement Group

```
rel[str,str] PackageOf
rel[str,str] ClassOf
rel[str,str] MethodCall

rel[&T1, int] indegree(rel[&T1,&T2] R)
  = { <D, #(domainR(R, {D}))> | <&T1 D, &T2 U> : R }

rel[str,str] ClassDepInterPackg
  = { <C1,C2> | <str P1, str C1> : PackageOf
              , <str P2, str C2> : PackageOf
              , <str C3, str C4> : ClassDep
              , C1 == C3, C2 == C4, P1 != P2 }

rel[str,str] AffCoupling = PackageOf o inv(ClassDepInterPackg)

rel[str,int] AfferentCoupling = indegree(AffCoupling)

rel [str,int] PackageInstability
 = { <P1, (100*N1)/(N1+N2)> | <str P1, int N1> : EfferentCoupling
                            , <str P2, int N2> : AfferentCoupling, P1 == P2 }
```

JRelCal

Software Improvement Group

```
Relation<String, String> packageDep
  = packageOf.compose(classDep.compose(packageOf.inverse()));

Relation<String, String> packgDepInterPackg
  = packageDep.difference(packageOf.domain().id());

Relation<String,String> classForwardRel
  = (packageOf.inverse()).compose(packgDepInterPackg).compose(packageOf);

Relation<String,String> classDepInterPackg
  = classForwardRel.intersection(classDep);

Relation<String,String> affCoupling
  = packageOf.compose(classDepInterPackg.inverse());

Relation<String,Int> afferentCoupling = affCoupling.indegree();
```


```
predicate classDepInterPackg( Class c1, Class c2) {
 c1.getPackage() != c2.getPackage() and classDep(c1,c2)
}
class MyPackage extends Package {
 MyPackage() { this.fromSource() }

 predicate affCoupling(Class c) {
  exists(Class c1 | this.contains(c1) and classDepInterPackg(c1, c))
 }
 int afferentCoupling() {
  result = count(Class c | this.affCoupling(c))
 }
 float packageInstability() {
  result = (1.0 * this.efferentCoupling()) / (this.afferentCoupling()
 + this.afferentCoupling())
 }
}
```

JGraLab

Software Improvement Group

```
from p : V {JavaPackage}
reportMap p,
   from outerClass : V {JavaClass}
   with
     (not p --> {PackageOf} outerClass) and
     (p --> {PackageOf} <-- {ClassDep} outerClass)
   report outerClass end
end store as AffCoupling

using AffCoupling:
from p : V {JavaPackage}
reportMap p, count(get(AffCoupling,p)) end
store as AfferentCoupling

using AfferentCoupling, EfferentCoupling:
from p : V {JavaPackage}
reportMap p, get(EfferentCoupling,p) /
         ( get(EfferentCoupling,p) + get(AfferentCoupling,p))
end store as PackageInstability
```

CrocoPat

Software Improvement Group

```
ClassDepInterPackg(c1,c2)
 := EX( p1, PackageOf(p1, c1) & EX( p2, PackageOf(p2, c2) &
        !=(p1,p2) & ClassDep( c1, c2)));

AffCoupling(p,c)
 := EX( c1, PackageOf( p, c1) & ClassDepInterPackg( c, c1));

Package(x) := PackageOf(x,_);

FOR p IN Package(x) {
 ca := #(AffCoupling(p,c));
 PRINT "AfferentCoupling ", p, " ", ca, ENDL;

 ce := #(EffCoupling(p,c));
 PRINT "EfferentCoupling ", p, " ", ce, ENDL;

 i := ce / (ca + ce);
 PRINT "Instability ", p, " ", i, ENDL;
}
```

JTransformer

Software Improvement Group

```
classDepInterPackg(C1, C2) :-
   packageOf(P1, C1), packageOf(P2,C2),
   not(P1 = P2), classDep(C1,C2).

affCoupling(P, C) :-
   packageOf(P, C1), classDepInterPackg(C, C1).

afferentCoupling(P,N) :-
   setof(C, affCoupling(P, C), AffClasses),
   length(AffClasses, N).

packageInstability(P, I) :-
  efferentCoupling(P, Ec),
  afferentCoupling(P, Ac),
  I is Ec/(Ec + Ac).
```

Language comparison

Software Improvement Group

Criteria vs. tools		Grok	Rscript	JRelCal	SemmleCode	CrocoPat	JGraLab	JTransformer
Style/paradigm		Relational	Relational & comprehensions	API Relational	OO & SQL-like	FO-logic & imperative	SQL-like & Path expr.	FO-logic
Types	String	x	x	x	x	x	x	x
	Int	x	x	x	x	x	x	x
	Real	x	-	x	x	x	x	x
	Bool	-	x	x	x	x	x	x
	Other	-	Composite & location	Java	Class	-	Edge & Node	Logic terms
Parametrization		-	x	x	-	x	x	x
Polymorphism		-	x	x	x	-	x	x
Modules		x	x	x	x	x	-	x
Libraries		-	-	x	x	-	-	x

Tool comparison

Software Improvement Group

Criteria vs. tools	Grok	Rscript	JRelCal	SemmleCode	CrocoPat	JGraLab	JTransformer
Output formats	Text	Rstore	Sets & Relations	Text, Charts, maps, graphs	Text, RSF	Text, HTML	Text
Interactive interface	CLI	CLI, GUI	-	CLI, Eclipse	CLI	CLI	Eclipse
API support	-	-	x	x	x	x	x
Interchange format	RSF, TA	Rstore	RSF	-	RSF	TGraph	Prolog
Extraction support	C++	-	-	Java, XML	-	Java, C	Java
Licensing	-	BSD	LGPL	Proprietary	LGPL	GPL 2 Proprietary	EPL

Summary

Software Improvement Group

- **Language criteria**
  - No significant differences found
  - It is not possible to implement Package Instability in Grok
- **Tool criteria**
  - Significant differences: interchange format, extraction, licensing
  - Poor support for extraction
- **Interactive use**
  - Only JRelCal is less suitable.
- **Tool integration**
  - JRelCal, SemmleCode, CrocoPat, JGraLab, JTransfomer
  - Grok, Rscript only through interchange format

Conclusion

- **Compared seven code query technologies**
  - Package instability example
  - Six language criteria
  - Six tool criteria
- **Comparison not evaluation**
- **Presented findings**
  - Allow an informed decision about which tool to choose

Future work & challenges

- **Future wok**
  - Add more tools / formalisms
  - Performance comparison
- **Challenges**
  - Adoption of each tool stronger points
  - Better support for libraries, interchange format and extractors
  - Availability of API
  - Interfacing through IDE
- **Research directions**
  - Analyze several versions of software
  - Architecture checking

Questions?

Software Improvement Group

W: www.sig.nl

T: +31 20 3140950

t.alves@sig.nl