

Semmlle

Querying Software Projects with SemmlleCode

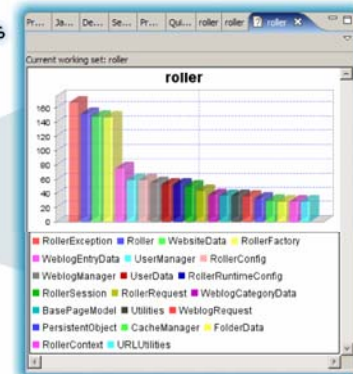
Q1: What are the types with many dependencies?

```
from RefType t, int n
where t.fromSource() and
      n = count(RefType s |
                depends(s,t))
      and n > 25
select t, n order by n desc
```

annotations
applications AP
fields
generics
calls
interfaces
frameworks
enums

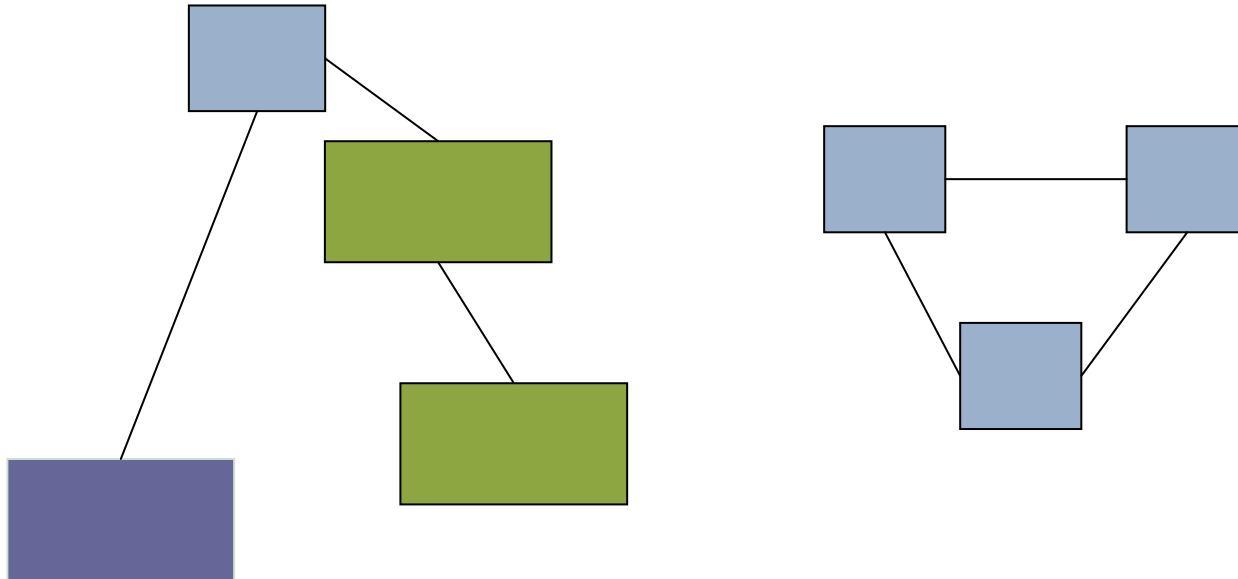


Relational
representation





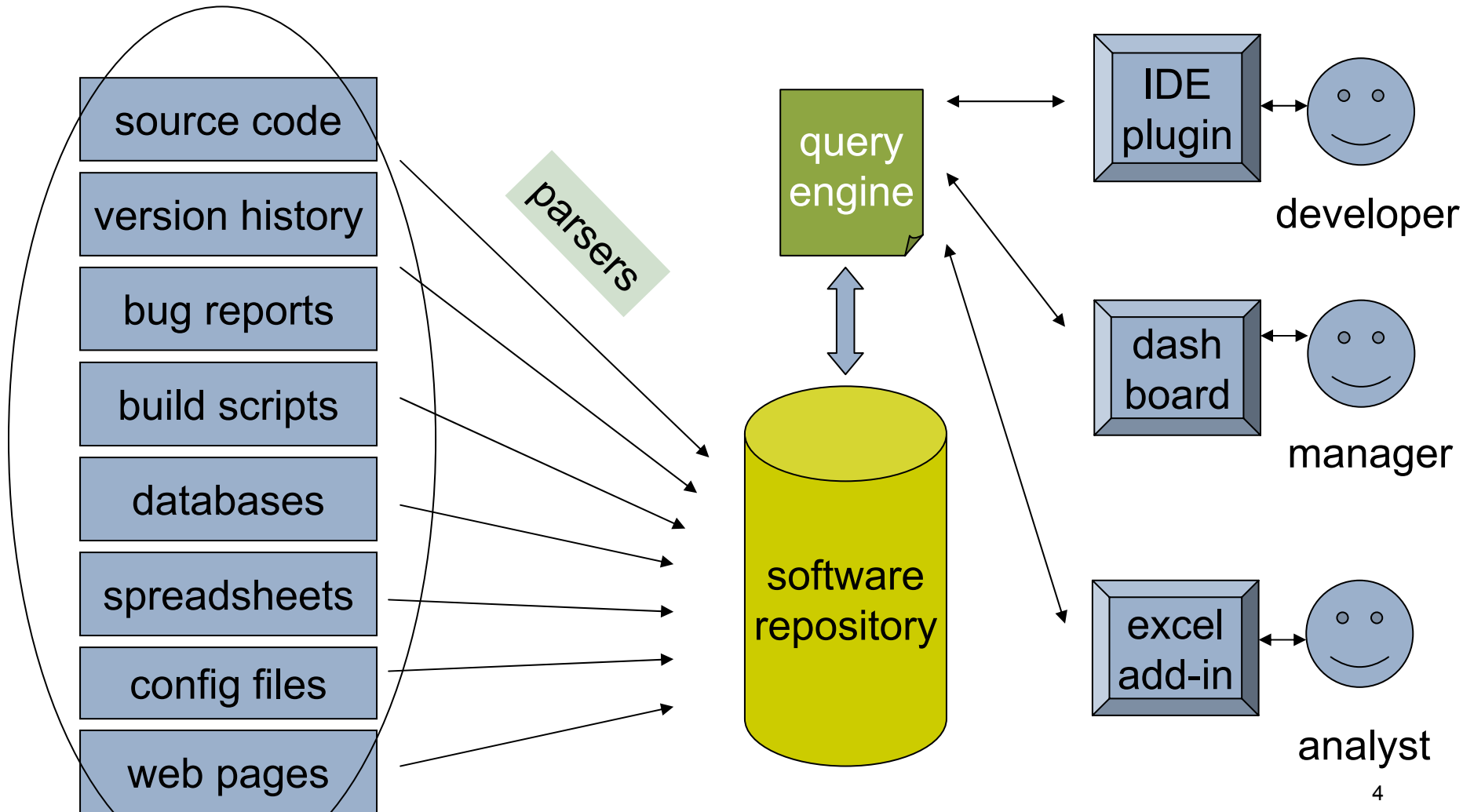
Analysing the glue is imperative: XML config files



Pieces of Java are glued together with XML config files: Spring, Hibernate, JBoss....
cannot analyse Java without these XML files



The vision





The problem

design query language and engine
for accessing vast repository of
different types of source artefact

libraries of queries:
tailor framework to different types of artefact



Not a solution 0: SQL

```
CREATE PROCEDURE sample AS  
  DECLARE @Count INTEGER  
  DECLARE @NewCount INTEGER  
  DECLARE @TempCount INTEGER
```

- verbose
- complex recursion
- impoverished type checking
- weird behaviour is standard

```
UNION  
SELECT DISTINCT m.id  
FROM foo.methods AS m
```

...



Not a solution 1: Prolog

```
cu(CU) :- nonvar(CU),cus(CU,A2),!.
```

```
cu(CU) :- var(CU),cus(CU,A2),!.
```

```
type(C) :-
```

```
type(C) :-
```

```
subtype(T,S)
```

```
lsubtypeplus(T,S)
```

```
lsubtypeplus(T,S)
```

```
rsubtypeplus(T,S)
```

```
rsubtypeplus(T,S)
```

```
subtypeplus(T,S)
```

```
subtypeplus(T,S)
```

- baroque execution model:

- non-termination
- hard to optimise

- need extra-logical annotations

- every intermediate result is named (SUBT,T).

```
:- subtype(T,S).
```

```
:- subtype(T,MID), lsubtypeplus(MID,S).
```

```
:- subtype(T,S).
```

```
:- rsubtypeplus(MID,S),subtype(T,MID).
```

```
:- var(T),rsubtypeplus(T,S).
```

```
:- nonvar(T),lsubtypeplus(T,S).
```



Not a solution 2: XQuery

```
for $A in $db:prj-files/bat:class[@name=$param]
```

```
let $methods := java:all-methods($A)
```

```
let $n := fn:count($methods)
```

```
let $v := fn:count($methods)
```

```
let $k := fn:count($methods)
```

```
return
```

- cryptic
- hard to optimise
- poor support for creating libraries
- + widely known

```
return fn:count(
```

```
    java:accessors($v, $methods)
```

```
    ) div $n
```

```
    ) div $k
```

```
    ) *100
```

```
)
```



Sometimes a solution: perl, sed, awk ...

- + no need for filetype-specific tools
- + great when it works
- tough to ask semantic questions

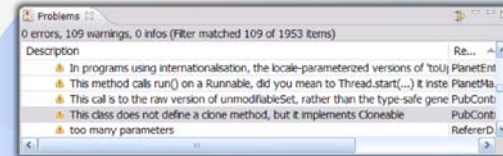
```
find . -type f | cut -f2- -d/ | perl -n -e '/.\.([\^.]*)$/ && print $1' |  
sort | uniq -c | sort -g
```



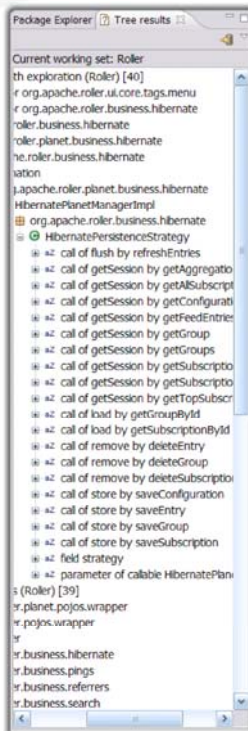
SemmlerCode: the power of .QL

SemmlerCode
<http://semmler.com>

Find bugs, enforce coding rules



Navigate code

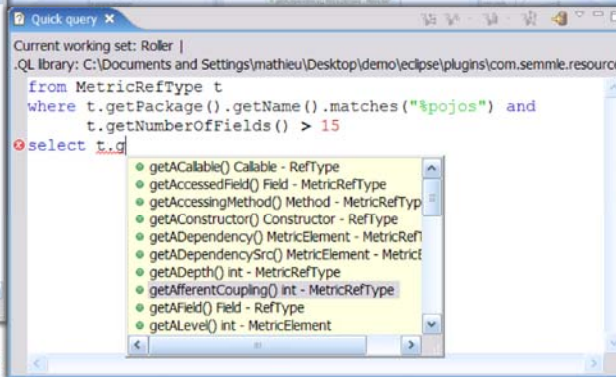


New Eclipse Plugin

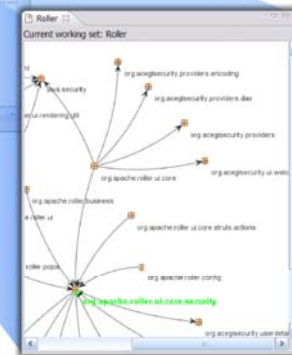
free to download, free to use



Compute metrics



Customise queries



Understand dependencies

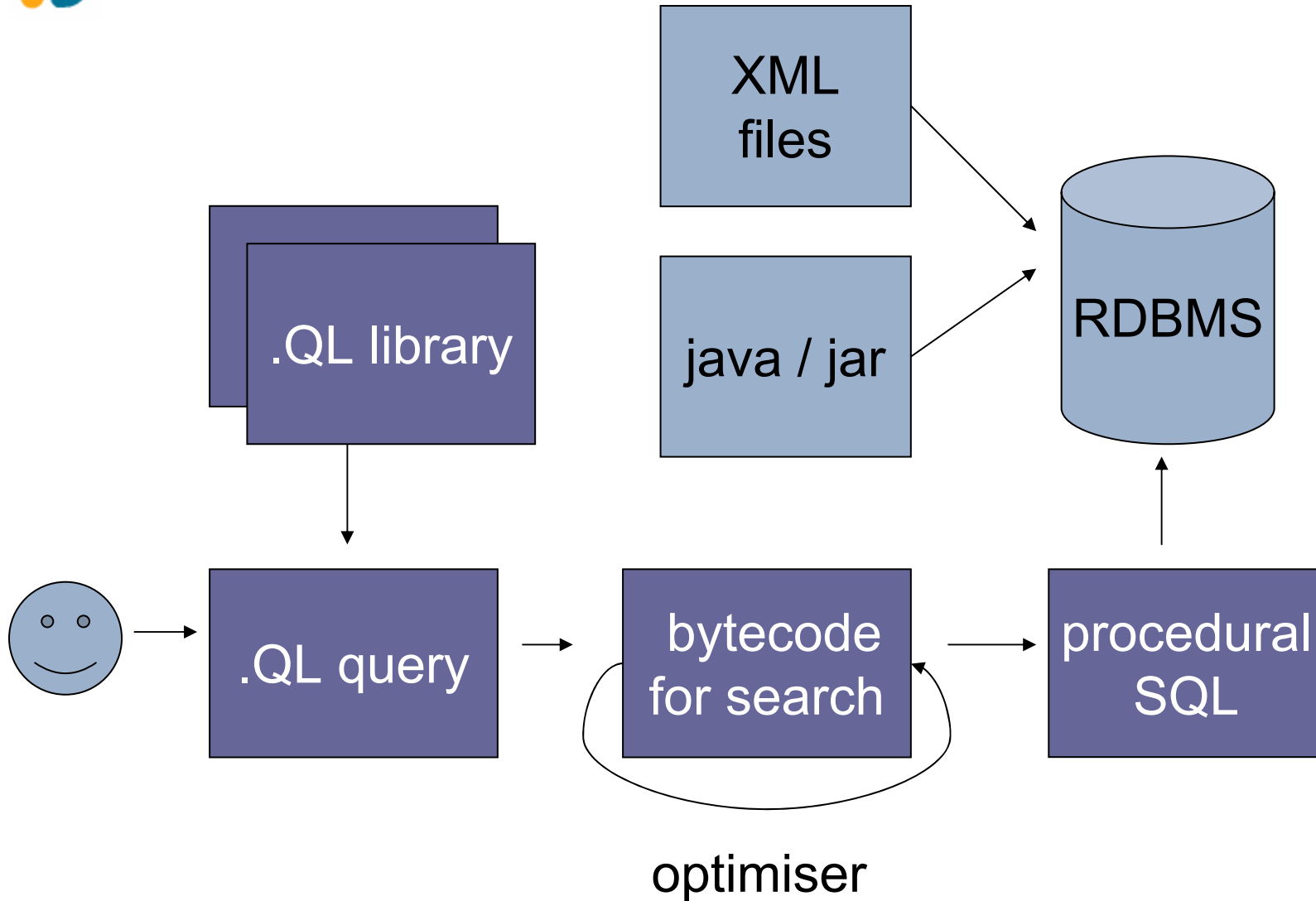


The query language .QL

- ❑ Object-oriented, for creating libraries of queries
- ❑ Recursive queries, as in logic programming
- ❑ Aggregates, borrowed from Edsger W. Dijkstra
- ❑ On top of any traditional relational database



How it works





Demo

The source we shall explore:

- ❑ Alfresco: Enterprise Content Management
- ❑ Spring: Java/JEE Application Framework
- ❑ Builds on Tomcat, JBoss, ...

Vital statistics:

50553	Java methods
6647	Java types
516	XML files

Demo parts:

- out-of-the-box
- writing your own queries
- querying XML config files



The key points of .QL

designed for creating libraries of queries

classes are predicates
inheritance is implication
nondeterministic expressions

recursion with super-simple semantics

syntax familiar to SQL and Java programmers

Semmlle

.QL classes for *any* database

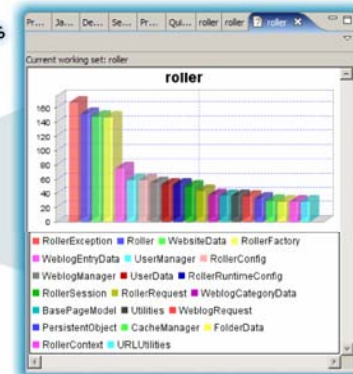
Q1: What are the types with many dependencies?

```
from RefType t, int n
where t.fromSource() and
      n = count(RefType s |
                depends(s,t))
      and n > 25
select t, n order by n desc
```

applications AP
generics
calls
interfaces
frameworks
enums
fields
annotations



Relational
representation





Annotating a database schema with column types

defines @method to be the values in the id column

```
methods(    int id: @method,  
            varchar(100) nodeName: string ref,  
            varchar(900) signature: string ref,  
            int typeid: @type ref,  
            int parentid: @reftype ref,  
            int cuid: @cu ref,  
            int location: @location ref    );
```

declares parentid to be members of @reftype

union column types:
@reftype = @class | @interface | ...



From primitives to classes

```
class MyMethod extends @method {  
    string getName() {methods(this,result,_,_,_,_)}  
    string toString() {result=this.getName()}  
}
```

Size of default library for Java:
~75 class definitions
~1000 SLOC



Too good to be true?

Jeff Ullman, 1991:

It is not possible for a query language to be seriously logical and seriously object-oriented at the same time.



Wrapping up

Java is not enough
source code analysis
tools must process a
multitude of artefacts

libraries of queries
a means to achieve such
heterogeneous tools

.QL
object-oriented queries made easy