bren:school
information and computer sciences

INSTITUTE FOR GENOMICS AND BIOINFORMATICS
University of California, Irvine

Sourcerer: Searching Internet-Scale Software Repositories

Erik Linstead
Department of Computer Science
University of California, Irvine

elinstea@ics.uci.edu

Overview

- Introduction
- Sourcerer Architecture
- Internet-Scale Software Repositories
  - Populating a code database
  - The shape of software
  - Observations/Lessons Learned
- Searching Software
  - Keyword based
  - Structure Based
  - Hybrid
  - Ranking
- Future Directions
  - Topic-based search
- Conclusions

Acknowledgements

- Baldi Group - Institute for Genomics and Bioinformatics
  - Professor Pierre Baldi
  - Paul Rigor
- Mondego Group – Institute for Software Research
  - Professor Crista Lopes
  - Sushil Bajracharya

Introduction

- Millions of lines of source code available
  - Don’t always know how good it is
  - Don’t always know what it does
  - Even if it’s in-house!
- Open source projects provide repositories of commonly used code
  - Increasing popularity in industry
- Projects are interdependent
- Desire to maximize code reuse
  - Time is money
- Explore code entity relationships
- Understand shape and function of software
  - Large scale
  - “in the wild”

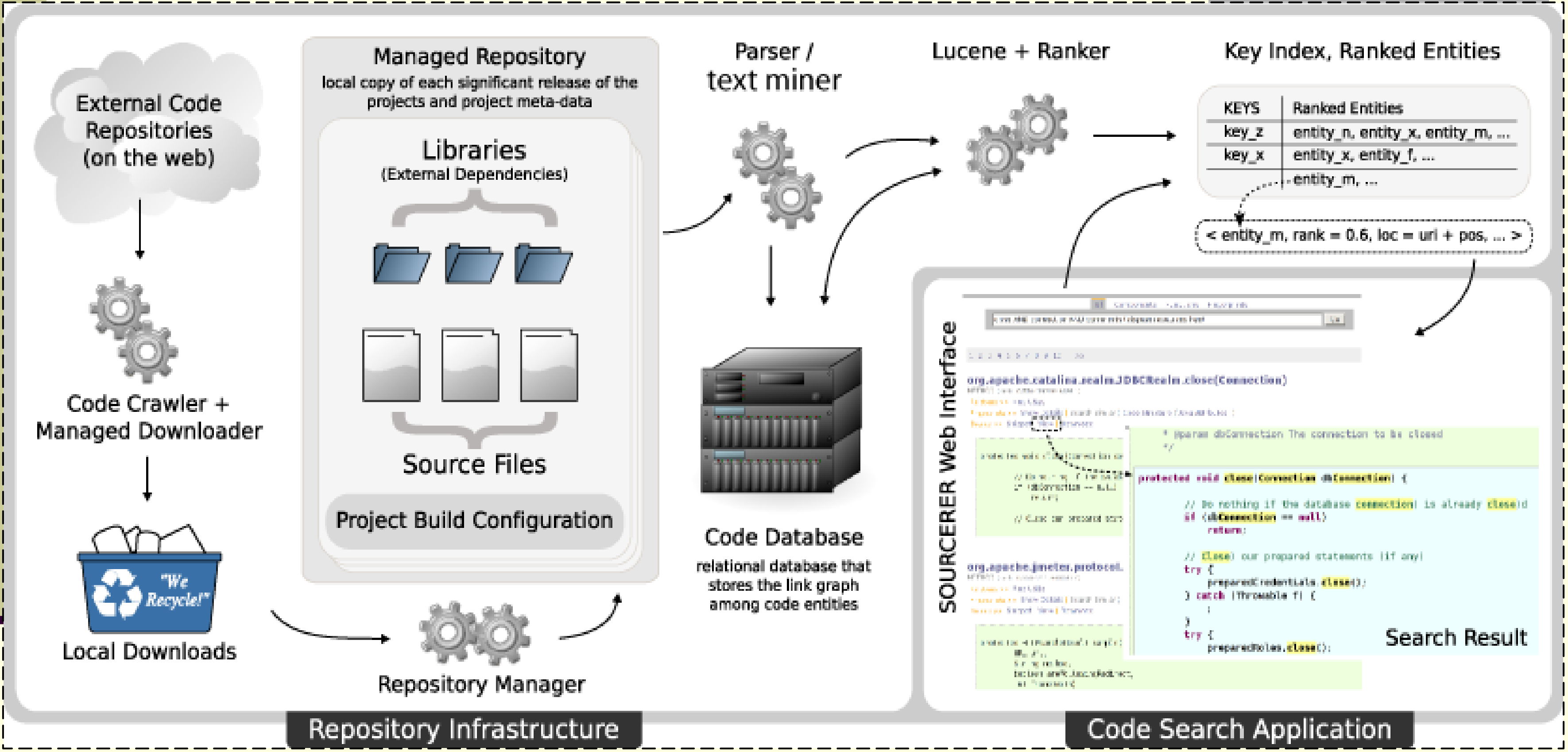
Introduction

- Code reuse has evolved
  - Reuse an entire system
  - Reuse a portion of a system
  - Reuse a class/interface
  - Reuse a function/algorithm
  - Reuse fields (eg. static definitions)
- Any system that facilitates reuse must deal with code at multiple granularities
- Any system that facilitates program understanding should be unsupervised and intuitive
- Searching/Mining for practical application
  - Functional analysis
  - Staffing – assignment, skill assessment
  - Refactoring
  - Reuse

Sourcerer

- UCI ICS project designed to:
    - Index publicly available source and provide fast search and mining
        - Emphasis on ranking, relevancy, functional analysis
        - Structure-based search, software shape
    - Statically analyze relationships among code entities
    - Leverage data to better understand code, facilitate reuse, provide tools for real-world software development
    - Do this all on an Internet scale across multiple programming languages
    - Explore new avenues for mining software
- Current Version
    - ~12k open source projects (4,600 with 38 million SLOC)
    - 9,250 contributors, 48k packages, 560k classes, 3.2M methods
    - Focused on java language as proof of concept
- Publicly Available
    - http://sourcerer.ics.uci.edu

Sourcerer Architecture


External Code Repositories (on the web)
Code Crawler + Managed Downloader
"We Recycle!"
Local Downloads
Managed Repository
local copy of each significant release of the projects and project meta-data
Libraries
(External Dependencies)
Source Files
Project Build Configuration
Repository Manager
Repository Infrastructure
Parser / text miner
Code Database
relational database that stores the link graph among code entities
Lucene + Ranker
Key Index, Ranked Entities
< entity_m, rank = 0.6, loc = url + pos, ... >
SOURCERER Web Interface
org.apache.catalina.realm.JDBCRealm.close(Connection)
org.apache.jmeter.protocol.
Search Result
Code Search Application

KEYS	Ranked Entities
key_z	entity_n, entity_x, entity_m, ...
key_x	entity_x, entity_f, ...
	entity_m, ...

```
     * @param dbConnection The connection to be closed
     */

protected void close(Connection dbConnection) {

        // Do nothing if the database connection is already closed
        if (dbConnection == null)
            return;

        // Close our prepared statements (if any)
        try {
            preparedCredentials.close();
        } catch (Throwable f) {
            ;
        }
        try {
            preparedRoles.close();
```

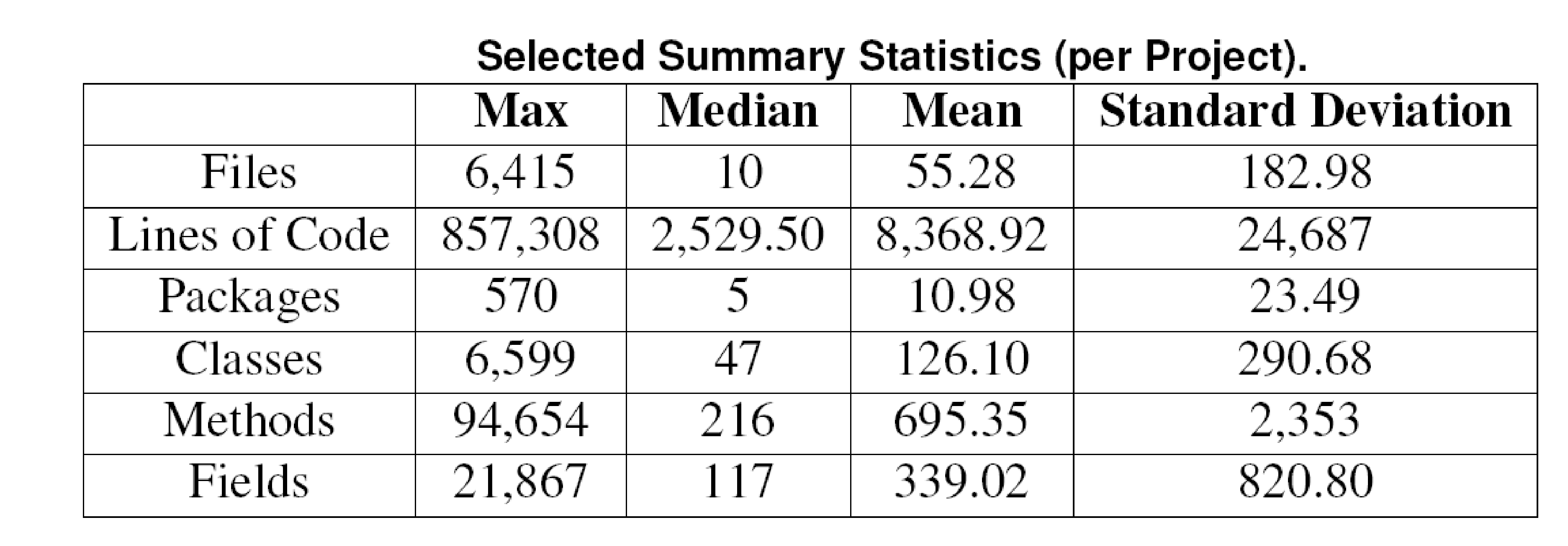
Database

- Parse Code and Store:
  - Entities (Classes, Methods, etc)
  - Relations
  - Documents (source files)
  - Repository Info (CVS, SVN, etc)
- Combination of
  - Relational Database (postgresql)
  - Text Indexing (Lucene)
  - Text Mining (source-tuned topic models)
  - Middleware for distributed parsing (our own)

Static Analysis of Code Relations

- Uses
  - generic
- Inside
  - Lexical containment
- Calls
- Throws
- Returns
- Overrides
- Overloads
- Instantiates
- Reads
- Writes

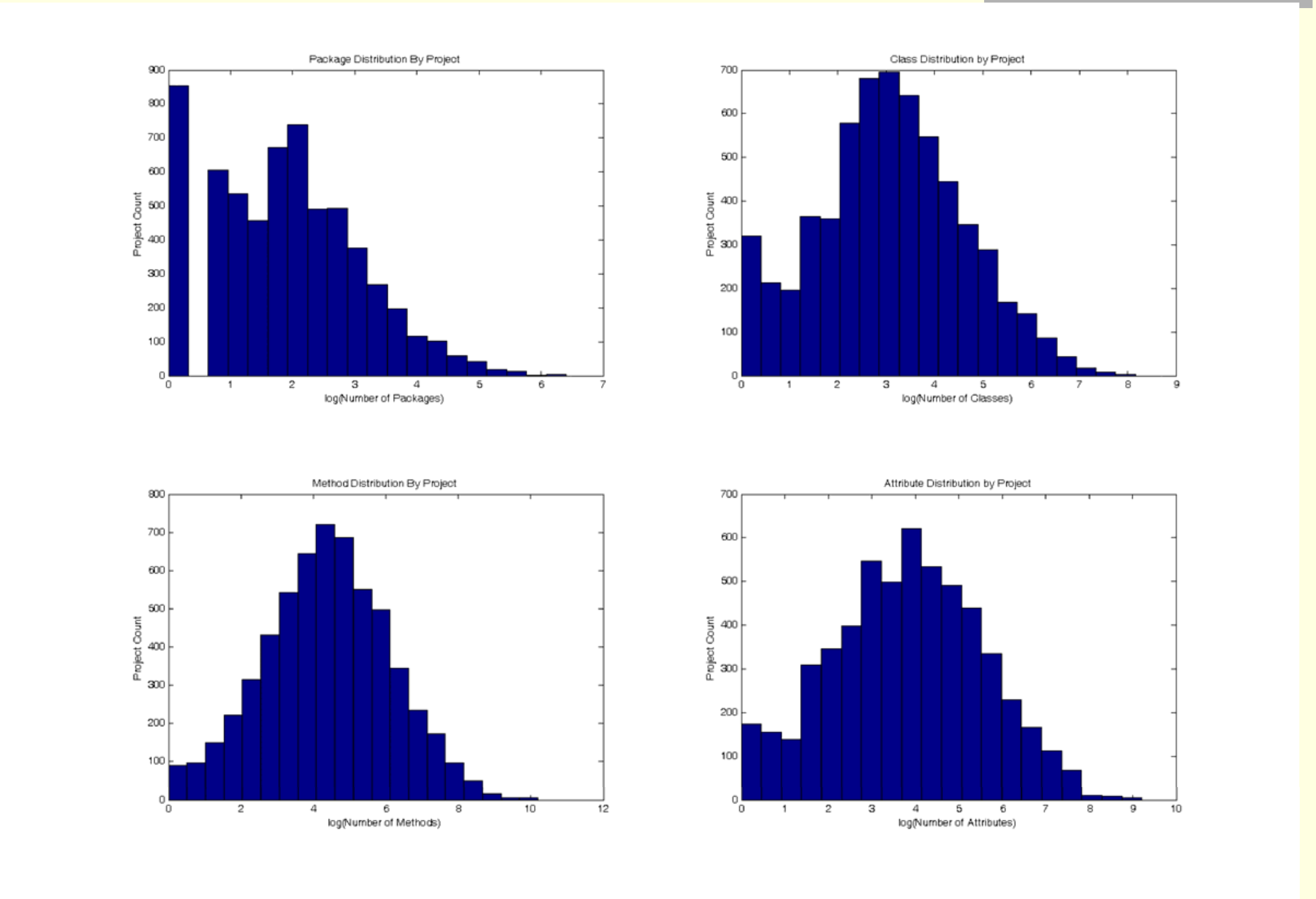
Simple Statistics



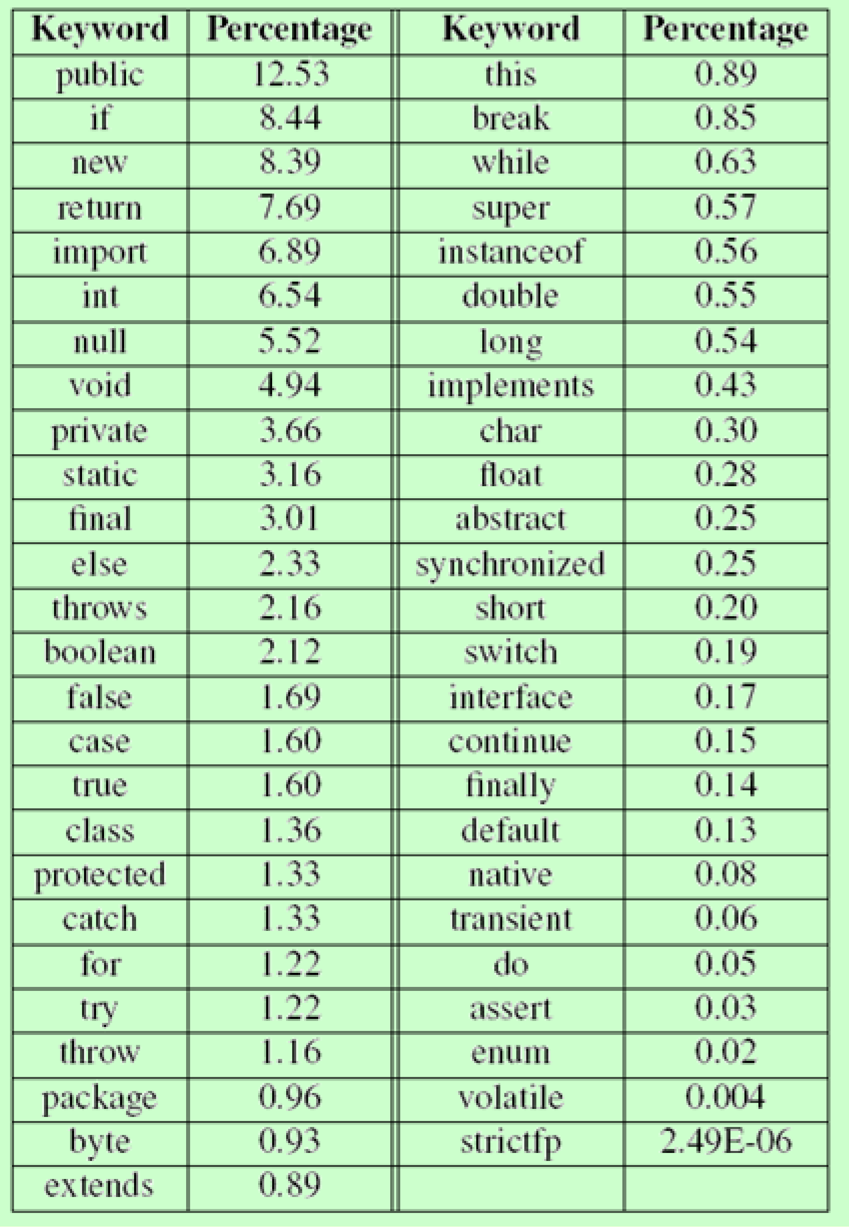
Selected Summary Statistics (per Project).

	Max	Median	Mean	Standard Deviation
Files	6,415	10	55.28	182.98
Lines of Code	857,308	2,529.50	8,368.92	24,687
Packages	570	5	10.98	23.49
Classes	6,599	47	126.10	290.68
Methods	94,654	216	695.35	2,353
Fields	21,867	117	339.02	820.80

Simple Statistics

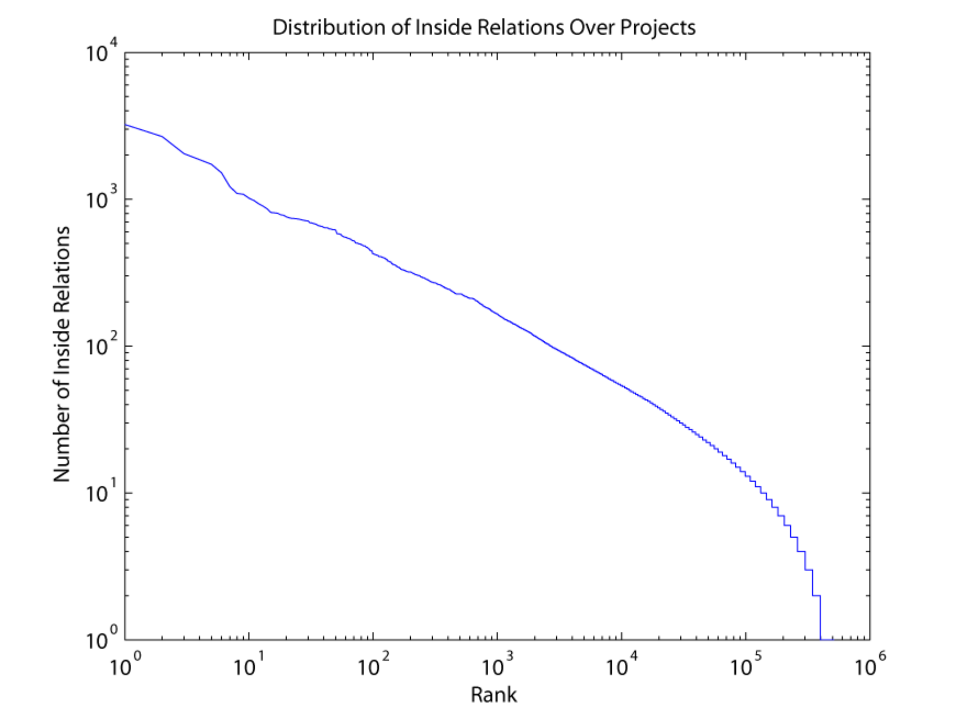


Keyword Occurrence

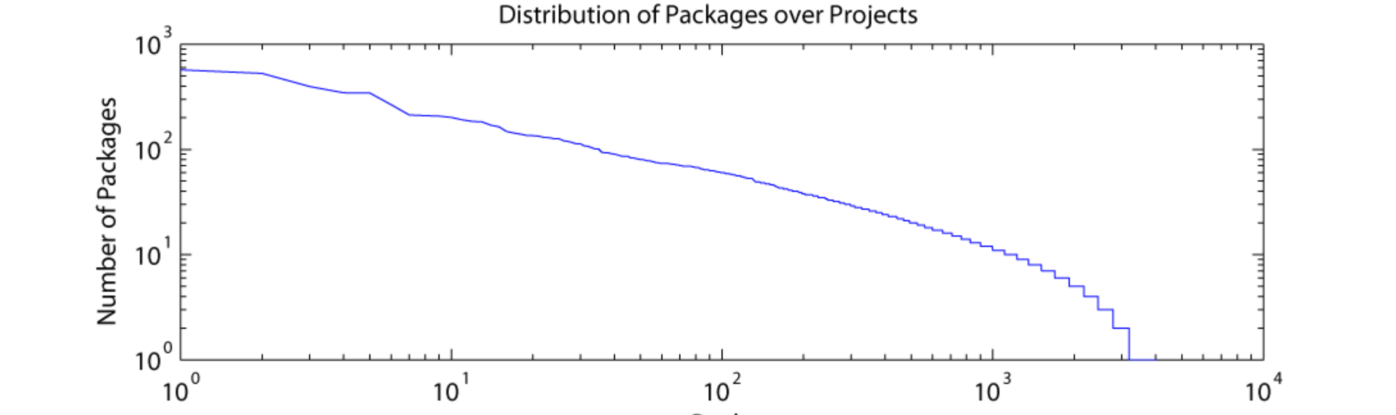


Keyword	Percentage	Keyword	Percentage
public	12.53	this	0.89
if	8.44	break	0.85
new	8.39	while	0.63
return	7.69	super	0.57
import	6.89	instanceof	0.56
int	6.54	double	0.55
null	5.52	long	0.54
void	4.94	implements	0.43
private	3.66	char	0.30
static	3.16	float	0.28
final	3.01	abstract	0.25
else	2.33	synchronized	0.25
throws	2.16	short	0.20
boolean	2.12	switch	0.19
false	1.69	interface	0.17
case	1.60	continue	0.15
true	1.60	finally	0.14
class	1.36	default	0.13
protected	1.33	native	0.08
catch	1.33	transient	0.06
for	1.22	do	0.05
try	1.22	assert	0.03
throw	1.16	enum	0.02
package	0.96	volatile	0.004
byte	0.93	strictfp	2.49E-06
extends	0.89		

Power Law Distributions



Distribution of Inside Relations Over Projects



Distribution of Packages over Projects

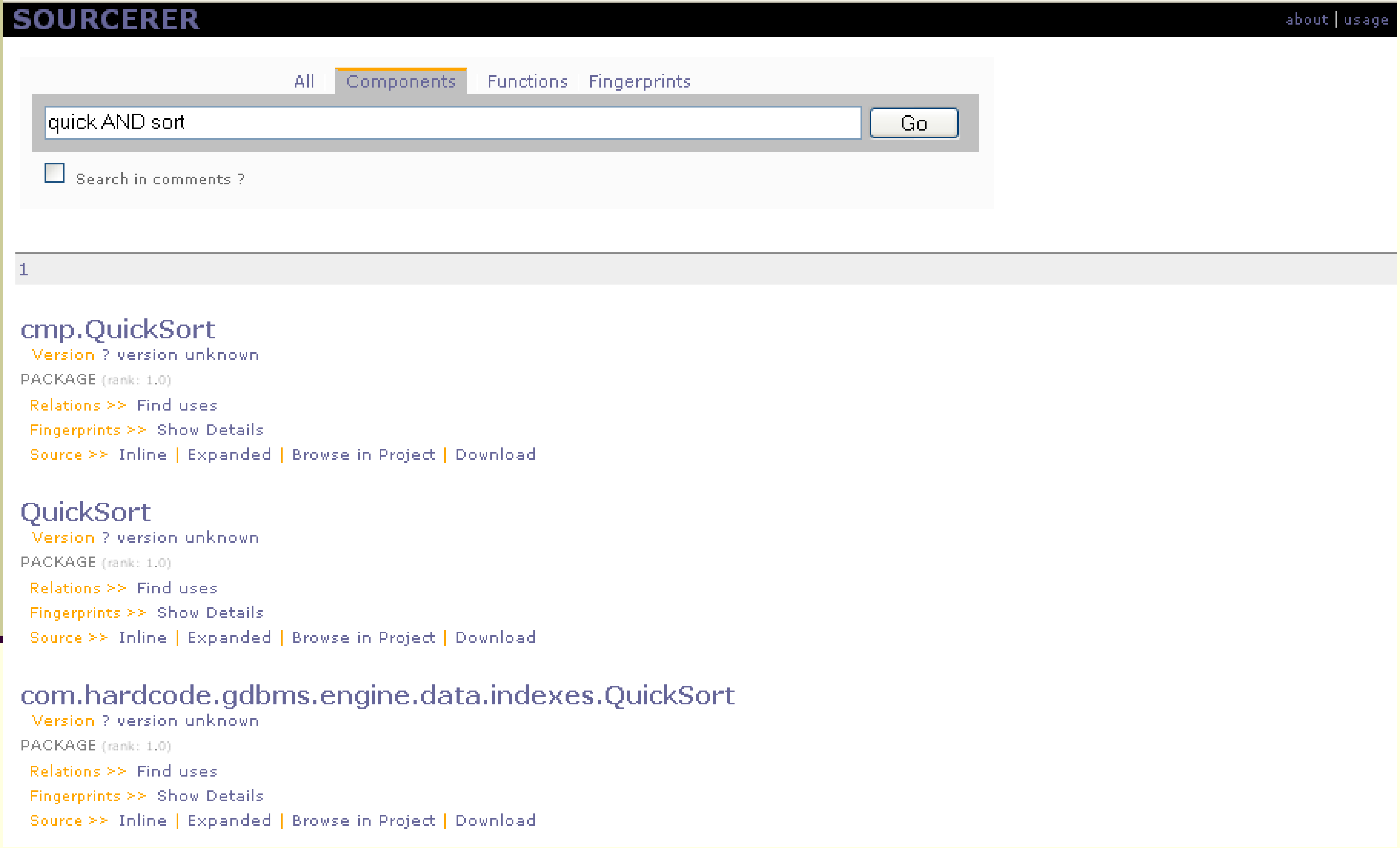
Searching Code

- Once you know what code does, how can you *find* code for a given programming task?
- Specialized keyword extraction
  - Comments
  - Code
    - calculateFastFourierTransform(Signal s)
    - calculate_Fast_Fourier_Transform(Signal s)
- What if you want to search based on structure
  - “Find thread-safe methods with 2 conditional statements”
- Commercial search engines gaining popularity
  - Google CodeSearch, Koders, Krugle
  - “glorified grep” over code
  - Lightweight analysis

Keyword-Based Search

- Known techniques for doing this well
  - Abundant IR literature
  - Open Source Search Engines
    - Lucene – incredibly fast, light-weight, and FREE!
- Tricky parts:
  - Associating comments with correct entity
  - Accounting for all naming conventions that are likely to appear in code.
    - Good luck!

Example – Quick Sort (classes)



quick AND sort

□ Search in comments ?

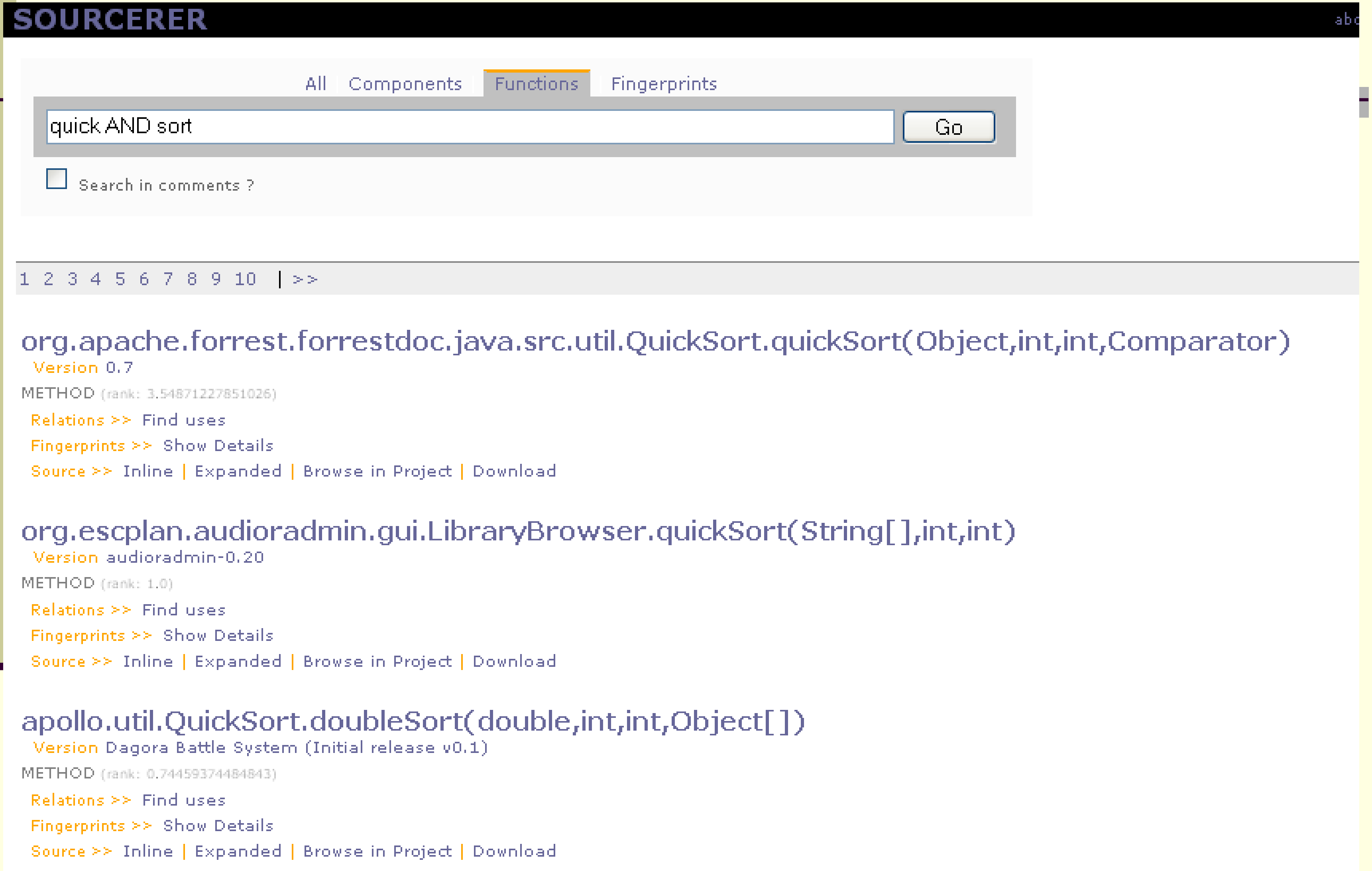
1

cmp.QuickSort
Version ? version unknown
PACKAGE (rank: 1.0)
Relations >> Find uses
Fingerprints >> Show Details
Source >> Inline | Expanded | Browse in Project | Download

QuickSort
Version ? version unknown
PACKAGE (rank: 1.0)
Relations >> Find uses
Fingerprints >> Show Details
Source >> Inline | Expanded | Browse in Project | Download

com.hardcode.gdbms.engine.data.indexes.QuickSort
Version ? version unknown
PACKAGE (rank: 1.0)
Relations >> Find uses
Fingerprints >> Show Details
Source >> Inline | Expanded | Browse in Project | Download

Example – Quick Sort (functions)



SOURCERER

All | Components | Functions | Fingerprints

quick AND sort

Go

□ Search in comments ?

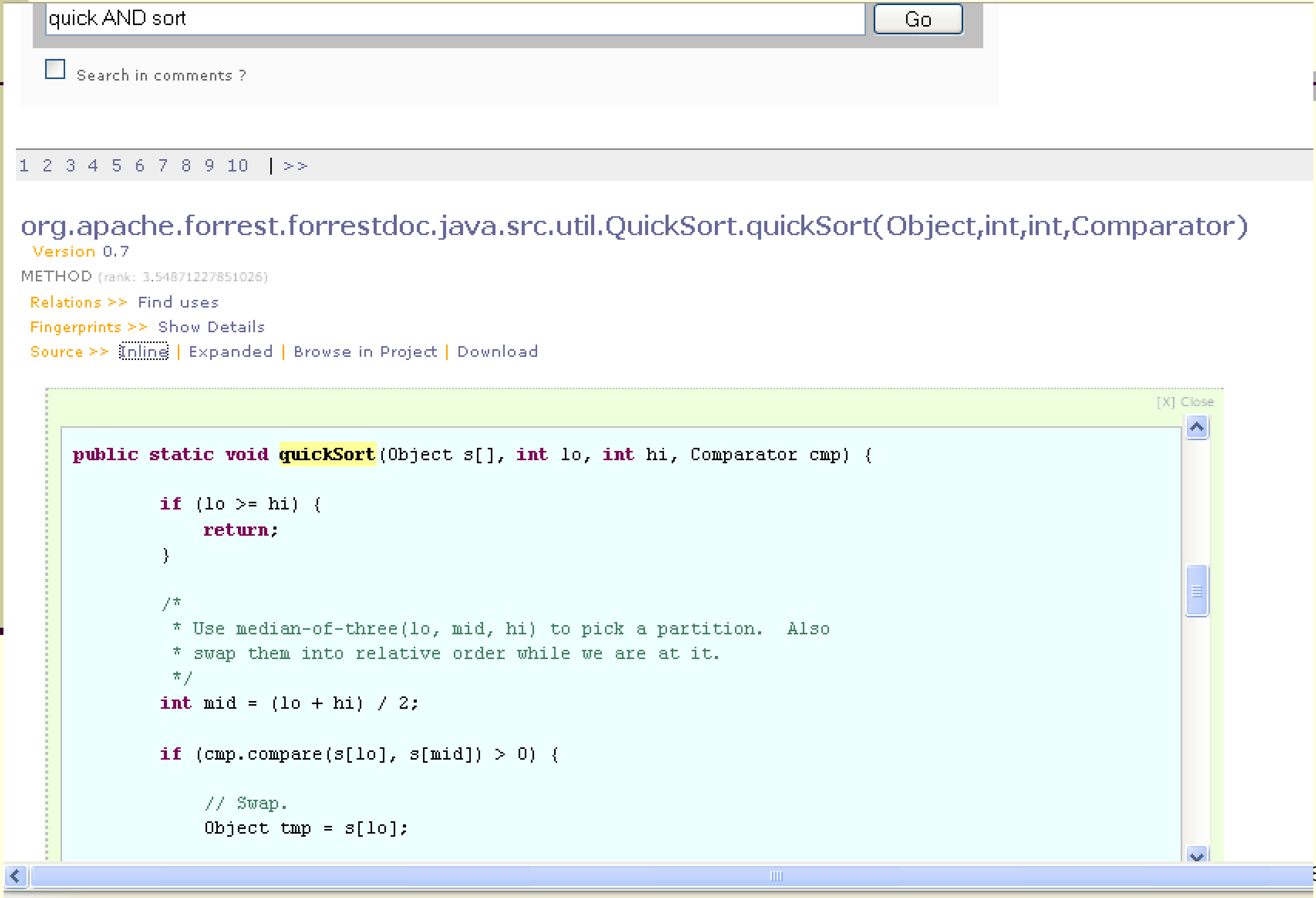
1 2 3 4 5 6 7 8 9 10 | >>

org.apache.forrest.forrestdoc.java.src.util.QuickSort.quickSort(Object,int,int,Comparator)
Version 0.7
METHOD (rank: 3.548712227851026)
Relations >> Find uses
Fingerprints >> Show Details
Source >> Inline | Expanded | Browse in Project | Download

org.escplan.audioradmin.gui.LibraryBrowser.quickSort(String[],int,int)
Version audioradmin-0.20
METHOD (rank: 1.0)
Relations >> Find uses
Fingerprints >> Show Details
Source >> Inline | Expanded | Browse in Project | Download

apollo.util.QuickSort.doubleSort(double,int,int,Object[])
Version Dagora Battle System (Initial release v0.1)
METHOD (rank: 0.744593744484843)
Relations >> Find uses
Fingerprints >> Show Details
Source >> Inline | Expanded | Browse in Project | Download

Example – Browse Code



quick AND sort

□ Search in comments ?

1 2 3 4 5 6 7 8 9 10 | >>

org.apache.forrest.forrestdoc.java.src.util.QuickSort.quickSort(Object,int,int,Comparator)

Version 0.7

METHOD (rank: 3.548712227851026)

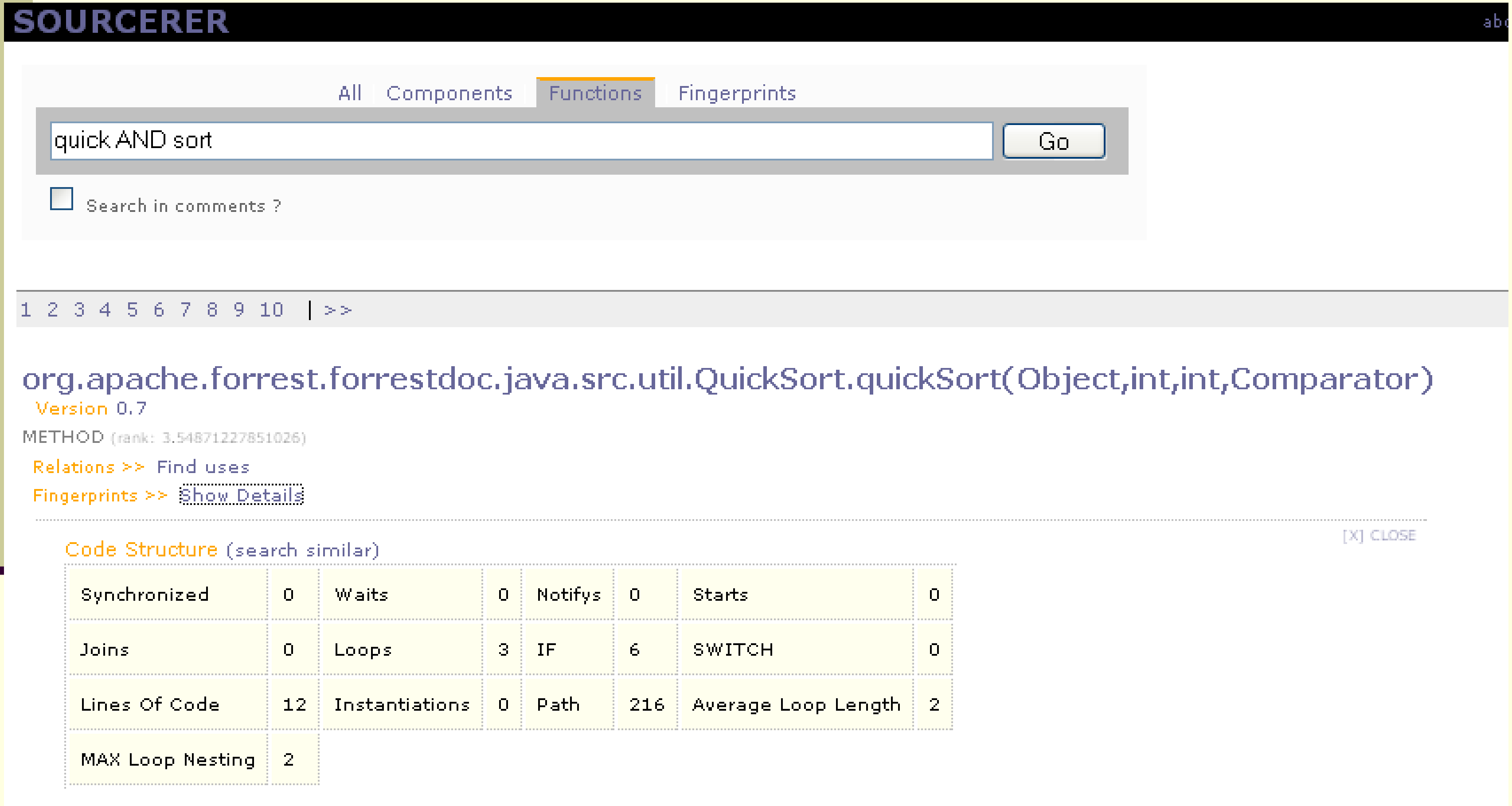
Relations >> Find uses

Fingerprints >> Show Details

Source >> Inline | Expanded | Browse in Project | Download

```
public static void quickSort(Object s[], int lo, int hi, Comparator cmp) {

        if (lo >= hi) {
            return;
        }

        /*
         * Use median-of-three(lo, mid, hi) to pick a partition.  Also
         * swap them into relative order while we are it.
         */
        int mid = (lo + hi) / 2;

        if (cmp.compare(s[lo], s[mid]) > 0) {

            // Swap.
            Object tmp = s[lo];
```

Example – Code Structure Fingerprint



SOURCERER

abo

All | Components | Functions | Fingerprints

quick AND sort

Go

□ Search in comments ?

1 2 3 4 5 6 7 8 9 10 | >>

org.apache.forrest.forrestdoc.java.src.util.QuickSort.quickSort(Object,int,int,Comparator)

Version 0.7

METHOD (rank: 3.54871227851026)

Relations >> Find uses

Fingerprints >> Show Details

[X] CLOSE

Code Structure (search similar)

Synchronized	0	Waits	0	Notifys	0	Starts	0
Joins	0	Loops	3	IF	6	SWITCH	0
Lines Of Code	12	Instantiations	0	Path	216	Average Loop Length	2
MAX Loop Nesting	2						

Fingerprint Types

- Structural Fingerprints
  - Control
  - Iteration
  - Synchronization
- Java Type Fingerprints
  - Fields
  - Methods
  - Constructors
  - etc
- MicroPattern Fingerprints
  - Occurrence of common design patterns
  - Gil & Maman, OOPSLA ‘05

Structural Fingerprints

- Types and counts of concurrency constructs (sync,wait,notify)
- Types and counts of branching constructs (if,switch)
- Number of loop structures
- Number of paths through code
- Number of dynamic memory allocations
- Average loop length
- Maximum nesting of loops

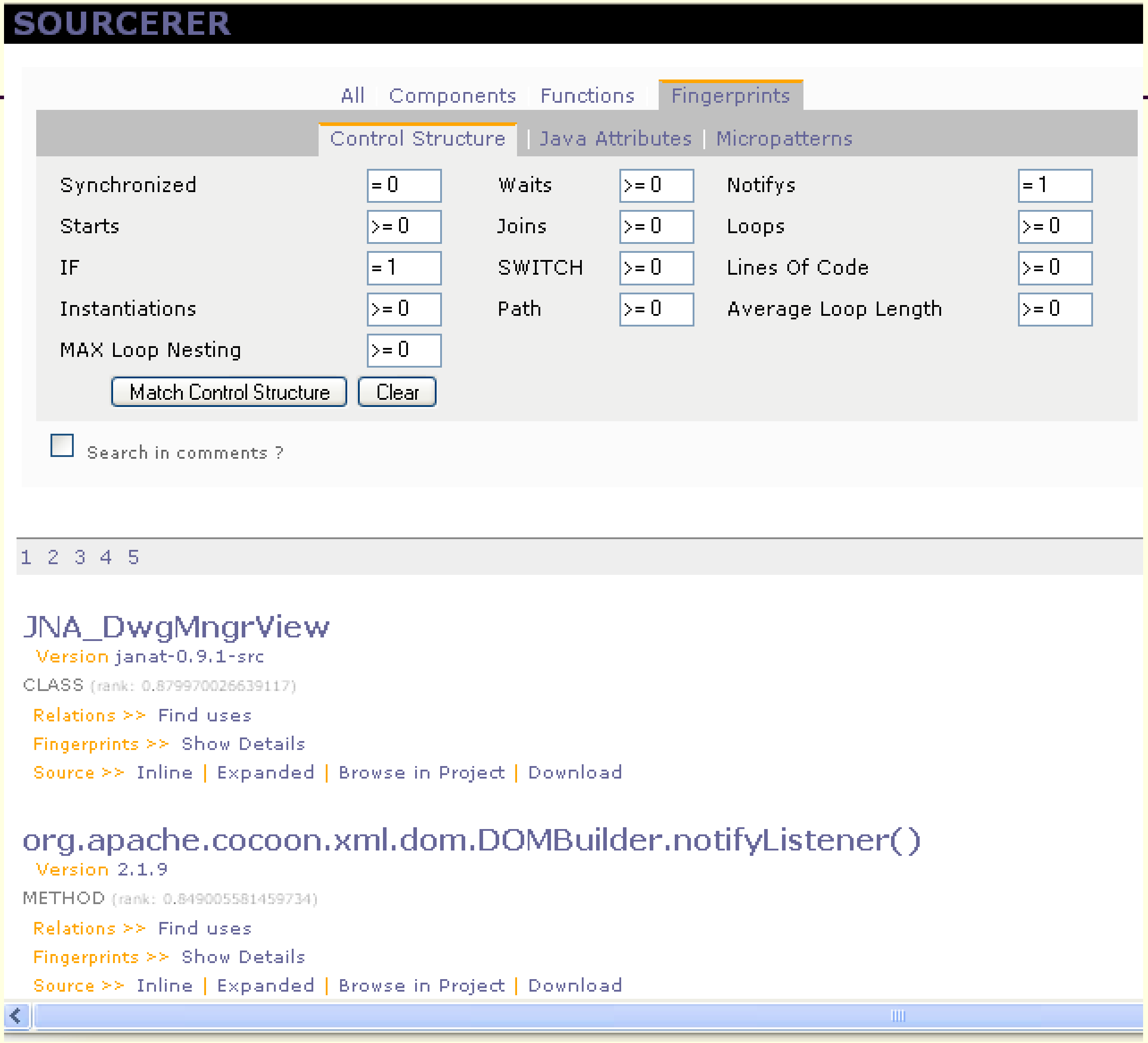
<sync,wait,notify,if,switch,loops,paths,allocs,avg_loop,max_nesting>

Similarity measure

- Many methods exist
- Popular IR technique is cosine distance
  - Calculate angle between query fingerprint, **q**, and code fingerprint, **f**

$$\mathrm{sim}(q,f) = (\mathbf{q} \blacklozenge \mathbf{f}) / (|\mathbf{q}| \mathrm{~x~} |\mathbf{f}|)$$

Example



Synchronized	= 0	Waits	>= 0	Notifys	= 1
Starts	>= 0	Joins	>= 0	Loops	>= 0
IF	= 1	SWITCH	>= 0	Lines Of Code	>= 0
Instantiations	>= 0	Path	>= 0	Average Loop Length	>= 0
MAX Loop Nesting	>= 0				

□ Search in comments ?

1 2 3 4 5

JNA_DwgMngrView
Version janat-0.9.1-src
CLASS (rank: 0.879970026639117)
Relations >> Find uses
Fingerprints >> Show Details
Source >> Inline | Expanded | Browse in Project | Download

org.apache.cocoon.xml.dom.DOMBuilder.notifyListener()
Version 2.1.9
METHOD (rank: 0.849005581459734)
Relations >> Find uses
Fingerprints >> Show Details
Source >> Inline | Expanded | Browse in Project | Download

Hybrid Search

- The best of both worlds
- Combine keywords with fingerprints
- “I think it should look something like…”
- Query is a code snippet
  - Parse for keywords and structure

Industry Applications

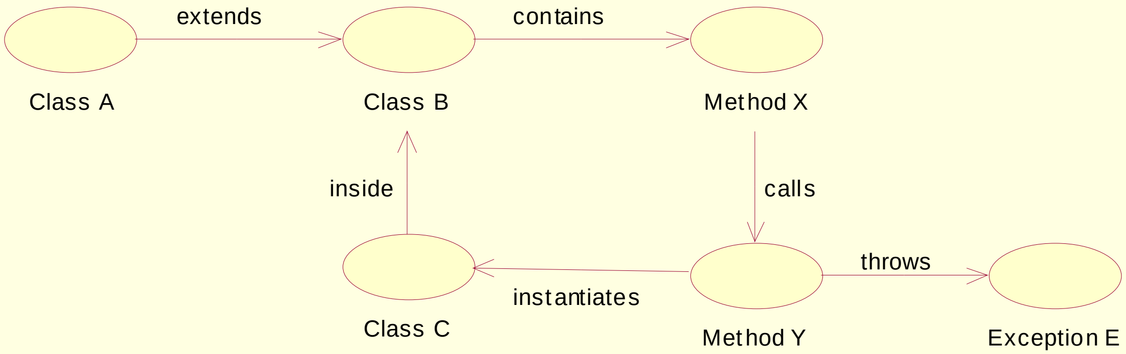
- General code reuse
  - Or at least a convenient way to find references
- Licensing issues
  - Worried that open source may have slipped in?
  - Find it with structure-based search
- Code analysis
  - Leverage relational databases
  - Examine statistics for best practices, etc.

Improving Results - Ranking

- Java-based heuristics
  - Boost hits to right of fully-qualified name (net.linstead.neuralNet)
  - Discount comment hits
  - Discount test code
  - Discount trivial implementations
- In addition, would like to give preference to code that is:
  - Heavily referenced
  - “popular”
  - Likely to be robust
- Need a systematic way to access code for these properties
- This is Google’s bread and butter

Link Analysis

- Start by building a directed graph of entities and relations
  - We have these in the database at parse time



PageRank (CodeRank)

$$PR(A) = (1-d) + d\,(PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn))$$

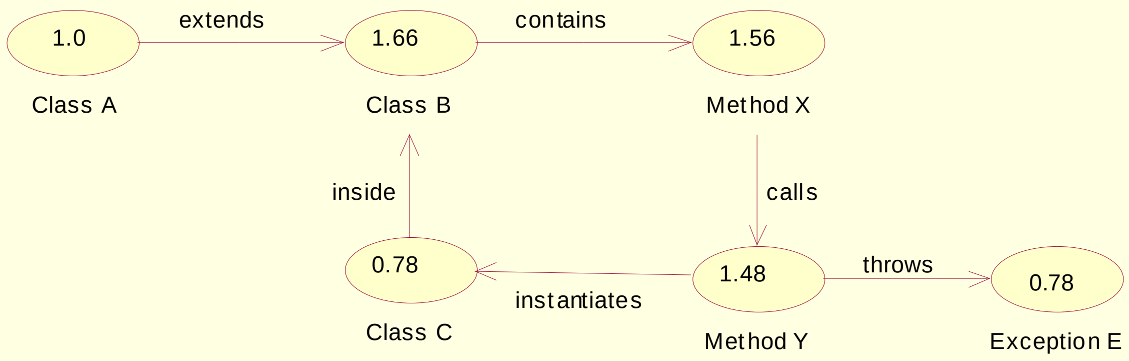
PR(X) – PageRank of Node X

T1…TN – Nodes pointing to X

C(Y) – number of outgoing edges of node Y

d – a damping factor

Example


1.0
extends
1.66
contains
1.56
Class A
Class B
Method X
inside
calls
0.78
1.48
throws
0.78
instantiates
Class C
Method Y
Exception E

Leveraging PR

- Used in combination with term frequency, similarity
- Currently Sourcerer uses PageRank almost verbatim
  - Local vs. Global rank
- Can modify algorithm to give different weight to different relationships
  - Is “calls” more or less important than “overloads”?
- Ultimately user will have to specify as a search parameter

Ranking Results

- 30% AUC (area under curve) improvement over competing engines
    - Google - 33% AUC
    - Google CodeSearch – 66% AUC
    - Sourcerer (code keywords only) – 73.6% AUC
    - Sourcerer (comment keywords only) – 44.7% AUC
        - 48.5% recall
    - Sourcerer (code + comments + heuristics + CodeRank) – 84% AUC
        - 100% recall
    - Sourcerer (code + heuristics) – 90.9% AUC
        - 74% recall
    - Sourcerer (code + heuristics + CodeRank) – 92% AUC
        - 74% recall
- Lessons Learned
    - Combination of keyword and structure-based heuristics yields greatest improvement
    - From a community standpoint need standardized benchmark

Conclusion

- Software repositories contain a wealth of information
  - Code metadata and implementations
  - Statistics for shape and function
- Recent statistical machine learning techniques make intuitive, scalable program understanding more feasible
- Effective IR techniques promise to aide in code reuse assuming
  - Give the searcher relevant results
  - Make it fast and easy
- Fingerprinting code provides an intuitive means for structure-based search
- New ranking techniques means the user finds what they want without wading through useless code
  - Leverage keywords and structure
  - Go beyond "glorified grep"