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 (e.g., 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
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

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Median</th>
<th>Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>6,415</td>
<td>10</td>
<td>55.28</td>
<td>182.98</td>
</tr>
<tr>
<td>Lines of Code</td>
<td>857,308</td>
<td>2,529.50</td>
<td>8,368.92</td>
<td>24,687</td>
</tr>
<tr>
<td>Packages</td>
<td>570</td>
<td>5</td>
<td>10.98</td>
<td>23.49</td>
</tr>
<tr>
<td>Classes</td>
<td>6,599</td>
<td>47</td>
<td>126.10</td>
<td>290.68</td>
</tr>
<tr>
<td>Methods</td>
<td>94,654</td>
<td>216</td>
<td>695.35</td>
<td>2,353</td>
</tr>
<tr>
<td>Fields</td>
<td>21,867</td>
<td>117</td>
<td>339.02</td>
<td>820.80</td>
</tr>
</tbody>
</table>
Simple Statistics
### Keyword Occurrence

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Percentage</th>
<th>Keyword</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>12.53</td>
<td>this</td>
<td>0.89</td>
</tr>
<tr>
<td>if</td>
<td>8.44</td>
<td>break</td>
<td>0.85</td>
</tr>
<tr>
<td>new</td>
<td>8.39</td>
<td>while</td>
<td>0.63</td>
</tr>
<tr>
<td>return</td>
<td>7.69</td>
<td>super</td>
<td>0.57</td>
</tr>
<tr>
<td>import</td>
<td>6.89</td>
<td>instanceof</td>
<td>0.56</td>
</tr>
<tr>
<td>int</td>
<td>6.54</td>
<td>double</td>
<td>0.55</td>
</tr>
<tr>
<td>null</td>
<td>5.52</td>
<td>long</td>
<td>0.54</td>
</tr>
<tr>
<td>void</td>
<td>4.94</td>
<td>implements</td>
<td>0.43</td>
</tr>
<tr>
<td>private</td>
<td>3.66</td>
<td>char</td>
<td>0.30</td>
</tr>
<tr>
<td>static</td>
<td>3.16</td>
<td>float</td>
<td>0.28</td>
</tr>
<tr>
<td>final</td>
<td>3.01</td>
<td>abstract</td>
<td>0.25</td>
</tr>
<tr>
<td>else</td>
<td>2.33</td>
<td>synchronized</td>
<td>0.25</td>
</tr>
<tr>
<td>throws</td>
<td>2.16</td>
<td>short</td>
<td>0.20</td>
</tr>
<tr>
<td>boolean</td>
<td>2.12</td>
<td>switch</td>
<td>0.19</td>
</tr>
<tr>
<td>false</td>
<td>1.69</td>
<td>interface</td>
<td>0.17</td>
</tr>
<tr>
<td>case</td>
<td>1.60</td>
<td>continue</td>
<td>0.15</td>
</tr>
<tr>
<td>true</td>
<td>1.60</td>
<td>finally</td>
<td>0.14</td>
</tr>
<tr>
<td>class</td>
<td>1.36</td>
<td>default</td>
<td>0.13</td>
</tr>
<tr>
<td>protected</td>
<td>1.33</td>
<td>native</td>
<td>0.08</td>
</tr>
<tr>
<td>catch</td>
<td>1.33</td>
<td>transient</td>
<td>0.06</td>
</tr>
<tr>
<td>for</td>
<td>1.22</td>
<td>do</td>
<td>0.05</td>
</tr>
<tr>
<td>try</td>
<td>1.22</td>
<td>assert</td>
<td>0.03</td>
</tr>
<tr>
<td>throw</td>
<td>1.16</td>
<td>enum</td>
<td>0.02</td>
</tr>
<tr>
<td>package</td>
<td>0.96</td>
<td>volatile</td>
<td>0.004</td>
</tr>
<tr>
<td>byte</td>
<td>0.93</td>
<td>strictfp</td>
<td>2.49E-6</td>
</tr>
<tr>
<td>extends</td>
<td>0.89</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)

**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)

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

org.escplan.audiорadmin.gui.LibraryBrowser.quickSort(String[],int,int)

apollo.util.QuickSort.doubleSort(double,int,int,Object[])
Example – Browse Code

```java
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 at it.
     */
    int mid = (lo + hi) / 2;

    if (cmp.compare(s[lo], s[mid]) > 0) {
        // Swap.
        Object tmp = s[lo];
    }
}
```
Example – Code Structure

Fingerprint

SOURCERER

All Components Functions Fingerprints

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

Methods (rank: 3.388/722/851025):

Relations >> Find uses

Fingerprints >> Show Details

Code Structure (search similar)

<table>
<thead>
<tr>
<th>Synchronized</th>
<th>Waits</th>
<th>Notify</th>
<th>Starts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Joins</th>
<th>Loops</th>
<th>IF</th>
<th>SWITCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lines Of Code</th>
<th>Instantiations</th>
<th>Fath</th>
<th>Average Loop Length</th>
<th>MAX Loop Nesting</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0</td>
<td>216</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[X] CLOSE
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, \( f \)

\[
sim(q,f) = \frac{(q \cdot f)}{(|q| \times |f|)}
\]
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

```
Class A extends Class B contains Method X
inside Method X calls instantiates

Class C instantiates Method Y throws Exception E
```
PageRank (CodeRank)

\[ PR(A) = (1-d) + d \left( \frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)} \right) \]

- \( PR(X) \) – PageRank of Node X
- \( T_1 \ldots T_N \) – Nodes pointing to X
- \( C(Y) \) – number of outgoing edges of node Y
- \( d \) – a damping factor
Example

- Class A extends 1.0
- Class B contains 1.66
- Method X calls 1.48
- Class C instantiates 0.78
- Method Y throws 0.78
- 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”