Code Search
and
Empirical Software Engineering
Kathryn Stolee -- Harpole-Pentair Assistant Professor

July 10, 2014
My Research Interests

- Code search and code reuse
- Empirical studies
- End-user programming
- Testing
My Research Interests

- Code search and code reuse
- Empirical studies
- End-user programming - spreadsheets, mashups, macros
- Testing
My Research Interests

- Code search and code reuse
- Empirical studies
- End-user programming - spreadsheets, mashups, macros
- Testing - finding effective tools, training students
My Research Interests

- Code search and code reuse
- Empirical studies
- End-user programming - spreadsheets, mashups, macros
- Testing - finding effective tools, training students
Programmers are Resourceful
Code Search is Common

- 89% of programmers search for code frequently or occasionally [1]
- 85% of programmers search for code at least weekly [2]

Code Search is Common

- 89% of programmers search for code frequently or occasionally [1]
- 85% of programmers search for code at least weekly [2]
- 48% of programmers copy/paste and modify found code [2]
- 67% of programmers use found code as a reference example [2]

Code Search, Typical
Code Search, Typical

How do I do X?

How have others done X?
Code Search, Typical

How do I do X?  How have others done X?

Query

“compute x^n”
Code Search, Typical

How do I do X?

How have others done X?

Query

“compute x^n”
Code Search, Typical

How do I do X?

How have others done X?

Query

“compute x^n”

Results
Code Search, Typical

How do I do X?
How have others done X?

Query

Search

"compute x^n"

Results

There are too many results!
Which results are relevant???
Programmers are Resourceful
Programmers are Resourceful

... but their resources can be better!
Code Search, New
What code does X?
Code Search, New

What code does X?

Query
Input: 3, 2
Output: 9
Code Search, New

What code does X?

Query
Input:
Output:

NCode Search
Input 3, 2
Output 9

11
What code does X?

Query
Input:
Output:

Results
```
int addIntegers(int x, int y) {
    if (x == 0)
        return x + y;
    else
        return x + y;
}

int computeValue(int x, int n) {
    int result = x;
    for(int i = 1; i < n; ++i) {
        result = result * x;
    }
    return result;
}

double power(double x, double n) {
    double pow = Math.pow(x, n);
    return pow;
}
```

double power(double x, double n) {
    double pow = Math.pow(x, n);
    return pow;
}
What code does X?

Query
Input:
Output:

Wow! These results all behave as specified!
Sometimes there are too many results...
Sometimes there are no exact matches.

Results

```java
int add(int x, int y) {
    if (x == y) {
        return x * 2;
    } else {
        return x * 4;
    }
}

double power(double x, double n) {
    double pow = Math.pow(x,n);
    return pow;
}

double computeValue(int x, int n) {
    int result = x;
    for(int i = 1; i < n; i++) {
        result = result * x;
    }
    return result;
}

double powerOne(int x) {
    double result = Math.pow(x, 1.0);
    return result;
}

double powerTwo(int x) {
    double result = Math.pow(x, 2.0);
    return result;
}

double powerThree(int x) {
    double result = Math.pow(x, 3.0);
    return result;
}
```
Method to the Madness

Source Code Repository

Input: susie@mail.com
Output: susie

Solver

Rank Results
Method to the Madness

Search by Solving

Input: susie@mail.com
Output: susie

Source Code Repository

Transform

Solver

Rank Results
Method to the Madness

Search by Solving

Input: susie@mail.com
Output: susie

Source Code Repository

Transform

(assert (= input “susie@mail.com”))
(assert (= output “susie”))

Solver

Rank Results
Method to the Madness

```lisp
(assert (= input "susie@mail.com"))
(assert (= output "susie"))
```

```
output = input.substring(0, input.indexOf('@'));
```

```
(assert (≤ (length input) (length output)))
(assert (= 'ċ' (charAt input upper)))
(assert (forall ((i Int))
  (⇒ (and (< i upper) (≥ i 0)) (= (charAt input i) (charAt output i)))))
```
Targeted Domains and Applications
Targeted Domains and Applications

Past

Present/Future
Targeted Domains and Applications

Yahoo! Pipes

Past
Present/Future
Targeted Domains and Applications

Yahoo! Pipes

MySQL

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice Bower</td>
<td>2008-08-19</td>
<td><a href="mailto:abower@uml.edu">abower@uml.edu</a></td>
</tr>
<tr>
<td>Carl Dent</td>
<td>2008-10-10</td>
<td><a href="mailto:cdent@msd.org">cdent@msd.org</a></td>
</tr>
<tr>
<td>Eleanor Fae</td>
<td>2008-08-19</td>
<td><a href="mailto:efae@csse.uml.edu">efae@csse.uml.edu</a></td>
</tr>
<tr>
<td>Greg Hagg</td>
<td>2008-08-19</td>
<td><a href="mailto:ghagg@uml.edu">ghagg@uml.edu</a></td>
</tr>
</tbody>
</table>
Targeted Domains and Applications

Yahoo! Pipes

MySQL

Java Strings
Targeted Domains and Applications

Yahoo! Pipes

MySQL

Java Strings

Past

Present/Future

Regex

(a*bc)d+\1
Targeted Domains and Applications

Yahoo! Pipes

MySQL

Java Strings

Past

Present/Future

Regex

(a*bc)d+\1

Test-Driven Development
Targeted Domains and Applications

Yahoo! Pipes

MySQL

Past Present/Future

Java Strings

Automated Fault Fixing

Regex

(a*b*c)d+\1

Test-Driven Development
Findings
Findings

• Highlights:

• Approach particularly well-suited when the problem can be expressed as a *transformation*

• With a smart ranking algorithm, we can out-perform Google in terms of finding relevant results
Findings

• Highlights:

  • Approach particularly well-suited when the problem can be expressed as a *transformation*

  • With a smart ranking algorithm, we can out-perform Google in terms of finding relevant results

• Shortcomings:

  • Limited in expressiveness of code that can be found

  • Performance compared to execution is unknown
Satsy - a senior design project
My Research Interests

- Code search and code reuse
- Empirical studies
- End-user programming - spreadsheets, mashups, macros
- Testing - finding effective tools, training students
Code Search at Google

- Surveyed and logged search activity for 28 developers over 2 weeks
- Research questions:
  - Why do programmers search for code?
  - What are the characteristics of their searches?
### Why do Programmers Search?

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>22%</td>
<td>API consumer needs (how to call a function)</td>
</tr>
<tr>
<td>14%</td>
<td>Check implementation details</td>
</tr>
<tr>
<td>10%</td>
<td>Fault localization</td>
</tr>
<tr>
<td>...</td>
<td>browsing, how to do something, dependencies, ...</td>
</tr>
</tbody>
</table>
Search Session Characteristics

- Average 5-6 search sessions per day per user
- Average 16 searches per day per user
- Average keywords per query: 2.21
- Sessions with only one result clicked: 22%
Implications

- Programmers are searching more than ever
- Searches in programming context are often directed, not blind
- Search tools are more than code finders
Crowdsourcing Evaluations
Crowdsourcing Evaluations

* **Why:** access to a large, diverse group of developers
Crowdsourcing Evaluations

- **Why**: access to a large, diverse group of developers
- **How**: Amazon’s Mechanical Turk framework
Crowdsourcing Evaluations

Why: access to a large, diverse group of developers

How: Amazon’s Mechanical Turk framework

What:
- Evaluate relevance of search results
- Preference evaluation for refactored code
- Survey developer habits, opinions, preferences
Crowdsourcing Evaluations

- **Why**: access to a large, diverse group of developers

- **How**: Amazon’s Mechanical Turk framework

- **What**:
  - Evaluate relevance of search results
  - Preference evaluation for refactored code
  - Survey developer habits, opinions, preferences

- **Is it good?**
  - Results are not significantly different than when using student populations
Thanks!

* Collaborators:
  * Yuriy Brun -- University of Massachusetts-Amherst
  * Matthew B. Dwyer -- University of Nebraska-Lincoln
  * Sebastian Elbaum -- University of Nebraska-Lincoln
  * Claire Le Goues -- Carnegie Melon University
  * Caitlin Sadowski -- Google

* Students:
  * Carl Chapman (MS)
  * Yalin Ke (PhD)
  * James Saylor (ugrad)