A Journey through Searching Similar Code

Miryung Kim
Professor and Vice Chair of Graduate Studies at UCLA
Amazon Scholar, Amazon Web Services
Outline: A *Journey* through Searching Similar Code

What motivated us?
What were early attempts?
How serious is this problem?
How can we automate?
How can we examine variations at scale?
How to search with a human in the loop?
What ideas have motivated searching similar code?

• To understand programmers’ copy and paste coding behavior, we built an Eclipse plug-in that records edits and replays the captured edits at IBM

• Programmers often create and manage code clones with clear intent
An Empirical Study of Code Clone Genealogies [FSE 2015]

- We developed an approach that automatically reconstructs the history of code clones from a source code repository.
- We studied clone evolution in several Java open source projects.
Clone Genealogy

Clone genealogy is a representation that captures clone change patterns over a sequence of program versions.
Dagstuhl: Multiversion Program Analysis in 2005

Mining Software Repositories

Code Element

P  P'

Time
Systematic Changes
(similar updates to similar code)

Consistent updates to clones
Managing multiple products, forked projects and versions
API evolution and ripple effects on client applications
Refactoring
What were early attempts to abstract systematic changes?
Miryung’s PhD @ U of Washington
Automated Change Rule Inference

Analyses of Software Evolution
- Evolution of Code Clones

High-level changes are often systematic at a code level

Automatic Inference of High-Level Change Descriptions
- Rule-based Change Representations
- Rule Learning Algorithms

David Notkin
(1 Jan 1955 – 22 Apr 2013)
API Change Rule Inference [ICSE 2007]

FOR ALL x:method-header IN
Factory.create*Chart(*)
argAppend(x, [int])
except {Factory.createPieChart()}

P

Factory.createChart()
Factory.createBarChart()
...
Factory.createPieChart()
Factory.createLineChart()
LSDiff Rule Inference [ICSE 2009]

- “Replace all calls to \texttt{SQL.exec} with \texttt{SafeSQL.exec}”

\begin{verbatim}
deleted Calls(m, "SQL.exec") =>
added Calls(m, "SafeSQL.exec")
\end{verbatim}

- “All \texttt{setHost} methods in Service’s subclasses in the old version deleted calls to \texttt{SQL.exec} except the \texttt{setHost} method in the \texttt{NameSvc} class.

\begin{verbatim}
past_subtype("Service", t) \land past_method(m, "setHost", t) => deleted Calls(m, "SQL.exec")
except t="NameSvc"
\end{verbatim}
How serious is this problem of searching similar code?
Baishakhi’s PhD @ UT Austin
Cross-system co-evolution

Co-Evolution of Forked Projects

Software Eco-Systems

API Stability and Adoption in Android Ecosystem
Porting consists of a significant portion of the BSD family evolution and a significant portion of active committers port changes.
SPA: Detecting Semantic Inconsistencies in Ported Code in Linux [ASE 2013]

13% to 25% changes are reused in Linux and Microsoft projects
Android is evolving fast at a rate of 115 API updates per month.

- 28% of API references in client apps are outdated with a median lagging time of 16 months.
- API usage adaptation code is **defect prone** than other code.
Microsoft: Quantifying Benefits of Windows Re-architecting [FSE ‘12, TSE ‘14]

Refactoring churn is less defect-prone than regular churn.
How can we *automate* systematic changes?
Na Meng’s PhD @ UT Austin
Automating Systematic Changes

Systematic Editing:
Generating Program Transformations from an Example

Na Meng  Miryung Kim  Kathryn S. McKinley
The University of Texas at Austin
mengna152173@gmail.com, miryung@ece.utexas.edu, mckinley@cs.utexas.edu

LASE: Locating and Applying Systematic Edits by Learning from Examples

Na Meng*  Miryung Kim*  Kathryn S. McKinley*†
The University of Texas at Austin*  Microsoft Research†
mengna94@cs.utexas.edu, miryung@ece.utexas.edu, mckinley@microsoft.com

Program differencing  Context extraction  Identifier & edit position abstraction

DELETE: config = (ILaunchConfiguration) iter.next()

DELETE: v1 = (t1)v2.m1();
1. public ILaunchConfiguration[] getLaunchConfigurations(ILaunchConfigurationType type) throws CoreException {
2.  Iterator iter = getAllLaunchConfigurations().iterator();
3.  List configs = new ArrayList();
4.  + ILaunchConfiguration config = null;
5.  while (iter.hasNext()) {
6.    - ILaunchConfiguration config =
7.      (ILaunchConfiguration)iter.next();
8.    + config = (ILaunchConfiguration)iter.next();
9.    + if (!config.isValid()) {
10.       + config.reset();
11.       if (config.getType().equals(type)) {
12.          configs.add(config);
13.       }
14.   }
15.  return (ILaunchConfiguration[])configs.toArray
16.   (new ILaunchConfiguration[configs.size()]);
17.}

1. ... method_declarion(... ...){
2.  T1 v1 = ml().m2();
3.  ... ... 
4.  while(v1.m3()){
5.    UPDATE: T2 v2 = (T2)v1.m4();
6.    TO: T2 v2 = null;
7.    INSERT: v2 = (T2)v1.m4();
8.    INSERT: if(!v2.m5()){
9.      INSERT: v2.m6();
10.  }  
11. ... ... 
12.  }  
13. }  
14. }

Abstract edit script
Abstract Edit Script

Variable Map | Method Map | Type Map
---|---|---
v1 <- v1 | m1 <- m1 | T1 <- T1
v2 <- v3 | m2 <- m2 | T2 <- T4
m3 <- m3
m4 <- m4

SEARCH

TRANSFORM

```
T1 v1 = m1().m2()
while(v1.m3())
T2 v2 = (T2)v1.m4()

T1 v1 = m1().m2()
while(v1.m3())
T2 v2 = new T2()
while(v1.m3())
return v2

T4 v3 = (T4)v1.m4()
T5 v4 = v3.m5()
if(v4!=null && v4.m6().m7(v5))
    then
cfgs.mi
    v2.m8(v4)
```
Lase: Locating and Applying Systematic Edits [ICSE 2013]

User selects examples

\[ A_{old} \rightarrow A_{new} \]

\[ B_{old} \rightarrow B_{new} \]

LASE selects methods & suggests edits

\[ D_{old} \rightarrow D_{suggested} \]

\[ I_{old} \rightarrow I_{suggested} \]

\[ X_{old} \rightarrow X_{suggested} \]
A chicken and egg problem: Users needs to find patch examples generalize from.
A chicken and egg problem: Users need hints on what to generalize.
How can we mine and examine variations at scale?
Tianyi Zhang’s PhD @ UCLA: Leveraging Commonalities and Variations at Scale

- AST Traversal
- Program Slicing
- Call Sequence Extraction

380K Java Repositories on GitHub

Structured API call sequences

Frequent Sequence Mining

SMT-based Guard Condition Mining

API usage patterns

GitHub

Stack Overflow

312K Java or Android posts

408K methods

873K Java files (100K GitHub repos)

739K distinct Java files

4M methods

5M clone pairs (57K groups)

Clones

SCC Clone Detector
API Usage Mining from GitHub and API Misuse Detection in StackOverflow [ICSE ‘18]
Analyzing and Supporting Adaptation of Online Code Examples [ICSE’19]
Visualizing API Usage Examples at Scale [CHI’18]

Focal API

new FileInputStream()

Many code examples using this call

What exception(s) are thrown?

What other methods are called before and after?

Interactive visualization showing common usage and frequency

github stackoverflow

GitLab Bitbucket CODE PROJECT
How can we construct a search pattern with a human in the loop?
ALICE: Active Inductive Logic Programming for Code Search [ICSE 2019]

Input: More instance labels
ALICE: Keep refining the query
Output: A smaller set of method locations that match the new query
public void getUserName(String id) {
    try {
        ResultSet set = db.executeQuery("select name from users where id=" + id);
        while (set.next()) { ... }
    } catch (SQLException e) { ... }
}

public void queryDatabase() {
    try {
        ResultSet result = s.executeQuery("select * from customers");
        while (result.next()) { ... }
    } catch (SQLException e) { ... }
}

public List get() {
    ResultSet set = stmt.executeQuery("select * from t");
    List l = new List();
    while (set.next()) { ... }
    return l;
}
A chicken and egg problem: Users need hints on how to pick a discriminatory atom.
Developer Tools for Big Data Systems & Heterogeneous Hardware

...
“Should I include `updateAAD` or `Base64.decode`?”
SURF: Scaling Code Pattern Inference with Interactive What-If Analysis [ICSE 2024]

- Matching on Program Dependence Graphs
- Simultaneous overlay
- Global distribution
- Impact Analysis, and
- What-if Analysis

The inferred pattern matched against

Population of graphs

2. Summary of code population overlaid on a skeleton

3. Global distribution

Feature Choices

4. Impact Analysis

5. What-If Analysis
### Design 1: Hint on Global Distribution

<table>
<thead>
<tr>
<th>S</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

#### Support:
Number of times a statement appears, regardless of label

#### String.getBytes()

#### Cipher.init()

#### String.equals()

#### Information gain:
Improvement in entropy for separating positive from negative instances

\[
\text{Entropy}(G_P) - \left( \frac{|G_m|}{|G_P|} \times \text{Entropy}(G_m) + \frac{|G_e|}{|G_P|} \times \text{Entropy}(G_e) \right)
\]
Design 2: Impact Analysis

- Focuses attention to which instances will be included/excluded for each statement choice

“If I include this specific code line, how many will I match?”
Design 3: What-If Analysis

Explore trade-offs involved in selecting one statement over another

“Which code statement is better to include?”
Recap: A Journey through Searching Similar Code

- What motivated us? Systematic changes
- What were early attempts? Rule-based change abstraction
- How serious is this problem? Pretty serious
- How can we automate? Generalized patch synthesis

Several chicken and egg problems: Users need example patches, hints on what to generalize, and hints on how to pick a discriminatory atom.

- How can we examine variations at scale? Simultaneous overlay
- How to construct a search pattern with a human in the loop? Hints on global distribution and interactive what-if analysis
Discussion: Would’ve, could’ve, should’ve, etc.

- DSL
- Code embeddings
- LLM
- Information retrieval
- Search with multi-modality: text, video, etc.
Thank you!

public Diff handleMouseInSides(Canvas canvas, MergeSourceViewer msv, int my) {
    int lineHeight = msv.getTextWidget().getLineHeight();
    ChangeDiffs changeDiffs = getChangeDiffs();
    if (changeDiffs != null) {
        int shift = msv.getVerticalScrollOffset() + (2 * lineRegion.y) + (2 - LW);
        Iterator iter = changeDiffs.iterator();
        while (iter.hasNext()) {
            Diff diff = (Diff) iter.next();
            if (diff.isDeleted()) continue;
            if (fShowCurrentOnly2 && isCurrentDiff(diff)) continue;
            Point region = new Point(0, 0);
            msv.getLineRange(diff.getPosition(msv), region);
            int y = region.x * lineHeight + shift;
            int h = region.y * lineHeight;
            if (my >= y && my < y + h) return diff;
        }
    }
}

Merger merger = getMerger();
int lineHeight = tp.getTextWidget().getLineHeight();
if (merger != null) {
    Point region = new Point(0, 0);
    char leg = getLeg(tp);
    int shift = msv.getVerticalScrollOffset() + (2 - LW);
    for (Iterator iterator = merger.changesIterator(); iterator.hasNext();) {
        Diff diff = (Diff) iterator.next();
        if (diff.isDeleted()) continue;
        if (fShowCurrentOnly2 && isCurrentDiff(diff)) continue;
        Point region = new Point(leg, region);
        msv.getLineRange(diff.getPosition(leg), region);
    }
}
For each change example, a user can view AST edit operations.
A user can view a hierarchy of edit scripts and select one of them.
A User can inspect a generalized edit script
A user can correct suggested edits before applying the suggestion.
On average, Lase finds edit locations with 99% precision, 89% recall, and applies edits with 91% accuracy.

For three bugs, Lase suggests in total 9 edits that developers missed and later confirmed.

<table>
<thead>
<tr>
<th>Index</th>
<th>Bug(patches)</th>
<th>mi</th>
<th>Edit Location</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Σ</td>
<td>✅</td>
</tr>
<tr>
<td>2</td>
<td>82429(2)</td>
<td>16</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>139329(3)</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>103863(5)</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>129314(3)</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>95409(3)</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>24</td>
<td>98198(2)</td>
<td>9</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>
Lab Study Results

“I am confident in my answers to the API usage questions.”

Participant Count

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Neutral</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

Search
Examplore

Median: 4
Median: 5
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db", "root", "root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
Formulate a Search Query

• A user selects a code example and annotate important features.

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
```

A code example with user annotations

search query
Logic-based Code Search

Search Query

methodDec (i₀, m) ∧
type (i₁, ResultSet) ∧
contains (i₀, i₁) ∧
methodCall(i₂, executeQuery) ∧
contains (i₀, i₂) ∧
looplike (i₃, "*.next()") ∧
contains (i₀, i₃)

Fact Base

Matched Code

```java
public void getUserName(String id) {
    try {
        ResultSet set = db.executeQuery("select name from users where id=" + id);
        while (set.next()) { … }
    } catch (SQLException e) { … }
}
```

```java
public void queryDatabase() {
    try {
        ResultSet result = s.executeQuery("select * from customers");
        while (result.next()) { … }
    } catch (SQLException e) { … }
}
```

```java
public List get() {
    ResultSet set = stmt.executeQuery("select * from t");
    List l = new List();
    while (set.next()) { … }
    return l;
}
```

and 32 other matched locations
Align and aggregate structured call sequences into a single view

Glassman* and Zhang* et al. CHI 2018
Explore less frequent but critical API usage features
Interactively building your own patterns
A within-subject user study
RQ1. Does the bird’s-eye view help build robust API knowledge?

**Key Result:** Users with the bird’s-eye view answered API usage questions more *correctly* and *comprehensively*.
Design 1: Infer Common PDG Subgraph

The inferred pattern matched against Population of graphs
Design 2: Simultaneous Overlay

“What constructors are called?”

“What exceptions are caught?”
Design 2: Simultaneous Overlay

Statement Choice 1
Statement Choice 2
Statement Choice 3
Challenge 1: Instance-level feedback provides too little information

Cipher.getInstance(AES)
Cipher.init(…)
System.println(…)

Cipher.getInstance(AES)
Cipher.init(…)
System.println(…)

Cipher.getInstance(DES)

Cipher.getInstance(DES)
Cipher.init(…)
System.println(…)

Cipher.getInstance(...)

Cipher.getInstance(AES)

Cipher.getInstance(AES)

Cipher.getInstance(AES)
System.println(…)

Rest of population
Results: Users can better understand the API usage distribution when given guidance.

- 30% more correct answers
- 20% less time required
- 1.8X more likely to construct the target pattern

Improvements are statistically significant ($p < 0.001$) following a mixed-effects linear model accounting for ordering, tool, and task.