The Many Faces of Software Analytics

David Lo
School of Information Systems
Singapore Management University
davidlo@smu.edu.sg

Talk at the University of Luxembourg, Dec 2014
A Brief Self-Introduction

6,496 miles or 10,454 km
A Brief Self-Introduction

From Wikipedia
A Brief Self-Introduction
Singapore Management University

- Third university in Singapore
- Number of students:
  - 7000+ (UG)
  - 1000+ (PG)
- Schools:
  - Information Systems
  - Economics
  - Law
  - Business
  - Accountancy
  - Social Science
School of Information Systems

- Undergraduates: 1000+
- Master students: 100+
- Doctoral students: 50+
Our Research Group @ SMU
Our Research Group @ SMU

- 9 PhD Students
- 1 Visiting Professor
- 1 Research Engineer (Jan 2015)
Software Analytics

"Data exploration and analysis in order to obtain insightful and actionable information for data-driven tasks around software and services" (Zhang and Xie, 2012)
Software Analytics: Definition

- Analysis of a large amount of software data stored in various repositories in order to:
  - Understand software development process
  - Help improve software maintenance
  - Help improve software reliability
  - And more
Software Analytics
Research Directions: Software Analytics

Analytics for Coding & Collaboration
- Intelligent Multimodal Code Search
- Recommender for Libraries and Online Resources
- Coding and Collaboration With New Media

Analytics for Testing & Debugging
- Bug Finding and Fixing
- Bug Report Management
- Privacy-Preserving Test Anonymization

Analytics for Requirement & Design Validation
- Specification Mining and Inference
- Tracing Requirement to Source Code
- Design Defect Detection and Amelioration

Empirical Studies
Our Past and Current Work

Analytics for Coding & Collaboration

Intelligent Multimodal Code Search

Recommendation for Libraries and Online Resources

Coding and Collaboration With New Media

Empirical Studies

Analytics for Testing & Debugging

Bug Finding and Fixing

Bug Report Management

Privacy-Preserving Test Anonymization

Analytics for Requirement & Design Validation

Specification Mining and Inference

Tracing Requirement to Source Code

Design Defect Detection and Amelioration
Intelligent Multi Modal Code Search
Intelligent Multi Modal Code Search

User Query

Code Search Engine

Code base

Version control system, collaboration sites...

e.g., structured query, free text, code example...

Relevant Code

e.g., code fragment, method, class, projects, …
Intelligent Multimodal Code Search

**Nodes:** func A, func B, var C, var D;

**Relations:** C dataDepends A, D dataDepends B, D isFieldOf C;

**Targets:** D

**How do I load properties from an XML file?**

**Dependence Query Language**

**Nodes:**
- func A
- func B
- var C
- var D

**Relations:**
- C dataDepends A, D dataDepends B
- D isFieldOf C

**Targets:** D

**Free Text**

**Code Search Engine**

**Code Examples**

```python
if (tag -> check (a, b, c)) { 
tag -> a = a;
 b = 0;
 c = b;
} else {
 return a
}
```

```python
if (tag -> check (a, b, c)) { 
tag -> a = a;
 b = 0;
 c = b;
} else {
 return a
}
```
A developer can define a query about the dependence relationship in a bug pattern or a need-to-refactor code pattern. Using our search engine, he/she can find x1, x2, and x3 which are instances of the code pattern.
Workflow of Our Approach

- Query
- Query Graph Construction and Splitting
- Query Graphs
- Post-Filtering and Merging
- Graph Query Processing
- Indexed SDG
- Code
- SDG
- Graph Indexing

Query Results
Dependence Query Language (DQL)

- Allows developers to describe a target
  - Involving several code elements
  - Including the dependencies between the elements

- Composed of 4 parts
  - Query identifier declarations [D]
  - Code element (node) constraints [N]
  - Relation constraints [R]
  - Desired target identifiers [T]
Dependence Query Language (DQL)

- **Node Description [N]:** Code element constraints
  - contains <Text>, inFile <FileName>, inFunction <FnName>, controlType <for/while/switch/if>, etc.

- **Relation Description [R]:** Relationship constraints
  - A *(transitively)* controls B, A calls B, A is data dependent on B
  - A is one step *(directly)* <depend-operation> on B
  - A textual contains B, etc.
Query Splitting

- Split a query with disjunctions of conditions
- Result: Multiple queries with only conjunctions

function/control-point A, variable B; A contains "abc" or contains "de"; A dataDepends B; want A

- function A, variable B; A contains "abc"; A dataDepends B; want A
- control-point A, variable B; A contains "abc"; A dataDepends B; want A
- function A, variable B; A contains "de"; A dataDepends B; want A
- control-point A, variable B; A contains "de"; A dataDepends B; want A
Query Graph Construction

- Query Declarations
  - Each identifier becomes a node in the query graph

- Relation Descriptions
  - Each dependence relation becomes an edge in the query graph
Query Graph Splitting

- Divide the query graph to two sub-graphs
  - Each only capture control OR data dependences
Graph Indexing and Query

- **Purpose:**
  - Locate all instances of a given graph pattern in a large graph (Cheng et al., ICDE08)

Graph

(a) Graph

(b) Query

Three results found:
- triangle
- square
- star
Result Filtering & Merging

- Result Filtering
  - Textual conditions (e.g., textual contains)
  - Other relation descriptions

- Result Merging
  - Split 1: Disjunctions
  - Split 2: Data vs. Control Dependences
  - Need to union the sub-results
Evaluation

- Two open source projects
  - expat, gpsbabel

<table>
<thead>
<tr>
<th>Project name</th>
<th>Description</th>
<th>Version</th>
<th>Size (LOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>expat</td>
<td>XML handling library</td>
<td>2002-05-17</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2002-05-22</td>
<td>13</td>
</tr>
<tr>
<td>gpsbabel</td>
<td>GPS toolkit</td>
<td>2004-10-27</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2005-03-21</td>
<td>54</td>
</tr>
</tbody>
</table>

- Four software maintenance tasks
  - From pairs of snapshots from version histories
  - Developer change = Gold standard
## Overall Results: Accuracy

<table>
<thead>
<tr>
<th>Task</th>
<th>#Targets</th>
<th>Text Search</th>
<th>Code Clone Detection</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FP</td>
<td>FN</td>
<td>FP</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>526</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>8(186)</td>
<td>829(651)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
<td>297</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td>86</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

For task 2, the number in the bracket:

Adjusted numbers after considering correct locations that are not modified yet by developers
Find optimum connected graph that meets user needs
Greedy subgraph search algorithm with shortest path indexing

<table>
<thead>
<tr>
<th>Portfolio</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F₁</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group I</td>
<td>0.24</td>
<td>0.51</td>
<td>0.33</td>
<td>0.53</td>
<td>0.56</td>
<td>0.54</td>
</tr>
<tr>
<td>Group II</td>
<td>0.48</td>
<td>0.64</td>
<td>0.55</td>
<td>0.79</td>
<td>0.72</td>
<td>0.75</td>
</tr>
</tbody>
</table>
Example Based Code Search (ASEJ 15)

**Example 1:**
```java
if(c>3) {
    c = getStr();
    c = ext();
}
```

**Example 2:**
```java
if(b>1) {
    b = ext() + foo();
}
```

**Node declarations:** `ctrlPoint A, func B;`

**Node descriptions:** `A contains if, B contains ext;`

**Relationship descriptions:** `A oneStep controls B;`

**Targets:** `A, B;`

<table>
<thead>
<tr>
<th></th>
<th>Our</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prec.</td>
<td>0.684</td>
<td>0.584</td>
</tr>
<tr>
<td>Recall</td>
<td>0.721</td>
<td>0.767</td>
</tr>
<tr>
<td>F1</td>
<td>0.702</td>
<td>0.664</td>
</tr>
</tbody>
</table>

**PDGs Generation Engine**

**Query Generation Engine**

- Extend to compilable codes
- Generate PDGs
- Generate dependency query
- Recover textual information
- Mine common subgraphs

Lightweight type inference, Closed subgraph mining
Coding & Collaboration

Intelligent Multimodal Code Search

Structured Code Search (ASE10)
Structured + Topic Model (WCRE10)

Free Text Code Search (FSE12)

Example Based Code Search (ASEJ15)

Recmeender for Libraries and Online Resources

Coding and Collaboration With New Media

Empirical Studies

Active Code Search (ASE14)
Multi-Criteria Project Search (ICECCS13)
Similar Project Search (ICSM12)
Coding & Collaboration

Intelligent Multimodal Code Search

Recommender for Libraries and Online Resources

Coding and Collaboration With New Media

Empirical Studies

Recommending Related Libraries (WCRE13)

Recommending API Methods Given Feature Requests (ASE13)

Recommending Answer Posts (ASE11)
Coding & Collaboration

Intelligent Multimodal Code Search

Recommender for Libraries and Online Resources

Coding and Collaboration With New Media

Empirical Studies

Automated Content Categorization (ICPC14)

Recommendation of Tags to Contents (MSR13, ICSME14)

Observatory of Tweets and Trends (ASE11)

Developer Recommendation (WCRE11)

Recommending Best Answerers (QMC13)

Identification of Relevant Microblogs (ICSM12)

Project Success Estimation (CSMR13)
Our Past and Current Work

Analytics for Coding & Collaboration
- Intelligent Multimodal Code Search
- Recommender for Libraries and Online Resources
- Coding and Collaboration With New Media

Analytics for Testing & Debugging
- Bug Finding and Fixing
- Bug Report Management
- Privacy-Preserving Test Anonymization

Analytics for Requirement & Design Validation
- Specification Mining and Inference
- Tracing Requirement to Source Code
- Design Defect Detection and Amelioration

Empirical Studies
Bug Finding and Fixing are Hard!

- Software bugs cost the US Economy **59.5 billion dollars annually**
  - Stated by the US National Institute of Standards and Technology in 2002 (Tassey, 2002)

- Software debugging is an expensive and time consuming task in software projects
  - Testing and debugging activities account **30-90%** of the labor expended on a project (Beizer, 1990)
Bug Finding Techniques

A buggy program

Analyze program

List of possible buggy program elements
Bug Finding Techniques

**Failure**

```java
public void processFirstItem(ArrayList<Item> itemList) {
    if (!itemList.isEmpty()) {
        itemList.get(0);
    }
}

public void processFirstStudent(ArrayList<Student> studList) {
    studList.get(0);
}
```

**Bug Report**

With no apparent reason, the entire perspective becomes blank. I have seen this problem occasionally from release 1.0, and I have been hoping the problem would go away with newer releases. It still occurs frequently with 1.x.

**Bug Finder**

**Anomaly**
## Spectrum-Based Fault Localization

### Program Spectra

<table>
<thead>
<tr>
<th>Block ID</th>
<th>Program Element</th>
<th>T1</th>
<th>T2</th>
<th>T3, T4, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>double a, x; double ap, del, sum; int n; double temp; if ( x &lt;= 0.0 )</td>
<td>![circle]</td>
<td>![circle]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>{return 0.0;}</td>
<td></td>
<td>![circle]</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>del = sum = 1.0 / (ap = a); for ( n = 1; n &lt;= ITMAX; ++n){</td>
<td>![circle]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>sum += del *= x / ++ap; if ( Abs( del ) &lt; Abs( sum ) * EPS)</td>
<td>![circle]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 5        | /*BUGS: supposed to be:* /
/* temp = sum * exp(-x + a*log(x)-Lgamma(a))* /
 temp = sum * exp( x + a*log( x )-Lgamma(a)); return temp; } | ![circle] |  |  |

### Status of Test Case Execution

- **F**: Failure
- **P**: Pass

**Program spectra**
# Measuring suspiciousness

## Program Elements

<table>
<thead>
<tr>
<th>Block ID</th>
<th>Program Elements</th>
<th>Ochiai</th>
<th>Klosgen</th>
<th>Pietatsky Shapiro</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>double a, x; double ap, del, sum; int n; double temp; if ( x &lt;= 0.0 )</td>
<td>0.82</td>
<td>0.31</td>
<td>-0.04</td>
</tr>
<tr>
<td>2</td>
<td>{return 0.0;}</td>
<td>0.39</td>
<td>0.06</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>del = sum = 1.0 / (ap = a); for ( n = 1; n &lt;= ITMAX; ++n ){</td>
<td>0.93</td>
<td>0.34</td>
<td>-0.15</td>
</tr>
<tr>
<td>4</td>
<td>sum += del *= x / ++ap; if ( Abs( del ) &lt; Abs( sum ) * EPS ){</td>
<td>0.93</td>
<td>0.34</td>
<td>-0.15</td>
</tr>
<tr>
<td>5</td>
<td>/<em>BUGS: supposed to be:</em>/ /<em>temp = sum * exp(-x + a</em>log(x)-LGamma(a))*/ temp = sum * exp( x + a * log( x ) - LGamma(a)); return temp; }</td>
<td>0.93</td>
<td>0.34</td>
<td>0</td>
</tr>
</tbody>
</table>

## e.g., spectrum-based fault localization

(Abreu et.al, TAI CPART-MUTATION’07, Lucia et al., ICSM’10)
Motivation

There is no single fault localization techniques that is the best in all cases. (Lucia et al., J SEP, 2014)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Version</th>
<th>Ochiai</th>
<th>Klosgen</th>
<th>Pietatsky Saphiro</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>v35</td>
<td>0.01</td>
<td>0.13</td>
<td>0.47</td>
</tr>
<tr>
<td>nanoxml</td>
<td>v2_b6</td>
<td>0.50</td>
<td>0.05</td>
<td>0.63</td>
</tr>
<tr>
<td>tcas</td>
<td>v23</td>
<td>0.56</td>
<td>0.56</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Combine different techniques?
Fusion Localizer (ASE14)

1. Scores Normalization
   - Normalized Scores

2. Technique Selection
   - Selected Techniques

3. Data Fusion
   - New Suspiciousness Scores Assigned to Program Elements

Fault Localization Techniques

Suspiciousness Scores Assigned to Program Elements
Step 2. Techniques selection

A set of fault localization techniques

Choosing the techniques to be fused

(A) Overlap-based selection

(B) Bias-based selection

Selected fault localization techniques
Step 2. Techniques selection

(A) Overlap-based selection

- Based on the overlap ratio
- Select 50% of the least overlap techniques

(Wu, Data Fusion in Information Retrieval, 2012)
Step 2. Overlap-based selection

<table>
<thead>
<tr>
<th>Technique</th>
<th>Top-K Most Suspicious Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ochiai</td>
<td>Block 2, Block 3, Block 4, Block 7, Block 8</td>
</tr>
<tr>
<td>Klosgen</td>
<td>Block 4, Block 5, Block 6, Block 7, Block 9</td>
</tr>
<tr>
<td>Piat. Shapiro</td>
<td>Block 1, Block 4, Block 5, Block 6, Block 8</td>
</tr>
<tr>
<td>Tarantula</td>
<td>Block 4, Block 5, Block 6, Block 8, Block 10</td>
</tr>
</tbody>
</table>

$L_{all}$: Block 1, Block 2, Block 3, Block 4, Block 5, Block 6, Block 7, Block 8, Block 9, Block 10

$L_{Ochiai}$: Block 2, Block 3

\[
o_{rate_i} = \frac{|L_{all}| - |L_i|}{|L_{all}|}
\]

Overlap Rate of Ochiai = \(\frac{10 - 2}{10} = 0.8\)
Step 2. Technique selection

(B) Bias-based selection

- Based on the similarity score
- $\text{Bias} = 1 - \text{similarity score}$
- Select 50% of the most biased techniques

(Nuray and Can, Information Processing and Management, 2006)
Step 2. Bias-based selection

<table>
<thead>
<tr>
<th>Block</th>
<th>Freq.</th>
<th>Block</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1</td>
<td>1</td>
<td>Block 1</td>
<td>0</td>
</tr>
<tr>
<td>Block 2</td>
<td>1</td>
<td>Block 2</td>
<td>1</td>
</tr>
<tr>
<td>Block 3</td>
<td>1</td>
<td>Block 3</td>
<td>1</td>
</tr>
<tr>
<td>Block 4</td>
<td>4</td>
<td>Block 4</td>
<td>1</td>
</tr>
<tr>
<td>Block 5</td>
<td>3</td>
<td>Block 5</td>
<td>0</td>
</tr>
<tr>
<td>Block 6</td>
<td>3</td>
<td>Block 6</td>
<td>0</td>
</tr>
<tr>
<td>Block 7</td>
<td>2</td>
<td>Block 7</td>
<td>1</td>
</tr>
<tr>
<td>Block 8</td>
<td>3</td>
<td>Block 8</td>
<td>1</td>
</tr>
<tr>
<td>Block 9</td>
<td>1</td>
<td>Block 9</td>
<td>0</td>
</tr>
<tr>
<td>Block 10</td>
<td>1</td>
<td>Block 10</td>
<td>0</td>
</tr>
</tbody>
</table>

Cosine Similarity

$$Sim(L_i, L_{all}) = \frac{\sum_{j=1}^{m} L_j \times L_{all_j}}{\sqrt{\sum_{j=1}^{m} L_j^2} \times \sqrt{\sum_{j=1}^{m} L_{all_j}^2}}$$

$$Bias(L_i, L_{all}) = 1 - Sim(L_i, L_{all})$$

$$Sim(L_{Ochiai}, L_{all}) = \frac{1 + 1 + 4 + 2 + 3}{\sqrt{5} \times \sqrt{(1 + 1 + 1 + 1 + 1 + 1 + 9 + 9 + 4 + 9 + 1 + 1)}} = 0.6822$$

$$Bias(L_{Ochiai}, L_{all}) = 0.3178$$
Data fusion methods

- **Score-based fusion**
  1. **CombSUM**: Sum up all scores (Fox et al., NIST, 1994)
  2. **CombANZ**: Average of the non-zero scores (Fox et al., NIST, 1994)
  3. **CombMNZ**: Sum up all scores multiplied by the number of techniques that assign a non-zero score (Fox et al., NIST, 1994)

- **Ranking-based fusion**
  5. **Borda Count**: Sum up all ranking (Aslam and Montague, SIGIR, 2001)
Variants of Fusion Localizer

- Fault Localization Techniques
  - Zero-One Normalization
  - Reciprocal Ranking Normalization
- New Suspiciousness Scores Assigned to Program Elements
  - CombSUM, CombANZ, CombMNZ
  - Correlation-based fusion, BordaCount

1. Score Normalization, Technique Selection
2. Data Fusion
   - 1. Overlap-based
   - 2. Bias-based
### Dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LOC</th>
<th>Num. of Buggy Version</th>
<th>Num. of Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_token</td>
<td>478</td>
<td>5</td>
<td>4,130</td>
</tr>
<tr>
<td>print_token2</td>
<td>399</td>
<td>10</td>
<td>4,115</td>
</tr>
<tr>
<td>replace</td>
<td>512</td>
<td>31</td>
<td>5,542</td>
</tr>
<tr>
<td>schedule</td>
<td>292</td>
<td>9</td>
<td>2,650</td>
</tr>
<tr>
<td>schedule2</td>
<td>301</td>
<td>9</td>
<td>2,710</td>
</tr>
<tr>
<td>tcas</td>
<td>141</td>
<td>36</td>
<td>1,608</td>
</tr>
<tr>
<td>tot_info</td>
<td>440</td>
<td>19</td>
<td>1,051</td>
</tr>
<tr>
<td>space</td>
<td>6,218</td>
<td>35</td>
<td>13,585</td>
</tr>
<tr>
<td>NanoXML v1</td>
<td>3,497</td>
<td>6</td>
<td>214</td>
</tr>
<tr>
<td>NanoXML v2</td>
<td>4,007</td>
<td>7</td>
<td>214</td>
</tr>
<tr>
<td>NanoXML v3</td>
<td>4,608</td>
<td>9</td>
<td>216</td>
</tr>
<tr>
<td>NanoXML v5</td>
<td>4,782</td>
<td>8</td>
<td>216</td>
</tr>
<tr>
<td>XML security v1</td>
<td>21,613</td>
<td>6</td>
<td>92</td>
</tr>
<tr>
<td>XML security v2</td>
<td>22,318</td>
<td>6</td>
<td>94</td>
</tr>
<tr>
<td>XML security v3</td>
<td>19,895</td>
<td>4</td>
<td>84</td>
</tr>
<tr>
<td>Rhino</td>
<td>49k</td>
<td>11</td>
<td>20-152</td>
</tr>
<tr>
<td>Lucene</td>
<td>88k</td>
<td>9</td>
<td>1,072-1,154</td>
</tr>
<tr>
<td>Ant</td>
<td>264k</td>
<td>10</td>
<td>1,024-1,555</td>
</tr>
</tbody>
</table>

Total: 230 Bugs
Avg. % of code inspected to localize all bugs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Average</th>
<th>Technique</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero-One, Overlap CombANZ</td>
<td>21.36%</td>
<td>Naish2</td>
<td>24.63%</td>
</tr>
<tr>
<td>Zero-One, Bias CombANZ</td>
<td>21.39%</td>
<td>GP13</td>
<td>24.78%</td>
</tr>
<tr>
<td>Zero-One, Bias CombSUM</td>
<td>22.94%</td>
<td>Ochiai</td>
<td>25.29%</td>
</tr>
<tr>
<td>Zero-One, Overlap CorrB_Top50%</td>
<td>23.11%</td>
<td>GP03</td>
<td>25.82%</td>
</tr>
<tr>
<td>Zero-One, Overlap CombSUM</td>
<td>23.15%</td>
<td>Tarantula</td>
<td>26.77%</td>
</tr>
<tr>
<td>Zero-One, Overlap CorrB_Top10%</td>
<td>23.23%</td>
<td>GP19</td>
<td>31.60%</td>
</tr>
<tr>
<td>Zero-One, Overlap CombMNZ</td>
<td>23.31%</td>
<td>Naish1</td>
<td>34.40%</td>
</tr>
<tr>
<td>Zero-One, Bias CorrB_Top10%</td>
<td>23.33%</td>
<td>GP02</td>
<td>39.48%</td>
</tr>
<tr>
<td>Zero-One, Bias CorrB_Top50%</td>
<td>23.38%</td>
<td>Russel&amp;Rao</td>
<td>42.48%</td>
</tr>
<tr>
<td>Zero-One, Overlap CorrA_Top10%</td>
<td>23.56%</td>
<td>Binary</td>
<td>52.04%</td>
</tr>
<tr>
<td>Zero-One, Bias CombMNZ</td>
<td>23.78%</td>
<td>Wong1</td>
<td>86.26%</td>
</tr>
<tr>
<td>Zero-One, Bias CorrA_Top10%</td>
<td>23.78%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Proportion/number of bugs localized

#### When 10% of blocks are inspected

<table>
<thead>
<tr>
<th>Technique</th>
<th>%Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{\text{Zero-One,Overlap CombANZ}}$</td>
<td>46.96%</td>
</tr>
<tr>
<td>$F_{\text{Zero-One,Bias CombANZ}}$</td>
<td>46.52%</td>
</tr>
<tr>
<td>Ochiai</td>
<td>42.17%</td>
</tr>
<tr>
<td>Naish2</td>
<td>36.96%</td>
</tr>
<tr>
<td>GP13</td>
<td>36.96%</td>
</tr>
</tbody>
</table>

#### When 10 blocks are inspected

<table>
<thead>
<tr>
<th>Technique</th>
<th>Hit@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{\text{Zero-One,Bias CombANZ}}$</td>
<td>91</td>
</tr>
<tr>
<td>$F_{\text{Zero-One,Overlap CombANZ}}$</td>
<td>87</td>
</tr>
<tr>
<td>Ochiai</td>
<td>74</td>
</tr>
<tr>
<td>Naish2</td>
<td>73</td>
</tr>
<tr>
<td>GP13</td>
<td>72</td>
</tr>
</tbody>
</table>
Report-Directed Bug Finding (ICSME14)

On average, AmaLgam\textsubscript{Composite} improves AmaLgam by 6.8%, 8.0%, 5.0%, 14.4%, and 6.5% in terms of Hit@1, Hit@5, Hit@10, MAP, and MRR respectively.

\[
\sum_{i=1}^{15} w_i \times VSM_i(b, f) + \sum_{j \in H, R, S} w_j \times score_j(b, f),
\]
Anomaly-Directed Bug Finding (ICSE12)

### Code Fragment 1
```
219: struct dentry * dentry = sd->s_dentry;
220: if (dentry) {
    /* the following parts are detected as clones */
222:     spin_lock(&dcache_lock);
223:     spin_lock(&dentry->d_lock);
224:     if (!d_unhashed(dentry) & dentry->d_inode) {
        dget_locked(dentry);
226:         __d_drop(dentry);
227:         spin_unlock(&dentry->d_lock);
228:         spin_unlock(&dcache_lock);
229:         .......
```

### Code Fragment 2
```
456: struct dentry *tmp;
457: tmp = lookup_one_len(name, parent, strlen(name));
460: spin_lock(&dcache_lock);
461: spin_lock(&tmp->d_lock);
462: if (!d_unhashed(tmp) & tmp->d_inode) {
463:     dget_locked(tmp);
464:     __d_drop(tmp);
465:     spin_unlock(&tmp->d_lock);
466:     spin_unlock(&dcache_lock);
468:     .......
```

### Feature Extraction + Classification

- Improve
  - Avg. % TP Found:
    - 11% for Linux
    - 87% for Eclipse
    - 86% for ArgoUML
Testing & Debugging

Bug Finding and Fixing

Bug Report Management

Privacy-Preserving Test Anonymization

Empirical Studies

Failure-Directed

Report-Directed

Anomaly-Directed

Extensions

Automated Patching

Eff. Estimate:
ISSRE14,
ICSM13-EMSE15

Reduce. Man. Eff.:
ASE12-ASEJ15

Comm. Resource:
FSE14
ICSE14

Post Mortem:
WCRE13

ASE14
ICSM10-JSEP14
ICSM12
ASE11
HASE11
ISSTA09

ICSME14
ICSE12
ICPC14x2

CSMR-WCRE14
COMPSAC14
SAC14
ICSE12
RV11
ASE10
KDD09

ASE12
ICSE12
Testing & Debugging

- Bug Finding and Fixing
- Bug Report Management
- Privacy-Preserving Test Anonymization
- Empirical Studies

Duplicate Detection
- ASE12
- CSMR12
- ASE11
- ICSE10

Report Prioritization
- ICSM13-EMSE15
- WCRE12
- ICSM12

Report Categorization
- COMPSAC14
- ICECCS14
- WCRE12

Report Assignment
- WCRE13

Reopen Prediction
- ASEJ 15
- CSMR13
Testing & Debugging

Bug Finding and Fixing

Bug Report Management

Privacy-Preserving Test Anonymization

Empirical Studies

1

Single-Data Release

PLDI 11

Multiple-Data Release

ASE12
Testing & Debugging

Bug Finding and Fixing

Bug Report Management

Privacy-Preserving Test Anonymization

Empirical Studies

Real Bugs

Test Adequacy

Fault Localization

Bug Trackers

Bug Linking

ASE12-ASEJ 15
IEICE Trans14
SAC14
ISSRE12

APSEC14
CSMR-QSI C13

ASE14
MSR12
ICSM13

CSMR-WCRE14
ISSRE13

CSMR13
Our Past and Current Work

- **Analytics for Coding & Collaboration**
  - Intelligent Multimodal Code Search
  - Recommender for Libraries and Online Resources
  - Coding and Collaboration With New Media

- **Analytics for Testing & Debugging**
  - Bug Finding and Fixing
  - Bug Report Management
  - Privacy-Preserving Test Anonymization

- **Empirical Studies**
- **Analytics for Requirement & Design Validation**
  - Specification Mining and Inference
  - Tracing Requirement to Source Code
  - Design Defect Detection and Amelioration
Specification Mining and Inference

- Most bugs are caused due to semantic errors (Tan et al., ESEJ 14)
  - Programs are not implemented according to requirements
- Developers often do not have the expertise or time to write formal specifications

- Viable solution: specification mining
  - Automated reverse engineering of specifications from programs
Specification Mining and Inference

**Strong Properties**
- Likely invariants
- Frequent patterns
- Temporal rules
- Live sequence charts

**Unified Model**
- Finite State Machine
- Message Sequence Graphs
- Class Diagram

**Execution Traces**

**Specification Miners**
Mining Temporal Rules [J SEP08, SCP12, ICDE12]

- **Aim:**
  - Find temporal rules observed within a trace set: “Whenever a series of events occurs, eventually another series of events will also occur”
  - Among most widely used temporal logic expression for verification (Dwyer et al. ICSE’99).

---

**LTL BNF Notation**

<table>
<thead>
<tr>
<th>rules</th>
<th>:=</th>
<th>G(prepost)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prepost</td>
<td>:=</td>
<td>event → post</td>
</tr>
<tr>
<td>post</td>
<td>:=</td>
<td>XF(event)</td>
</tr>
</tbody>
</table>
Significance, Soundness and Completeness

- **Distinguish Significant Rules via Statistical Notions**
  - **Support**: The number of traces supporting the premise
  - **Confidence**: The likelihood of the premise being followed by the consequent

- **Ensure Soundness and Completeness**
  - Sound: All mined rules are statistically significant
  - Complete: All statistically significant rules are mined/represented
Scalability Challenge

Existing Method (Yang06)

Check all possible 2-event rules (n x n of them) for statistical significance

Need to check n^L rules for L-event rules

> 50^1000 operations

vs.

< 25 seconds

Our Method

Explore the search space depth first and identify significant ones

Employ a number of search space pruning strategies

Linear to the size of the output significant rules and the length of traces

Good results on standard benchmarks datasets
Specification Mining Strategies – I & II

**[Apriori – Support]**

\[ Rx = p \rightarrow c; \text{ sup}(Rx) < \text{min}_\text{sup} \]

\[ p \sqsubset q \]

\[ \frac{\text{sup}(Rx) < \text{min}_\text{sup}}{\text{sup}(Ry) < \text{min}_\text{sup}} \]

**Ry is not significant**

**Rx:** \( a \rightarrow z \); \( \text{sup}(Rx) < \text{min}_\text{sup} \)

\( a, b \rightarrow z \)

\( a, b, c \rightarrow z \)

\( a, c \rightarrow z \)

\( a, b, d \rightarrow z \)

\( \ldots \)

\( \text{Ry}_s \)

**Non-significant**

**[Apriori – Confidence]**

\[ Rx = p \rightarrow c; \text{ Ry} = p \rightarrow d \]

\[ c \sqsubset d \]

\[ \frac{\text{conf}(Rx) < \text{min}_\text{conf}}{\text{conf}(Ry) < \text{min}_\text{conf}} \]

**Ry is not significant**

**Rx:** \( a \rightarrow z \); \( \text{conf}(Rx) < \text{min}_\text{conf} \)

\( a \rightarrow b, z \)

\( a \rightarrow b, c, z \)

\( a \rightarrow c, z \)

\( a \rightarrow b, d, z \)

\( \ldots \)

\( \text{Ry}_s \)

**Non-significant**

**School of Information Systems**

**SMU**

**Singapore Management University**

64
Detecting Redundant Rules

\[ Rx = p \rightarrow c; Ry = q \rightarrow d \]
\[ p \sqcup c \sqsubseteq q \sqcup d \]
\[ sup(Rx) = sup(Ry) \]
\[ conf(Rx) = conf(Ry) \]

\[ Rx \text{ is redundant} \]

Redundant rules are identified and removed early during mining process.

Rx: \( a \rightarrow b, c, d \)

\[ \begin{align*}
Ry_s & : a \rightarrow b \\
 & a \rightarrow c \\
 & a \rightarrow b, c \\
 & a \rightarrow b, d \\
& \ldots
\end{align*} \]

Redundant iff sup and conf are the same
A series of transaction **set up events** (connection to server instance, transaction manager and implementation set up) is eventually followed with transaction **termination events** (transaction completion, resource release).
Program Verification: VCS Application

Bug: Store (S) and rename (N) without appropriate next actions

[Bug-1]

Bug: Deletion (D) without log update

Mined Bug-Identifying Rules/Properties

\(<W;X;G;T;N> \rightarrow <S;O;Y>\)

[Bug-2]

\(<W;X;G;C;I;D> \rightarrow <A;O;Y>\)

[Bug-3] [Bug-4]
Library Usage Rules: Windows (WCRE09,SCP12)

- Collect traces from 10 Windows Applications:
  - Excell, OneNote, TextPad, VS.Net, Visio, WMPlayer, Virtual PC, Movie Maker, WordPad, Access

- Collect traces pertaining to:
  - Registry, Memory Management, GDI (Device Control and UI related API)
  - Produces several million events
Library Usage Rules: Windows

V HeapAlloc(,,); ->HeapFree(,,V);
V GlobalAlloc(,); -> GlobalFree(V);
V VirtualAlloc(,,); ->VirtualFree (,,V);

... 

HeapFree(,,v); -P> V HeapAlloc(,,,,);

Detect **double free**, which is disallowed
“Calling HeapFree twice with the same pointer can cause heap corruption, resulting in subsequent calls to HeapAlloc returning the same pointer twice.” [MSDN]
Library Usage Rules: Windows

RegCreateKeyExA(V,.) -> RegCloseKey(V);
Not all opened registry need to be closed
Predefined keys need not be closed

V CreateCompatDC(); -> DeleteDC(V);
V CreCompatBmap(,,); -> DeleteObj(V);
V CreRectRgn(,,,) -> DeleteObj(V);
DeleteDC(V) – precede -> V CreCompDC()
SetBkColor(,V); -> V SetBkColor(,)

...
Mining Live Sequence Charts (ASE10, ASEJ 12)

Method | Pre                                              | Post
-------|---------------------------------------------------|--------------------------------------------------
send(...) | code=257 subID="PWD"                             | subID="PWD"

<table>
<thead>
<tr>
<th>Method</th>
<th>CFTP</th>
<th>Jeti</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario Min.</td>
<td>53s</td>
<td>2s</td>
</tr>
<tr>
<td>Daikon (All/ Sli)</td>
<td>163s</td>
<td>31s</td>
</tr>
<tr>
<td>Slicing</td>
<td>11s</td>
<td>3s</td>
</tr>
</tbody>
</table>

Traces (T) → Scenario-Based Specification Miner → Scenario-Based LSCs → Sliced Trace (ST) → Daikon → Invariants on ST

Traces (T) → Integrate Invariants into LSCs → Scenarios/ LSCs → Scenario Specific Invariants → Invariant Comparator → Invariants on T

send(…) code=257 subID="PWD" ...

resetState(…)
getFileSystemView(…)
send(…)
getLanguage(…)
Mining Finite State Machines (FSE09)

- FSM learner often overgeneralizes
  - Generates a prefix tree acceptor
  - Merge nodes (generalization)

Identification of bad merges using mined temporal rules

| System Model                        | |Evs.| |kTail| |With Refinement| |
|------------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|                                    |                 | Precs. | Recall | Time | Precs. | Recall | Time |
| X11 Windowing Library              | 356.400         | 0.873  | 1.000  | 0.211 | 0.905  | 1.000  | 0.218 |
| CVS Client                         | 2121.000        | 0.169  | 0.970  | 0.557 | 1.000  | 0.970  | 0.616 |
| WebSphere Business Processes       | 9317.080        | 1.000  | 0.999  | 1.453 | 1.000  | 0.999  | 1.528 |
Mining Message Sequence Graphs (ICSE12)

<table>
<thead>
<tr>
<th>Trace Set</th>
<th>Concrete-class Trace Set</th>
<th>Abstract Model</th>
<th>Class-Level Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
<td>CPU</td>
<td>Dev</td>
<td></td>
</tr>
<tr>
<td>Req</td>
<td>ℓ. ends(ℓ/Release)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grant</td>
<td>ℓ. ends(Req) Addr</td>
<td>ℓ. true</td>
<td></td>
</tr>
<tr>
<td>ℓ. bet(Grant, Release)</td>
<td>Ack</td>
<td>ℓ. true</td>
<td></td>
</tr>
<tr>
<td>ℓ. bet(Grant, Release)</td>
<td>Data</td>
<td>ℓ. ends(Data)</td>
<td></td>
</tr>
<tr>
<td>Release</td>
<td>ℓ. bet(Grant, Release)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guard Inference</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace Set</td>
<td>Aggregate Model</td>
</tr>
<tr>
<td>Concrete</td>
<td>Symbolic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Prec.</th>
<th>Recall</th>
<th>F1</th>
<th>Prec.</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIP</td>
<td>0.8</td>
<td>0.05</td>
<td>0.09</td>
<td>0.64</td>
<td>0.66</td>
<td>0.65</td>
</tr>
<tr>
<td>XMPP-Core</td>
<td>1.0</td>
<td>0.19</td>
<td>0.32</td>
<td>1.0</td>
<td>0.66</td>
<td>0.79</td>
</tr>
<tr>
<td>XMPP-MUC</td>
<td>0.61</td>
<td>0.36</td>
<td>0.45</td>
<td>0.67</td>
<td>0.63</td>
<td>0.65</td>
</tr>
<tr>
<td>CTAS</td>
<td>0.25</td>
<td>0.43</td>
<td>0.31</td>
<td>0.88</td>
<td>0.90</td>
<td>0.89</td>
</tr>
</tbody>
</table>
Requirement & Design

**Specification Mining and Inference**

**Tracing Requirement to Source Code**

**Design Defect Detection and Amelioration**

**Empirical Studies**

---

**Strong Properties**

- Inv.
- Patterns
- Rules
- LSC

**Unified Model**

- FSM
- MSG

**Class Diagram**

- ICDE09
- KDD07
- SCP12
- ICDE11
- ASE13
- ICECCS11
- TKDE11
- ASE10-ASEJ12
- IS09
- ASE09
- WODA08
- PASTE08
- DASFAA08
- ASE08
- JSEP08
- ASE07
- FSE09
- ICSE12
- FSE06
- ICSE11
- ICPC14
Requirement & Design

Specification Mining and Inference

Tracing Requirement to Source Code

Design Defect Detection and Amelioration

Empirical Studies

Concern Location

ICSM13
WCRE11

Design Defect Detection In Tiered Architecture

SEKE11

Empirical Evaluation on Specification Miners

JSS12
WCRE06

Empirical Evaluation on Interestingness Measures
Future Directions
Big Data for Software Engineering

Millions of Projects
Millions of Bug Reports
Millions of Repositories
Millions of Blogs and Posts
Thousands of APIs
Wealth of Software Engineering Data

There is a wealth of information about what’s new, what works, and what doesn’t in the Web
Difficulty in “Making Sense” of Data

Thousands of new tutorials, blogs, and microblogs are being generated daily.
Our Vision: Personalized Observatory

- Highlights new developments, new solutions, and new pitfalls personalized to a target developer
- Gathers, groups, filters, and summarizes information obtained from various channels
- Automatically updates itself when relevant new information is released in the web
Proposed Process

Data Collection

Data Linking and Grouping

Data Matching and Filtering

Data Summarization

Automatic Update
Process: Data Collection (1)

- Gather data from various information channels
Process: Grouping (2)

- Link related pieces of data together
- Group them to a higher level concept
- Approaches:
  - Topic modeling, Clustering
Process: Matching (3)

- Match user interest to community data
- Approaches:
  - Information retrieval approaches
Process: Summarization (4)

- **Motivation:** A large collection of documents from the community might match user interests
  - Need to summarize them to a manageable size
- **Approach:** Text summarization approaches
Process: Update (5)

- Continually update considering new user and community data
Proposed Infrastructure

**Client Side Component**
- Data Collector
- Automatic Updater
- Data Linker

**Server Side Component**
- Data Collector
- Data Summarizer
- Data Linker
- Automatic Updater
- Data Matcher

**Automatic Updater**
Challenges: Vocabulary Mismatch

- Assumption: Related pieces of information are textual similar.
- Reality: Developer might use peculiar words that are not commonly used by others in the same community.
- How to bridge the differences in the vocabulary used by various developers?
Challenges: Privacy Concern

- Client component needs to send queries to server component
  - Includes developer personal data
- Raises privacy concern:
  - Can some private information be leaked?
  - Sensitive web data, source code, industry project, etc.
- How to minimize privacy leak while not reducing utility?
Challenges: Near Real-Time Update

- Huge amount of information being generated constantly on the web.
- Scale-up the server side component:
  - How to design efficient, incremental and parallel algorithms to collect, group, match, and summarize data?
- Reduce the size of queries being sent from clients to servers:
  - How to produce informative yet succinct queries?
State of Research @ SMU

- **Data Collection:**
  - Observatory of trends in software related microblogs. ASE 2012
  - Automatic classification of software related microblogs. ICSM 2012
State of Research @ SMU

- **Dealing with Vocabulary Mismatch:**
  - Automated construction of a software-specific word similarity database. CSMR-WCRE 2014.

- **Dealing with Privacy Concern:**
  - kb-anonymity: a model for anonymized behaviour-preserving test and debugging data. PLDI 2011.
Summary: Software Analytics

Analytics for Coding & Collaboration
- Intelligent Multimodal Code Search
- Recommender for Libraries and Online Resources
- Coding and Collaboration With New Media

Analytics for Testing & Debugging
- Bug Finding and Fixing
- Bug Report Management
- Privacy-Preserving Test Anonymization

Analytics for Requirement & Design Validation
- Specification Mining and Inference
- Tracing Requirement to Source Code
- Design Defect Detection and Amelioration
Summary: Future Directions

Millions of Projects
Millions of Bug Reports
Millions of Repositories
Millions of Blogs and Posts
Thousands of APIs

...
Thank you!

Questions? Comments? Advice?
davidlo@smu.edu.sg