Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis

Chu-Pan Wong\textsuperscript{1}, Yingfei Xiong\textsuperscript{1}, Hongyu Zhang\textsuperscript{2}, Dan Hao\textsuperscript{1}, Lu Zhang\textsuperscript{1}, Hong Mei\textsuperscript{1}

\textsuperscript{1}Peking University
\textsuperscript{2}Microsoft Research Asia
INTRODUCTION
Background

Large amount
- Eclipse got 4414 bug reports in 2009

Painstaking
- 11892 source code files in Eclipse 3.1
- No prior knowledge for new developers
Bug-Report-Oriented Fault Localization

Bug reports as queries

Rate source files by heuristics

Ranked list of source code files

Developers
This Talk

Two new heuristics
A Typical Approach -- BugLocator

• Combining three heuristics

• First heuristic: VSM (vector space model) similarity between the bug report and files
  – Each document represented as a vector of token weights
  – Token weight = token frequency × inverse document frequency
An Example for VSM

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>80720</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Pinned console does not remain on top</td>
</tr>
<tr>
<td>Description</td>
<td>Open two console views, ... Pin one console. Launch another program that produces output. Both consoles display the last launch. The pinned console should remain pinned.</td>
</tr>
</tbody>
</table>

Corresponding source code file: `ConsoleView.java`

```java
public class ConsoleView extends PageBookView implements IConsoleView IConsoleListener {

    ...

    public void display(IConsole console){
        if (fPinned&&fActiveConsole!=null) {return;}
    }

    ...

    public void pin(IConsole console){
        if (console == null) {
            setPinned(false);
        } else{
            if (isPinned()){ setPinned(false); } display(console);
            setPinned(true);
        }
    }

} 
```
A Typical Approach -- BugLocator

• Second heuristic: large files
  – Existing studies show that large files has higher fault density

• Third heuristic: similar bug reports
  – The files modified in the fix of a previously similar bug report are more likely to contain faults

• Final score = VSM score × large file score + similar bug report score
Existing Problem 1

• Noise in large source code files
  – When file size changes, fault density may change more than an order of magnitude
  – BugLocator: large file score range from 0.5~0.73
  – Large files may have much noise
Motivation Example - Noise

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>87692</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>setConsoleWidth() causes Invalid thread access</td>
</tr>
<tr>
<td>Description</td>
<td>calling setConsoleWidth() outside UI thread causes Invalid thread access.</td>
</tr>
</tbody>
</table>

- If BugLocator is used
  - Accessible.java is ranked 1\textsuperscript{st}
  - TextConsoleViewer.java (real fix) is ranked 26\textsuperscript{th}

- Problems
  - Noisy words
    - “access”
    - “invalid”
    - “call”
Our solution - Segmentation

Using segmentation technique, TextConsoleViewer.java is ranked to 1st
Existing Problem 2

- Stack Traces Information
  - Direct clues for bugs
  - Often treated as plain text
<table>
<thead>
<tr>
<th>Bug ID</th>
<th>87855</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>NullPointerException in Table.callWindowProc</td>
</tr>
</tbody>
</table>

Table.java is suspicious

Table.java is ranked to 252nd in BugLocator.
APPROACH
Segmentation

• Extract a corpus
  – Lexical tokens
  – Keywords removal (e.g. float, double)
  – Separation of concatenated word (e.g. isComitable)
  – Stop words removal (e.g. a, the)

• Evenly divide corpus into segments
  – Each segment contains $n$ words

• VSM score = the highest score of all segments
Fixing Large File Scores

- \( \text{LargeFileScore}(\#\text{terms}) = \frac{1}{1 + e^{-\beta \times \text{Nor}(\#\text{terms})}} \)
- Function \( \text{Nor} \) normalize values to \([0, 1]\) based on even distribution
- Parameter \( \beta \) in BugLocator is always 1
- Can be a larger number in our approach
Stack-Trace Analysis

- Extract file names from stack traces ($D$)
- Identify closely related files by imports ($C$)
- A defect is typically located in one of the top-10 stack frames

$$\text{BoostScore}(x) = \begin{cases} 
\frac{1}{\text{rank}} & x \in D \& \text{rank} \leq 10 \\
0.1 & x \in D \& \text{rank} > 10 \\
0.1 & x \in C \\
0 & \text{otherwise}
\end{cases}$$
Calculating Final Scores for Source Code Files

Modified BugLocator Score

BoostScore

Final Score
EVALUATION
## Subjects and Parameters

<table>
<thead>
<tr>
<th>Project</th>
<th>Studied Period</th>
<th>#Bug Reports</th>
<th>#Source Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse 3.1</td>
<td>Oct 2004 - Mar 2011</td>
<td>3075</td>
<td>11892</td>
</tr>
<tr>
<td>AspectJ 1.5</td>
<td>Jul 2002 - Oct 2006</td>
<td>286</td>
<td>5487</td>
</tr>
<tr>
<td>SWT 3.1</td>
<td>Oct 2004 - Apr 2010</td>
<td>98</td>
<td>484</td>
</tr>
</tbody>
</table>

- Parameters
  - Segmentation Size $n = 800$
  - Large File Factor $\beta = 50$
  - No universally best values
Metrics

- Standard ones also used in BugLocator
- Top N Rank of Files (TNRF)
  - The percentage of bugs whose any related files are listed in top N of returned files
- Mean Reciprocal Rank (MRR)
  - How high the first related files are ranked
    - \[ MRR = \frac{\sum_{i=1}^{\|BR\|} 1/\text{rank}(i)}{\|BR\|} \]
- Mean Average Precision (MAP)
  - How high all related files are ranked
    - \[ AvgP = \frac{\sum_{i=1}^{m} i/\text{Pos}(i)}{m} \]
    - \[ MAP = \text{the mean value of } AvgP \text{ for all bug reports} \]
## Overall Effectiveness

<table>
<thead>
<tr>
<th>Subject</th>
<th>Approach</th>
<th>Top N (%)</th>
<th>MRR</th>
<th>MAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>N=1</td>
<td>N=5</td>
<td>N=10</td>
</tr>
<tr>
<td>Eclipse</td>
<td>BugLocator</td>
<td>29.4</td>
<td>52.9</td>
<td>62.8</td>
</tr>
<tr>
<td></td>
<td>BRTracer</td>
<td>32.6</td>
<td>55.9</td>
<td>65.2</td>
</tr>
<tr>
<td>AspectJ</td>
<td>BugLocator</td>
<td>26.5</td>
<td>51.0</td>
<td>62.9</td>
</tr>
<tr>
<td></td>
<td>BRTracer</td>
<td>39.5</td>
<td>60.5</td>
<td>68.9</td>
</tr>
<tr>
<td>SWT</td>
<td>BugLocator</td>
<td>35.7</td>
<td>69.3</td>
<td>79.5</td>
</tr>
<tr>
<td></td>
<td>BRTracer</td>
<td>46.9</td>
<td>79.6</td>
<td>88.8</td>
</tr>
</tbody>
</table>
## Effectiveness of Segmentation

<table>
<thead>
<tr>
<th>Subject</th>
<th>Approach</th>
<th>Top N (%)</th>
<th>MRR</th>
<th>MAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>N=1</td>
<td>N=5</td>
<td>N=10</td>
</tr>
<tr>
<td>Eclipse</td>
<td>BugLocator</td>
<td>29.4</td>
<td>52.9</td>
<td>62.8</td>
</tr>
<tr>
<td></td>
<td>Segmentation</td>
<td>30.5</td>
<td>54.2</td>
<td>64.0</td>
</tr>
<tr>
<td>AspectJ</td>
<td>BugLocator</td>
<td>26.5</td>
<td>51.0</td>
<td>62.9</td>
</tr>
<tr>
<td></td>
<td>Segmentation</td>
<td>31.1</td>
<td>54.5</td>
<td>67.1</td>
</tr>
<tr>
<td>SWT</td>
<td>BugLocator</td>
<td>35.7</td>
<td>69.3</td>
<td>79.5</td>
</tr>
<tr>
<td></td>
<td>Segmentation</td>
<td>45.9</td>
<td>76.5</td>
<td>85.7</td>
</tr>
</tbody>
</table>
Effectiveness of Stack-Trace Analysis

<table>
<thead>
<tr>
<th>Subject</th>
<th>Approach</th>
<th>Top N (%)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>N=1</td>
<td>N=5</td>
<td>N=10</td>
<td>MRR</td>
</tr>
<tr>
<td>Eclipse</td>
<td>BugLocator</td>
<td>29.4</td>
<td>52.9</td>
<td>62.8</td>
<td>40.7</td>
</tr>
<tr>
<td></td>
<td>Stack-Trace</td>
<td>31.8</td>
<td>54.8</td>
<td>64.0</td>
<td>42.8</td>
</tr>
<tr>
<td>AspectJ</td>
<td>BugLocator</td>
<td>26.5</td>
<td>51.0</td>
<td>62.9</td>
<td>38.8</td>
</tr>
<tr>
<td></td>
<td>Stack-Trace</td>
<td>37.4</td>
<td>59.0</td>
<td>68.5</td>
<td>47.7</td>
</tr>
<tr>
<td>SWT</td>
<td>BugLocator</td>
<td>35.7</td>
<td>69.3</td>
<td>79.5</td>
<td>50.2</td>
</tr>
<tr>
<td></td>
<td>Stack-Trace</td>
<td>38.7</td>
<td>72.4</td>
<td>81.6</td>
<td>53.3</td>
</tr>
</tbody>
</table>
Summary of Main Findings

Our approach is able to significantly outperform BugLocator

Either segmentation or stack-trace analysis is an effective technique

Segmentation and stack-trace analysis complement each other
RELATED WORK
Parallel Work


The two heuristics in our approach are different from all parallel work.
Comparison with L2R and BLUiR

• AspectJ
  – Better than L2R, Better than BLUiR

• SWT
  – Better than L2R, Worse than BLUiR

• Eclipse
  – Worse than L2R, Similar to BLUiR

The two heuristics are probably orthogonal to other heuristics, and can be combined
More Parallel Work

• Laura Moreno, John Joseph Treadway, Andrian Marcus, Wuwei Shen. On the Use of Stack Traces to Improve Text Retrieval-Based Bug Localization. ICSME 2014
• Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung and Sunghun Kim. CrashLocator: Locating Crashing Faults based on Crash Stacks, ISSTA 2014
• Shaowei Wang, David Lo, Julia Lawall. Compositional Vector Space Models for Improved Bug Localization. ICSME 2014
Thanks for your attention!

Code and data available at:
http://brtracer.sourceforge.net/