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

Chu-Pan Wong[1], Yingfei Xiong[1], Hongyu Zhang[2], Dan Hao[1], Lu Zhang[1], Hong Mei[1]

[1]Peking University

[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

Software Project Team

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

Bug ID	80720
Summary	**Pinned console** does not remain on top
Description	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**.

Corresponding source code file: **ConsoleView.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 $\times$ large file score + similar bug report score

Existing Problem 1

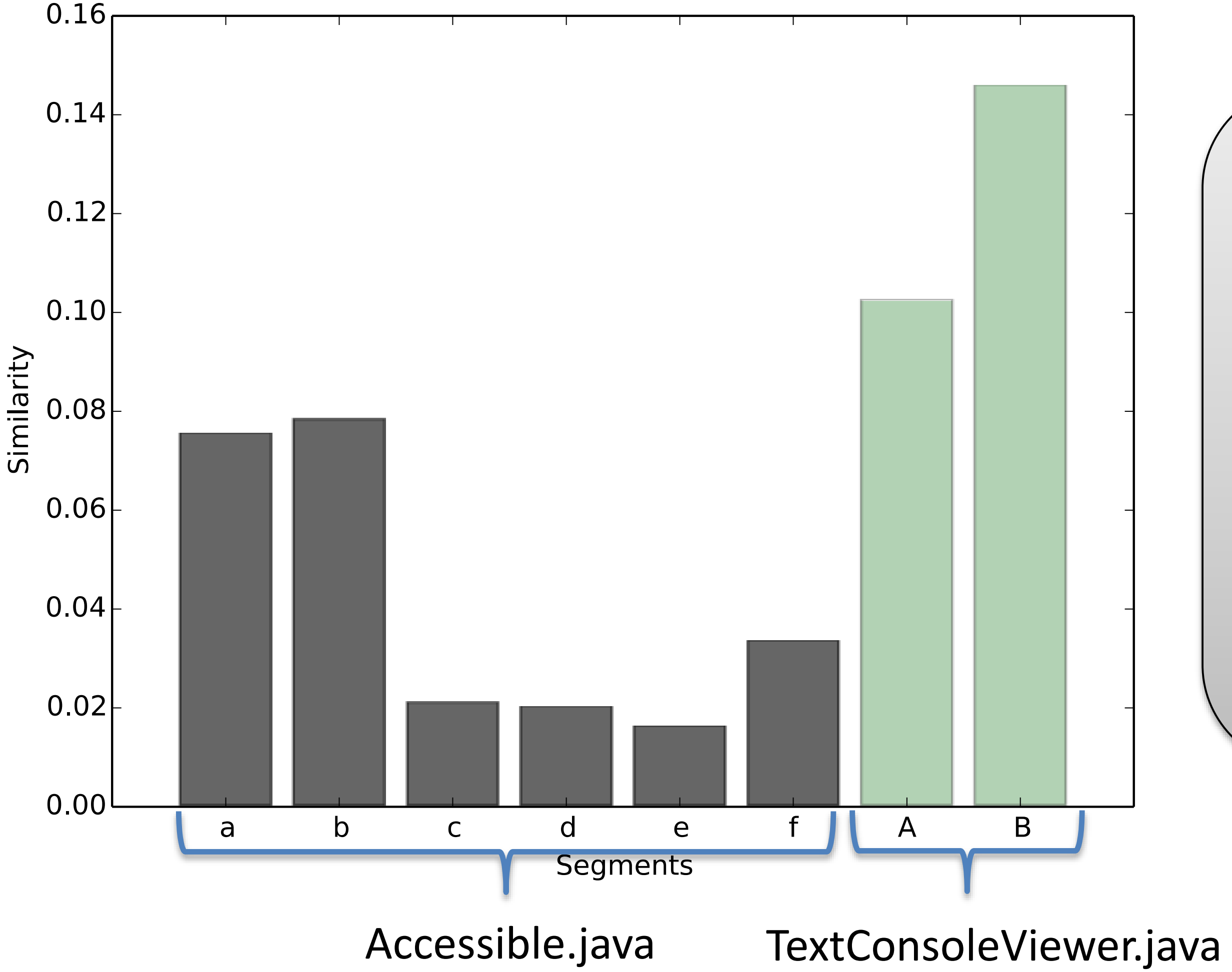
- Noise in large source code files
  - When file size changes, fault density may change more than an order of magnitude
  - BugLocator: large file score range from 0.5~0.73
  - Large files may have much noise

Motivation Example - Noise

Bug ID	87692
Summary	setConsoleWidth() causes Invalid thread access
Description	calling setConsoleWidth() outside UI thread causes Invalid thread access.

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

Motivation Example – Stack Traces

Bug ID	87855
Summary	NullPointerException in Table.callWindowProc
Description	Here is a stack trace I found when trying to kill a running process by pressing the "kill " button in the console view. I use 3.1M5a. !ENTRY org.eclipse.ui 4 0 2005-03-12 14:26:25.58 !MESSAGE java.lang.NullPointerException !STACK 0 java.lang.NullPointerException at org.eclipse.swt.widgets.Table.callWindowProc(**Table.java**:156) at org.eclipse.swt.widgets.Table.sendMouseDownEvent(**Table.java**:2084) at org.eclipse.swt.widgets.Table.WM_LBUTTONDOWN(**Table.java**:3174) at org.eclipse.swt.widgets.Control.windowProc(Control.java:3057) at org.eclipse.swt.widgets.Display.windowProc(Display.java:3480) at org.eclipse.swt.internal.win32.OS.DispatchMessageW(Native Method) at org.eclipse.swt.internal.win32.OS.DispatchMessage(OS.java:1619) at org.eclipse.swt.widgets.Display.readAndDispatch(Display.java:2539) at org.eclipse.ui.internal.Workbench.runEventLoop(Workbench.java:1612) at org.eclipse.ui.internal.Workbench.runUI(Workbench.java:1578) at org.eclipse.ui.internal.Workbench.createAndRunWorkbench (Workbench.java:293) at org.eclipse.ui.PlatformUI.createAndRunWorkbench(PlatformUI.java:144) at org.eclipse.ui.internal.ide.IDEApplication.run (IDEApplication.java:102) .... at java.lang.reflect.Method.invoke(Method.java:585) at org.eclipse.core.launcher.Main.invokeFramework(Main.java:268) at org.eclipse.core.launcher.Main.basicRun(Main.java:260) at org.eclipse.core.launcher.Main.run(Main.java:887) at org.eclipse.core.launcher.Main.main(Main.java:871)

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. isCommitable)
  - 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

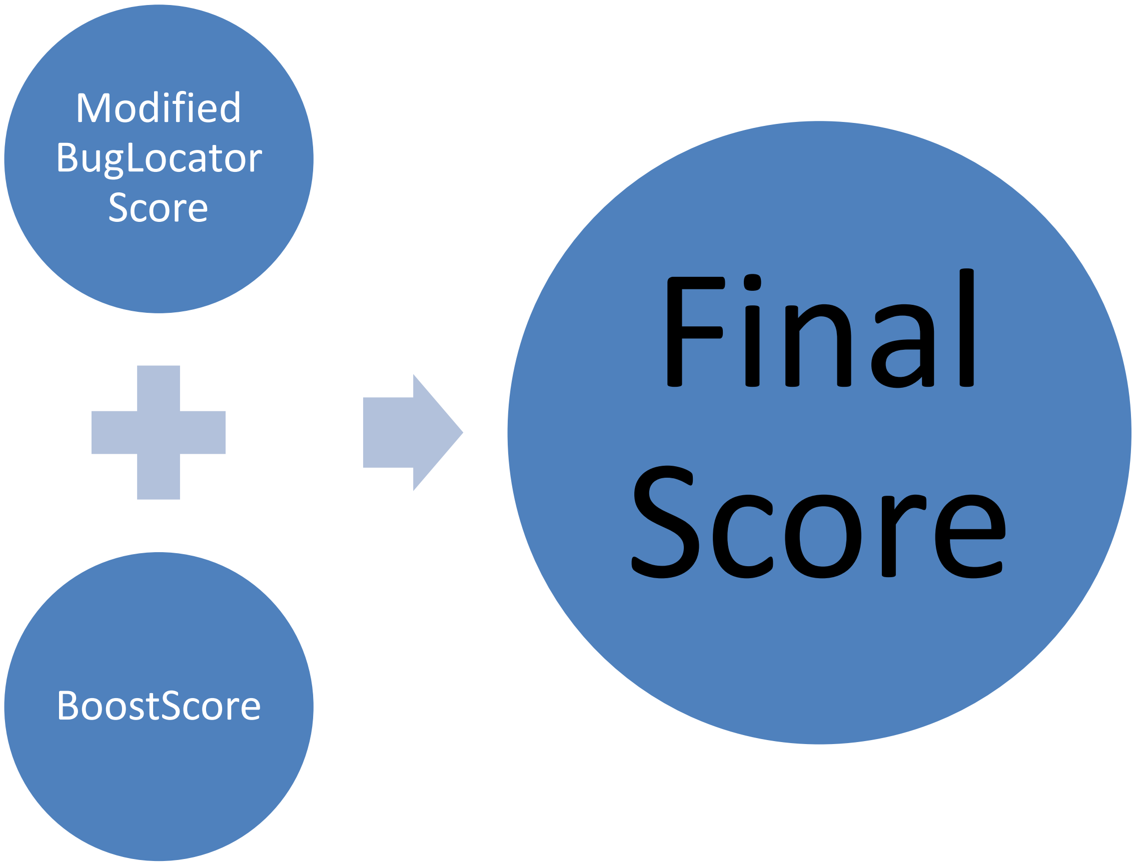
- $LargeFileScore(\#\text{terms}) = \frac{1}{1+e^{-\beta \times Nor(\#terms)}}$
- Function Nor normalize values to [0, 1] based on even distribution
- Parameter β in BugLocator is always 1
- Can be a larger number in our approach

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

$$BoostScore(x) = \begin{cases} \frac{1}{rank} & x \in \mathcal{D} \;\&\; rank \leq 10 \\ 0.1 & x \in \mathcal{D} \;\&\; rank > 10 \\ 0.1 & x \in \mathcal{C} \\ 0 & otherwise \end{cases}$$

Calculating Final Scores for Source Code Files



EVALUATION

Subjects and Parameters

Project	Studied Period	#Bug Reports	#Source Files
Eclipse 3.1	Oct 2004 - Mar 2011	3075	11892
AspectJ 1.5	Jul 2002 - Oct 2006	286	5487
SWT 3.1	Oct 2004 - Apr 2010	98	484

- Parameters
  - Segmentation Size n = 800
  - Large File Factor β=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
  - $MRR = \frac{\sum_{i=1}^{|BR|} 1/rank(i)}{|BR|}$
- Mean Average Precision (MAP)
  - How high all related files are ranked
  - $AvgP = \frac{\sum_{i=1}^{m} i/Pos(i)}{m}$
  - $MAP =$ the mean value of $AvgP$ for all bug reports

Overall Effectiveness

Subject	Approach	Top N (%)			MRR	MAP (%)
		N=1	N=5	N=10		
Eclipse	BugLocator	29.4	52.9	62.8	40.7	30.7
	BRTracer	32.6	55.9	65.2	43.4	32.7
AspectJ	BugLocator	26.5	51.0	62.9	38.8	22.3
	BRTracer	39.5	60.5	68.9	49.1	28.6
SWT	BugLocator	35.7	69.3	79.5	50.2	44.5
	BRTracer	46.9	79.6	88.8	59.5	53.0

Effectiveness of Segmentation

Subject	Approach	Top N (%)			MRR	MAP (%)
		N=1	N=5	N=10		
Eclipse	BugLocator	29.4	52.9	62.8	40.7	30.7
	Segmentation	30.5	54.2	64.0	41.6	31.1
AspectJ	BugLocator	26.5	51.0	62.9	38.8	22.3
	Segmentation	31.1	54.5	67.1	42.2	24.0
SWT	BugLocator	35.7	69.3	79.5	50.2	44.5
	Segmentation	45.9	76.5	85.7	58.2	51.6

Effectiveness of Stack-Trace Analysis

Subject	Approach	Top N (%)			MRR	MAP (%)
		N=1	N=5	N=10		
Eclipse	BugLocator	29.4	52.9	62.8	40.7	30.7
	Stack-Trace	31.8	54.8	64.0	42.8	32.5
AspectJ	BugLocator	26.5	51.0	62.9	38.8	22.3
	Stack-Trace	37.4	59.0	68.5	47.7	27.7
SWT	BugLocator	35.7	69.3	79.5	50.2	44.5
	Stack-Trace	38.7	72.4	81.6	53.3	47.2

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

- [L2R] X. Ye, R. Bunescu, and C. Liu, “Learning to rank relevant files for bug reports using domain knowledge,” in *Proc. FSE*, 2014, pp. 66–76.
- [BLUiR] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, “Improving bug localization using structured information retrieval,” in *Proc. ASE*, 2013, pp. 345–355.
- B. Sisman and A. C. Kak, “Assisting code search with automatic query reformulation for bug localization,” in *Proc. MSR*, 2013, pp. 309–318.
- T.-D. B. Le, S. Wang, and D. Lo, “Multi-abstraction concern localiza- tion,” in *Proc. ICSM*, 2013, pp. 364–367.
- C. Tantithamthavorn, A. Ihara, and K. ichi Matsumoto, “Using co- change histories to improve bug localization performance,” in *Proc. SNPD*, 2013, pp. 543–548.

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
- Ripon K. Saha, Julia Lawall, Sarfraz Khurshid, Dewayne E. Perry. On the Effectiveness of Information Retrieval Based Bug Localization for C Programs. ICSME 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/