Fault Localization Based on Natural Language Document Similarity

Zak Fry

Qualifying Exam Presentation

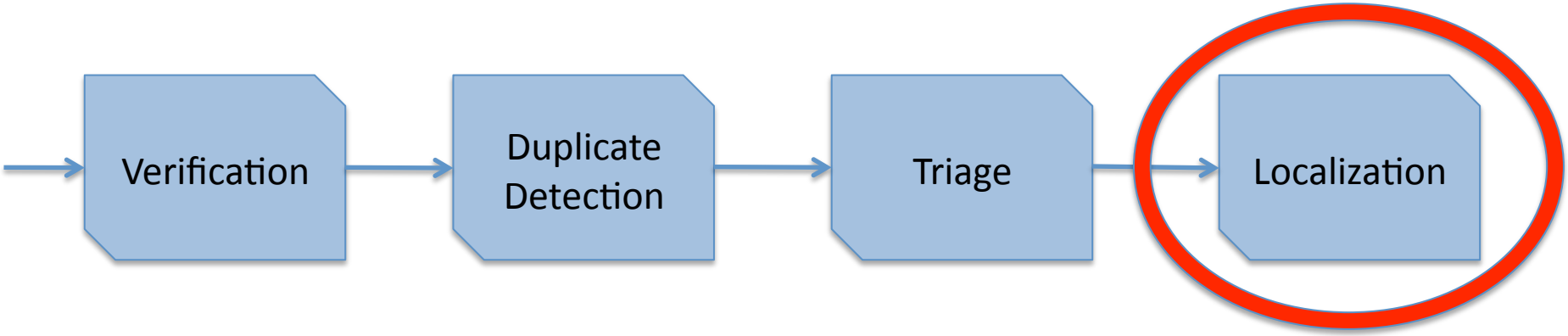
An Abundance of Bugs

- In 2005: “Every day almost 300 bugs appear that need triaging. This is far too much for the Mozilla programmers to handle”[1]
- From 2003-2005 half of all bugs fixed in Mozilla still took longer than 29 days to resolve
- Over 10% of submitted Mozilla bugs during this time still have not changed status from “new” or “unverified”

1. J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In International Conference on Software Engineering (ICSE), pages 361–370, 2006.

Fault Localization

- Thus, we want to speed up the bug resolution process



- Fault localization (FL): the task of identifying the files that could be changed to address the defect.

State-of-the-Art Fault Localization

- Testing
  - Jones et al. – Tarantula: Coverage based on failed/ passed cases
- Model Checking
  - Ball et al. – Coverage using SLAM counterexample and passing program runs
- Monitoring
  - Liblit et al. – Remote sampling and statistical bug isolation

Desired Solution Properties

- Use only limited static information
  - Require little additional developer effort
- Use lightweight, low overhead analysis
  - Time overhead consistent with current defect reporting rates
- Scale to large programs and diverse bug types

Motivating Example

Eclipse Bug 91543:
“Exception when placing a breakpoint (double click on ruler).”
With M6 and also with build I20050414-1107 i get the stacktrace below now and then when wanting the place a breakpoint when double clicking in the editor bar. if i close the editor and reopen it again it goes ok.

!MESSAGE Error within Debug UI: !STACK 0
org.eclipse.jface.text.BadLocationException at
org.eclipse.jface.text.AbstractLineTracker.getLineInformation(AbstractLineTracker.java: 251) ...

Actual Change:
ToggleBreakpointAction
RulerToggleBreakpointAction

Stack Trace:
AbstractLineTracker
AbstractDocument
AbstractRulerActionDelegate
PluginAction
PluginAction
AbstractTextEditor
AbstractTextEditor
TypedListener
EventTable
Widget
Display
Display
…

Motivating Example

Eclipse Bug 91543:
“Exception when placing a breakpoint (double click on ruler).”
With M6 and also with build I20050414-1107 i get the stacktrace below now and then when wanting the place a breakpoint when double clicking in the editor bar. if i close the editor and reopen it again it goes ok.

!MESSAGE Error within Debug UI: !STACK 0
org.eclipse.jface.text.BadLocationException at org.eclipse.jface.text.AbstractLineTracker.getLineInformation(AbstractLineTracker.java: 251) ...

Actual Change:
ToggleBreakpointAction
RulerToggleBreakpointAction

Eclipse Code Search:
Breakpoint
MethodBreakpointType
BreakpointsLocation
IChangeRulerColumn
BreakpointSpec
BreakpointEventImpl
MethodBreakpointSpec
RulerColumnBreakpoint
ClearAllBreakpoints
JavaExceptionBreakpoint
ValidBreakpointLocationLocator
TaskRulerAction
…

Motivating Example

Eclipse Bug 91543:
“Exception when placing a breakpoint (double click on ruler).”
With M6 and also with build I20050414-1107 i get the stacktrace below now and then when wanting the place a breakpoint when double clicking in the editor bar. if i close the editor and reopen it again it goes ok.

!MESSAGE Error within Debug UI: !STACK 0
org.eclipse.jface.text.BadLocationException
at
org.eclipse.jface.text.AbstractLineTracker.ge
tLineInformation(AbstractLineTracker.java:
251) ...

Actual Change:
ToggleBreakpointAction
RulerToggleBreakpointAction

Insights

- Developers, Users, Maintainers use different structure and language to describe the same concepts
  - Natural language information is encoded in both defect reports and source code
- Word frequency and semantic language similarities present but not always obvious

Thesis

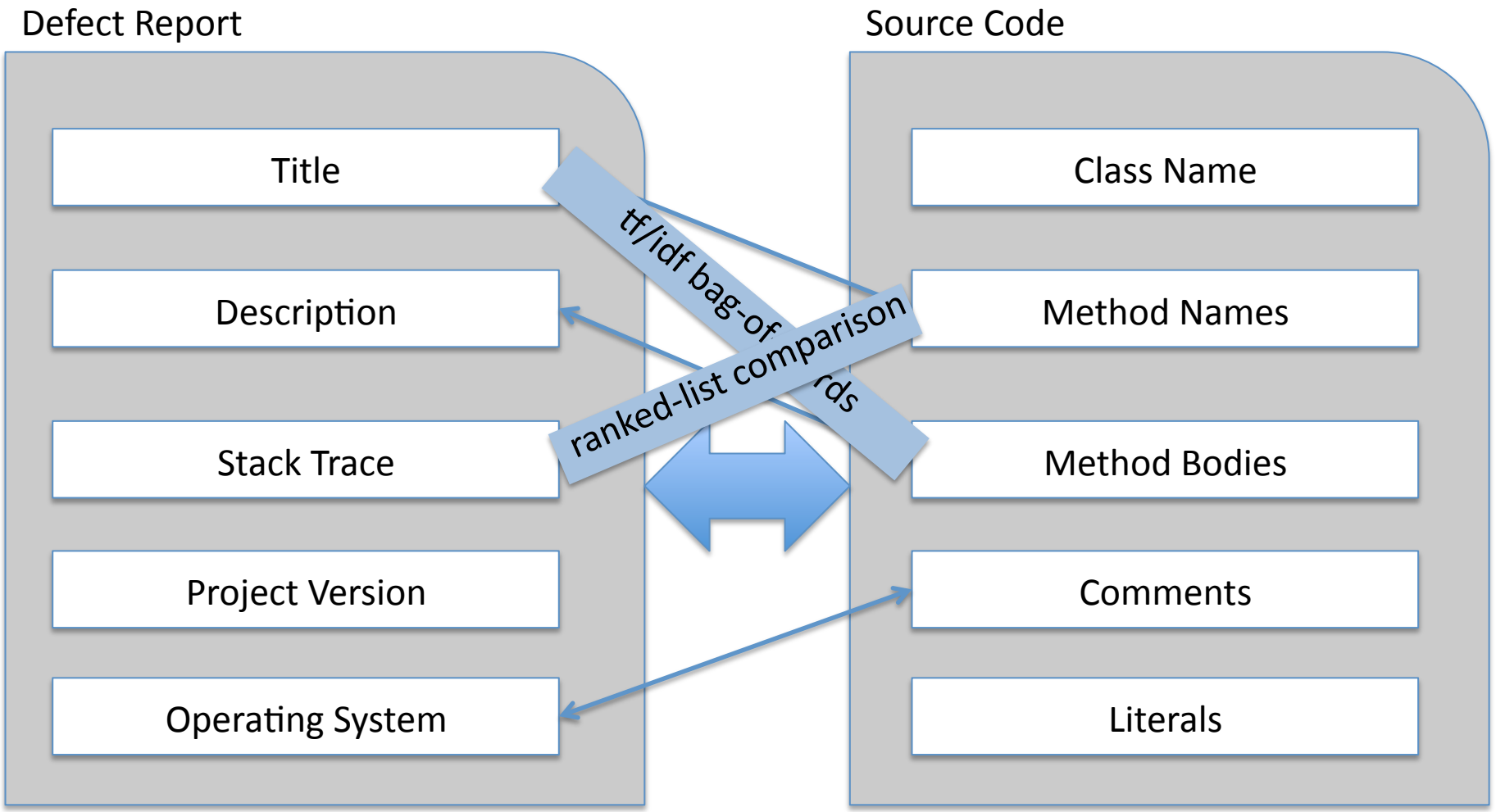
- Developers and users can experience and use the same concepts but will encode them differently (code vs. runtime behavior)
- We hypothesize that:
  - We can construct a model that exposes the mapping between the two types of documents such that we can locate faults at least as accurately as state-of-the-art tools on indicative benchmarks while limiting overhead
  - Our model's success is explained by natural language information and not mitigating non-natural-language features

Proposed Algorithm Structure

- Input: Bug report(s) and the source code
- Basic Approach: develop specific document substructure similarity metrics that can be combined in a model to localize faults accurately
- Output: Ranked list of files in terms of likelihood of containing the bug in question

Methodology

- Structured Document Comparison



Feature Selection

- ANOVA of all possible document comparisons
  - For each bug: used all correct files and 150 distractors
  - Used only as a starting point – not a purely linear model
- Principle Component Analysis
  - Showed that a combination of 12 accounted for 99% of the overall variance in the data
  - Compared model using all features and one with the top 12 from the ANOVA to verify both steps

Feature Selection

- Final parameter space optimization via hill climbing
  - Started with ANOVA F-values and varied by +/- 10% each iteration
  - Stopped after 5 iterations when overall accuracy changed by less than 0.01%

Model Features

Report Substructure	Code Substructure	Relative Weight
Report title	Method bodies	23.06
Report body	Method signatures	41.48
Report title	Comments	10.53
Report body	Class name	9.46
Report body	Comments	7.89
Stack trace	Class name	7.79
Report body	Method bodies	5.72
Component	Method bodies	4.30
Operating System	Comments	3.48
Component	Comments	3.03
Product	String literals	1.94
Report title	Method signatures	1.32

Evaluation

- Eclipse, Mozilla, and OpenOffice
  - 5345 confirmed bug reports with known fixes based on CVS check-in messages
- 48k files, 65mil LOC
  - Files totaling 6.5mil LOC involved in user fixes
- Closest previous work (Tarantula) used only 7 small programs comprising 2,557 LOC total

Score Metric

- Used by Jones et al. to compare against Cleve and Zeller, Renieris and Reiss
- Represents the number of files that can be eliminated from the overall search space
- Given our ranked list of 10,000 files and 1 faulty file: if the correct file appears as number 2,000:

  (10,000 – 2,000) / 10,000 = 80%

Experiment 1 – "Score" Accuracy

- Compare against two logical baselines
  - Stack traces
  - Code churn
- Indirectly compare against the reported results of three state of the art tools
  - Incompatible benchmark requirements
- This work is successful if we are more accurate than the simple baselines and do comparably to existing tools while limiting overhead

Experiment 1 - Results

Set of Defects	Reports Used	Our Approach	Stack trace Baseline	Code churn Baseline
Reports from Openoffice	1040	67.9%	60.0%	72.8%
Reports from Eclipse	1272	86.9%	56.3%	73.1%
Reports from Mozilla	3033	92.2%	50.1%	93.9%
Reports with Stack traces	325	86.2%	65.1%	76.4%
All reports	**5345**	**88.2%**	**53.1%**	**84.8%**

Technique	Score
Renieris and Reiss	56.000%
Cleve and Zeller	63.415%
Jones et al. (Tarantula)	77.797%
Our approach	**88.193%**

Experiment 2:
Natural Language Validation

- We measure the correlation of non-natural-language features and score
- Then we systematically degrade the natural language information and measure performance degradation to confirm
- This work is successful if correlations with extraneous features are low and performance degrades proportionally to word replacement

Experiment 2 – Results

- All non-natural-language features had correlations with our Score measures of less than .15
- Degraded language by
  - Replace from same corpus
  - Replace from a dictionary
  - Replace with random characters
- In all cases, performance decreased monotonically
- Overall, word choice accounts for 10% Score
  - Explains some of the increased performance over both baselines and existing work

Performance

- Parsing all code after build releases
  - Under an hour for all projects
- Once indexed, read index files
  - Under a minute for all projects
- Create a ranked list for an incoming report
  - Under 10 seconds for all instances tested
- Expected use case: maximum of 1min 10sec per incoming report

Conclusion

- H1: We can build a static model that achieves higher score accuracy than existing tools when localizing faults on large indicative benchmarks

- H2: The model's success is due to human chosen natural language inherent in defect reports and source code