
# *A Static Technique for Bug Localization Using Character N-Gram Based Information Retrieval Model*

***Sangeeta***

Indraprastha Institute of Information Technology, Delhi (IIIT-D), India

Thesis Advisor: Dr. Ashish Sureka

# Presentation Outline

- Bug Localization
- Research Motivation
- Literature Survey
  - Static Vs Dynamic
  - Advantages of Static
  - Traditional approaches (IR based bug localization)
- Proposed Solution
  - Research aim & contribution
  - Character level N-Gram model
  - Example illustrating advantages of character N-Gram
  - Similarity function
- Empirical Evaluation
  - Evaluation dataset
  - Performance metrics
  - Results
- Conclusions & Future Work

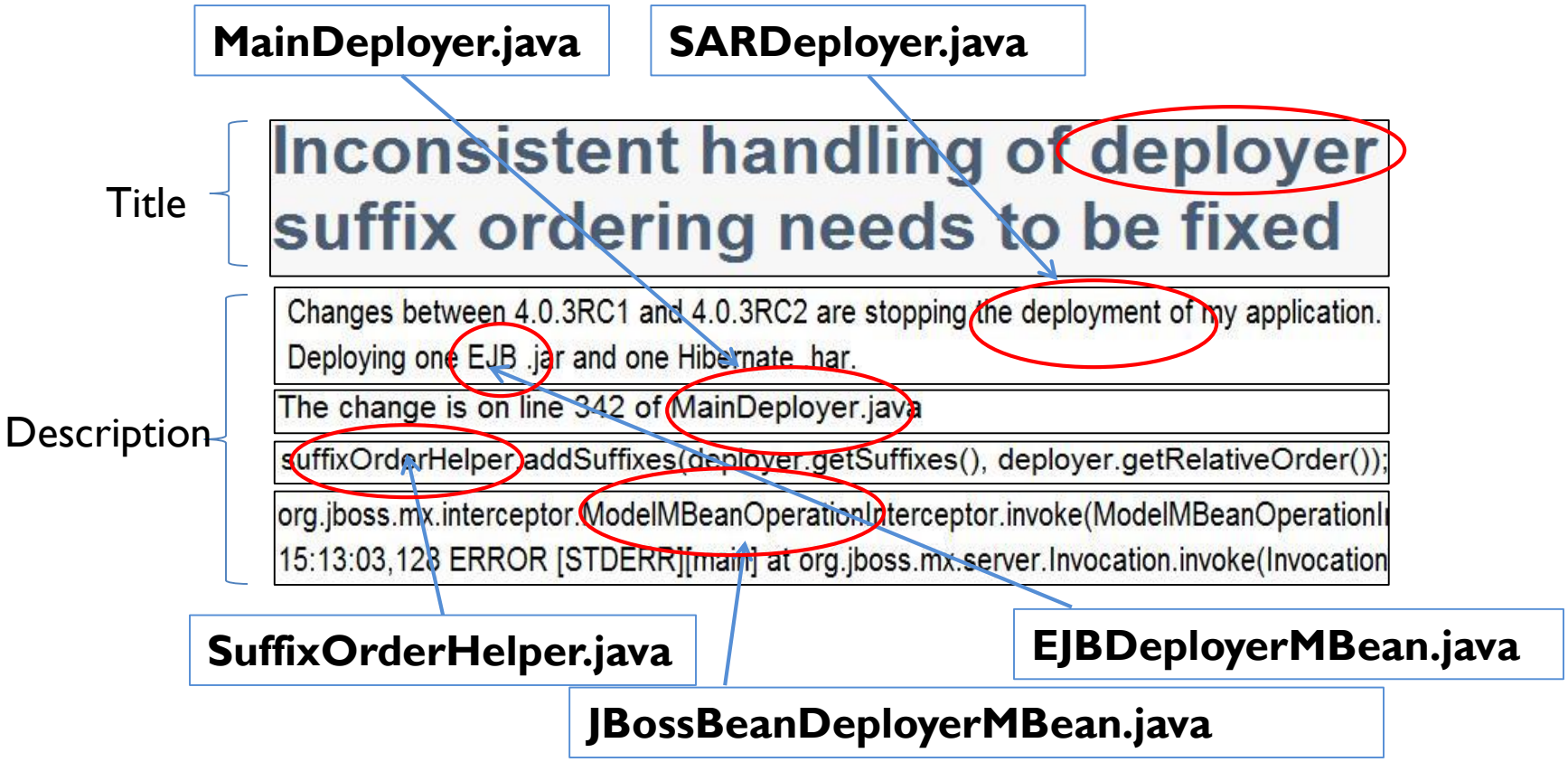
# Bug Localization

# Bug Localization

- *"Bug localization is a process of identifying the specific location(s) or region(s) of source code (at various granularity levels such as the directory path, file, method or statement) that is faulty and needs to be modified to repair the defect"*

# Example

- Bug localization at file level granularity



# Research Motivation

# Research Motivation

- Huge Number of bugs to fix:
  - Windows 2000 had about 63, 000 known bugs at its time of release, 2 bugs per 1000 lines
  - Mozilla project received 300 bug per day (year 2005) and total of 51, 154 bugs in 4 year (2002-2006)[1]

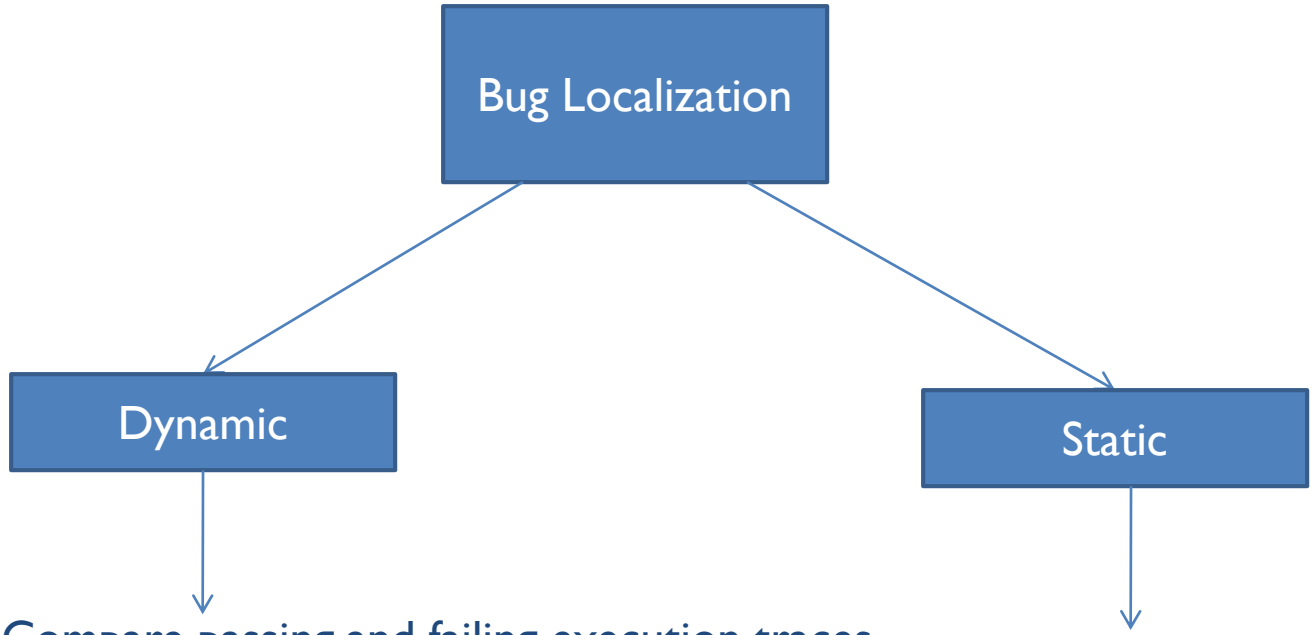
- Testing and debugging are laborious and expensive
  - Exhaustive testing not possible because of time and resource limitations
  - Corrective maintenance cost 21% of overall maintenance
  - Software faults cost the U.S economy about $59.5 billion annually[2]



[1] Anvik et al., Automating bug report assignment, 2006.
[2] http://www.nist.gov/director/prog-ofc/report02-3.pdf

# Literature Survey

# Literature Survey




- Compare passing and failing execution traces
- Hard to apply in real world defect tracking system

- Collect information from source code (or source code model)
- Don't require execution information
- Can be applied at any stage of software development

# Literature Survey…

| | Study | Bug Reports | Source Code | Granularity | IR Model | Case Study |
|---|---|---|---|---|---|---|
| 1 | [Chen2008] | Meta Information and Description | Source File and bug report social network | --------- | Page Rank, Vector Space Model | SVN, AgroUML |
| 2 | [Lukins 2008] | Manual extraction of keywords from Title and Description | Comments, Identifiers and String-Literals | Method Level | LDA Based | Rhino, Eclipse and Mozilla |
| 3 | [Fry 2010] | Title, Description, Stack Trace, Operating System | Class Name, Method Names, Method Bodies, Comments, Literals | File Level | Word-based cosine similarity vector comparison | Eclipse Mozilla OpenOffice |
| 4 | [Khansari 2010 ] | Title and Description | Source File Path | Path Level | SVM Classifier | Eclipse |

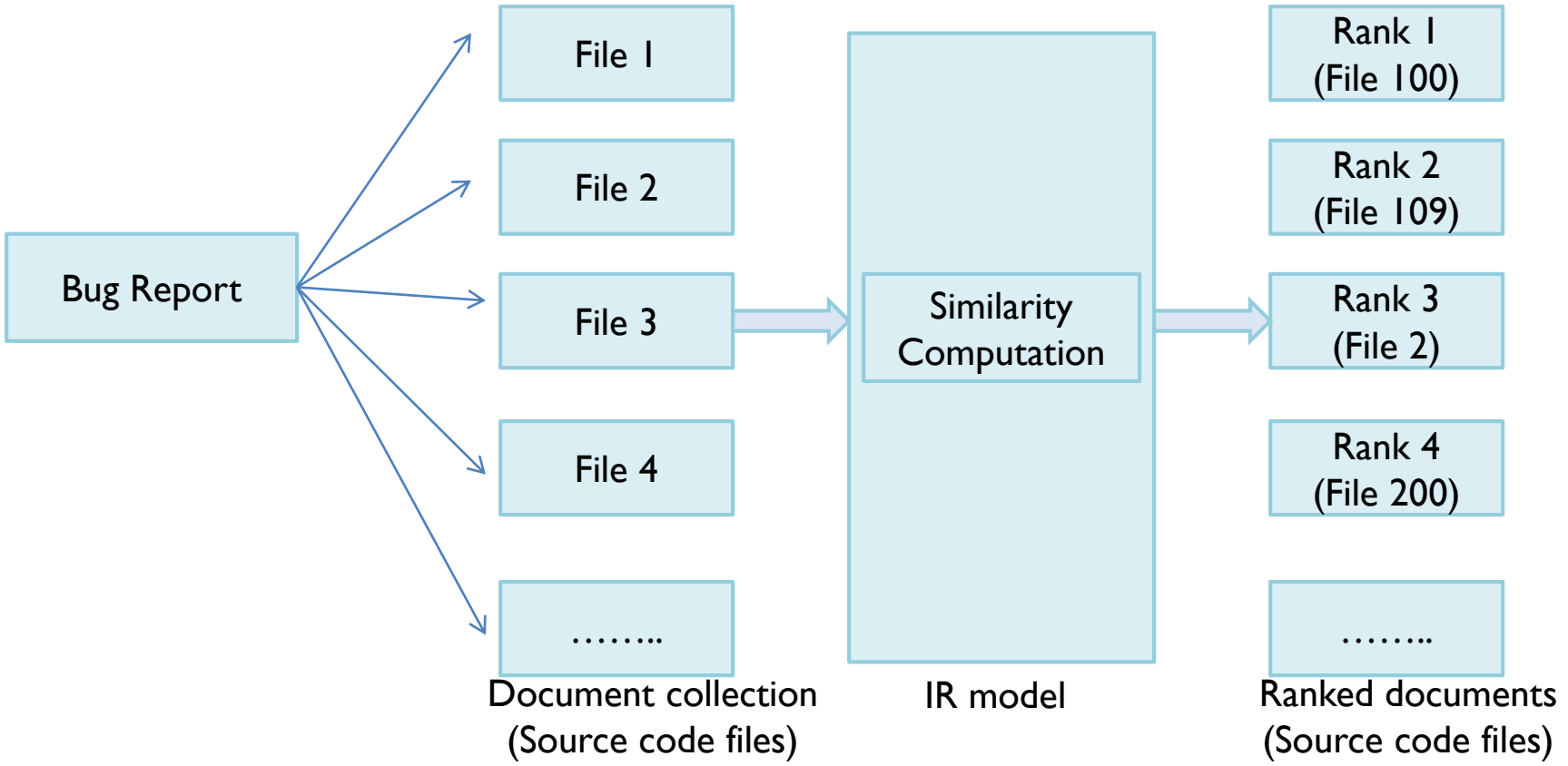
# Literature Survey…

| | Study | Bug Reports | Source Code | Granularity | IR Model | Case Study |
|---|---|---|---|---|---|---|
| 5 | [Canfora, 2005] | Title and Description | Revision comments | File Level | Probabilisticmodel described in Jones [2000] | Mozilla Firefox and KDE (kcalc, kpdf, kspread,koffice) |
| 6 | [Nichols, 2010] | Bug Description of old and new bugs | Identifier names, string literals | Method Level | LSI Based | Rhino |
| 7 | [Antoniol, 2000] | Maintenance requests (change log files version control) | Source code or higher level documents (Specifications) | Class level | Vector space model and stochastic language model | LEDA C++ Class Library |
| 8 | [Marcus, 2005] | change requests | Source code | Class level | Latent semantic indexing, dependency search | AOI 3D modeling studio, Doxygen |

# Research Aim

# Research Aim

- To investigate text analytics based solution for automated bug localization



# Research contribution (main idea and key points)

# Research contribution (main idea and key points)

- A novel method for "*automated bug-localization*" using "*character N-Gram based IR model*"

- An empirical evaluation of proposed solution on real world and publically available datasets

# Character N-Gram Model

# Character N-Gram Model

- N-Gram means subsequence of N contagious items within sequence of items
  - Word N-Gram represents sequence of words
  - Character N-Gram represents sequence of characters

  - Example (word N-Gram): *Mining Software Repositories Using Ngram*
    - bi-grams (N=2): *Mining Software, Software Repositories, Repositories Using, Using Ngram*
    - tri-grams (N=3): *Mining Software Repositories, Software Repositories Using, Repositories Using Ngram*

  - Example (Character N-Gram): *Software*
    - bi-grams (N=2): *So, of, ft, tw, wa, ar, re*
    - tri-grams (N=3): *Sof, oft, ftw, twa, war, are*

# Intuitions of Using Character N-Gram

- Ability to handle misspelled words
  - (Managment, Management),
  - (Periode, Period)
- Ability to match short forms with expanded forms
  - (Config, Configuration), (Auth, Authentication), (repl, replication), (impl, implementing), (Spec, Specs, Specification),
- Ability to match term variants to common root
  - Call: (Caller, Calling),
  - Manage: (Manager, Managed)
- Ability to match words joint using special characters
  - [TimerImpl], TimerImpl
  - JaasSecurityManagerService, ("jboss.security:service= JaasSecurityManager")

# Intuitions of Using Character N-Gram…

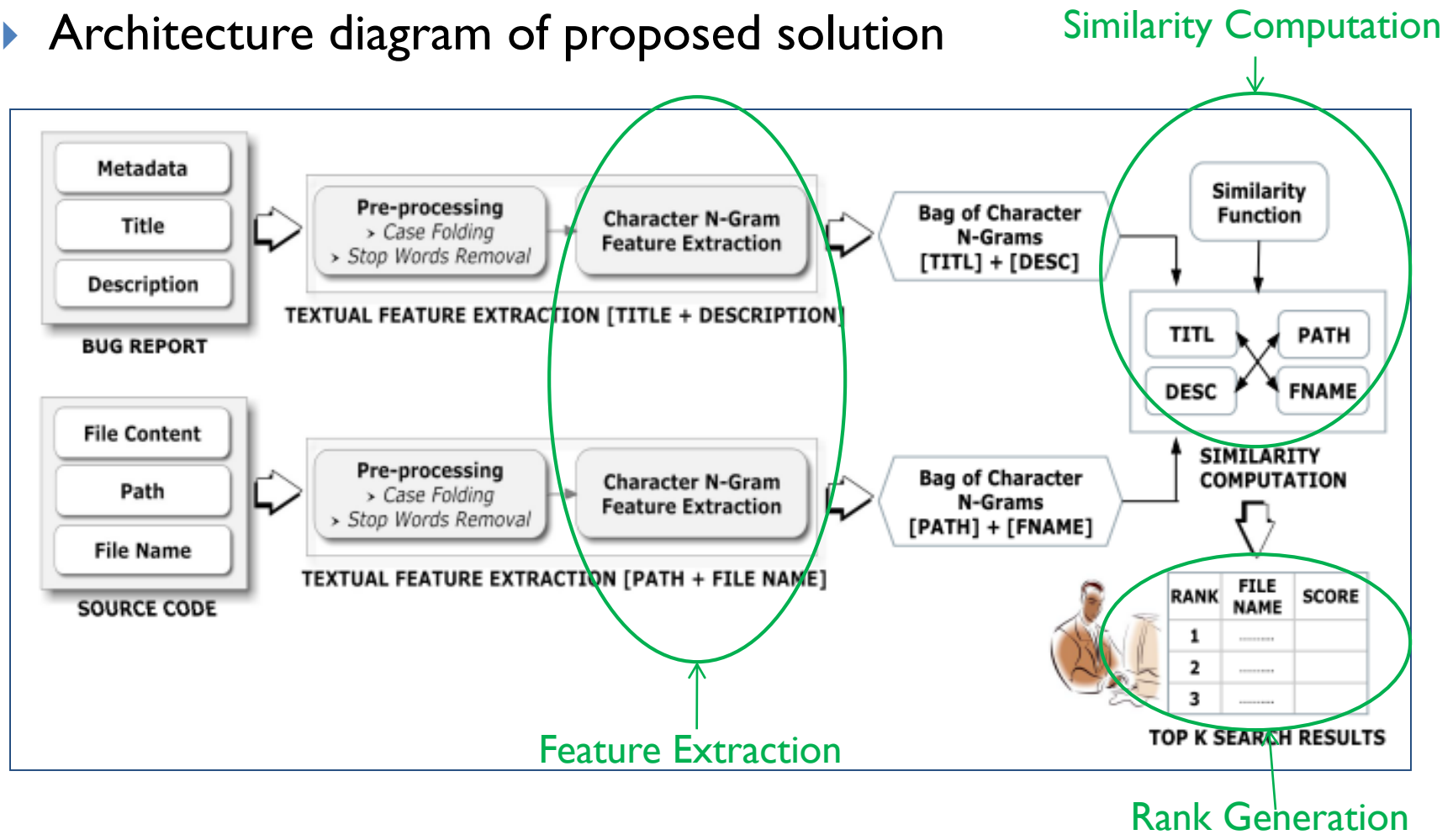
- Ability to match concept present in source code
  - "JRMPInvokerProxyHA", "invoker HA proxies"

- Ability to match concept permutations
  - "JMSManagedConnection" and "JMSConnectionManager"
  - After splitting followed by stemming of these two strings, a word level approach will get substring "jms", "connection", "manager"
  - Word based approach represented as "bag- of- words" can not differentiate between these two

# Solution Approach

# Solution Approach

- Architecture diagram of proposed solution



# Solution Approach

- Feature Extraction
  - Character N-Grams of size (3-10)
- Similarity Function
  - U,V: bag- of - Character N-Grams

$$SIM(U,V) = \sum_{u \in U} \sum_{v \in V} Match(u,v) \times Length(u)$$

- Rank generation
  - Tuning parameters (weight) : ($W_1$ ,$W_2$ ,$W_3$ ,$W_4$ )

$$SIM_{SCORE}(BR,F) = W_1 * SIM(T\text{-}F) + W_2 * SIM(T\text{-}P) + W_3 * SIM(D\text{-}F) + W_4 * SIM(D\text{-}P)$$

# Empirical Evaluation

# Empirical Evaluation

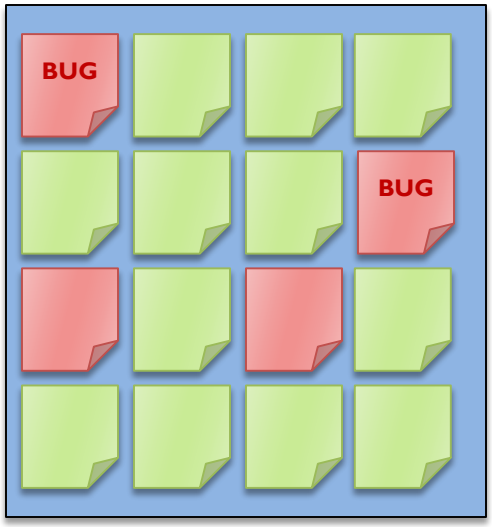
- Evaluation data set

| | | JBOSS | Apache |
|---|---|---|---|
| A | Start Issue ID | JBAS-1 | GERONIMO-1 |
| B | Last Issue ID | JBAS-8879 | GERONIMO-6143 |
| C | Number of Issue Reports Downloaded | 8072 | 4092 |
| D | Number of Issue Reports Unable to Download | 807 | 2051 |
| E | Number of Issue Reports(Status =CLOSED/ RESOLVED, Resolution = DONE and containing SVN commit) | 2433 | 1915 |
| F | Number of Issue Reports of Type = BUG AND in SET E | 1114 | 1269 |
| G | Number of Issue Reports in SET F AND containing at least one modified Java file | 1090 | 939 |
| H | Number of Issue Reports in SET G AND a specific version | 576 (VERSION =4.x) | 736 (VERSION = 1.x, 2.x, 3.x) |
| **I** | **Number of Issue Reports in SET H AND for which MODIFIED file Found in specified version** | **569** | **637** |

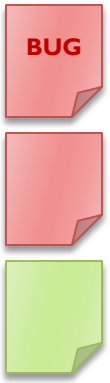
# Empirical Evaluation…

- Data set contains ground-truth
- Performance metrics
  - Score
  - Mean Average Precision (MAP)
  - Rank

# Performance Evaluation Metrics[Score]




*Search Space*




File predicted and impacted [*True Positive*]

File predicted and not impacted [*False Positive*]

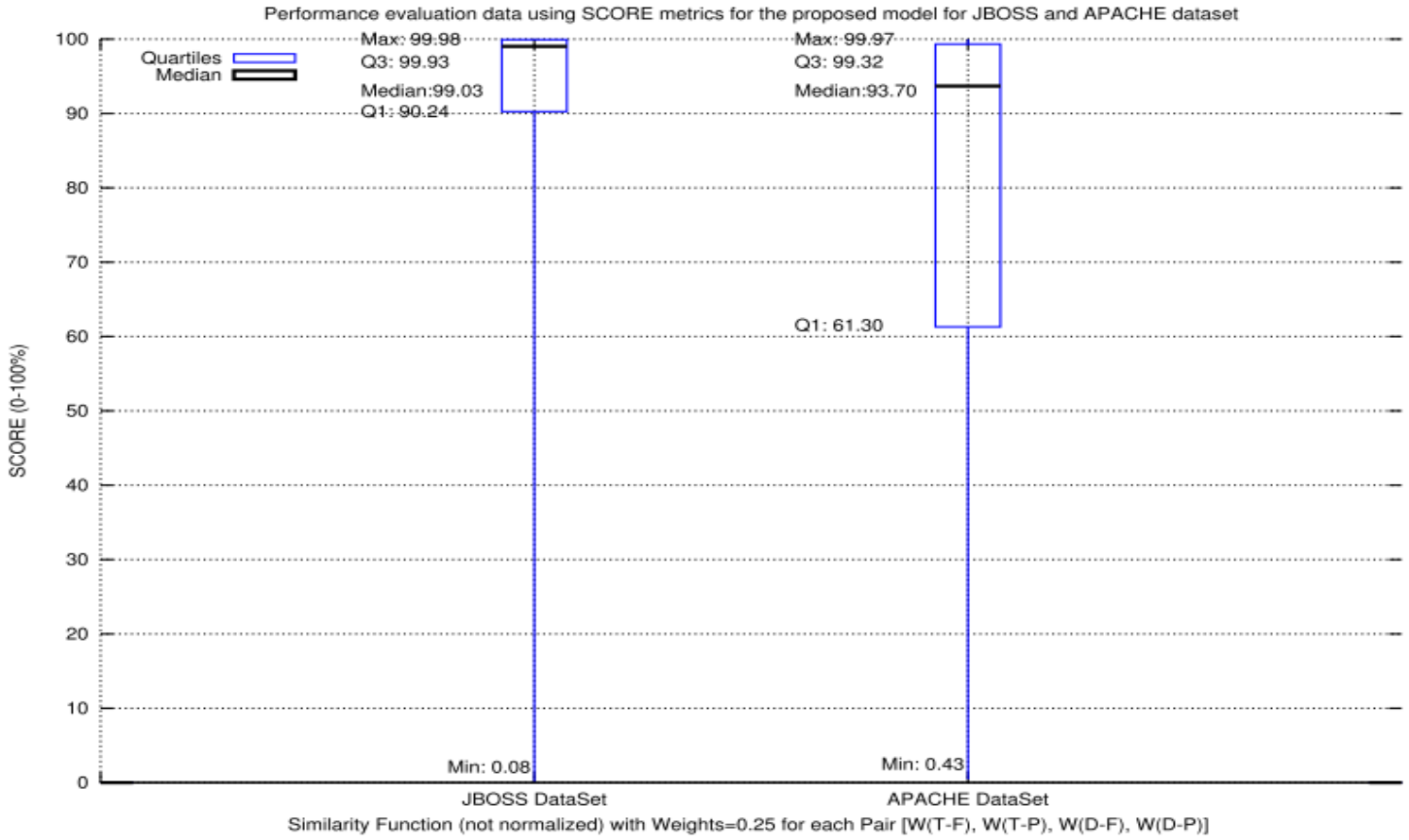
File not predicted and not impacted [*True Negative*]

Score = (1 - 4/16)*100 = 75%

- Score = Percentage of files in the source-code that need not be investigated to search the bug
- In the above example: the bug-fixer needs to inspect 4 files (out of 16) to find the bug

# Results[Metric-Score]

- JBOSS(Median Score): 99.03%
- Apache(Median Score): 93.70%



# Performance Evaluation Metrics[MAP]

| Rank | File No | Relevant |
|---|---|---|
| 1 | 549 | Y |
| 2 | 345 | |
| 3 | 567 | Y |
| 4 | 234 | |
| 5 | 123 | Y |
| 6 | 567 | |
| 7 | 345 | |
| 8 | 589 | Y |

Let total # of relevant docs = 4

R=1/4=0.25; P=1/1=1

R=2/4=0.4; P=2/3=0.667

R=3/4=0.75; P=3/5=0.6

R=4/4=1; P=4/8=0.5

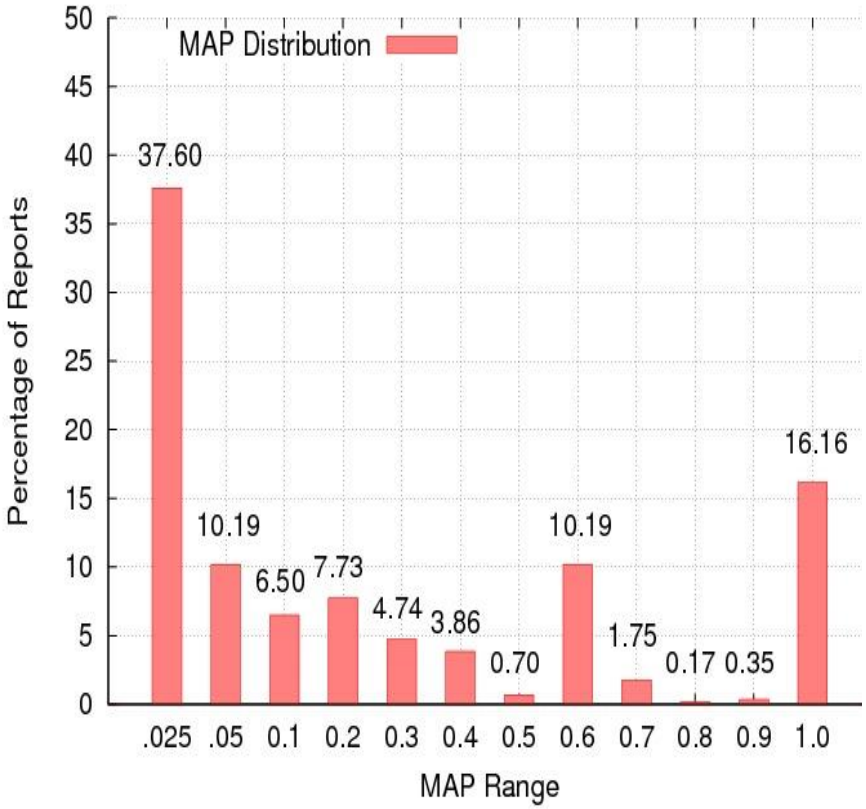
Average Precision(AP)=(1+0.667+0.6+0.5)/4

Mean Average Precision(MAP)= $\sum (AP_i)/N$

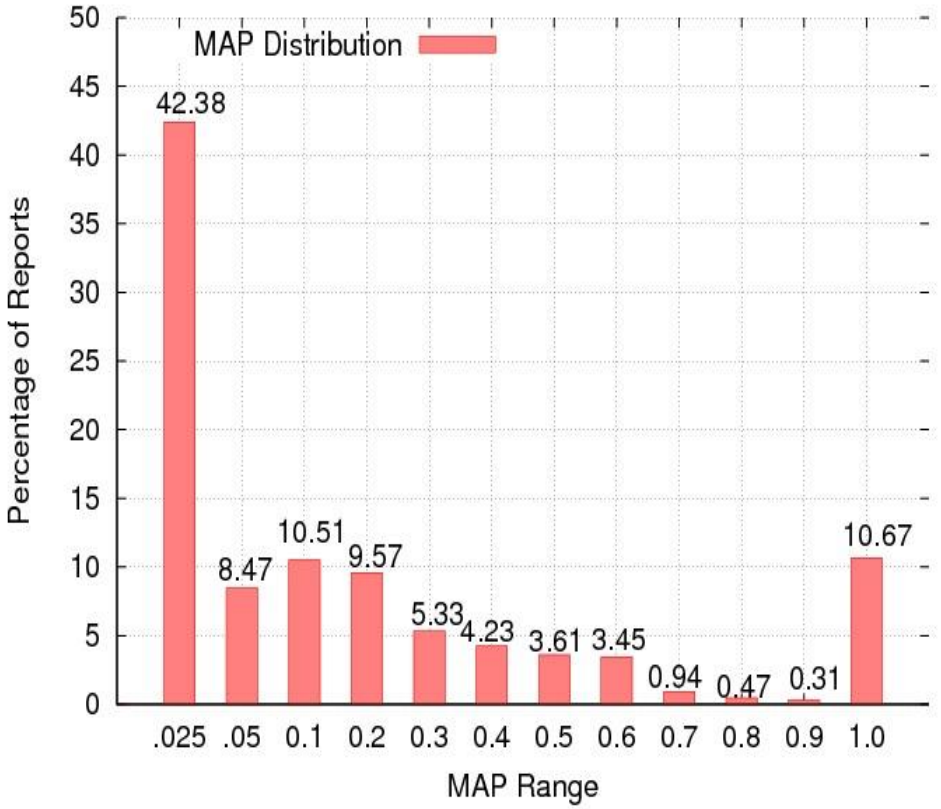
- N = No. of bugs

# Results [Metric-MAP]

- JBOSS(0.9-1.0): 16.16% (Bug report)
- Apache(0.9-1.0): 10.67% (Bug report)



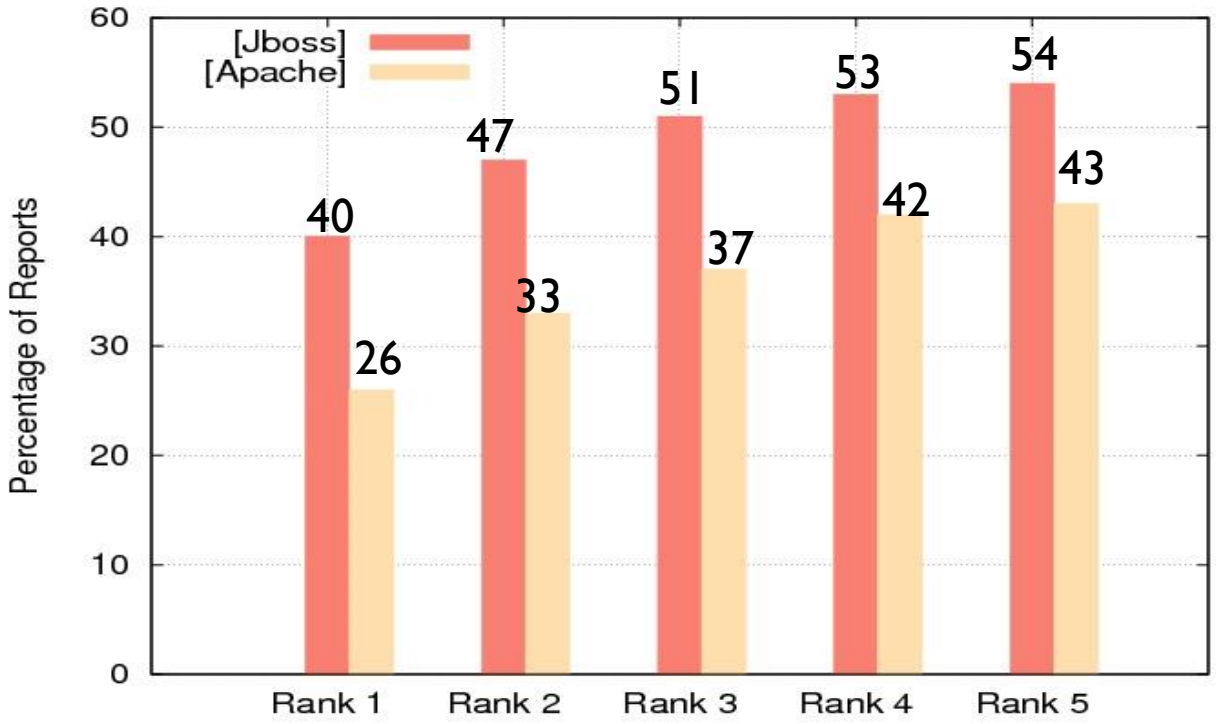
JBOSS



Apache

# Performance Evaluation Metrics[Rank]

- Percentage of bug for which at least one of the relevant retrieved at specified rank
- Rank 1:
  - JBOSS: 40%
  - Apache:26%



# Results (Predictive Power of Each Attribute)

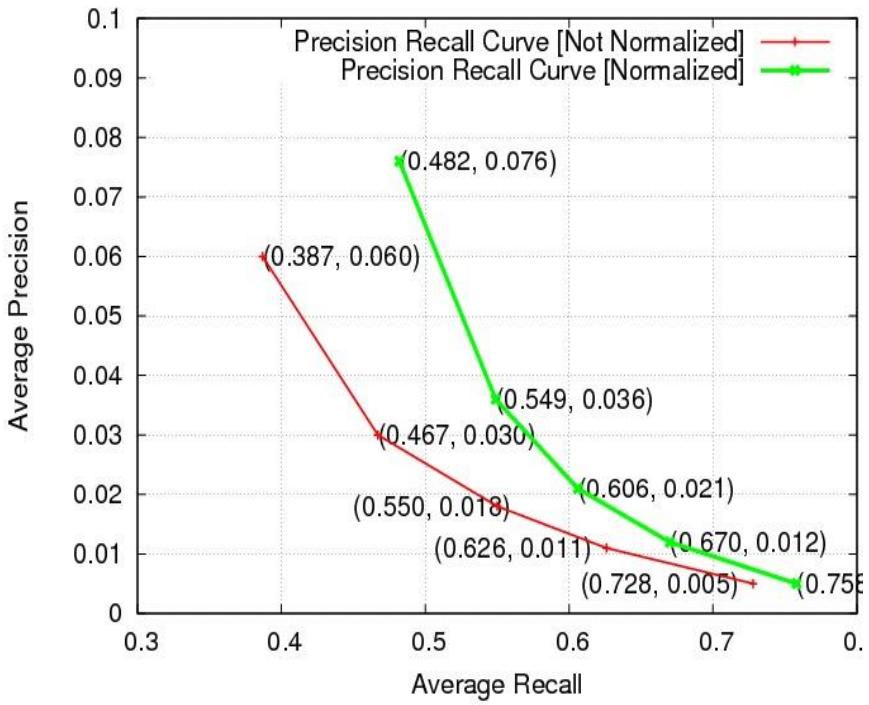
- Predictive power of each attribute (title Vs description)
  - Score: JBOSS, Apache
  - Description is better than title for bug localization (using *Score* metrics)

| Attribute | JBOSS(SCORE Median) | Apache (SCORE Median) |
|---|---|---|
| Title-File Name | 94.77 | 74.36 |
| Title-File Path | 85.56 | 78.92 |
| Description-File Name | 97.03 | 82.81 |
| Description-File Path | 94.08 | 83.90 |

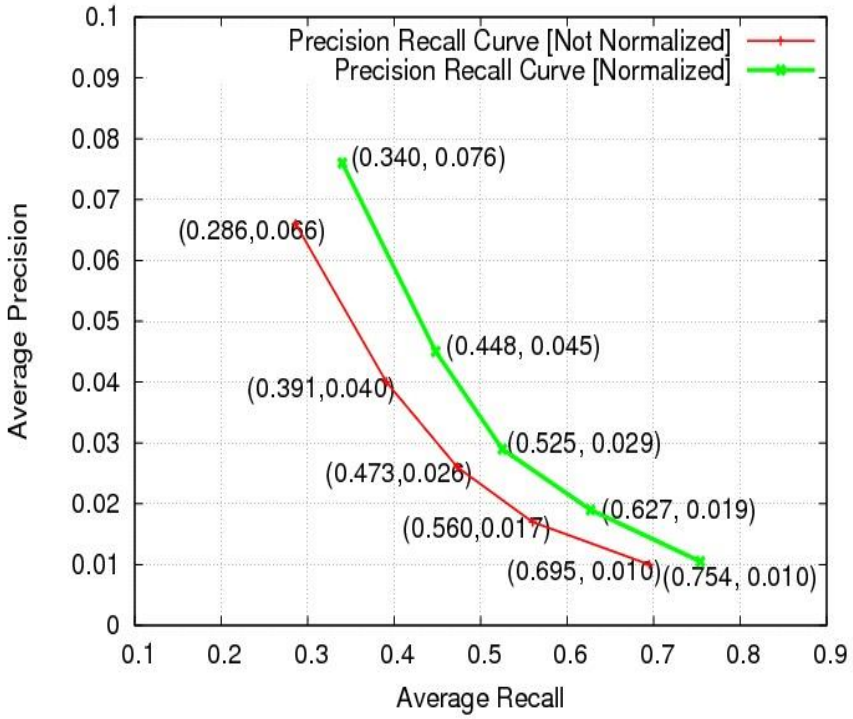
# Results (Normalization Effect)

- Normalization improved bug localization precision and recall

$$SIM(U,V) = \frac{\sum_{u \epsilon U} \sum_{v \epsilon V} Match(u,v) \times Length(u)}{|u| \times |v|}$$




JBOSS




Apache

# Low Performance on Apache?

- Project "Geronimo" have lots of external dependencies such as "Tomcat"
- Even if bug is in tomcat it is sometimes reported in Geronimo
- Currently hard to detect using textual similarity

# Advantages & Disadvantages of Proposed System

- Advantages of the proposed system
  - Simple to apply
  - Partial match without information loss
  - No need to train
  - Language independent

- Disadvantages of the proposed system
  - Scalability issue
  - False positives are reported due to partial match

# Conclusions

# Conclusions

- Proposed an IR based static technique for bug localization to assist developer in bug fixing
- Central idea: application of character N-Gram as low level feature
- Character N-Gram based technique is effective in bug localization
  - Evaluation dataset: JBOSS and Apache
    - MAP(0.9-1.0): 16.16%(JBOSS), 10.67%(Apache)
    - Score (median): 99.03%(JBOSS), 93.70%(Apache)
    - Rank (rank1): 40% (JBOSS), 26% (Apache)
- Description is better than title for bug localization
- Normalization increases bug localization precision and recall

# Future Work

# Future Work

- To propose this technique as a base line approach for this dataset
- Including more features to measure effect on bug localization accuracy

# Publications

- S.Lal, A.Sureka, A Static Technique for Bug Localization Using Character N-Gram Based Information Retrieval Model, 5th India Software Engineering Conference , ISEC 2012 (Submitted)

- A. Sureka, S. Lal, L. Agarwal , Applying Fellegi-Sunter (FS) Model for Traceability Link Recovery Between Bug Databases and Version Archives, The Eighteenth Asia-Pacific Software Engineering Conference (APSEC 2011)

# Thank You