5 Papers

Title	Publication source	Year
Topic Modeling for Feature Location in Software Models: Studying Both Code Generation and Interpreted Models	IST	2021
Changeset-Based Topic Modeling of Software Repositories	TSE	2020
DeepFL: Integrating Multiple Fault Diagnosis Dimensions for Deep Fault Localization	ISSTA	2019
On Combining IR Methods to Improve Bug Localization	ICPC	2020
Bug Localization Using Latent Dirichlet Allocation	IST	2010

Bug Localization Using
Latent Dirichlet Allocation

汇报人：王昭丹

CONTENTS

INTRODUCTION

Static bug localization

- gather information from the source code (or a model of the code)

1

Dynamic bug localization

- gather information from execution traces of the system

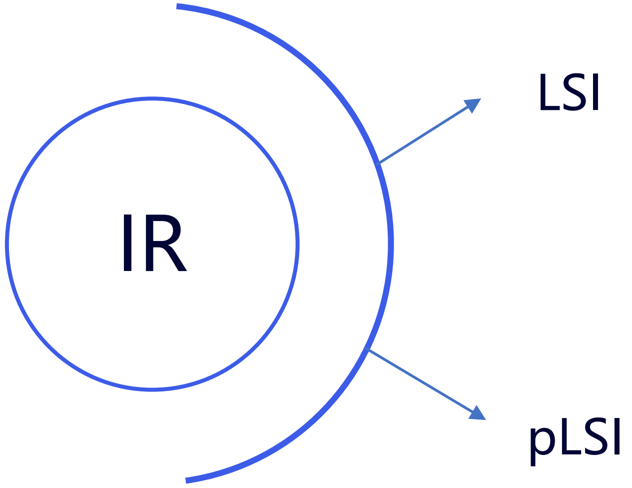
2

Advantages

Do not require a working subject software system

Can be applied at any stage of the software development or maintenance processes

INTRODUCTION



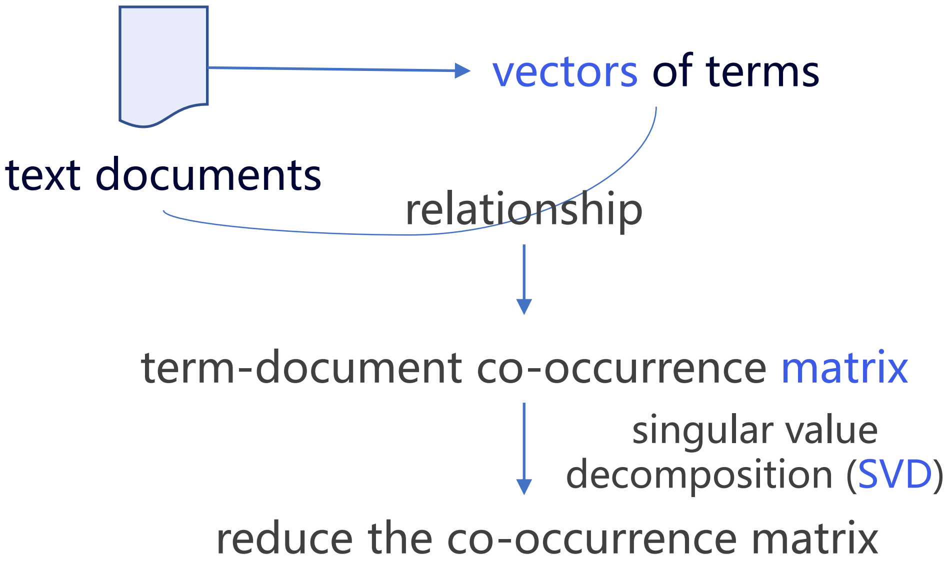
LDA-based static technique

- Latent Dirichlet Allocation (LDA)
- Modularity and extensibility
- Provide advantages over both LSI and pLSI
- Stability of the software

RELATED WORK —— IR models for source code retrieval

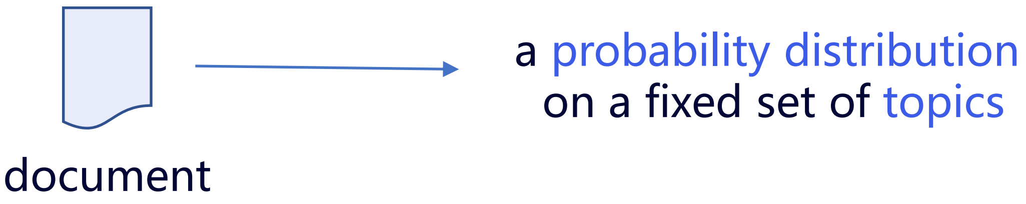
LSI

- An algebraic model
- Similarity — cosine of angle between vectors
- User query is first transformed into a document
- dimensionality reduction parameter
- does not do well when representing polysemy,



pLSI

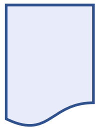
- A generative topic model
- each term in a document is modeled as a mixture over a set of multinomial random variables (topics)
- pLSI is susceptible to overfitting
- pLSI is not able to predict appropriate topic distributions for new documents



RELATED WORK —— IR models for source code retrieval

Latent Dirichlet allocation (LDA)

- A probabilistic and fully generative topic model



document

a finite mixture over the set of topics

- Each topic in this set is a probability distribution over the set of terms that make up the vocabulary of the document collection.

- Similarity between a document **di** and a query **Q** is computed as the conditional probability of the query given the document:

$$Sim(Q, d_i) = P(Q|d_i) = \Pi_{q_k \in Q} P(\boxed{q_k}|d_i)$$

qk is the kth word in the query

RELATED WORK —— IR models for source code retrieval

Latent Dirichlet allocation (LDA)

- In the results returned by LDA, the most likely terms in each topic – the terms with the highest probability – can be examined to determine the likely meaning of the topic.
- Approximation techniques; Gibbs sampling
- Choosing the number of topics in LDA modeling

Table 1
Top 10 terms in Topic 0 extracted from Mozilla.

Topic 0									
Set	Str	Print	Setup	Prt	Page	Preview	Engine	Pm	Footer
0.4740	0.2671	0.1455	0.0173	0.0117	0.0099	0.0077	0.0052	0.0049	0.00440

related to settings

related to printing

related to a page

Topic 0 —— print settings for a page

RELATED WORK —— Source code stability

Stability metric SDI_{inh}

- compute the percentage of change from the design in **iteration t** **(D_t)** to the design in **iteration t+1** **(D_{t+1})** of a software system

- **a** the number of classes whose names were modified from Dt to Dt+1,
- **b** the number of new classes added to Dt+1,
- **c** the number of classes removed from Dt,
- **d** the number of classes whose inheritance hierarchies were modified from Dt to Dt+1,
- **m** the total number of classes in Dt

$$SDI = \frac{(a+b+c)}{m} \times 100$$

$$SDI_{inh} = \frac{(a+b+c+d)}{m} \times 100$$

SDIe metric

- **Newly created**: Classes that were added to iteration t + 1 of the software.
- **Removed**: Classes removed from iteration t of the software.
- **Changed**: Classes with internal changes from iteration t to t + 1 .
- **Unchanged**: Classes that remained the same from iteration t to iteration t + 1.

j is the number of categories, value 1, 2, ...

$$SDI_e = -\sum_{i=1}^{\boxed{j}} \frac{C-i}{\boxed{N}} \log_2 \frac{\boxed{C_i}}{N}$$

Ci represents the number of classes that belong to category i

N is the total number of classes in iteration t + 1

RELATED WORK —— CK object-oriented metrics suite

$$SDI_{e,ck}$$

- SDIe,ck is a simpler version of the SDIe metric
- Use only the Chidamber and Kemerer (CK) suite of object-oriented class metrics

- The Chidamber and Kemerer (CK) suite of OO metrics is defined as follows: (collected these metrics using Understand for Java™)

  - WMC (Weighted Methods Per Class)
  - DIT (Depth of Inheritance Tree)
  - NOC (Number of Children)
  - CBO (Coupling Between Object Classes)
  - RFC (Response for a Class)
  - LCOM (Lack of Cohesion in Methods)

RESEARCH APPROACH

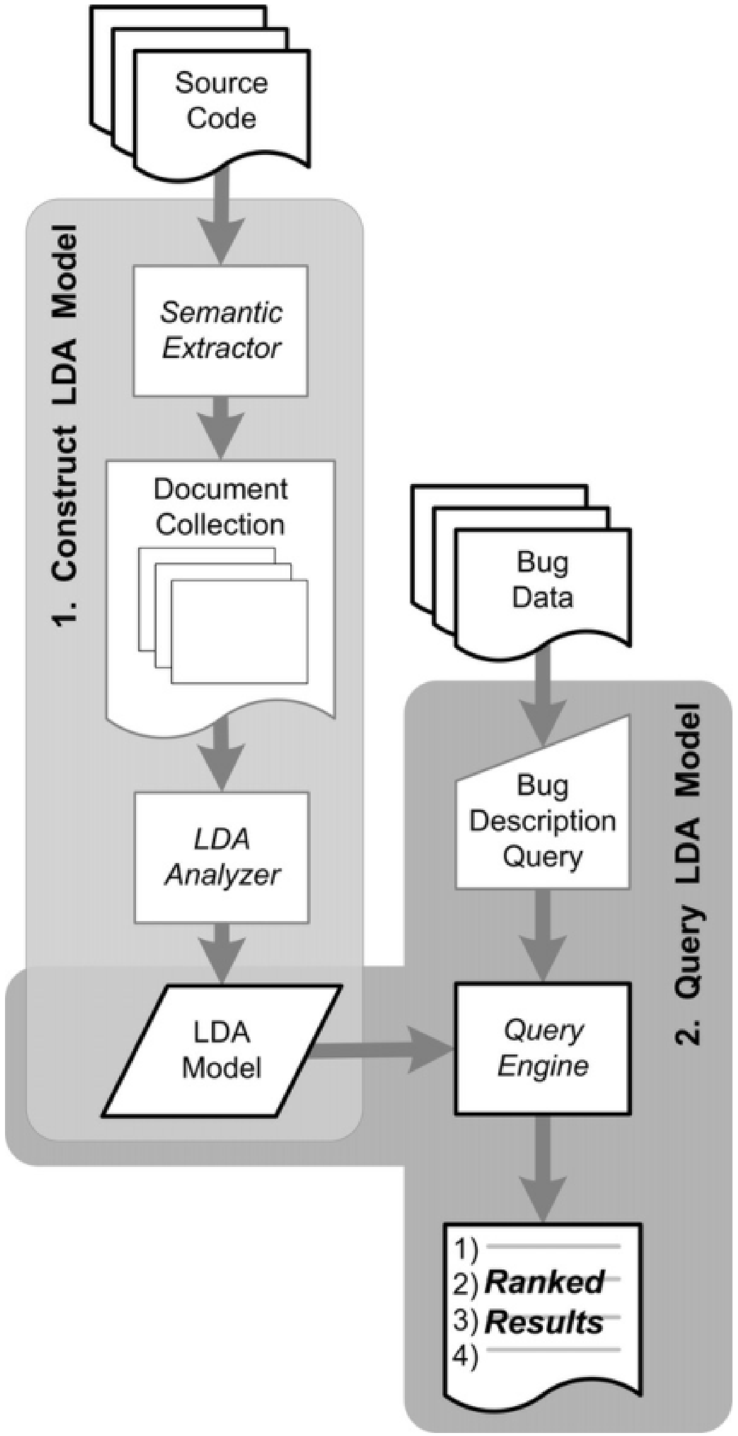
LDA-based bug localization approach

Two steps are necessary to construct an LDA model of a software system:

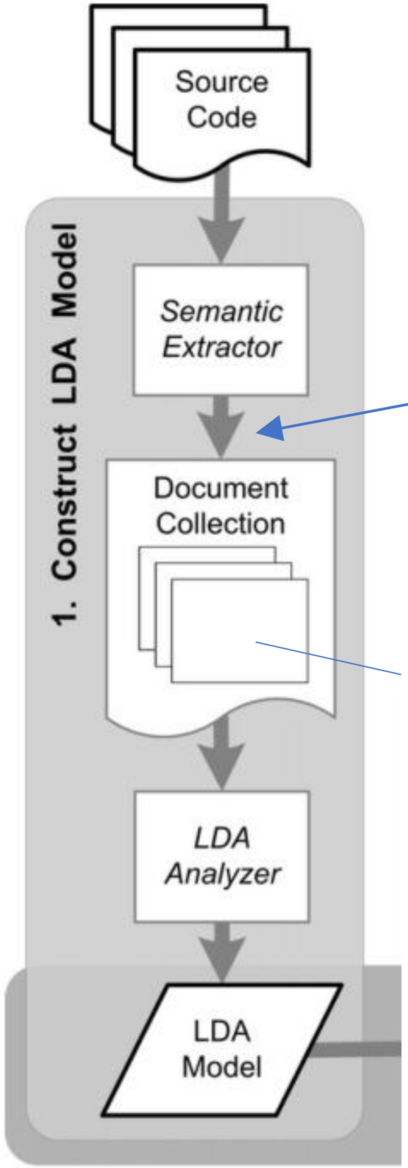
① build a document collection from the source code;

② perform an LDA analysis on the document collection.

1



RESEARCH APPROACH



Preprocess the semantic information

Store the preprocessed data extracted from each source element as a separate document in the collection

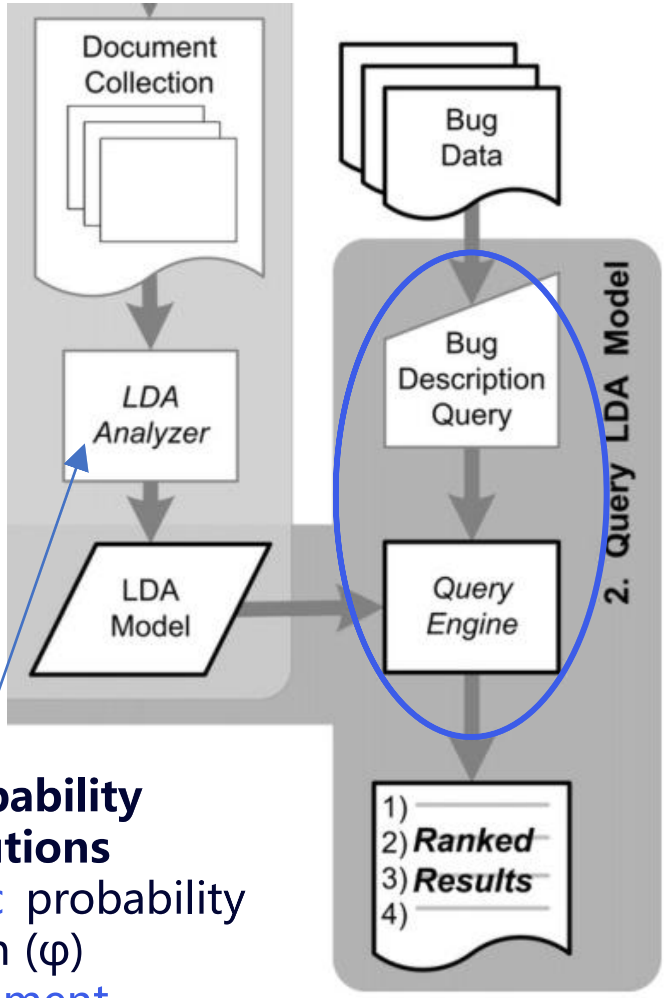
Step 1: Build a document collection from the source code

- at the method level of granularity
- extract string-literals in addition to comments and identifiers

Preprocess

- Multi-word identifiers are split into separate words based on common coding practices, e.g. printFile → print + file
- Each word is stemmed using a Porter stemming algorithm

RESEARCH APPROACH



Two probability distributions

① word–topic probability distribution (φ)
② topic–document distribution (Θ)

Step 2: Generate an LDA model

- an open-source software tool for LDA analysis called **GibbsLDA++**
- estimate topics from the document collection, estimate the word–topic and topic–document probability distributions

Parameters

- The number of topics
- Number of iterations for the Gibbs sampling process
- **α**, a hyperparameter of LDA
- **β**, a hyperparameter of LDA

RESEARCH APPROACH

Words added

- **Common abbreviations** or whole words when an abbreviation was used, e.g., eol for end-of-line, management for mgmt
- **Variants** of words already in the query, e.g., parse for parser
- **Synonyms** of words in the query, e.g., quit for kill [process]
- **Sub-words** of words in the query, e.g., name for rename

Process used to create queries

- bug title and description

Form an **initial query** by manually extracting **keywords from the bug title;** Words not related to the bug domain were ignored.

Form a **second query** by manually **adding keywords from the summary** of the initial bug report to the first query

Form a **third query** by **adding words related to the bug** and/or **removing words from query 1 or 2** that were less applicable to the bug domain

RESEARCH APPROACH —— Case studies

Data examined in the case studies

Table 2
Software used in case studies.

Case study	Software	Versions analyzed
1	Mozilla	1.5.1, 1.6, 1.6a
	Eclipse	2.0, 2.1.3, 3.0.2
2	Rhino	1.5R5
3, 4, 5	Rhino	1.4R3, 1.5R1, 1.5R2, 1.5R3, 1.5R4, 1.5R5, 1.6R1, 1.6R2, 1.6R3, 1.6R4, 1.6R6, 1.6R7
	Eclipse	3.0, 3.0.1, 3.0.2, 3.1, 3.1.1, 3.1.2, 3.2, 3.2.1, 3.2.2, 3.3, 3.3.1, 3.3.2, 3.4

RESEARCH APPROACH —— Case studies

Design of the case studies

- Comments, identifiers, and string-literals
- At the method level of granularity
- Minimal preprocessing —— all words were stemmed before being added to the collection
- Initial simple tests —— stop word removal was not found to substantially improve the results
- Did not remove stop word
- Rhino: 100 topics; Eclipse: 500 topics; Mozilla: 200 topics
- $\alpha = 50/K$, $\beta = 0.01$
- Accuracy: the rank of the first relevant method returned by each query; average of the ranks of the first relevant method returned for all bugs

RESEARCH APPROACH —— Case studies

Case Study 1

Goal: To examine whether the accuracy of LDA-based bug localization is better than LSI, over the same data used in previous LSI studies.

Metric: Rank of first relevant method returned for individual queries.

1

Case Study 2

Goal: To examine whether LDA is sufficiently accurate over all bugs in a single software system.

Metric: Percentage of bug queries with first relevant method in top 10 results.

2

RESEARCH APPROACH —— Case studies

Case Study 3

Goal: To examine whether LDA is sufficiently accurate over all bugs in the given software system (Rhino or Eclipse).

Metric: Percentage of bug queries with first relevant method in top 10 results (Rhino); percentage of bug queries with first relevant method in top 1000 results (Eclipse).

3

Case Study 4

Goal: To assess the impact of software size on the accuracy of bug localization.

Metric: A correlation between the average of the ranks of the first relevant method returned for each bug in each software iteration, and the Lines of Code, Number of Classes, and Number of Methods size metrics.

4

RESEARCH APPROACH —— Case studies

Case Study 5

Goal 1: To assess the impact of stability on the accuracy of bug localization.

Metric 1: A correlation between the average of the ranks of the first relevant method returned for each bug in each software iteration, and the SDI, SDIinh, and SDIe,ck stability metrics.

Goal 2: To assess the impact of complexity on the accuracy of bug localization.

Metric 2: A correlation between the average of the ranks of the first relevant methods returned for each bug in each software iteration, and the CK metrics.

5

RESULTS OF THE CASE STUDIES

Table 4
Eclipse bugs analyzed with LDA/LSI query.

Software version	Bug no.	Bug title	LDA/LSI query [28] NOTE: same query used as in Poshyvanyk et al. [28]
2.1.3	5138	Double-click-drag to select multiple words does not work	Double click drag select mouse up down release text offset document position
2.0.0	31779	UnifiedTree should ensure file/folder exists	Unified tree file folder node system location
3.0.2	74149	The search words after "" will be ignored	Search query quoted token

- LDA-based approach performed as well as or better than LSI when **100** topics were used.
- LDA-based approach outperformed LSI for all bugs when **500** topics were used.

LDA does perform better than **LSI** over the same corpus previously used by Poshyvanyk et al., using the same queries previously published by Poshyvanyk et al.

Table 5
Comparison of LDA to LSI over Eclipse. *Note*: same queries were used as in Poshyvanyk et al. [28].

Bug no.	First relevant method	LDA rank (100 topics)	LDA rank (500 topics)	LSI rank
5138	TextDoubleClickStrategyConnector.mouseUp(LDA)/JavaStringDoubleClickSelector.doubleClicked (LSI)	2	2	7
31779	UnifiedTree.createChildNodeFromFileSystem (both LDA and LSI)	2	1	2
74149	QueryBuilder.tokenizeUserQuery (both LDA and LSI)	1	1	5

RESULTS OF THE CASE STUDIES

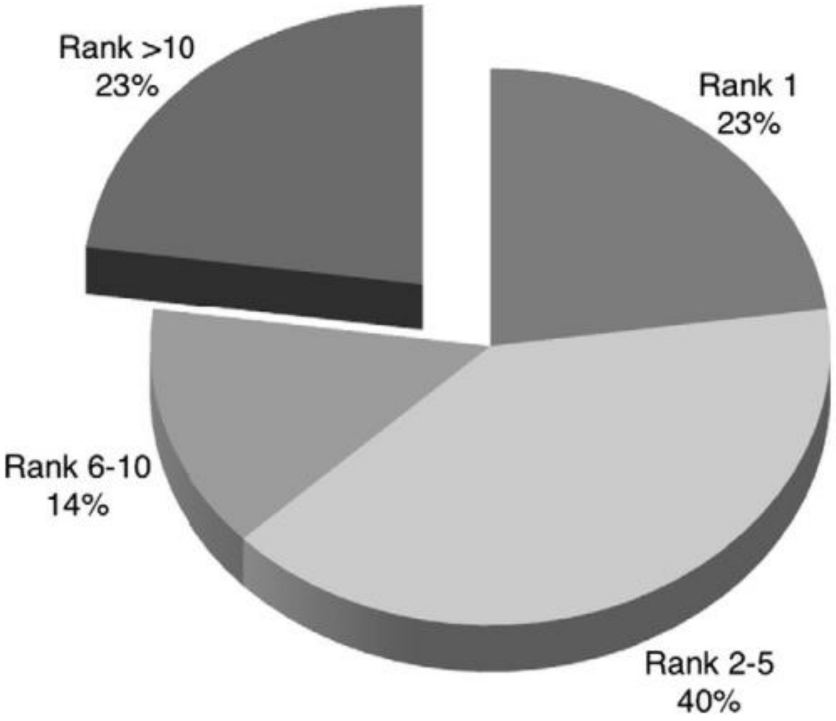


Fig. 2. Rank of first relevant method returned for Rhino 1.5R5 bugs.

- For **77%** (27/35) of the bugs analyzed the first relevant method was returned in the top 10 results
- For **63%** (22/35) of the bugs the first relevant method was returned in the top five results.

LDA-based bug localization technique does possess sufficient accuracy over **all the bugs** available in **one** complete software system.

RESULTS OF THE CASE STUDIES

Table 10
Accuracy of LDA-based bug localization per iteration of Rhino.

Version	No. bugs	Percentage of bugs			Average rank	Stdev of rank
		Rank 1	*Top 5*	*Top 10*		
1.4R3	4	25	25	25	291	474.08
1.5R1	7	43	57	57	39	52.24
1.5R2	3	0	0	33	44	40.67
1.5R3	11	27	40	50	40	61.69
1.5R4	13	23	46	62	176	517.13
1.5R5	35	23	63	77	17	46.39
1.6R1	11	9	27	36	99	233.17
1.6R2	6	33	50	50	290	654.20
1.6R3	1	100	100	100	1	0
1.6R4	12	17	25	25	1062	1107.67
1.6R6	1	0	0	0	104	0
1.6R7	2	0	0	50	252	345.07

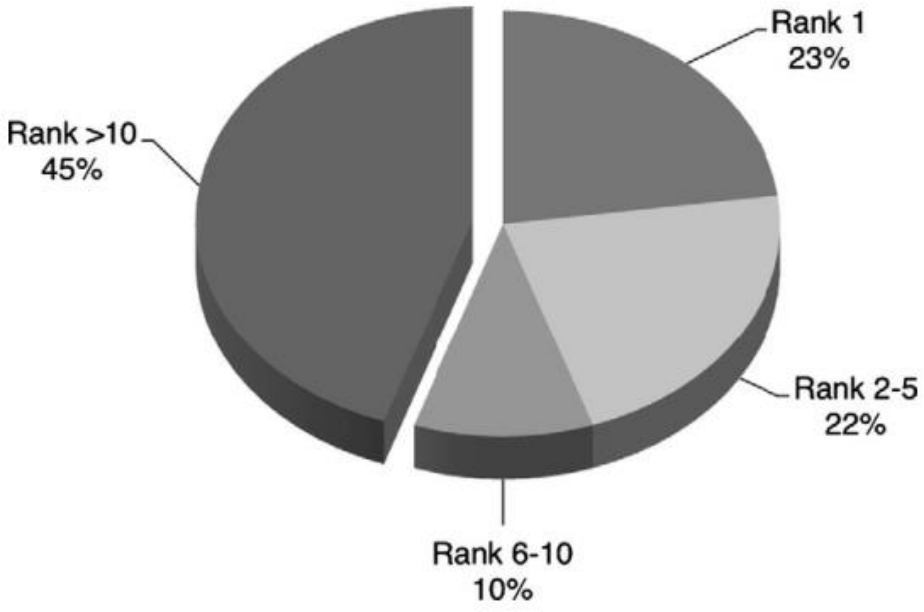


Fig. 3. Accuracy of LDA-based bug localization across 12 versions of Rhino.

- Over one-half (**55%**) of the bugs in Rhino resulted in the top ranked method in the top **10** results returned.
- Almost one-quarter (**23%**) resulted in the first relevant method being returned with a rank of **one**.

LDA-based bug localization technique does possess sufficient accuracy over **all the bugs** available in the given software system.

RESULTS OF THE CASE STUDIES

Table 15
Software size measures vs. average rank in Rhino.

Software size measures (Rhino)	Rhino average rank	
	r_s	*p*-Value
Lines of Code	0.203	0.527
Number of Classes	0.151	0.639
Number of Methods	0.196	0.541

Table 16
Software size metrics vs. average rank in Eclipse.

Software size measures (Eclipse)	Eclipse average rank			
	100 Topics		500 Topics	
	r_s	*p*-Value	r_s	*p*-Value
Lines of Code	0.038	0.900	0.269	0.374
Number of Classes	0.080	0.796	0.237	0.436
Number of Methods	0.159	0.603	0.308	0.306

- This test using Spearman’ s r_s is performed at the 95% significance level (**α = 0.05**).
- The correlations all have **p-values > α**.

There is no significant relationship between the software size measures and the accuracy of LDA-based bug localization as measured by average rank in Rhino and Eclipse

RESULTS OF THE CASE STUDIES

Table 17
Stability metrics vs. average rank in Rhino.

Stability metrics	Average rank	
	r_s	*p*-Value
SDI	−0.437	0.179
SDI_{inh}	−0.309	0.355
$SDI_{e,ck}$	−0.391	0.235

Table 18
Stability metrics vs. average rank in Eclipse.

Stability metrics	Eclipse average rank			
	100 Topics		500 Topics	
	r_s	*p*-Value	r_s	*p*-Value
SDI	−0.104	0.746	−0.056	0.863
SDI_{inh}	−0.077	0.812	−0.084	0.795
$SDI_{e,ck}$	−0.126	0.697	−0.084	0.795

- This test using Spearman's r_s is performed at the 95% significance level (**α = 0.05**).
- The correlations all have **p-values > α**.
- In both Rhino and Eclipse, no significant relationship was found between the SDI metrics and the accuracy of the LDA-based bug localization technique.

The accuracy of the approach is not affected by the stability of the system design.

RESULTS OF THE CASE STUDIES

Table 19
Average CK metrics vs. average rank in Rhino.

Average CK metrics	Average rank	
	r_s	*p*-Value
WMC	0.147	0.649
DIT	0.133	0.681
NOC	−0.233	0.466
CBO	0.228	0.477
RFC	0.113	0.727
LCOM	0.308	0.330

Table 20
Stability metrics vs. average rank in Eclipse.

Average CK metrics	Eclipse average rank			
	100 Topics		500 Topics	
	r_s	*p*-Value	r_s	*p*-Value
WMC	−0.297	0.324	0.028	0.929
DIT	0.115	0.707	0.220	0.471
NOC	0.198	0.517	0.275	0.363
CBO	−0.231	0.448	0.077	0.803
RFC	−0.258	0.394	0.027	0.929
LCOM	−0.308	0.306	−0.281	0.353

- This test using Spearman' s r_s is performed at the 95% significance level (**α = 0.05**).
- The correlations all have **p-values > α**.
- The CK metrics were not significantly correlated with average rank.

CK metrics would not be good indicators of the accuracy of the technique.

论 文 汇 报 展 示

感谢您的聆听

汇报人：王昭丹