Recovering Traceability Links between Code and Documentation

Paper by: Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo

Presentation by: Brice Dobry and Geoff Gerfin

Problem Statement and Proposed Solution

- Problem: Documentation is usually created in a very informal manner
  - In large software systems with large amounts of free-text documentation, this makes it difficult to associate the correct document for a certain piece of code.
- Solution: Use Information Retrieval (IR) to recover traceability links between source code and free-text documents.

Benefits of Traceability Links (1/3)

- Program Comprehension
  - For both top-down and bottom-up code analysis, traceability links can aid in:
    - Forming a hypothesis about how the code functions (bottom-up)
    - Locating code that supports a hypothesis (top-down)
- Maintenance
  - Determine legacy system functionality
  - Links can associate domain concepts to code fragments

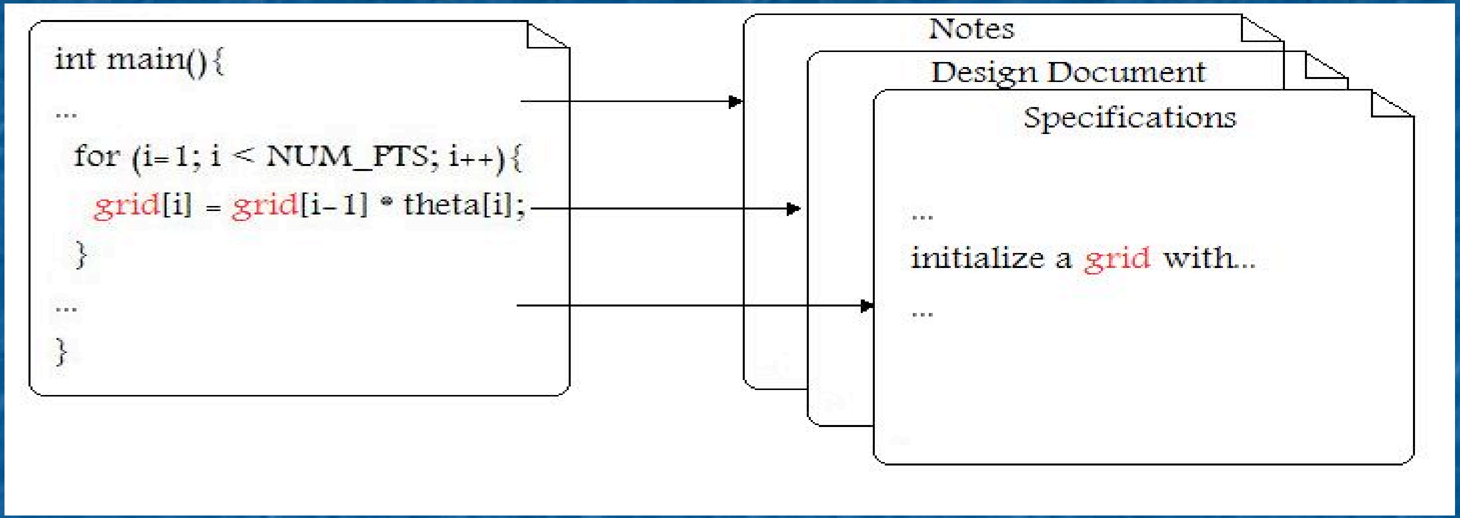
Benefits of Traceability Links (2/3)

- Requirement Tracing (Specifications)
  - Locate source code that corresponds to a program specification
  - Enables assessing program completeness / code inspection
- Impact Analysis
  - Discover pieces of code affected by a change to a program's specification
  - Discover pieces of documentation affected by a change to a program's code

Benefits of Traceability Links (3/3)

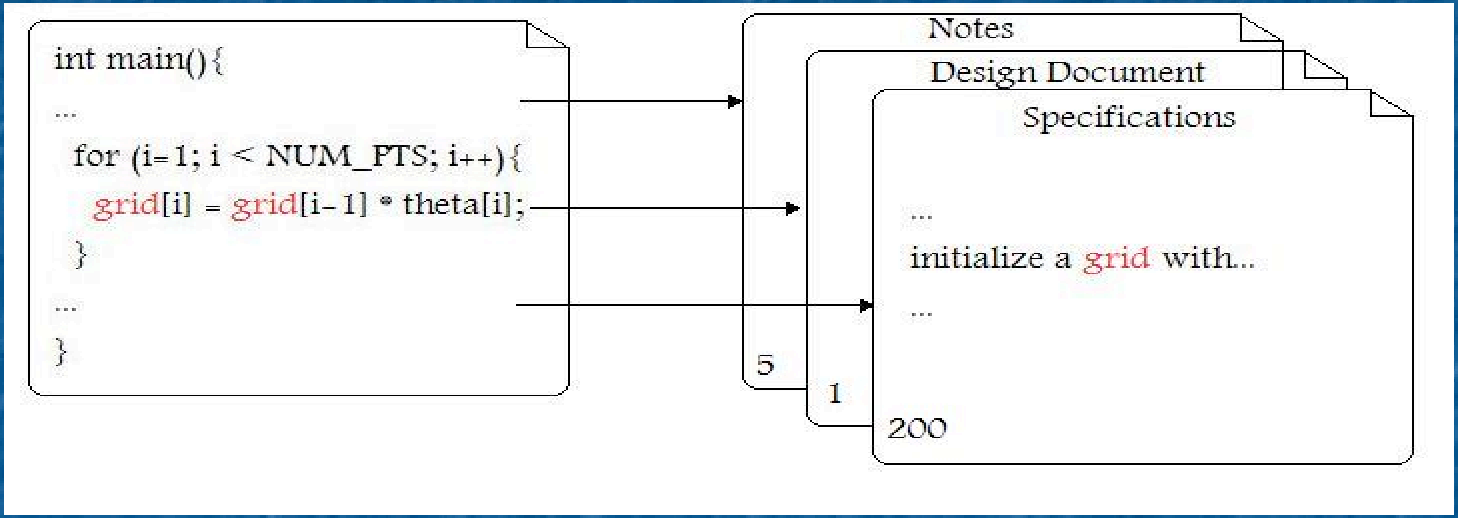
- Code Reuse
  - Concepts about existing code could exist in a wide range of documents (specifications, man pages, design documents, etc.)
  - Traceability links could aid in locating code that could be reused

Proposed Method



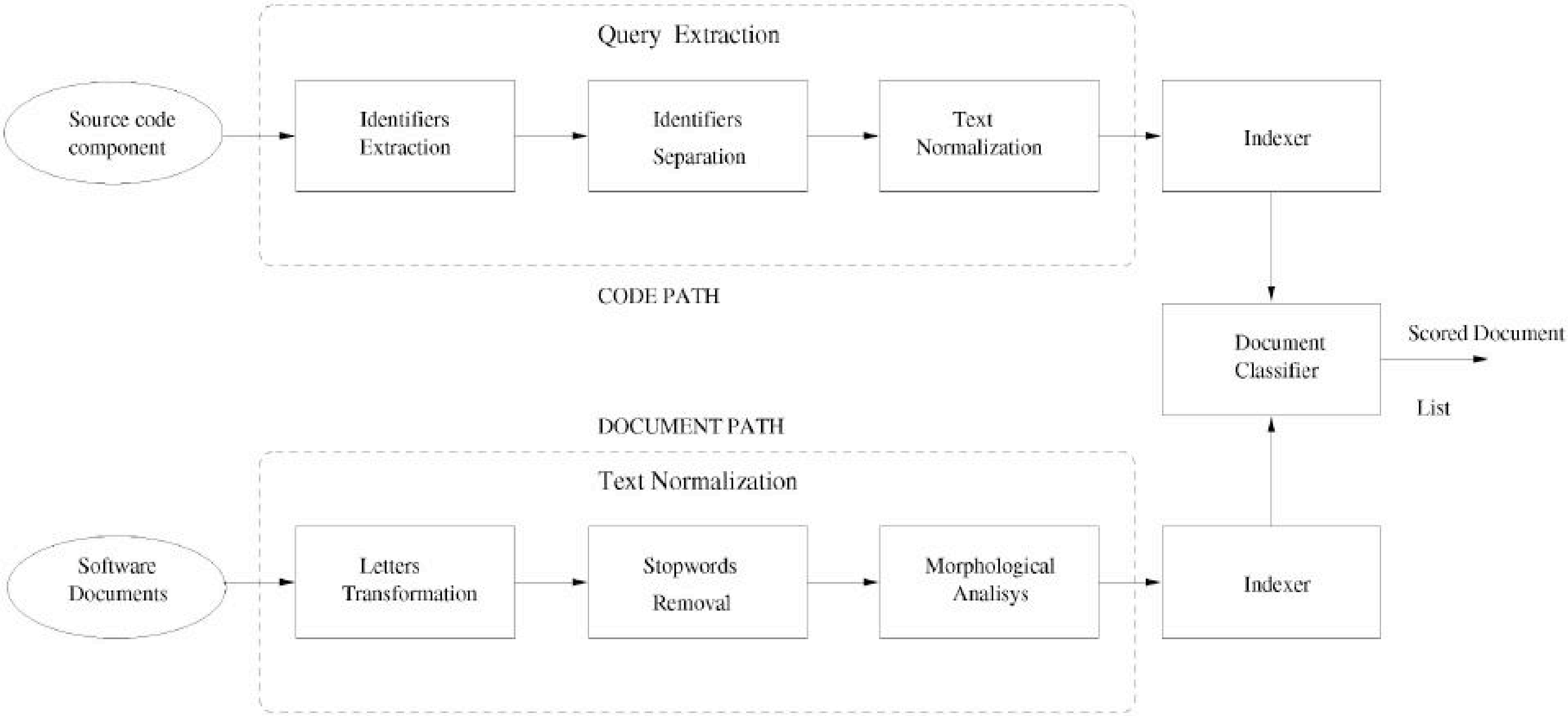
- Authors chose not to base their method on traditional compiler methods
  - Too difficult to apply syntactic analysis to the natural language sentences that occur in free-text documents
  - However, parsing technology can be applied when identifying source code elements

Proposed Method

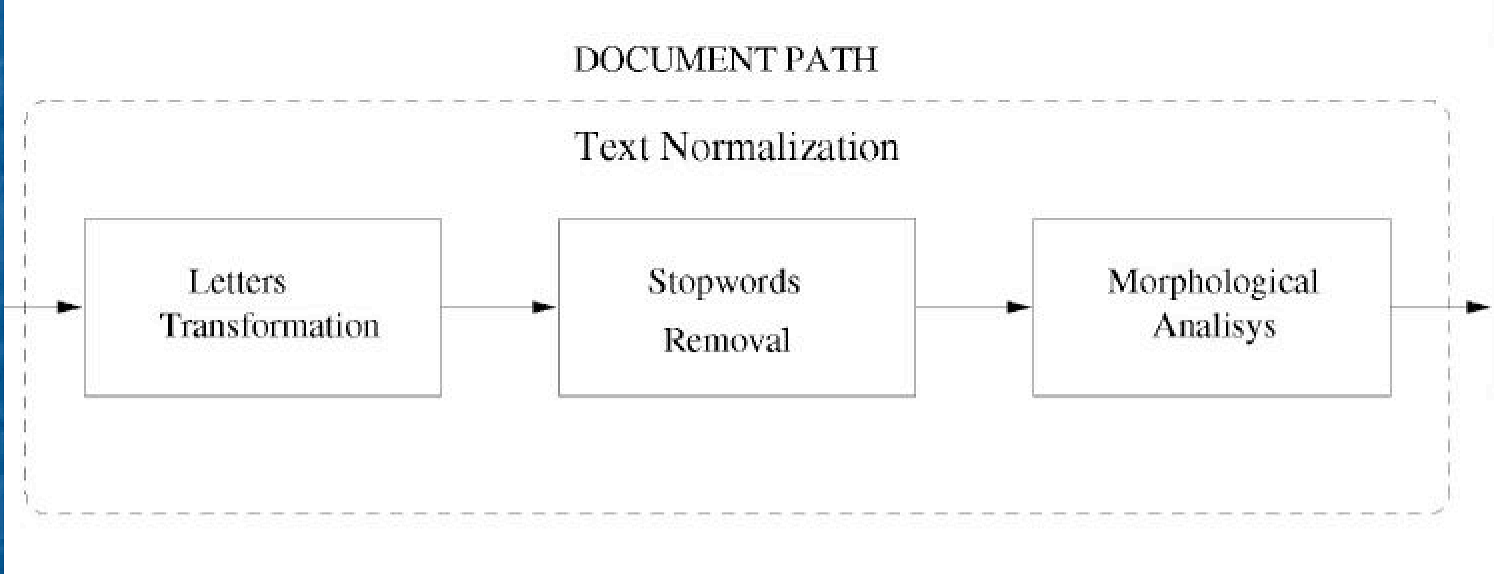


- Ranking is done in two different ways:
  - Probabilistic Model
  - Vector Space Model

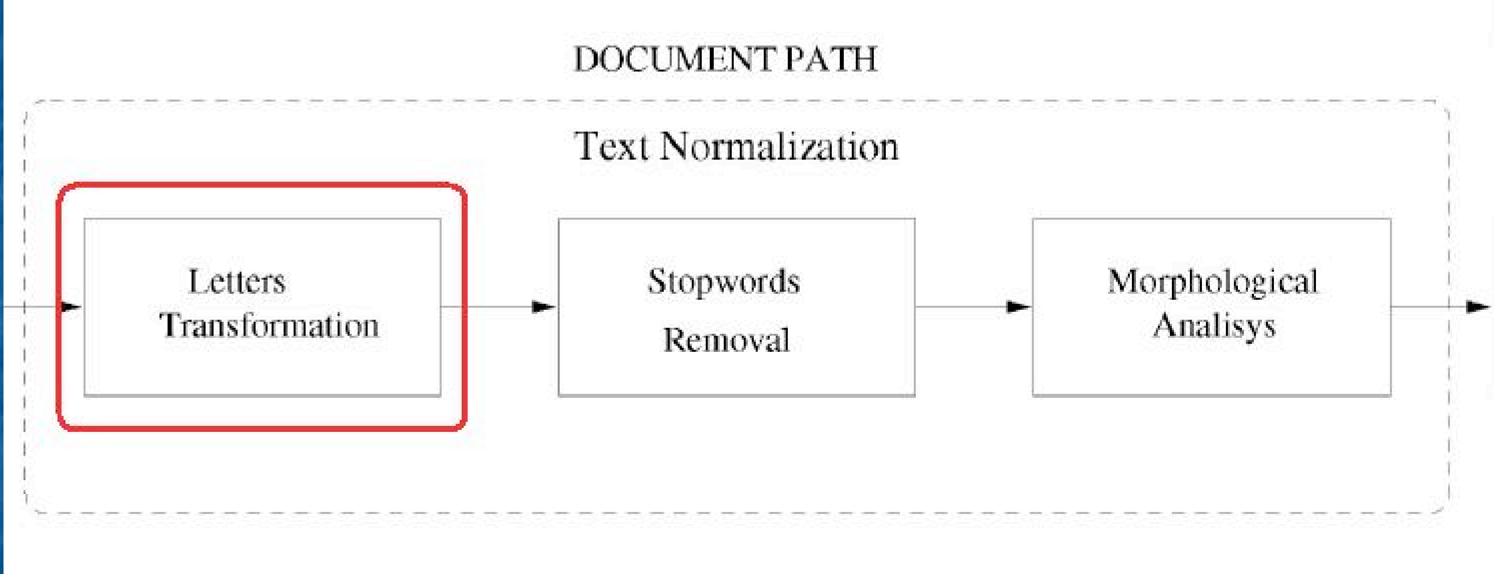
Architecture



Document Path (1/4)

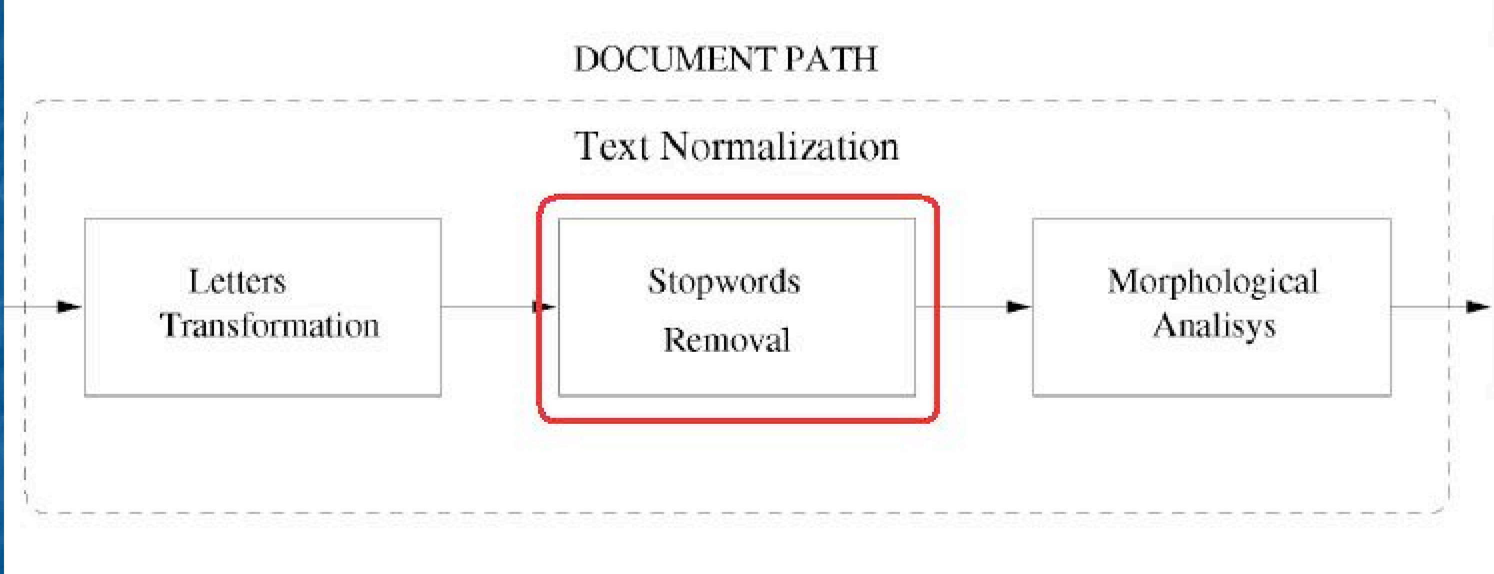


Document Path (2/4)



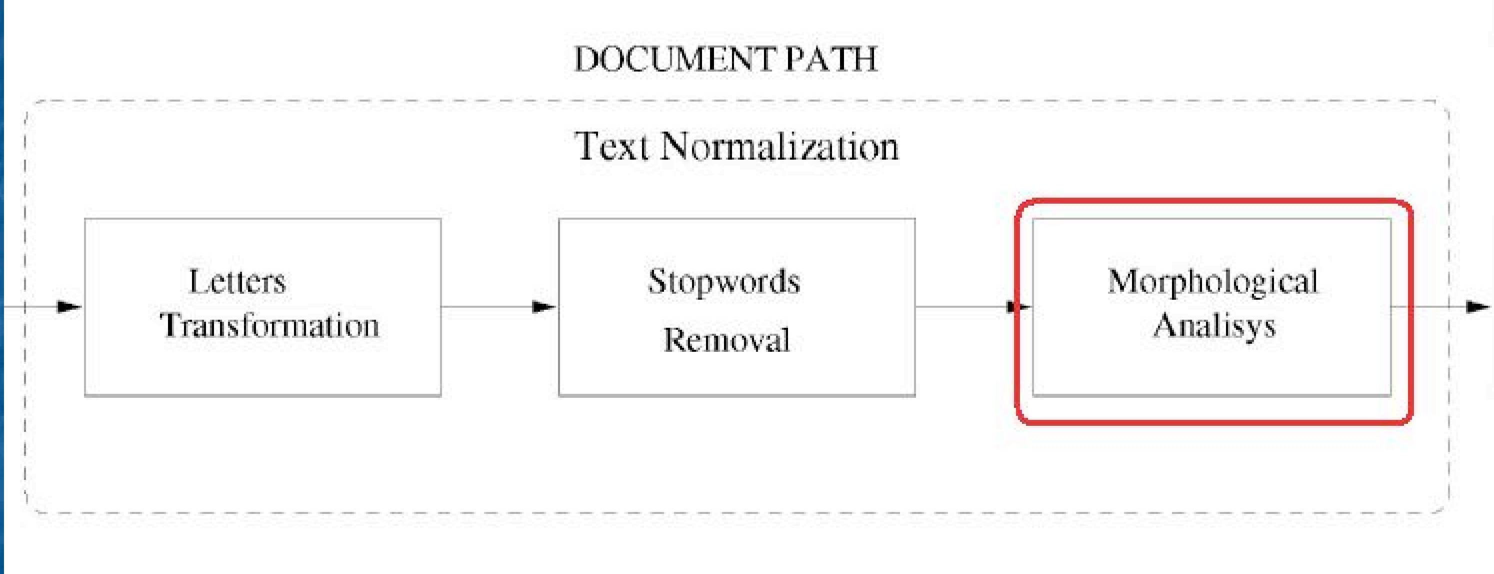
- Capital Letters -> Lowercase Letters

Document Path (3/4)



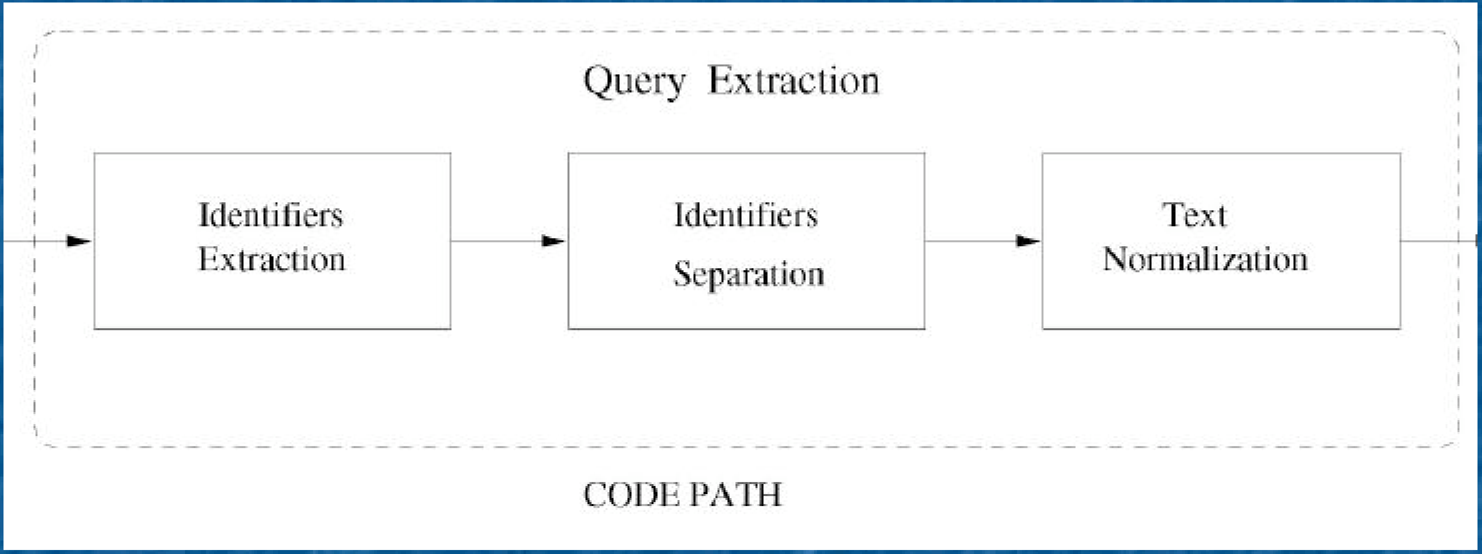
- Stop words are removed:
  - articles, punctuation, numbers
  - Ex source code cmpnt: "areaOfARectangle"
  - Ex sentence: "...calculate the volume **of a** cylinder."

Document Path (4/4)



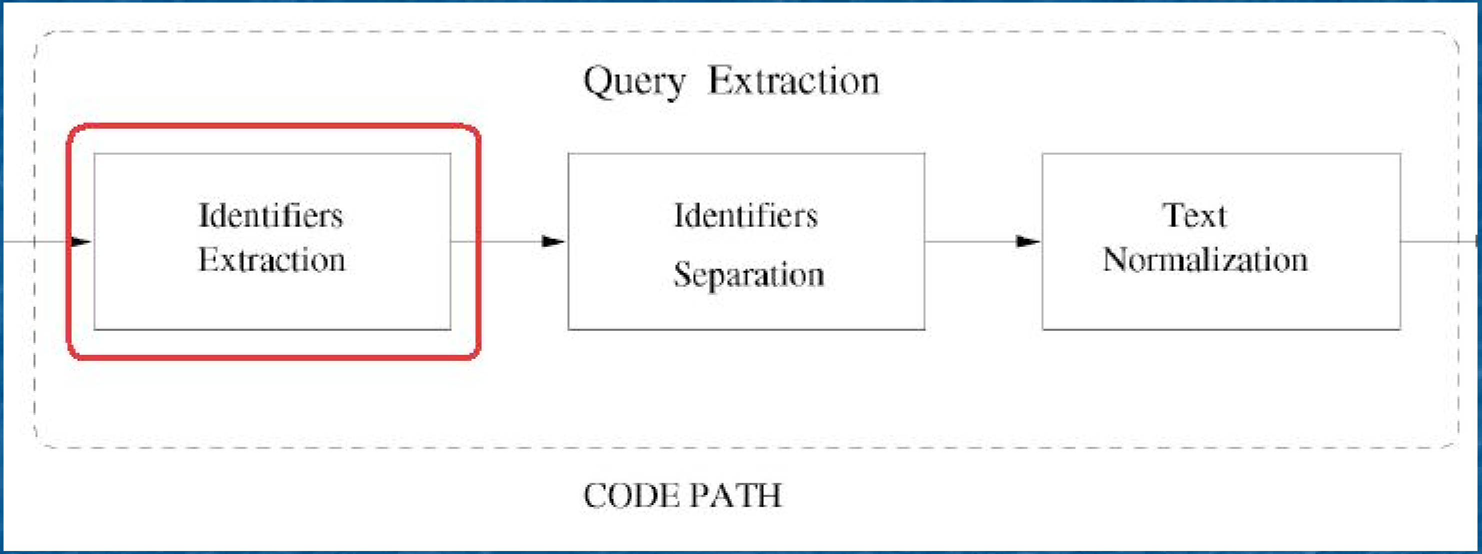
- Plurals -> Singulars
  - Ex: "Rectangles" -> "rectangle"
- Conjugated Verbs -> Infinitives
  - Ex: "jumps" -> "to jump"

Code Path (1/4)



Query Extraction
Identifiers Extraction
Identifiers Separation
Text Normalization
CODE PATH

Code Path (2/4)




```
double areaOfARectangle(float height, float width){
  double area;

  if (height == 0 || width == 0)
    return -1.0;

  area = height*width;
}
```

<u>Extracted Source Code Components</u>

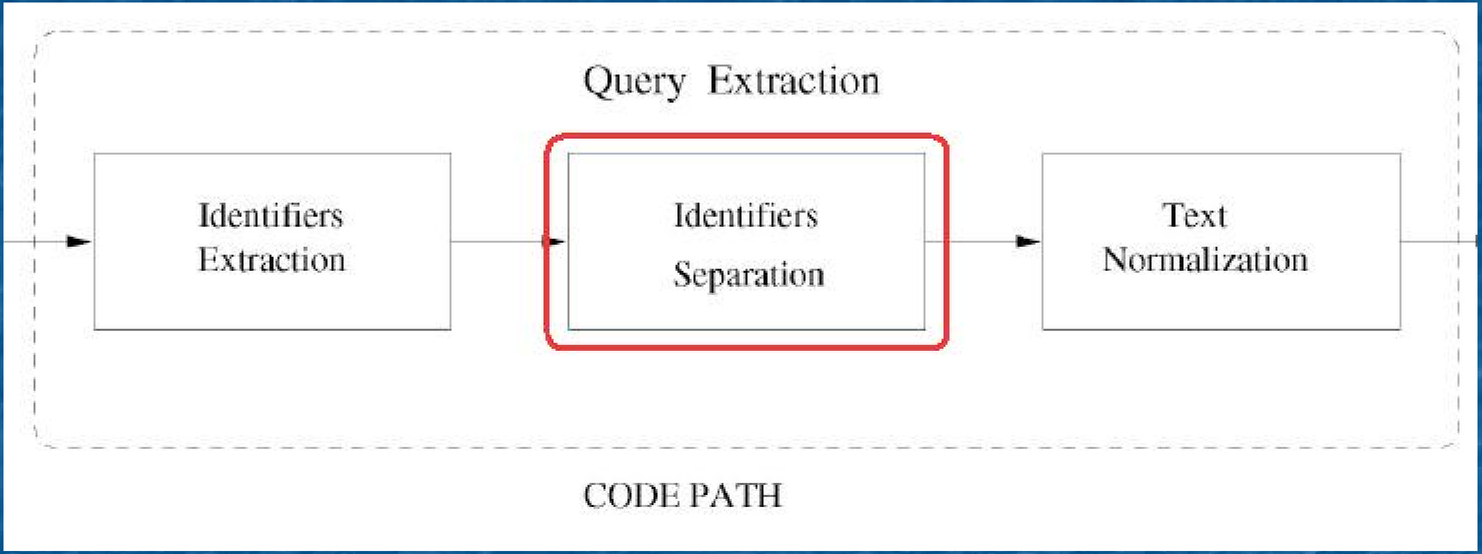
-"areaOfARectangle"

-"area"

-"height"

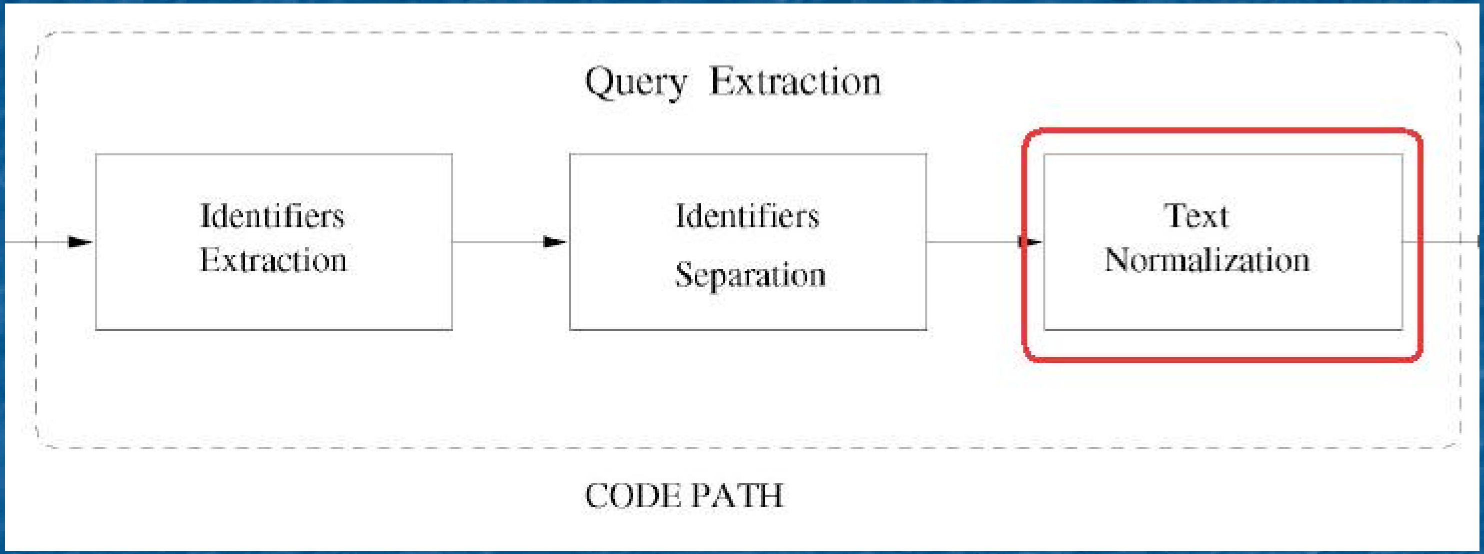
-"width"

Code Path (3/4)



- Text that contains two or more identifiers is split into single identifiers:
  - "areaOfARectangle" → "area", "Of", "A", "Rectangle"

Code Path (4/4)



- Text normalization includes the components of the document path:
  - Capital -> Lowercase
  - Stop Words Removal
  - Plurals -> Singulars; Conjugated Verbs -> Inifinitves

Ranking Methods - Probabilistic

- Probabilistic Model
  - Free-text documents are ranked according to the probability that they are relevant to a given query
  - Each string of words in a given vocabulary is assigned a probability within each document
  - The source code components are scored against the model
    - Higher scores indicate higher probability of relevancy

Ranking Methods - Probabilistic

- The similarity between a source code component and a document can be represented as a conditional probability:

$$Similarity(D_i, Q) = Pr(D_i | Q).$$

- Using Baye's Rule:

$$Pr(D_i|Q) = \frac{Pr(Q|D_i)Pr(D_i)}{Pr(Q)}.$$

- Pr(Di) same for all docs, P(Q) is constant:
  - Similarity(Di, Q) = Pr(Q | Di)

Ranking Methods - Probabilistic

- Q can be represented as a sequence of words:

$$Pr(w_1, w_2, \ldots, w_m \mid D_i) = Pr(w_1 \mid D_i) \prod_{k=2}^{m} Pr(w_k \mid w_1, \ldots, w_{k-1}, D_i).$$

- Computation can become exhaustive, so it is better to be less precise and limit to the last n-1 words (where n < m):

$$Pr(w_1, w_2, \ldots, w_m \mid D_i) \simeq Pr(w_1, \ldots, w_{n-1} \mid D_i) \prod_{k=n}^{m} Pr(w_k \mid w_{k-n+1}, \ldots, w_{k-1}, D_i).$$

Ranking Methods - Probabilistic

- Even the n-1 limit could become exhaustive if there is a large amount of words in the vocabulary
- It is rare that multiple words from the same source code component occur in the same document, therefore we can compute independently:

$$Similarity(D_i, Q) = Pr(Q|D_i)$$
$$= Pr(w_1, w_2, \ldots, w_m \mid D_i) \simeq \prod_{k=1}^{m} Pr(w_k \mid D_i).$$

- Problem: If any one word doesn't occur, P = 0.
  - Solution: Smoothing Function → If a word doesn't occur, P = lambda; otherwise P = P(wk|Di) + lambda

Ranking Methods - Vector

- Vector Space Model
  - Documents are classified in n-dimensions
    - n is the number of words in the vocabulary (n = |V|)
  - 2 vectors are created:
    - Vector 1: $[d_{i1}, d_{i2}, \ldots, d_{i|V|}]$ created for each doc.
      - Represents the occurrence of a Vocab word in Doc. i
    - Vector 2: $[q_1, q_2, \ldots, q_{|V|}]$ is the same for each doc
      - Represents the occurrence of a source code component Q in the Vocab.

Example Vectors

Component 1: double areaOfRectangle();

Component 2: double volumeOfCylinder();

This document describes the specifications for finding the area of a rectangle and the volume of a cylinder.

- Vocabulary: area, volume, rectangle, cylinder:

	Component 1	Component 2
	D = [1, 1, 1, 1]	D = [1, 1, 1, 1]
	Q = [1, 0, 1, 0]	Q = [0, 1, 0, 1]

Ranking Methods - Vector

- A distance function is used to compute the similarities between the vectors (overlap indicates high similarity):

$$Similarity(D_i, Q) = \frac{\sum_{j=1}^{V} d_{i,j} q_j}{\sqrt{\sum_{h=1}^{V} (d_{i,h})^2 * \sum_{k=1}^{V} (q_k)^2}}.$$

- This is the cosine of the angle between vectors d and q. A higher cosine of an angle indicates less difference; this is used as a common distance function

Case Study

Background Info

- Metrics
  - Recall
    - Ratio of number of relevant documents retrieved over the total number of relevant documents

$$Recall = \frac{\sum_i \#(Relevant_i \wedge Retrieved_i)}{\sum_i \#Relevant_i} \%$$

    - 100% recall means that all relevant documents were retrieved

Background Info

- Metrics (cont'd)
  - Precision
    - Ratio of number of relevant documents retrieved over the total number of documents retrieved

$$Precision = \frac{\sum_i \#(Relevant_i \wedge Retrieved_i)}{\sum_i \#Retrieved_i} \%$$

    - 100% precision means that all no irrelevant documents were retrieved

Background Info

- Metrics (cont'd)
  - Ideal results would have recall and precision both equal to 100%
  - For a tool to be most useful, it should have 100% recall with precision as high as possible (make sure all relative documents are included but include as few false positives as possible)

Background Info

- Test Subjects
  - **LEDA** (**L**ibrary of **E**fficient **D**ata types and **A**lgorithms)
    - C++
    - 95 KLOC
    - 208 classes
    - 88 manual pages
    - Manual pages were generated with scripts that extract comments from the source code

Background Info

- Test Subjects (cont'd)
  - Albergate
    - Java
    - 20 KLOC
    - 95 classes (60 looked at for this experiment)
    - 16 functional requirements
    - Documentation was produced early in the development cycle so much more distance between documentation and code

LEDA Results

- Many names (functions, arguments, etc.) from the code appear exactly in manual pages so traceability link recovery task is relatively easy
- Simplified steps
  - Identifier separation: Only split identifiers containing underscores
  - Text normalization: Only transform capital letter to lowercase

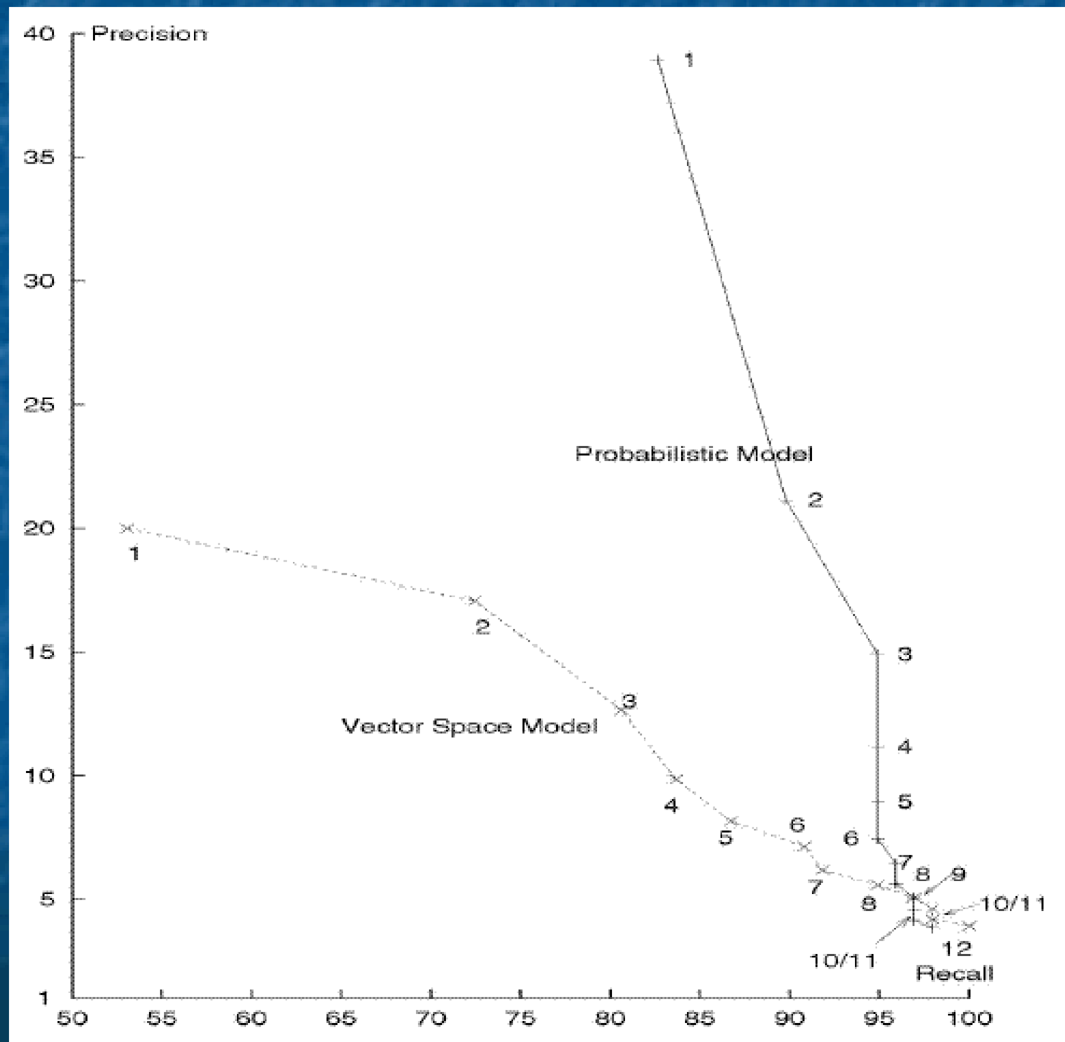
LEDA Results

- 208 classes, 88 manual pages
- Each class described by at most one man page
- 110 classes were not described anywhere
- Total number of links: 98

LEDA Results

		Probabilistic IR model			Vector Space IR model		
Cut	Retrieved	Relevant	Precision	Recall	Relevant	Precision	Recall
1	208	81	38.94 %	82.65 %	52	25.00 %	53.06 %
2	416	88	21.15 %	89.79 %	71	17.06 %	72.44 %
3	624	93	14.90 %	94.89 %	79	12.66 %	80.61 %
4	832	93	11.17 %	94.89 %	82	9.85 %	83.67 %
5	1040	93	8.94 %	94.89 %	85	8.17 %	86.73 %
6	1248	93	7.45 %	94.89 %	89	7.13 %	90.81 %
7	1456	94	6.45 %	95.91 %	90	6.18 %	91.83 %
8	1664	94	5.64 %	95.91 %	93	5.58 %	94.89 %
9	1872	95	5.07 %	96.93 %	95	5.07 %	96.93 %
10	2080	95	4.56 %	96.93 %	96	4.61 %	97.95 %
11	2288	95	4.15 %	96.93 %	96	4.19 %	97.95 %
12	2496	96	3.84 %	97.95 %	98	3.92 %	100.00 %

LEDA Results



LEDA Results

- Probabilistic model has higher recall value at low cut values but vector space model reaches 100% sooner (cut value of 12 versus 17)

- Precision results are greatly affected by the fact that more than half of the classes (110/208) are not referenced in any document

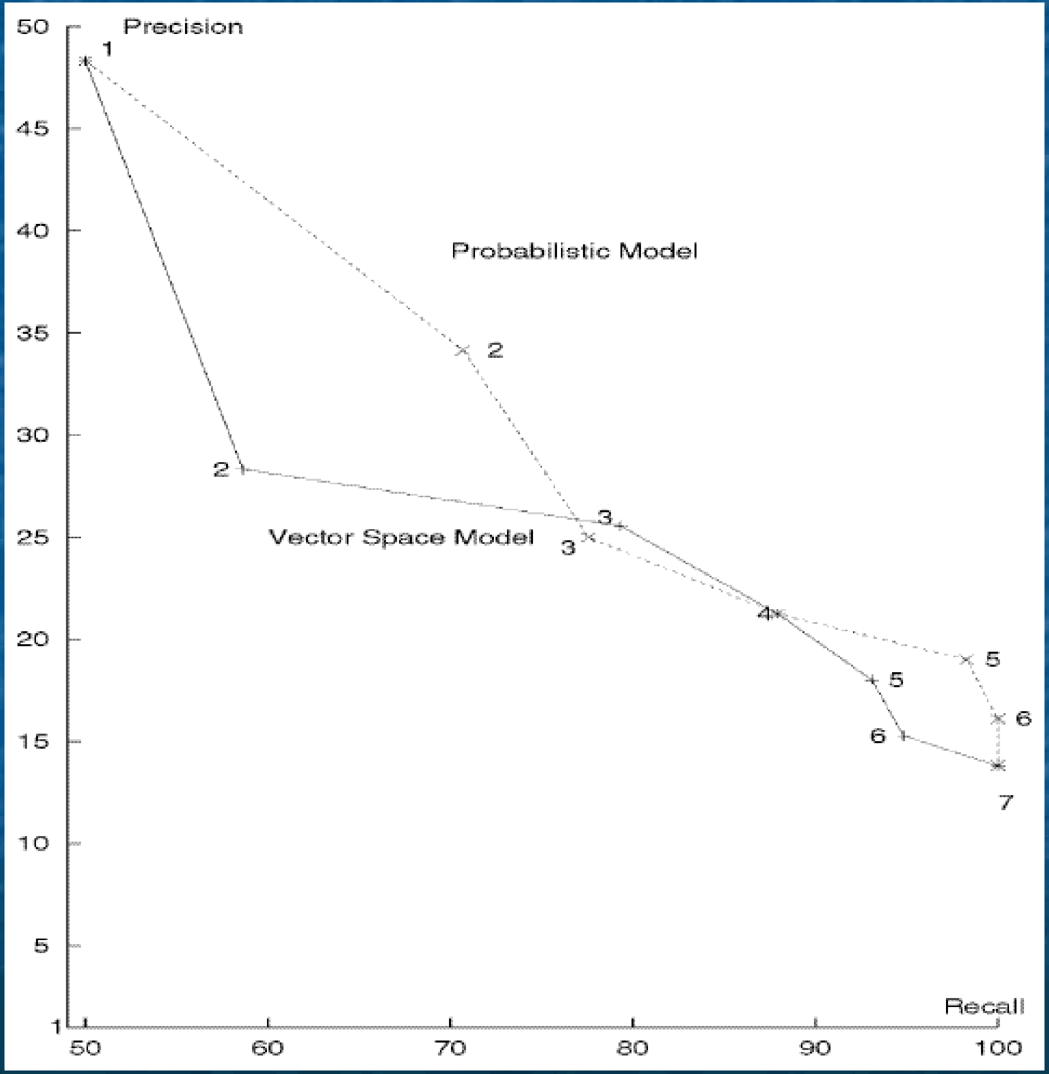
Albergate Results

- 60 classes, 16 functional requirements
- On average a requirement was implemented by about 4 classes with a maximum of 10
- Most classes were associated with only one requirement (6 were associated with two, 8 were associated with none)
- Total number of links: 58

Albergate Results

		Probabilistic IR model			Vector Space IR model		
Cut	Retrieved	Relevant	Precision	Recall	Relevant	Precision	Recall
1	60	29	48.33 %	50.00 %	29	48.33 %	50.00 %
2	120	41	34.16 %	70.68 %	34	28.33 %	58.62 %
3	180	45	25.00 %	77.58 %	46	25.55 %	79.31 %
4	240	51	21.25 %	87.93 %	51	21.25 %	87.93 %
5	300	57	19.00 %	98.27 %	54	18.00 %	93.10 %
6	360	58	13.80 %	100.00 %	55	15.27 %	94.82 %
7	420	58	13.80 %	100.00 %	58	13.80 %	100.00 %

Albergate Results



Albergate Results

- Two models performed similarly
- Probabilistic model reached 100% recall sooner (cut value of 6 versus 7)

Probabilistic vs. Vector Space Model

- Observations
  - Probabilistic model gets high recall values with small cut values then makes little progress towards 100% as cut value increases
  - Vector space model starts with lower recall values at low cut values then makes regular progress towards 100% as cut value increases

Probabilistic vs. Vector Space Model

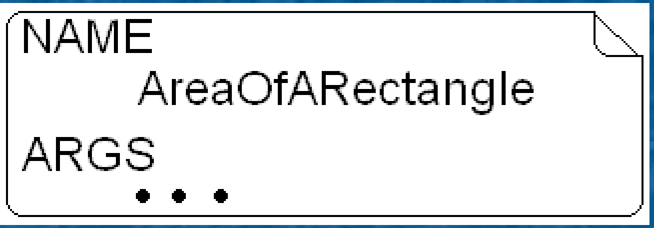
- Explanation
  - Probabilistic mode
    - Associate a class with a document based on the product of the unigram probabilities with which each class identifier appears in the document
    - Class identifiers that do not appear in the document are assigned a very low probability
  - Vector space mode
    - Only account for the class identifiers which appear in the document
    - Weigh the frequency of occurrence of the words in the document with respect to their distribution in other documents

Probabilistic vs. Vector Space Model

- Explanation (cont'd)
  - Probabilistic model
    - Best-suited for cases where the presence of class identifiers that are not included in the document is low
  - Vector space model
    - Best-suited for cases where each group of words is common to a relatively small number of documents
    - Aims to regularly achieve the maximum recall with a low number of retrieved documents, and not necessarily to pick the best match

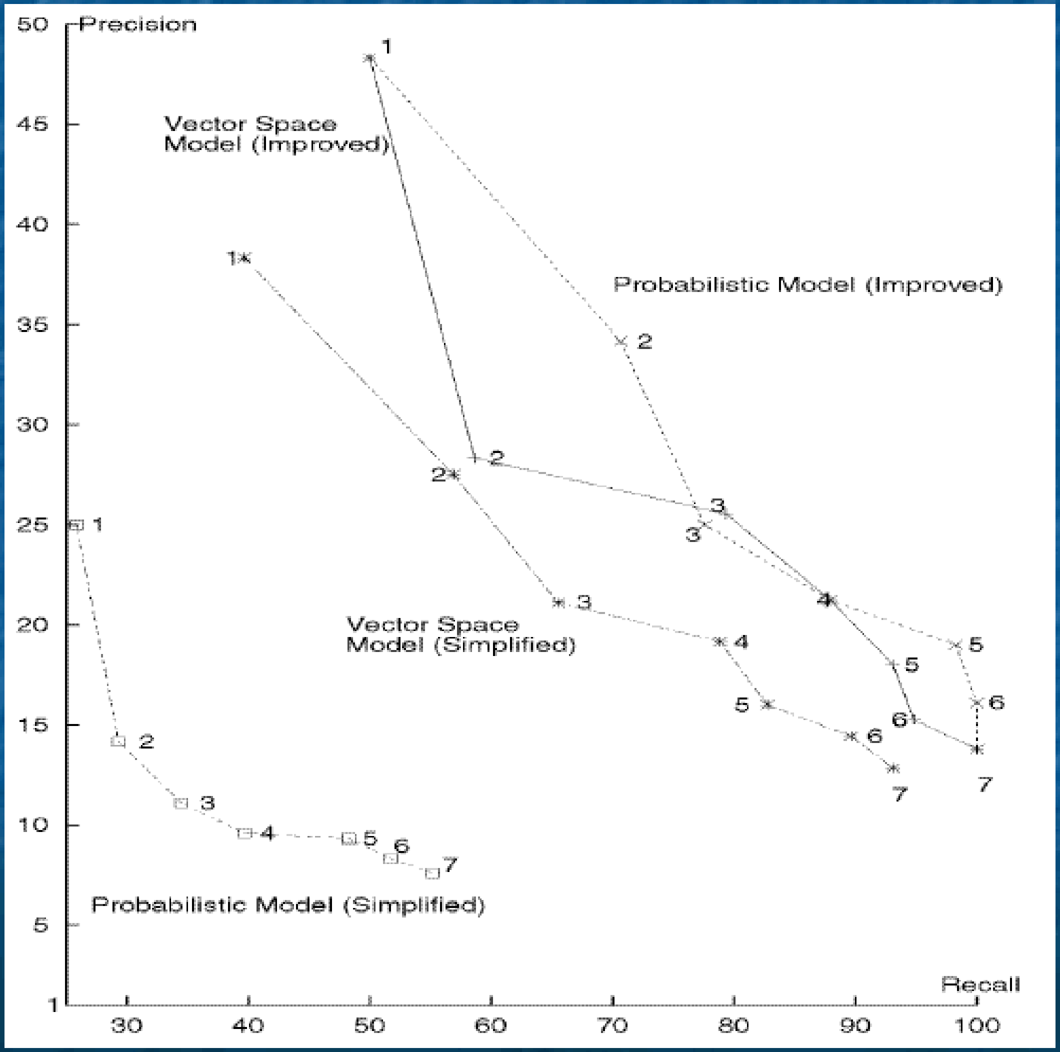
Probabilistic vs. Vector Space Model

- Explanation (cont'd)
  - Simplified process is only acceptable for documents close to the code



  - When used for the Albergate study, the difference is clear
    - The vector space model is affected very little while the probabilistic model is effected greatly by the simplification

Albergate Results (simplified)



Evaluation

Comparing IR models with `grep`

- Single Code Item
  - grep with each class identifier individually
- Code Items or Combined
  - grep with the or of all of the class's identifiers

	Single Code Item				Code Items *or* Combined			
	#Queries	#Empty Set	Mean Size	Max Size	#Queries	#Empty Set	Mean Size	Max Size
Albergate	4834	4575	5	14	60	0	11	13
LEDA	4670	451	20	88	208	1	75	88

Considerations of Effort Saving

- Recovery Effort Index (REI)

$$REI = \frac{\#Retrieved}{\#Available}\%$$

  - A person with no tool would have to look through every document to find links (REI = 1)
  - The lower the REI, the less effort is required (less effort identifying false positives)

Considerations of Effort Saving

- Recovery Effort Index (cont'd)
  - Can also be seen as the ratio between the precision of results achieved by a manual process and a semiautomatic tool with recall equal to 100% for the same software system

$$\frac{Precision_m}{Precision_t} = \frac{\#(Relevant \wedge Retrieved_m)}{\#(Relevant \wedge Retrieved_t)} \frac{\#Retrieved_t}{\#Retrieved_m}$$

$$\frac{Precision_m}{Precision_t}\% = \frac{\#Retrieved_t}{\#Available}\%$$

Considerations of Effort Saving

- Recovery Effort Index (cont'd)
  - Albergate (vector space): REI = 43.75%
  - LEDA (vector space): REI = 13.63%
  - Higher REI for Albergate is because there are not that many documents total (16)
  - IR methods are designed to work with huge document spaces
  - Albergate (grep): REI = 54.54
  - LEDA (grep): REI = 16

Retrieving a Variable Number of Documents

- Instead of a cut value, we could also have a variable number of documents based on some threshold of the similarity values

$$t_Q = c * [\max_i s_{i,Q}]$$

- Return all documents with $s_{k,Q} \geq t_Q$

Retrieving a Variable Number of Documents

Percentage	Retrieved	Relevant	Precision	Recall
90 %	59	29	49.15 %	50.00 %
70 %	101	38	37.62 %	65.51 %
50 %	158	50	31.64 %	86.20 %
30 %	265	55	20.75 %	94.82 %
10 %	484	58	11.98 %	100.00 %

- Worse than fixed cut values but still decent
- Mixed version: take the minimum of the above technique with 10% constant or the best 7 (constant cut value)

Percentage	Retrieved	Relevant	Precision	Recall
min(10 %, best 7)	329	58	17.62 %	100.00 %

Conclusion

Summary

- IR is a practical solution to the problem of (semi-)automatically recovering traceability links
- Both IR models (probabilistic and vector space) achieve 100% recall with approximately the same number of documents retrieved
- Probabilistic model achieves higher recall with a smaller number of documents retrieved
- Vector space model shows regular increase in recall with higher numbers of documents retrieved

Summary

- IR approach easily reduces the effort required by the user over `grep`
- Increased text normalization provides better results, especially when the "distance" between the documents and the code is higher

Future Work

- Use known existing traceability links to ease the recovery of additional links
  - Can be especially useful when the number of common words between the code and documentation is very low (or 0)
- Investigate using this technology for impact analysis
  - Take a textual maintenance request and determine which sections of code will be affected to make this change