Assisting Concept Location in Software Comprehension

Brendan Cleary and Chris Exton

Lero, University of Limerick, Ireland,
brendan.cleary@ul.ie

Abstract. Concept location, the problem of associating human oriented concepts with their counterpart solution domain concepts, is a fundamental problem that lies at the heart of software comprehension. Recent research has attempted to alleviate the impact of the concept location problem through the application of methods drawn from the Information Retrieval (IR) community. Here we present a new approach based on a complimentary IR method which also has a sound basis in cognitive theory. We compare our approach to related work through an experiment and present our conclusions. ...

1 Introduction

While how software engineers understand or comprehend the systems they work with is frequently described in terms of processes, models or strategies [1] [2], it can also be described in terms of a person's ability to communicate intelligently in human oriented terms about a systems implementation. Biggerstaff describes a person as understanding a program when able to explain the program, its structure, its behavior, its effects on its operational context, and its relationships to its application domain in terms that are qualitatively different from the tokens used to construct the source code of the program [3]. This categorization of how an engineer understands a software system rests on two different descriptions of "computational intent" [4] and the ability of the engineer to associate concepts appearing in one description of intent with the concepts in another.

In software engineering we are usually concerned with two descriptions of intent; one described using a human language another using a programming language. For clarity and to compare related work, we refer to the former as problem domain and the latter as solution domain descriptions of intent. These different descriptions or domains of intent are separated by constraints on the sets of concepts expressible using the language in which they are described which constitute a "conceptual gap" between domains [5]. Recent research [6] [7] [8] [9] [10] has seen attempts to bridge this conceptual gap based on the application of methods drawn from the Information Retrieval (IR) community. These approaches typically index the source code "documents" of a System Under Study (SUS) constructing a vocabulary of terms and models of term-document relationships within the SUS. When a user specifies a query the vocabulary and term-document models are consulted in order to determine a set of documents (usually methods or functions) which are ranked in descending order and returned to the user as being related to the terms defined in their original (problem domain) query.

Approaches which follow this general schema tend to focus on information derivable solely from the source code of the SUS in constructing their models and ranking

documents as being related to a query. While source code is one artifact used by software engineers to express their intent, engineers have traditionally had recourse to use other artifacts to communicate and record concerns which either were not easily expressed directly in code (crosscutting concerns for example) or which they intended to express in code at some other time. Artifacts such as design and requirements documentation have traditionally been used to express such concerns while more recently bug tracking databases, online forums and even email have become in some development environments the primary mechanisms for expressing and communicating such concerns. These artifacts serve as a ”human oriented” repository of information related to a SUS that parallels the source code ”machine orientated” artifacts of the SUS. By concentrating on source code artifacts alone the concepts and concerns described in these artifacts cannot be taken into account when evaluating a user's query. In this paper we present a new approach to the concept location problem based on a cognitively motivated information retrieval technique which while similar to the general schema outline above, differs primarily in that it allows us to transparently incorporate information derived from non-source code artifacts in implementing a ranked search over the source code artifacts of a SUS.

In the next section we look at related work on the application of information retrieval techniques to the concept location problem while also giving a brief overview of other approaches to the problem. In section 3 we present our approach its theoretical foundations and implementation. In sections 4 and 5 we present a small experiment where we evaluate the performance of our technique against related techniques. Finally in section 6 we present our conclusions and discuss future work.

2 Related Work

The concept location problem put simply is the problem of identifying the subset of elements comprising a software system related to a set of problem domain concepts. As such the concept location problem is very similar in intent to the concern localization and feature location problems in that each is concerned with mapping from the problem domain to the solution domain. However while the three terms are frequently used interchangeably we can envisage a distinction in terms of the constraints placed (usually informally) on the definition of the set of problem domain concepts in each. While concept location places very little constraint on its definition of a problem domain concept (except that it be defined using human-orientated terms), the concern in concern localization implies that it is motivated by problem domain concepts that are also system stakeholder interests. Finally feature location goes one step further in constraining problem domain concepts to be those stakeholder concerns which are also executable using test cases.

2.1 Feature Location and Concern Localization

Feature location approaches, due to the more stringent constraints they place on the definition of the problem domain concept set, are able to capitalize on the formal relationships expressed in a function call trace that results from the execution of one or

more test cases related to a feature or features. Dynamic software analysis or feature location techniques such as software reconnaissance [11] and formal concept analysis [12], focus on localizing concepts that are expressible either through test cases or through navigation of control and data flow. Unfortunately while a system's implementation may imply the intent that led to its development, the intent is not expressed explicitly in that implementation [13]. As such these techniques are only able to localize concepts which are expressible as test cases. While this is a limitation, in cases where there exists no system expert or documentation, they can be of great benefit in assisting software engineer comprehension. These approaches are also able given enough test cases to identify sets of elements specific to particular features.

Concern localization approaches, due to their definition of problem domain concepts as being that which a system stakeholder is concerned with, tend to involve those stakeholders (usually software engineers) and their existing knowledge or actions in defining the mapping between concerns and software elements. Semi-automated or manual approaches such as IBM's Concern Manipulation Environment (CME) [14] and FEAT [15] provide an environment which allows engineers to explicitly describe and record associations between software elements and user defined concerns. Similarly artifact recommender systems or agents such as Hipikat [16] suggest pertinent artifacts (both source code and documentation) to engineers as they engage in an understanding task by attempting to automatically model the concerns the engineer is currently working on and making inferences from that model. Mylar [17] is another tool that attempts to model the concerns or tasks that an engineer is concerned with as a set of evolving software elements; this information is then used to perform filtering of the elements presented to the engineer in and IDE on a task by task basis in an attempt to reduce information overload.

2.2 IR Based Approaches

The concept location problem being very much more liberal in its definition of a problem domain concept requires approaches which are able to construct relationships between arbitrary problem domain concepts described in human orientated terms and the elements of a software system from only that evidence which already exists in the corpus of software system artifacts. When described like this the concept location problem would seem to be well fitted by techniques from the information retrieval community where the goal is to find material (usually documents) of an unstructured nature (usually text) that satisfy an information need from within large collections [18]. In [6] Zhao et al describe their attempts at using an IR technique, the Vector Space Model (VSM), in identifying specific and relevant sets of functions for a set of given feature descriptions. Their approach sees them construct feature descriptions from natural language texts such as requirements and design documentation. These feature documents are then matched, using the VSM, against query documents derived from identifiers found in functions in the source code of the SUS.

One of the fundamental problems associated with the VSM, as used by Zhao et al in [6], is that correlation between term sets is used to compute the similarity measure between documents or between documents and queries, That is, two documents are considered similar if they share the same terms. While logical, this scheme requires

that if documents are discussing a particular concept, then to be considered similar those documents need to use the same terms when describing the concept. Where a document or a user generated query uses different terms when referring to the same concept then documents that should be considered similar will likely not be classified as such by VSM. These problems are termed synonymy and polysemy respectively where synonymy describes the problem of different terms being used to describe the same concept and polysemy describes the problem of a single term (depending on context) having more than one distinct meaning [19]. This is of potentially great significance to novice software engineers or engineers encountering an unfamiliar system for the first time who may not possess a large vocabulary of terms with which to describe their concerns and so may be unsuccessful in constructing queries which would elicit the desired results.

Latent Semantic Indexing (LSI) is an extension to the VSM model that attempts to increase classification accuracy by reducing the impact of the synonymy and polysemy problems. Whereas VSM computes similarity based on a raw term-document occurrence matrix (with possibly some term weighting) LSI attempts to alleviate the problems we have just described by performing an extra analysis step in which the overall distribution of a term over its usage context, independent of its correlations with other terms, is first taken into account prior to computing a similarity measure between documents [20]. This allows otherwise unrecognized or ”latent” relationships between terms to be exposed and recognized when performing a search. Marcus et al. in [10] expand on the work in [9] to apply the LSI method directly to the concept location problem. The authors first index the source code of the SUS identifying and extracting index terms from identifiers and comments. The authors then partition the SUS into a set of documents (defined in terms of the index term set) corresponding to the set of functions defined in the SUS. Finally each document is mapped, using a technique called Singular Value Decomposition, into the LSI space. Query documents are similarly mapped into the LSI space and the similarity between documents and the query is computed in a similar fashion as in the VSM. In an attempt to further resolve the lack of vocabulary problem described previously the authors investigate two mechanisms for defining the query documents, the first is a simple user generated natural query. The second sees the authors implement a limited form of query expansion where given a single query term other potentially related query terms are derived from the LSI space based on their relationship to the ”seed” query term.

While using the LSI space to identify latent relationships between terms in the source code of a SUS will identify some previously unrecognized relations, it is only making use of one part of the available corpus of information about the SUS. The work presented in this paper similarly to [10] employs a sophisticated IR method that seeks to address the problems of synonymy, polysemy and lack of vocabulary mentioned previously. However unlike [10] the method we describe provides a structured and cognitively motivated mechanism for incorporating information derived from the potentially large (depending on the SUS) corpus of non-source code artifacts in generating sets of ranked software elements in response to a users query. The next section describes our approach, its cognitive motivation and implementation.

3 Assisting Concept Location through Language Modeling

Language Modeling (LM) is an approach used in many recent studies in IR that not only produces promising experimental results comparable with the best IR systems but also provides a sound theoretical setting [21]. The LM approach to information retrieval calculates the conditional probability $P(Q|D)$ of generating a query Q given an observed document D. Where $P(Q|D)$ is calculated based on a probabilistic language model derived from document D.

3.1 Classical LM Framework

The classical LM framework can be described as follows; given a query Q consisting of a sequence of query terms $Q = \{q_1, q_2, ..., q_n\}$ the probability of generating Q from document D is equal to the probability of observing the sequence of n query terms from Q in D.

$$P(Q|D) = P(q_1, q_2, ..., q_n|D) \tag{1}$$

For reasons of computational tractability the independence assumption which states that terms are statistically independent from each other [22] is often invoked. This assumption results in a unigram model being calculated from D whereby terms are considered to be conditionally independent. This means that the order or sequence of occurrence of query terms in the document does not need to be considered when calculating the correspondence of the query and document.

$$P(Q|D) = \prod_{q_i \in Q} P(q_i|D) \tag{2}$$

Models such as this have already been used by for recovering traceability links between source code and high-level documentation. In [23] Antoniol et al. used unigram estimation based on term frequency to create links that describe the similarity between elements of the code base (object-orientated classes) and high level system documentation. Antoniol et al. use a stochastic language model based on identifiers found in the source code elements to calculate the set of conditional probabilities between a given source code element and the set of system documents.

An alternative formulation of LM in information retrieval is KL-divergence which estimates two models; one for the query and one for the document, the similarity of the query and document or score then being determined by the KL-divergence between the two models.

$$Score(Q,D) = \sum_{q_i \in Q} P(q_i|Q) \times logP(q_i|D) \tag{3}$$

3.2 Extending the Classical LM Framework

While the independence assumption makes the development of retrieval models easier and the retrieval operation tractable, it does not hold in textual data [22]. In reality a

word may be related to other words [21]. As such the unigram model has the potential to miss potentially significant dependencies between words; for instance the synonymy relationship. This deficiency has prompted recent research into extending the classical LM framework.

One approach to extending the classic LM framework, and the one which we investigate in this paper, is to consider indirect correspondences between query terms and document terms so that documents can be retrieved even if they don't contain the original query terms. In these approaches a model of relationships between terms is first defined, then when two terms are compared, if there is no direct correspondence this relationship model is consulted to see if an indirect match can be made. How the model of term relationships is implemented allows us classify these extensions to the classical LM framework as practicing either document expansion or query expansion.

Document expansion approaches [22] use the relationship model to enrich the document model so that terms in the relationship model which have related terms in the document are artificially inserted into the document model. In these approaches then matching of a query to a document proceeds as normal except that terms not originally in the document will now also be considered in the evaluation. Query expansion approaches [21] (investigated in this paper) do not manipulate the document model but instead modify the query model, identifying terms in the relationship model that are related to the query terms and then considering these terms when evaluating the query against document models.

3.3 Query Expansion with Term Relationships

More formally we can define query expansion in the LM framework as an extension to classical smoothing techniques. In the classical LM framework for a document to be retrieved given a query that document would have to contain all the query terms. In order to allow documents which contain only some of the query terms to be retrieved the document model or query model can be smoothed in terms of the collection model so that the probabilities of terms not actually in the document or query are increased to some small non-zero value.

Unfortunately this form of smoothing while solving the zero probability problem increases the probabilities of all the non-occurring terms in the document uniformly or proportionally to the term distribution in the whole collection [21]. This means that terms which are actually not related to those already in the document are artificially incorporated into the document model (for example), while those terms that deserve to be incorporated in the model (by their relatedness to terms in the document) receive no special treatment. For example if the term "engine" appears in a document the probability that a query with the term "car" should be matched against that document is higher than if the query contained the term "desk" instead. In this case it is intuitively more reasonable to assign a higher probability to the term "car" due to the relationship between "engine" and "car".

An alternative solution which we investigate here to smoothing the query model is to incorporate terms derived from some explicit model of term relationships. Bai et al. in [21] smooth the original query model $P_{ML}(t_i|Q)$ by another probability function

defined over an explicit model of term relationships $P_R(t_i|Q)$ so that the query model from (3) $P(t_i|Q) = P(q_i|Q)$ now becomes;

$$P(t_i|Q) = \lambda P_{ML}(t_i|Q) + (1-\lambda)P_R(t_i|Q) \tag{4}$$

Given this definition for $P(t_i|Q)$, $Score(Q,D)$ now becomes;

$$Score(Q,D) = \sum_{q_i \in Q} P(q_i|Q) \times logP(q_i|D) \tag{5}$$

$$= \sum_{t_i \in V} [\lambda P_{ML}(t_i|Q) + (1-\lambda)P_R(t_i|Q)] \times logP(q_i|D) \tag{6}$$

$$= \lambda \sum_{q_i \in Q} P_{ML}(q_i|Q) \times logP(q_i|D) + (1-\lambda) \sum_{t_i \in V} P_R(t_i|Q) \times logP(t_i|D) \tag{7}$$

Where λ is a mixture parameter used to control the influence of the two models on the document scoring function and where V is the vocabulary or set of unique terms t_i in the term relationship model. Given this equation for estimating the query model the question becomes how does one define a model of explicit term relationships, that is, how do we define $P_R(t_i|Q)$?

3.4 Hyperspace Analogue to Language

A human encountering a new concept derives its meaning via an accumulation of experience of the contexts in which the concept appears [24]. HAL (Hyperspace Analogue to Language) is a cognitively motivated and validated semantic space model for deriving term co-occurrence relationships from a corpus of text [25]. HAL is significant because the term associations computed by the HAL model correlate with human judgments in word association tasks [24].

HAL represents words/terms/concepts as vectors in a high dimensional space based on lexical co-occurrences. A simple windowing based co-occurrence analysis can be used to construct a HAL space, whereby a window of size l-words is passed in one word increments over the corpus of text. Where two words occur within the window a co-occurrence relationship is defined between them. For an n-word vocabulary this co-occurrence analysis results in an $n \times n$ matrix of co-occurrence relationships. A concept c_i then can be represented a vector drawn from this matrix $c_i = \langle w_{c_i p_1}, w_{c_i p_2}, ... w_{c_i p_n} \rangle$ where $p_1, p_2, ..., p_n$ are called dimensions of c_i and correspond to the other concepts/words from the vocabulary which c_i participates in a co-occurrence relationship with. $w_{c_i p_i}$ is then the weight of p_i in the c_i concept vector [24]. An example HAL vector for the word "HAL" derived from the first paragraph of section 3.4 is given in Fig 1.

Given a HAL vector representation for a concept c_i, a set of quality properties $QP(c_i)$ for that concept can be derived. Quality properties are those properties of the concept which frequently co-occur in the same context as the concept. A property p_i of a concept c_i is declared a quality property iff $w_{c_i p_i} > \partial$, where the threshold ∂ is usually the mean weight for the concept vector [24].

HAL = <analogue:1, appears:1, associations:2, because:2, co-occurrence:1, cognitively:1, computed:2, concept:1, contexts:1, corpus:1, correlate:1, human:1, hyperspace:1, judgments:1, language:1, model:1, motivated:1, relationships:1, significant:2, term:2, text:1, word:1>

Fig. 1. Example HAL Vector

3.5 Information Flows

While the co-occurrence matrix at the heart of the HAL representation can be used directly to make inferences about term relationships, in [26] Song and Bruza propose a more complex model of term relationships based on HAL vectors. The goal of the HAL-based information flow model developed by Song and Bruza is to produce information-based inferences which correlate with inferences made by humans [27].

Given a source term or set of source terms $t_i,...,t_k$ and a target concept t_j there is an information flow from the set of source terms to the target term $t_i,...,t_k|-t_j$ if the former suggest or entails the latter to some degree [21]. The degree of information flow from $t_i,...,t_k$ to t_j is given by $degree(c_1 \triangleleft c_2)$ (the interested reader is referred to [26] for the formal definition of information flow). Essentially information flow measures how many of the quality properties of the source vector are also properties of the target vector [21], that is, the ratio of the intersection of the set of quality properties of c_i and c_j to the number of quality properties in c_i.

3.6 Query Expansion using Information Flows

Using this measure of the degree of information flow between concepts, given a concept or set of concepts in the form of a query, we can compute information flow values for each term in the vocabulary and by imposing a threshold or by selecting a set of the top ranked terms define a set of terms related by information flow to the terms in the query. That is we can use HAL derived information flow to define $P_R(t_i|Q)$.

More formally if we define information flow between terms as a probability:

$$P_{IF}(t_2|t_1) = \frac{degree(c_1 \triangleleft c_2)}{\sum_{t_k \in Vocabulary} degree(c_1 \triangleleft c_2)} \tag{8}$$

then we can define $P_R(t_i|Q)$ as follows:

$$P_R(t_i|Q) = P_{IF}(t_i|Q) = \sum_{Q_i \subseteq Q} P_{IF}(t_i|Q_j) \times P(Q_j|Q) \tag{9}$$

where Q_j can be a single query term or a group of query terms but usually corresponding to the query itself and where $P(Q_j|Q) = \frac{1}{|Q|}$.

To limit the number of term relationships considered we can then define a set of the top ranked IF (Information Flow) relationships E using some threshold and only consider those terms which are part of a relation in E.

$$P_R(t_i|Q) = P_{IF}(t_i|Q) = \sum_{Q_i \subseteq Q \wedge R(t_i,Q_j) \in E} P_{IF}(t_i|Q_j) \times P(Q_j|Q) \tag{10}$$

This definition of $P_R(t_i|Q)$ can then be used to smooth our query model (7):

$$Score(Q,D) = \lambda_{IF} \sum_{q_i \in Q} P_{ML}(q_i|Q) \times logP(q_i|D)$$

$$+(1-\lambda_{IF}) \sum_{Q_i \subseteq Q \wedge R(t_i,Q_j) \in E} P_{IF}(t_i|Q_k) \times P(Q_j|Q) \times logP(t_i|D) \quad (11)$$

3.7 Application to Concept Location

While recognized for their potential importance in assisting software engineer comprehension of unfamiliar systems, it has not been immediately obvious as to how to make use of intent rich non source code artifacts when implementing software comprehension tools or techniques. The Hipikat system [16] being a notable exception in that it directly incorporates non-source code artifacts in its recommendations of pertinent software artifacts related to the concerns software engineers are working on.

However direct and explicit use of non-source code artifacts is only one way in which these artifacts can be used to assist software comprehension. These non-compliable development artifacts can also serve as a repository of term relationships specific to the particular system. The query expansion LM approach, based on the cognitively motivated HAL & IF representation of term relationships discussed in this paper then serves as a principled foundation which allows us to incorporate these previously under utilized development artifacts in combating the concept location problem in software comprehension.

The modification of the basic HAL & IF query expansion LM framework is trivial. Instead of calculating a term relationship model from the source code of a SUS, we generate a HAL space from the non-source code artifacts related to the SUS. This HAL space is then used in combination with IF analysis to generate sets of terms that are potentially related to query terms specified by the user which are then used to smooth the classic LM query model as discussed in this paper. That is we calculate the query model smoothing function $P_R(t_i|Q)$ not from the source code of the system but from other documentation artifacts related to the system. The hypothesis for doing this being that by using natural language non-source code documents to build a model of term relationships we will identify relationships between terms that may not be expressed in source code documents due to the unrestricted nature of natural language documents.

4 Evaluation

To evaluate our technique, we extended our cognitive assignment Eclipse plug-in [28] to incorporate the HAL & IF based query expansion model discussed in this paper. We then conducted a small experiment to quantitatively assess the performance of our technique in terms of precision and recall versus sets of software elements considered by a system expert to be relevant to 4 system concern descriptions. Finally we compared the performance of our technique (QEKLD) against 3 other IR techniques including; a classic LM approach (LM), a dependency based language modeling based approach (DLM) and an LSI implementation (LSI).

4.1 System Under Study

The experiment was performed over the CHIVE software visualization tools framework [29] and associated non-source code documentation. The CHIVE has been employed in the implementation of several software understanding tools [30] and has been in development for over 3 years. The CHIVE core, the framework itself, consists of 7 packages, 25 classes and over 15 KLOC of Java. Finally between the client applications and the framework there is over 40,000 words of academic and technical text documenting CHIVE and its client applications. We chose the CHIVE framework as the basis of this case study because it constitutes a non trivial system with which the authors of this paper are intimately familiar and because both the source code and documentation of CHIVE are publicly available allowing the experiment to be replicated.

4.2 Experiment Procedure

Prior to the experiment we constituted a HAL term relationship matrix from the CHIVE documentation corpus using the freely available AutoMap tool [31], this term relationship model was then loaded into the cognitive assignment Eclipse plug-in. The 4 concern descriptions used in the experiment consisted of 2 feature descriptions, 1 feature request and a bug report. Query term sets for each of the 4 concern descriptions were arrived at based on queries generated manually by participants to an earlier experiment [32] where we used the same concern descriptions.

For each of the 4 concerns the corresponding query term sets were then used to generate a probability ranking for each source code document in the SUS based on the HAL & IF query expansion LM approach discussed in this paper. For this experiment we chose to use the method or function as our document, this decision is in keeping with other research on this topic [10] [6]. We also computed rankings for each document using the other IR techniques listed previously.

5 Results and Analysis

To assess and compare the performance of the various IR techniques, we used the average precision measure of document scoring function performance. Average precision is defined as the mean of the precision scores obtained after each relevant document is retrieved, and measures how accurately a document scoring function ranks a set of known relevant documents. A scoring function that returns more relevant documents with higher rankings will perform better under the average precision measure when compared with a scoring function which does not. For example if we have a set of 10 relevant documents out of a corpus of 100, a scoring function that returned the 10 relevant documents in the top 10 ranking positions would have an average precision of 1 where as a document scoring function which did not rank the relevant documents in the top 10 positions would get an average precision score of < 1. The interested reader is referred to [33] for a more formal explanation. The average precision *v* for a document scoring function is defined over the set of document rankings produced by that function as follows;

$$v = \frac{1}{R} \sum_{i=1}^{n} \frac{x_i}{i} \sum_{k=1}^{i} x_k \tag{12}$$

Where R is the total number of relevant documents in the collection and n is the number of documents included in the ranked document list. And

$$where\, x_i = \left\{ \begin{matrix} 1\, if\, the\, ith\, document\, is\, relevant \\ 0\, otherwise \end{matrix} \right\} \tag{13}$$

$$where\, x_k = \left\{ \begin{matrix} 1\, if\, the\, kth\, document\, is\, relevant \\ 0\, otherwise \end{matrix} \right\} \tag{14}$$

Table 1 presents an analysis of the average precision performances of 4 different document scoring functions against the experiment SUS.

Table 1. Average Precision Analysis

task	LM	DLM	LSI	QEKLD
Task 1	0.084070797	0.266397466	0.464817945	0.527995294
Task 2	0.035807103	0.200502069	0.296435054	0.528178496
Task 3	0.038667382	0.498058764	0.194446073	0.405287366
Task 4	0.029411765	0.076923077	0.5	0.25

Here we see that QEKLD (Query Expansion Kullback-Leibler Divergence - the approach discussed in the paper) significantly out performs the LSI document scoring function in all of the first 3 tasks. In task 1 we see a relatively small 4% difference between the two techniques however in tasks 2 and 3 we see a 23% and 21% difference respectively. The last task (task 4 the bug location task) sees the LSI technique outperform all other techniques by a significant margin. Table 2 presents the mean average precision for each of the techniques.

Table 2. Mean Average Precision

	LM	DLM	LSI	QEKLD
3 MAP	0.052848428	0.321652766	0.318566357	0.487153719
4 MAP	0.046989262	0.260470344	0.363924768	0.427865289

To reduce the distorting impact of task 4 (the bug location task) on the mean average precision analysis we have also calculated an average over the first 3 tasks only. From this analysis we can see that QEKLD is able to outperform LSI on average by between 6% and 17%.

Overall we were pleased with the performance of the QEKLD technique. While the experiment was small the results generated by our technique were, we think, sufficient to have a positive impact on a software engineer attempts to understand the experiment system. In comparison with the LSI technique, while QEKLD was able to out perform it on all tasks (excluding the bug location task), it remains to be seen if an average 10% to 20% difference in average precision has any impact on the performance of a software engineer using a search tool based on the technique. Even given a set of documents ranked with reasonably good recall and precision, an engineer still has to make a decision as to what elements they consider relevant to the task or concern they are attempting to localize. In this respect ranked search only offers suggestions and as such a small difference in precision and recall of one technique vs. another may not have a significant impact. What is potentially more significant is the case where documents which are related to the concern are ranked very low (out side the top 10 or 20 documents) in the result set. In this case it is unlikely that the engineer will identify the document as being related to their concern from search alone and will need to consult other sources of information, it is here that hybrid dynamic analysis - IR approaches such as [7] would likely be of great benefit.

6 Conclusions & Future Work

In this paper we have presented a new approach to assisting concept location in software comprehension. While recent research has seen the application of several IR techniques to the concept location problem, these approaches tend to focus on information derivable from the source code of the system under study alone. While the source code is an incredibly rich and useful source of information produced by software development and maintenance processes it is not the only artifact generated. Engineers have traditionally used other non-source code artifacts to express concepts and concerns which they were either to later express in code or which could not be expressed directly in the code. The approach we have presented is based on a cognitively motivated information retrieval technique which allows us to incorporate information from non-source code artifacts in implementing a ranked search over the source code documents in a system under study. We have presented the background behind our approach as well as its application to software engineering and software comprehension research. We also describe a small experiment we conducted to compare the performance of our technique against some other IR techniques. We show through this experiment that our technique equals or out performs other similar techniques. In future work we intend to perform more and larger experiments comparing our approach with similar approaches against different code bases and document corpuses to generalize our findings and to assess the performance of our technique.

7 Acknowledgments

This research was supported by Lero. Lero is supported by Science Foundation Ireland (under grant no. 03/CE2/I303_1).

References

1. Pennington, N.: Comprehension strategies in programming. presented at Empirical Studies of Programmers: Second Workshop. New Jersey. (1987)
2. Good, J.: Programming Paradigms, Information Types and Graphical Representations: Empirical Investigations of Novice Program Comprehension. University of Edinburgh. (1999)
3. Biggerstaff, T. J., Mitbander, B. G., Webster, D. E.: Program understanding and the concept assignment problem. Commun. ACM. **37** (1994), 72–82
4. Simonyi, C.: Intentional Programming. The Intentional Software Corporation. (2005)
5. Rajlich V., Wilde N.: The role of concepts in program comprehension. Proceedings 10th International Workshop on Program Comprehension. (2002)
6. Zhao, W., Zhang, L., Liu, Y., Sun, J., Yang, F.: SNIAFL: Towards a Static Non-Interactive Approach to Feature Location. Proceedings of International Conference on Software Engineering. Edinburgh. Scotland. (2004)
7. Poshyvanyk, D., Marcus, A., Rajlich, V., Gueheneuc, Y.-G. and Antoniol, G.: Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification. Proceedings of 14th IEEE International Conference on Program Comprehension. (2006)
8. Poshyvanyk, D., Marcus, A., Dong, Y.: JIRiSS - an Eclipse plug-in for Source Code Exploration. Presented at 14th IEEE International Conference on Program Comprehension. (2006)
9. Marcus, A., Maletic, J. I.: Recovering documentation-to-source-code traceability links using latent semantic indexing. Proceedings of 25th International Conference on Software Engineering. (2003)
10. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J. I.: An information retrieval approach to concept location in source code. Proceedings of 11th Working Conference on Reverse Engineering. (2004)
11. Wilde, N., Scully, M. C.: Software reconnaissance: Mapping program features to code. Journal of Software Maintenance: Research and Practice. **7** (1995) 49–62
12. Eisenbarth, T., Koschke, R., Simon, D.: Locating features in source code. IEEE Transactions on Software Engineering. **29** (2003) 210–224
13. Greenfield, J., Short, K.: Software Factories: Assembling Applications with Patterns Frameworks Models and Tools. John Wiley and Sons (2004)
14. Chung, W., Harrison, W., Kruskal, V., Ossher, H., Stanley, J., Sutton, M., P. Tarr.: Working with Implicit Concerns in the Concern Manipulation Environment. Presented at Linking Aspect Technology and Evolution Co hosted with Aspect Orientated Software Development. Chicago. USA. (2005)
15. Robillard, M. P.: Representing Concerns in Source Code. The University of British Columbia. (2003)
16. Cubranic, D., Murphy, G. C., Singer, J., Booth, K. S.: Hipikat: a project memory for software development. Software Engineering, IEEE Transactions on, **31** (2005) 446–465
17. Murphy, G. C., Kersten, M., Findlater, L.: How Are Java Software Developers Using the Eclipse IDE?. IEEE Software **23** (2006) 76–83
18. Manning, C. D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval: Cambridge University Press. (2007)
19. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., Harshman, R.: Indexing by Latent Semantic Analysis, Journal of the American Society of Information Science **41** (1990) 391–407
20. Landauer, T. K., Foltz, P. W., Laham, D.: Introduction to Latent Semantic Analysis. Discourse Processes (1998) 259–248
21. Bai, J., Song, D., Bruza, P., Nie, J.-Y., Cao, G.: Query expansion using term relationships in language models for information retrieval. Presented at 14th ACM international conference on Information and knowledge management. Bremen. Germany. (2005)

22. Gao, J., Nie, J.-Y., Wu, G., Cao, G. Dependence language model for information retrieval. Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. Sheffield. United Kingdom. ACM Press. (2004) 170–177
23. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. Software Engineering. IEEE Transactions on. **28** (2002) 970–983
24. Bruza, P. D., Song, D.: Inferring query models by computing information flow. Proceedings of the eleventh international conference on Information and knowledge management. McLean. Virginia. USA. ACM Press. (2002) 260–269
25. Lund, K., Burgess, C.: Producing high-dimensional semantic spaces from lexical co-occurrence. Behavior Research Methods. Instruments and Computers. (1997) 203–208
26. Song, D., Bruza, P.: Discovering information flow suing high dimensional conceptual space. Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval. New Orleans. Louisiana. United States. ACM Press. (2001) 327–333
27. Song, D., Bruza, P.: Towards Context-sensitive Information Inference. Journal of the American Soceity for Information Science and Technology (JASIST). **4** (2003) 321-334
28. Cleary, B., Exton, C.: The Cognitive Assignment Eclipse Plug-in (ICPC 06). Presented at Internation Conference on Program Comprehension. Athens. Greece. (2006)
29. Cleary, B., Exton, C.: CHIVE - a program source visualisation framework. Presented at 12th IEEE International Workshop on Program Comprehension. Bari. Italy. (2004)
30. LeGear, A., Cleary, B., Buckley, J., Collins, J. J., Exton, C.: Making a Reuse Aspectual View Explicit in Existing Software. Presented at Linking Aspect Technology and Evolution Co hosted with Aspect Orientated Software Development (ASOD 05). Chicago. USA. (2005)
31. Carley K. M., Diesner, J. AutoMap1.2 - Extract, analyze, represent, and compare mental models from texts. Carnegie Mellon University. School of Computer Science. Institute for Software Research International. Technical Report CMU-ISRI-04-100. (2004)
32. Cleary, B., Exton, C.: Assisting Concept Assignment using Probabilistic Classification and Cognitive Mapping. Presented at 2nd International Workshop on Supporting Knowledge Collaboration in Software Development (KSCD2006). Tokyo. Japan. (2006)
33. Kishida, K.: Property of Average Precision and its Generalization: An Examination of Evaluation Indicator for Information Retrieval Experiments. Technical Report. National Institute of Informatics. Tokyo. Japan. NII-2005-014E (2005)