Using Semantic Networks and Context in Search for Relevant Software Engineering Artifacts

George Karabatis[1], Zhiyuan Chen[1], Vandana P. Janeja[1], Tania Lobo[1], Monish Advani[1], Mikael Lindvall[2], and Raimund L. Feldmann[2]

[1] Department of Information Systems, University of Maryland, Baltimore County (UMBC)
1000 Hilltop Circle, Baltimore, MD 21250, USA
[2] Fraunhofer USA Center for Experimental Software Engineering
4321 Hartwick Rd., College Park, MD 20742, USA

Abstract. The discovery of relevant software artifacts can increase software reuse and reduce the cost of software development and maintenance. Furthermore, change requests, which are a leading cause of project failures, can be better classified and handled when all relevant artifacts are available to the decision makers. However, traditional full-text and similarity search techniques often fail to provide the full set of relevant documents because they do not take into consideration existing relationships between software artifacts. We propose a metadata approach with semantic networks[1] which convey such relationships. Our approach reveals additional relevant artifacts that the user might have not been aware of. We also apply contextual information to filter out results unrelated to the user contexts, thus, improving the precision of the search results. Experimental results show that the combination of semantic networks and context significantly improve the precision and recall of the search results.

Keywords: software engineering, search for artifacts, semantic networks, context.

1 Introduction

In the domain of software engineering software changes are inevitable, for example, due to requirements change, but cause several well-known problems if not handled properly. They can lead to severe time pressure and project delays due to underestimation of the scope of the change. Major studies of today's software intensive systems consistently find surprisingly large numbers of failed, late, or excessively expensive systems [36] and according to [29] requirement change is one of the most common causes of software project failure.

Thus, searching for relevant software development artifacts (requirements documents, design documents, source code, etc.), has become increasingly important. For

[1] Semantic networks are graphs which represent knowledge by interconnecting nodes through edges. They have been used to describe and classify concepts for many centuries. According to Sowa the earliest known semantic network was drawn in the third century AD by the Greek philosopher Porphyry (Porfyrios) to graphically illustrate the categories of Aristotle (Sowa 1992). For a detailed background on semantic networks see (Sowa).

S. Spaccapietra, L. Delcambre (Eds.): Journal on Data Semantics XIV, LNCS 5880, pp. 74–104, 2009.

example, software developers often need to find out whether there are some similar software components or software designs to better respond to a software change request. Finding such related information may greatly reduce the cost of software change or allow more accurate estimation of the cost of the change, which may lead to better decisions (e.g., whether to accept or reject the change request). One study found that the amount of necessary software changes was three times higher than originally predicted [45], indicating that a better search technology may be one possible solution to this problem. We also believe that our technique can also help in the development of new software.

Importance of Capturing Ad hoc Relationships: There are several obstacles to finding relevant artifacts, especially in the domain of software engineering. First, the relationships between artifacts are often ignored by existing full-text or similarity search technologies [9, 24, 1], but are extremely important for finding relevant software artifacts. For example, many software projects have various versions (thus, these versions are related), a large number of requirements documents (related to the code that implements them), and often implement overlapped functionality (thus, different projects are related to each other). However, it is extremely difficult for someone to find relevant artifacts if that person is unfamiliar with the software project structure and its history including its relationships to other software products and their evolution.

Motivating Example: The following example describes some of these difficulties focusing on the ones that arise during a search for relevant information that would be triggered for example, by a change request. Fig. 1 shows the relationships between two closely related software projects: The Tactical Separation Assisted Flight Environment (TSAFE) and the Flight Management System (FMS). The arrows are directional and indicate that the source node is related to the target node and the number identifies the degree of relevance between the two connected nodes. Details of these two projects can be found in Section 5.

Suppose a developer of FMS version 5 receives a change request to add the capability to change geographical area in run time. The developer tries to find artifacts related to the keyword "FMS 5". Certainly, an existing text search tool such as Google Desktop or a search tool that matches artifacts with similar attributes could be used. Unfortunately, related artifacts such as TSAFE II Loss Of Separation (LOS) (which is functionally equivalent to FMS 5), TSAFE II (without the LOS option), TSAFE II Dynamic Map (which implements the sought functionality in a different project TSAFE), and the requirement document for Dynamic Map (which is the design document for the sought functionality) are unlikely to be retrieved. The reason is that these artifacts do not contain the keyword FMS. In order to overcome this problem, one could make TSAFE a synonym of FMS or define a similarity score between these two terms. However, in this context, FMS has two meanings: Flight Management System and Finance Management System (an accounting system). Thus, a search for TSAFE would retrieve artifacts for both the Flight Management System and Finance Management System. In conclusion, the design information related to TSAFE II DM, which implements the sought functionality, cannot be retrieved without extensive searching due to its distant and indirect relationship to FMS 5 (see Fig. 1).

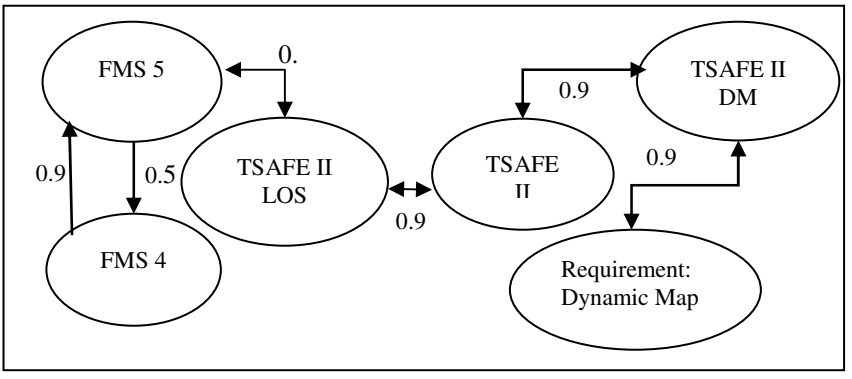

Fig. 1. A (partial) semantic network for Air Traffic Control Software

Our Approach and Contributions: We use two techniques to solve this problem: Semantic networks and context. Next, we give a brief description of these two techniques. We use semantic networks to capture ad hoc relationships between artifacts. Fig. 1 shows a partial view of a semantic network in our example. The nodes represent artifacts and the links represent relationships between artifacts. The number on each edge quantifies the degree of relevance (i.e., the strength of the relationship) of two artifacts. We also infer indirect relevance scores between two indirectly linked artifacts. For example, the relevance score between FMS 5 and TSAFE II DM equals the product of relevance scores between FMS 5 and TSAFE II LOS, TSAFE II LOS and TSAFE II, and TSAFE II and TSAFE II DM.

Now, given a search for information pertinent to FMS 5, we can use the semantic network to add relevant artifacts. A full-text search engine (or a similarity search engine) returns FMS 5 as result. We can then expand the results by adding any artifacts whose relevance score to FMS 5 exceeds a certain user defined threshold (let's say 0.7). This threshold identifies the degree of semantic closeness (relevance score) between related artifacts in the semantic network. For example, only artifacts which are relevant with a relevance score above threshold t are included in the semantic network recommendations. Thus the results will include FMS 5, TSAFE II LOS, TSAFE II, TSAFE II DM, and Requirement: Dynamic Map.

Although semantic networks recommend semantically relevant answers, these answers can fit the user's query more precisely if additional information about the user, the project, the environment, etc. were to be provided. We use context, which contains information relevant to the user (such as user's roles and the current and/or previous projects) to enable the semantic network to target the user queries more accurately. For example, some users may be only interested in requirements or design documents (since they may not be programmers and can not read source code). In our approach, we store different types of context information and use it to filter the results generated by semantic network. In the above example, the context associated with that user, is that the answer must be requirement documents, thus only the requirement for Dynamic Map is returned, which is exactly what the user wants.

We consider each artifact to be a node associated with certain characterizing attributes. We begin by constructing a first mode semantic network, referred to as Similarity based Network, which is a graph that identifies the similarity between the nodes in terms of the features associated with each node. We next utilize semantic rules to augment the first mode network and discover a second mode network, which we call Rule Enforced Network. Although both semantic networks and context have been used individually in search [26] and other applications, there are three important differences in this paper.

First, we use semantic networks to model the ad hoc relationships between artifacts while most existing work such as WordNet [26] and concept maps [74] model relationships between keywords. We find that in many domains such as software engineering, the relationships between artifacts are often difficult to model at the keyword level. For example, it is very difficult to model the relationship between a software release and its subsequent releases at the keyword level. On the other hand, it is straightforward to add such links between artifacts: a simple rule can be created to add such links automatically.

Second, this paper combines semantic network with context. Using a semantic network alone may improve the recall of search results because more artifacts are returned. However, it may not improve precision of the search results because not all these additional artifacts may be considered pertinent by the user. This paper uses context to filter out irrelevant artifacts based on existing contexts. We have conducted experiments which show that the combination of semantic network and context leads to better precision and recall.

Third, there has been little work on searching for software engineering artifacts, which has become increasingly important for the software industry – especially in regard to the increasing number of change requests and extended life cycles of today's software products. To the best of our knowledge, we are not aware of any study that uses both semantic networks and context in the software engineering domain. Specifically our contributions are as follows:

1. We automatically construct a semantic network from artifacts and their associated characterizing attributes.
2. We keep a provision for adding external semantic rules supplied by a domain expert, that when applied to the semantic network, they augment and enhance it.
3. We automatically find, using our semantic network, not only the requested artifacts based on a user query, but additional relevant ones that the user might have not been aware of.
4. We apply context on the result set of the user query to enhance the quality of search over artifacts and include only contextually related artifacts.
5. We demonstrate through experiments, using software artifacts from a software test bed, that the combination of semantic networks and context significantly improve both the precision and recall of search results.

Although this paper focuses on the software engineering domain, we believe the proposed approach is suitable to other domains too. The rest of the paper is organized as follows: Section 2 describes related work and Section 3 gives preliminaries on semantic networks and context. In Section 4 we describe our approach, and in Section 5 we present and discuss validating experiments. Section 6 concludes the paper.

2 Related Work

Related work on software change and reuse. The problem of determining software change impact has a long history. Haney's early model for module connection analysis predicts the propagation of change among modules of a system [33]. It assumes that a change in one module is likely to cause change in other modules. A probability connection matrix subjectively models the dependencies between modules. Our approach is different from Haney's in that we do not only model dependencies between modules, but between all artifacts carrying design information as well as relationships between projects, and we use context.

Many useful theoretical models for impact analysis and change-related processes are collected in the excellent overview by [11]. Different approaches to identify change are described; for instance, traceability analysis for change propagation between work-products, ripple-effect analysis for propagation within work-products, and change history analysis to understand relationships to previous changes. Many approaches address reuse of various artifacts [62, 57, 61]. These approaches make use of metadata (i.e., tags) to describe the artifacts, which are used to classify [57, 63] and retrieve them [56, 7].

Latent Semantic Indexing (LSI), an information retrieval technique, has been used to recover links between various artifacts [47, 50] that share a significant number of words (not necessary the words being searched). However, in the software engineering domain the usefulness of this approach is sometimes limited. As described in our Motivating Example, the critical keywords TSAFE and FMS do not appear in the same artifacts and the description of the change request uses different terminology. Furthermore, the similarities between two artifacts are symmetric in LSI, which is often not true in practice. For example, given a software release, the next and newer release is likely more interesting than the previous one. Since semantic networks do not require the existence of such shared keywords and they do not require that similarities are symmetric, our approach does not have these limitations.

Canfora et al. describe the state of the art of information retrieval and mention three areas in which information retrieval is applied to software maintenance problems: Classifying maintenance request (i.e., change request), finding an expert for a certain maintenance request, and identifying the source that will be impacted by a maintenance request [15]. For example, in [48] the authors use various technologies such as Bayesian classifiers to classify maintenance requests. In [5, 18] [54] the authors determine who is the expert for a certain change request based on who resolved a similar change request in the past based on data from version control systems and try to identify similar change requests from the past. The main difference with our approach is that we model distant relationships connecting projects and artifacts that are similar, but would most likely never show up using similarity-based searchers. The impact on source code from a certain change request has been studied in [14] by correlating change request descriptions with information provided in version management systems such as Bugzilla.

Related work on semantic networks. Semantic Networks have been used in philosophy, psychology and more recently in information and computer science. Sowa gives a descriptive outline on the types and use of semantic networks in different

disciplines in [70, 69]. Semantic networks have long been used to represent relationships [51]. Pearl used probabilities in semantic networks and performed extensive work in applying statistics and probability in causal semantic networks [59, 58] to derive such networks from observed data.

There has also been work on discovering semantic similarity in [22] based on generalization/specialization, and positive/negative association between classes; the topic of discovering and ranking semantic relationships for the semantic web is also relevant [3, 67]. Our work is also linked to the specification of relationships among database objects stored in heterogeneous database systems [38, 28, 68, 65]. We have used semantic networks to enhance the results of a user query in different application domains such as the environmental [17] and e-government of water quality management [16]. However these early approaches do not support automated creation of the semantic network and do not incorporate context as part of the solution. We are now applying our approach in the domain of software engineering: see [43] for an early and initial approach describing a limited scope of this problem in software engineering. Quite related is the work on ConceptNET a large scale concept base which captures common sense concepts [46].

Related work on context. A significant part of scientific literature is related to the use of context information related disciplines and in the social sciences such as psychology and sociology. Pomerol and Brezillon examined notions of context and identified its forms as external, contextual, and proceduralized [60]. Bazire and Brezillon made an analysis on 150 definitions of context found on the web, and concluded that the definition of context depends on the field of knowledge it belongs to [8]. For a comprehensive examination of an operational context definition see [78] and for context definitions in artificial intelligence, databases, communication, and machine learning see [13]. Lieberman and Selker presented context in computer systems and described it as "everything that affects the computation except the explicit input and output" [42]. There is related research performed within the scope of data integration and interoperability using context [37, 76, 30, 23]. Context has also been used as an aid in defining semantic similarities between database objects [39].

Sowa provides an overview on facts and context in [71]. Context has been used in multiple settings: Semantic knowledge bases utilize a partial understanding of context; WordNet is such an example, where context is expressed in natural language [26]. It has also been used to provide better algorithms for ranked answers by incorporating context into the query answering mechanism [2] and to improve query retrieval accuracy [66]. Significant work on context-sensitive information retrieval was performed in [66, 72, 35, 27]. However, we focus on how to take context into consideration when using semantic networks. Graphs that represent context have been used to provide focused crawling to identify relevant topical pages on the web [20]. Methods to model and represent context for information fusion from sensors using relational database model are described in [77]. Context has also been used to prune semantic networks to improve performance, by marking and thus using only nodes which are pertinent in specific contexts [31]. In addition, graphs were used by [55] to infer the context and fit it into an existing semantic network. However, in our work we keep the semantic networks separate from context, and we avoid automated inference of context. Finkelstein et al. identify the difficulties in automatic extraction of context,

especially with text, as documents may be large, could contain multiple concepts, and could inject a lot of noise [27]. We have decided to collect context either by observing user actions, or explicitly by the user, an approach which is also used in [41] where the context is used by the system; however it differs from our approach as it they use it to perform a rewrite of the original user query, whereas we apply a filtering technique on the expanded result.

Related work on the semantic web and ontologies. A significant amount of work on semantics and the meaning of concepts has been done for the semantic web [10, 75]. The Web Ontology Language OWL [53] has been used to model concepts of a domain that are to be shared by others providing a relevance to the concept of semantic networks. McCarthy introduced the *ist(C,p)* predicate to disambiguate when a proposition *p* is true in context *C* [52] and in [32] the authors adapt the *ist* construct of context and address the contextual problems which arise in aggregation on the semantic web. The restrictions of the standard OWL specification, such that it allows neither directionality of information flow, nor local domain (which is of utmost importance for contexts), nor context mappings, are overcome by extending the syntax and the semantics of OWL via explicit context mappings [12]. The notion of relationships between concepts is also related to the topic maps or concept maps [74]. The major thrust of our work is to create a methodology that utilizes semantic networks and contextual information to support software engineers in their search of relevant artifacts. It can be implemented in a variety of ways:

- As a stand-alone system, as we present in this paper
- On the web, using semantic web technologies, such as OWL and RDF [4]
- In a combination of the above two techniques

The concepts presented in this paper can also be adapted and implemented on the semantic web, for example, expressing relationships using OWL. However, such effort is not within the scope of this paper, but we plan to investigate semantic web technologies in the future.

3 Preliminaries

In this Section we provide an introduction to some topics and notation that are being used in the remainder of the paper, around the concepts of semantic networks and context.

Artifacts. In our software engineering setting, we assume that each artifact is associated with metadata represented as a set of attribute-value pairs. For example, the FMS version 5 has the following attributes: Name = Flight Management System Version 5, Type = Code, Programming Language = Java. In addition to these attributes the artifacts can be parsed to derive additional attributes. For instance in a Java program, import statements, function names etc. also provide valuable information about the artifact and can indeed be used as attributes describing the artifacts.

Semantic networks. A semantic network represents ad hoc relationships among artifacts.

Definition 1 [Semantic Network]*. A Semantic Network N(V, E) is a directed graph where V is a set of nodes and E is a set of edges. Each node corresponds to an artifact, and each edge links two relevant artifacts v_i and v_j and has a score $w(v_i,v_j)$ in the range of 0 to 1, representing the degree of relevance between the source artifact and the destination artifact.*

Fig. 1 illustrates a partial semantic network for the FMS and the TSAFE projects. The network contains knowledge of multiple people, e.g., an individual programmer of TSAFE may not know the relationships of artifacts in FMS, and vice versa, but a software architect may know that TSAFE is related to FMS, although the software architect may not know in detail the relationships between the artifacts within each system. That is, each of them only has knowledge of a part of the semantic network. However, based on the relevance scores between neighboring nodes in the network, it is possible to infer the relevance between any two nodes (as far apart as FMS 5 is to TSAFE II DM – see Fig. 1). Thus, one can discover more semantically related information compared to individual knowledge. Next we define the relevance score between any two artifacts in the network.

Definition 2 [Relevance Score]*. If v_i and v_j are two nodes in a semantic network N(E,V), there are k paths $p_1,\ldots, p_k$ between v_i and v_j, where path p_l ($1 <= l <= k$) consists of nodes $v_{l1},\ldots, v_{l|pl|+1}$ ($|p_l|$ is the length of path p_l). The relevance score rs as defined by [17] between v_i and v_j is*

$$rs = \max\left(\prod_{1 \le i \le |pl|} w\left(v_{l_i}, v_{l_{i+1}}\right) \right)$$

The above formula computes the relevance score between v_i and v_j as the maximum relevance score of all paths connecting v_i and v_j. The relevance score of such a path is computed using conditional probabilities under the assumption that they are independent.

For instance, the relevance score between 'FMS 5' and 'TSAFE II DM' can be considered as the conditional probability of a software developer interested in the TSAFE II DM given that the developer is interested in the related product line FMS 5. Using the standard notation for conditional probability, we have:

$$P(\text{TSAFE II DM} \mid \text{FMS 5}) = P(\text{TSAFE II DM, TSAFE II, TSAFE II LOS} \mid \text{FMS 5})$$

because the developer considers that TSAFE II DM and FMS 5 are related if all artifacts on the path from FMS 5 to TSAFE II DM (TSAFE II LOS and TSAFE II) are considered to be related. Using chain rules and assuming all conditional probabilities are independent [64], we have:

$$P(\text{TSAFE II DM, TSAFE II, TSAFE II LOS} \mid \text{FMS 5}) =$$

$$P(\text{TSAFE II LOS} \mid \text{FMS 5}) \times P(\text{TSAFE II} \mid \text{TSAFE II LOS}) \times P(\text{TSAFE II DM} \mid \text{TSAFE II}) = 0.9 \times 0.9 \times 0.9 = 0.73.$$

Thus, a developer receiving a change request for FMS 5 and using the semantic network, would be able to find relevant artifacts such as FMS 4, TSAFE II LOS, TSAFE II, LOS Detector requirement, TSAFE II DM, and Requirement: Dynamic Map. Note that we can easily specify "not related" information in the semantic network by simply not adding a link between them. For example, there shall be no link between the Finance Management System and TSAFE. We also assume that relationships between nodes do not have to reflect the same attribute. By design, we just need to have any relationships established between the nodes of the graph, and we do not necessarily need to have probabilities of the same attribute to calculate paths.

Context. We consider context to be significantly important in the search for semantically related information as every single search is performed within a specific context. Although this context may not explicitly appear in the query terms, nevertheless it does exist, and the user expects the system to provide information relevant to this context. In general, users may not be aware of context when they first search for information. However they become cognizant of context when they receive results that are irrelevant or not applicable to the current context, i.e., when a search for FMS returns Finance Management System. A highly beneficial characteristic of our system is that it takes advantage of context in a transparent way to filter and return the most appropriate answers tailored to each user. We consider the following four types of context:

- *User Context* contains information specific to users such as the role of the user (e.g., developers or design analyst), the programming language skills, etc.
- *Application Context* contains information about the application or project the user is working on, such as the name and type of project, etc.
- *Environment Context* includes information about the environment around the user, such as the organization the user belongs to, the operating system of the user's computer, etc.
- *Other Context* is used as a place holder for additional contextual information which does not fit in any of the previous context categories, but still is relevant to the domain of discourse.

Note that there are additional categories of context which do exist, such as security considerations, policies, etc., which are not captured in our system. We acknowledge that it is unlikely to capture all possible types of context and their values in a computer system, since there will always be additional information contributing to context. We limit ourselves to collecting information about the above categories of context, and we do not claim that we can capture all possible context types. For an extensive work on an operational definition of context see [78]. In the domain of software engineering, we claim that such context information is relatively easy to collect as it was the case in our experiments and described below, and assume will be similar in most software engineering settings. Unlike domains such as generic search on the Internet where users submit ad hoc queries and want to find answers immediately, the users in software engineering domain are typically software developers, analysts, project managers, etc., who are regular users of the system and are more willing to provide contextual information in return for more precise search results. User contexts can be gathered by asking the users about past project experience, or by

contacting their manager, etc. Application contexts can be obtained by asking the project managers. Environment contexts that are relatively static (i.e., the name of an organization) can be obtained easily, while those that are volatile (i.e., the current software version information) may need additional effort to collect and maintain. However, maintaining contexts falls outside the scope of this paper as we assume that the various types of contexts are already collected, stored, and maintained in a database. Formally a context can be represented in the following format.

Definition 3 [Context]. *The context $C(U_j, T)$ of type T for user U_j, where $T \in$ {User, Application, Environment, Other} is represented as a conjunctive normal form $\wedge_i(L_{i1} \vee L_{i2} \vee \cdots \vee L_{ini})$ where each L_{ij} is an attribute value pair or its negation .*

For example, the user context of a user who is a design analyst (i.e., interested in design and requirements documents) and does not know C++ is: $(\text{Type} = \text{Design} \vee \text{Type} = \text{Requirements}) \wedge \neg(\text{Programming Language} = \text{C++})$.

From a systems viewpoint, context is metadata information stored in database tables and it is used in conjunction with semantic networks as follows: Artifacts represented as nodes in semantic networks contain characterizing attributes, which may participate in the attribute value pairs of a context definition. These attributes link semantic networks and context. Relevance related information comes from semantic networks, and in turn is pruned by context-related information through the attribute value pairs. It is important to note that semantic networks and context are somehow orthogonal dimensions, but both use the attributes of the artifacts. Further details are described in Section 4.3.

4 Approach

Our approach consists of the following distinct steps: we first provide a high level overview of the major components of our system and the lifecycle of a user query through the system. Then we present details on the creation of a semantic network: we first derive the universal feature vector which has all the potential attributes across the set of artifacts. Based on the vector a feature vector for each artifact is generated. We utilize the similarities between these feature vectors to generate a similarity based network. This is further enhanced by semantic rules to generate a Rule enhanced network. We also define relevance scores between artifacts. Subsequently we define the context for the semantic network for a more refined result set. Lastly we apply transformation functions on this semantic network. The approach is discussed in the following subsections: Section 4.1 the system overview and the lifecycle of a user query. Section 4.2 describes how we construct semantic networks. Section 4.3 discusses how to use context in our system. Finally, Section 4.4 describes a framework of transformation functions which formalize our overall approach.

4.1 System Overview

In this section we outline the system architecture and the flow of a query from the time it is submitted until the results are returned back to the user.

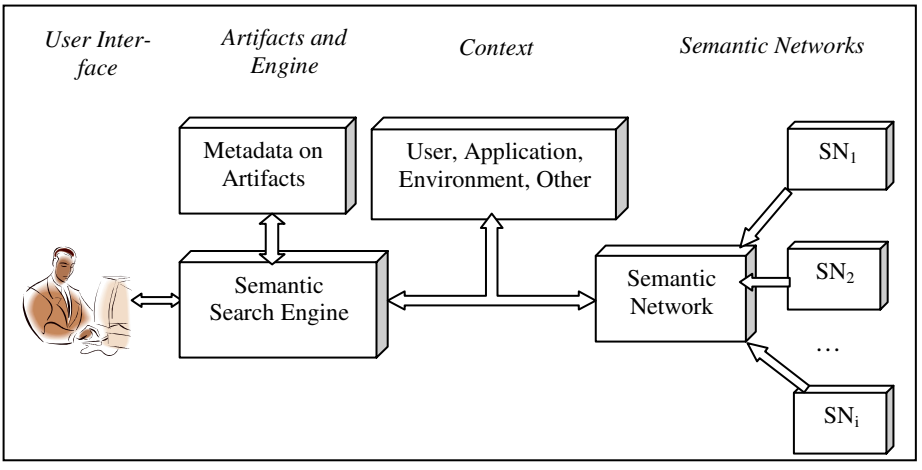

Fig. 2. Conceptual system architecture

A high-level conceptual architecture of our system with its major components is illustrated in Fig. 2. Our prototype system has been implemented using an Oracle database in which we store all types of metadata about software artifacts (attribute-value pairs), semantic networks, and context. Our system stores only metadata, e.g., an identifier (such as a uri) pointing to the location of each actual artifact. A set of semantic networks is depicted to the right part of Fig. 2, each representing a separate software project. All these semantic networks are merged into a larger Semantic Network, which integrates the individual semantic networks into a consolidated one. The edges connecting these networks identify the existence of a potential relationship between them. The strength of this relationship is represented as a relevance score. Another component of our system contains information about the different types of contexts that are collected (User, Application, Environment, etc.) and it is used to identify semantically related information and filters out irrelevant information. Context has been implemented as a set of tables in an Oracle database. Metadata about software artifacts is also stored in the database to be used in the extraction of the initial artifacts based on the user query. The Semantic Search Engine interacts with the users and all major components of our system. It oversees all operations at each component, from the submission of a user query, to its execution, the use of semantic networks and contexts, all the way to displaying the final results to the user.

Semantic networks and context information significantly improve the quality of the query result, since: (1) they enhance the result set with semantically relevant information that the users might not be aware of, and (2) they incorporate contextual knowledge to streamline the result according to user, application and other contexts. We demonstrate the improvement in quality by measuring recall and precision of the results (see Section 5). In this Section we present the lifecycle of a query submitted by a user to our system as illustrated in Fig. 3. Initially, a text search is performed to

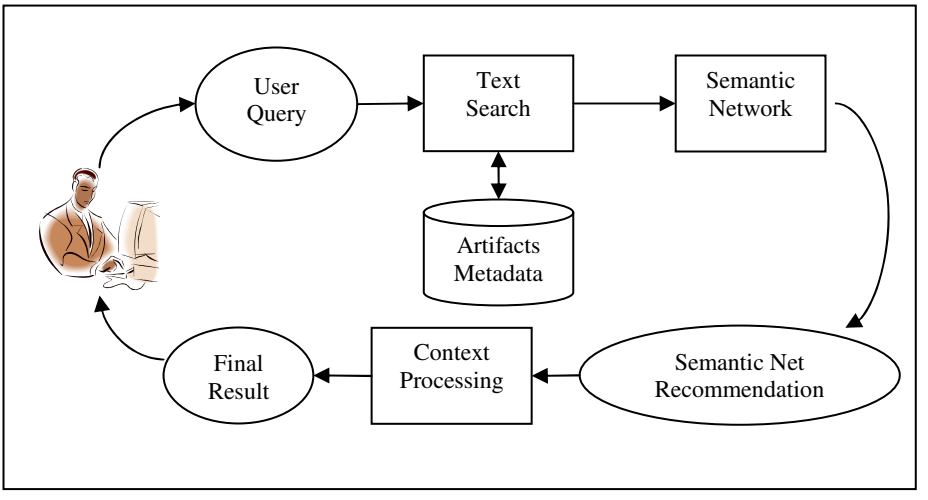

Fig. 3. Life cycle of a user query

collect all related information which matches the user query. This search is performed on a set of metadata about the artifacts[2]. Then, the returned artifacts are given as input to the semantic network. Using the algorithms described in detail in section 4.2 and depending on the value of the user-defined threshold on the relevance score, the semantic network produces an augmented list of recommended artifacts, whose relevance scores to any of the initial artifacts exceed the threshold and thus are semantically relevant to the initial user query. However, this augmented list may not reflect the contextual information pertaining to the user, project, etc. Consequently, the contexts are used next, to filter information accordingly. As a result, only the recommended artifacts which are pertinent to the contexts will be collected and given to the user in the final result set of the original query. For example, assume that a requirement analyst asks a query on "automated collision avoidance," the system first performs a full-text search and returns all artifacts from the database containing these keywords. At this point, the current result set may not contain all relevant artifacts as there could be additional artifacts that are semantically relevant but which are not included in the search results. Then, the system utilizes the semantic network to find all additional artifacts which are semantically related to the current search results; i.e. additional artifacts that do not contain the search keywords explicitly, but are closely related to them (e.g., the Loss of Separation Detection Module, which detects situations where two aircrafts are too close). However, the augmented results containing all semantically relevant artifacts may not be pertinent to the user's context. Therefore, contextual information is extracted from the database and is applied to the set of augmented results to filter out artifacts that are out of context keeping only those that are within context. In our example, only requirement documents (but not source code) are kept in the final result.

[2] In this paper, whenever we refer to artifacts we mean the metadata about the artifacts and not the artifacts themselves.

Of course we allow users to evaluate the recommended artifacts and they have the ability to accept/reject each one of them. They can also fine-tune the search query, resubmit to the semantic network and possibly provide a different threshold for the relevant artifacts until they are presented with recommended and contextualized artifacts to their satisfaction.

4.2 Construction of Semantic Networks

To avoid the daunting task of manually constructing and maintaining the semantic network, we adopt an approach for the construction of semantic networks in an automatic manner, consisting of two layers (first mode and second mode network). The first mode network identifies relevant artifacts based on similarity, whereas the second mode network is build on top of the first mode and enhances it by adding semantic information. As shown in Fig. 4, each node in the semantic network represents an artifact and part of the metadata for this artifact is a set of attributes which describe the artifact. We utilize the similarity between the attributes of the artifacts to construct the semantic network. We construct it as follows.

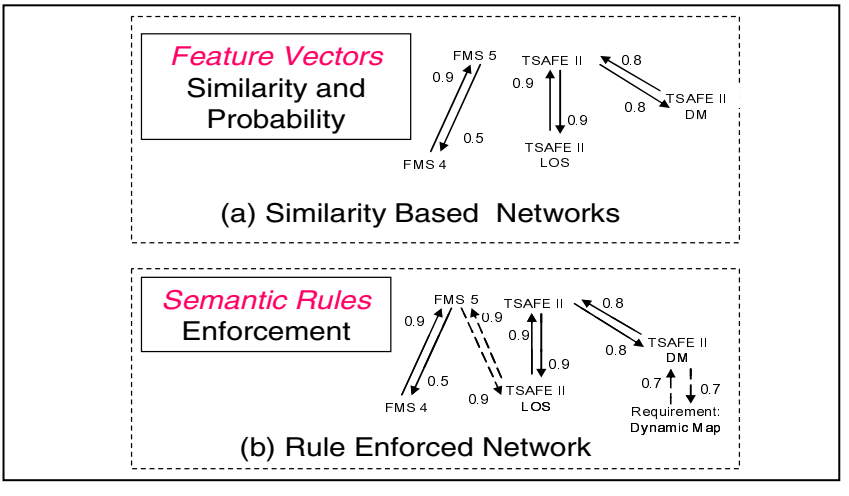

Fig. 4. Creating a semantic network

Automatic Generation of Feature Vectors

Let $X = \{x_1,...x_n\}$ be the set of artifacts, where each $x_i \in X$ is associated with a set of characterizing attributes $a_i = \{a_{i1},....a_{im}\}$. The values of these attributes can be transformed into categorical values (binary) and form a feature vector $f_i = \{f_{i1}, \ldots, f_{im}\}$. In order to automatically create a similarity based network we first need to generate the feature vector associated with each artifact. Our approach is generalizable to continuous attributes such that they can be discretized into categorical variables. Additionally we can also handle textual variables since we can parse the features from code files. These feature vectors are used to determine how similar the artifacts are in terms of

the attributes characterizing them and they are utilized to generate the semantic network; therefore, we outline our process and algorithm here for the generation of feature vectors. We discover features by class of artifacts. For instance, software artifacts can be viewed as programs, requirement specifications, test cases and so on. For each such class of artifacts we produce a universal feature vector by creating a parser for the artifacts. This can be seen as a preprocessing step necessary to acquire the data about the artifacts. In the case of Java programs, using a text file parser, we extract features such as import statements and function names, and we add them to the feature vector. The union of all feature vectors creates a universal vector *(V)* containing all artifacts. Thus $V=\{ a_{11} \cup a_{21},....\cup a_{im} \}$ and it will be used for the similarity based network.

Algorithm 1 The Feature Vector generation algorithm

Require: Set of artifacts X where each $x_i \in X$
Ensure: Set of artifacts X where each $x_i \in X$ is associated with a set of attributes $a_i = \{a_{i1},..,a_{im}\}$ and a set of features $f_i = \{f_{i1},..,f_{im}\}$

```
 1: for i = 1 to |X| do
 2:   {Read artifact x_i}
 3:   {a_i ← Parse Attributes(x_i)}
 4:   {add(U, a_i)}
 5: end for
 6: {Initialize Feature Vector f}
 7: for i = 1 to |X| do
 8:   for z = 1 to |U| do
 9:         if u_z ∈ x_i then
10:               {f_iz = 1}
11:         else
12:               {f_iz = 0}
13:         end if
14:   end for
15: end for
```

Algorithm for Automatic Feature Vector Generation. We outline our algorithm for the generation of the Feature Vector. The algorithm takes as an input the set of artifacts $X=\{x_1,...., x_n\}$. On lines 1-5 we generate a universal vector U, by parsing through the artifacts. This essentially finds all the attributes from the various artifacts, from x_1 to x_n, and stores them in U. So for instance if we are parsing a java program then the import statements will be the attributes of the Universal vector. We then create a feature vector from U on lines 7-15. We parse the artifacts to note the presence or absence of an attribute in the artifact. For example if we have a Java program artifact with an import java.util statement then the feature in the vector for this artifact

will have a value 1 vs. another java program without the import statement will have a value 0 for the feature. The parser can be modified to handle other types of languages such as c++, python etc. Programming languages provide a structured environment to handle such a parsing. However documents may not be parsed easily using this method since their structure is not very well defined. The complexity of the algorithm is O(N |U|) where N is the number of artifacts and |U| is the size of the Universal Vector.

Similarity Based Network

Let us assume that we have a set of n artifacts $X = \{x_1,...x_n\}$, where each $x_i \in X$ is associated with a set of m features captured in a feature vector $f_i = \{f_{i1}, \ldots, f_{im}\}$. We use a Jaccard similarity coefficient[3] to quantify the similarity among the feature vectors of the artifacts. Based on the Jaccard coefficients we connect similar nodes using edges and start creating the semantic network. We add probabilities on the edges as follows: given a pair of nodes x_p and x_q such that there exists a similarity between the two nodes the probability $w(xp , xq)$ of traversing from node x_p to x_q is:

$$w(x_p, x_q) = \frac{J_{pq}}{deg_p} \quad \text{where } deg_p = \sum_{j=1}^{k} J_{pj}$$

J_{pq} is the Jaccard similarity coefficient between the feature vectors of artifacts (nodes) x_p and x_q. J_{pj} is the weighted degree of the node p, and k is the number of incident edges on p. Thus, based on the similarity and probability computations we get a first mode semantic network as shown in Fig. 4(a), which we refer to as *Similarity based Network*. There could be several disconnected first mode semantic networks as shown in Fig. 4(a). The probabilities are shown close to the tip of each edge. We formally define the first mode Semantic Network as follows:

Definition 4 [Similarity based Network]. *Let $X = \{x_1,...x_n\}$ be the set of artifacts, where each $x_i \in X$ has a feature vector $f_i = \{f_{i1}, \ldots, f_{im}\}$ then a first mode Similarity based Network $N^{sn}(V^{sn}, E^{sn})$ is a directed graph where V^{sn} is a set of nodes and E^{sn} is a set of edges, such that $V^{sn} \subseteq X$ and $|V^{sn}| \leq |X|$, and each edge links two relevant artifacts $<v_i, v_j>$ and has a probability score $w(v_i,v_j)$ where $0 < w(v_i,v_j) \leq 1$.*

Rule Enforced Network

The automatically created first mode networks reflect similarity based on the feature vectors of each artifact but they do not include any additional semantic information. For example, there could be strong relevance between two nodes representing files from different projects, but because some attributes in the feature vectors (e.g. the name) are completely different, the Jaccard similarity coefficients may not rank them similar enough to create an edge between them. Such semantic knowledge is usually captured in the minds of experienced users, and it can be described in terms of *semantic rules* that explicitly identify connectivity between two nodes in the semantic

[3] A Jaccard similarity coefficient (Jaccard index) measures the similarity of sets and is defined as the size of the intersection divided by the size of the union of the sample sets.

network. This is required in two scenarios first the two nodes that were not deemed to be similar according to similarity measures (although they are similar indeed), second, there may be a situation where two nodes have a high similarity as per the similarity measures but have a low similarity. We define a semantic rule as follows:

Definition 5 [Semantic Rule]. *Given two artifacts x_p and x_q a semantic rule r is defined as r: x_p, x_q, w(xp , xq) where w(xp , xq) is the probability score associating the two artifacts.*

When these semantic rules are enforced, they add edges connecting nodes on the first mode semantic network(s), thus, they augment the network. The probabilities on the new edges are also calculated and the result is the second mode semantic network as shown in Fig. 2(b), which we refer to as *Rule enforced Network*. The new edges are depicted as dashed arrows. When multiple experts with similar roles create the same rule connecting two edges, we add a link to the network having as relevance score the average probability of all occurrences of the rule. When experts with different roles create new rules it is possible that these rules would expand the network in completely different directions. In such cases, we do not try to consolidate these rules into a single network, but we create separate networks each one specific to a role. We formally define this second mode Semantic Network.

Definition 6 [Rule enforced Network]. *Given a first mode Semantic Network $N^{sn}(V^{sn}, E^{sn})$, where V^{sn} is a set of nodes and E^{sn} is a set of edges in N^{sn}, and a set of semantic rules R, a second mode rule enforced Semantic Network $N^{re}(V^{re}, E^{re})$ is a directed graph where V^{re} is a set of nodes and E^{re} is a set of edges such that $V^{re} \subseteq X$, $|V^{re}| \leq |X|$ and $|V^{re}| \geq |V^{sn}|$, and each edge links two relevant artifacts $<v_i, v_j>$ and has a probability score $w(v_i,v_j)$ where $0 < w(v_i,v_j) \leq 1$.*

The probability scores encompass the similarity between the features of each artifact and the semantic rules enforced on the network. Such a network contains knowledge of multiple people, e.g., an individual programmer of TSAFE may not know the relationships of artifacts in FMS, and vice versa, but a software architect may know that TSAFE is related to FMS, although the software architect may not know in detail the relationships between the artifacts within each system. However, based on the relevance scores between neighboring nodes in the network, we can infer the relevance between any two nodes (as far apart as FMS 5 is to TSAFE II DM – see Fig. 1). Thus, one can discover more semantically related collective information compared to individual knowledge. If a semantic rule links two nodes that are already connected in the previously created similarity network, the semantic rule link replaces the similarity link (the expert's opinion supersedes the feature based similarity).

Algorithm for Automatic Semantic Network Generation

Once we have the feature vectors we then use the Jaccard coefficient to quantify the similarity among the feature vectors of the artifacts. We use the Jaccard coefficient since it does not give importance to a positive dissimilarity of features (marked as 0-0 in bits identifying that there is no similarity between two features that do not match) but gives importance to a positive match (1-1) and positive mis-match(1-0). We outline

the approach to identifying the similarity of the feature vectors in Algorithm 2. The complexity of the algorithm is $O(N^2|U|)$ where N is the number of artifacts and |U| is the size of the Universal Vector.

Algorithm 2 The Similarity based Network generation algorithm

Require: Set of artifacts X where each $x_i \in X$ is associated with a set of attributes $f_i = \{f_{i1}, .. , f_{im}\}$

Ensure: Similarity based Network $N^{sn}(V^{sn}, E^{sn})$ where V^{sn} is a set of nodes and E^{sn} is a set of edges, each edge links two relevant artifacts $< v_i, v_j >$ and has a probability score $w(v_i, v_j)$

```
 1: jc=0
 2: deg=0
 3: for i = 1 to |X| - 1 do
 4:   for j = i + 1 to |X| do
 5:        jc_ij = jc_ji = JC(f_i, f_j)
 6:         deg_i = deg_i + jc_ij
 7:         deg_j = deg_j + jc_ji
 8:  end for
 9: end for
10: for p = 1 to n do
11:   for q = 1 to n do
12:        w(xp , xq)  ←  jc_pq/deg_p
13:        if w(xp , xq)  < W_threshold then
14:              { w(xp , xq)  = 0}
15:        end if
16:    end for
17: end for
```

4.3 Using Context

We store context in relational tables. One table stores user context, with columns user ID, project ID, role of user, programming language, etc. A second table stores application context, including project ID, functionalities, etc. A third table stores environmental context, including user ID, operating system, organization name, etc. After these tables have been initialized, we create a mapping table to map information in these tables to conditions on attribute-value pairs over the artifacts. For example, if the user's role is developer, we map it to the condition: Type = Code $\vee$ Type = Requirement as a developer needs to read both code and requirements.

We can then combine all context information of a user into a single filtering condition. This condition is the conjunction of all conditions mapped from the context information of a user. For example, a user's filtering condition may be:

(Type = Code $\vee$ Type = Requirement) $\wedge$ (Programming Language = JAVA) $\wedge$ (Project = Flight Control) $\wedge$ (Operating System = LINUX)

At run time, this condition is used to filter the artifacts returned by the full-text search and semantic network. This step checks the attribute-value pairs of a returned artifact, and if any of those attributes in the artifact appears in the filtering condition, the value of that attribute will be checked against the filtering condition. If the value violates the condition, the artifact will be pruned. For example, if an artifact with Programming Language = C++ is returned, this artifact violates the above filtering condition and will be pruned.

Note that if an attribute of an artifact does not appear in the filtering condition, no check will be done and the artifact will remain in the result. For example, if programming language is not specified in an artifact (e.g., when the artifact is a design document), then this artifact will not be pruned based on the condition on programming languages.

4.4 Transformation and Composition Functions

It is quite intriguing to evaluate the effect of applying context during the different phases of the query lifecycle. For example, is it better to apply context before using semantic networks or after? Can we apply context both before and after using the semantic network? Questions like this might affect greatly the artifacts that will be retrieved and we investigate answers to these questions in this Section.

Each user query submitted to our system undergoes a series of transformations as it passes through its various phases and completes its cycle though our system (Section 4.1). During each of these different phases a transformation function is applied to a specific input available in the current phase, and produces a specific output applicable to the next phase. For example, extracted keywords of the initial user query are used as input to a function f_{MAS}, which conducts a Metadata Artifacts Search (MAS) and produces as its output a result containing artifacts R_A. Formally,

Definition 7 [Metadata Artifacts Search Function]. *Assume that Q_A is a set of keywords of a user query, and A is the domain of all artifacts. The function f_{MAS} is the Metadata Artifacts Search function which takes as input Q_A and produces as output a set of artifacts R_A .*

$$f_{MAS}(Q_A) = R_A \quad (\text{alternatively } Q_A \xrightarrow{f_{MAS}} R_A), \text{ where } R_A \subset A.$$

In a similar fashion we define two more transformation functions: f_{SN} and f_C which apply the semantic network techniques and the context techniques respectively. Therefore we have:

Definition 8 [Semantic Network Transformation Function]. *The function f_{SN} applies the input R_A through a semantic network and produces as output a set of related artifacts R_{SN} .* $f_{SN}(R_A) = R_{SN}$ $(\text{alternatively } R_A \xrightarrow{f_{SN}} R_{SN}),$ *where* $R_A, R_{SN} \subset A.$

Definition 9 [Context Transformation Function]. *The function f_C filters the input R_{SN} utilizing the appropriate context $C(U_j, T)$, and produces as output a set of filtered artifacts R_C .* $f_C(R_{SN}) = R_C$ $(\text{alternatively } R_{SN} \xrightarrow{f_C} R_C),$ *where* $R_{SN}, R_C \subset A.$

Definition 10 [Lifecycle Composition Function]. *A lifecycle composition function L of a user query in our system is a composition of the transformation functions f_{MAS}, f_{SN}, and f_C defined as L : f_C (f_{SN},(f_{MAS}(Q_A)))= R_C. Alternatively, L* $: f_C \circ (f_{SN} \circ f_{MAS}) :$ $Q_A \xrightarrow{f_{MAS}} R_A \xrightarrow{f_{SN}} R_{SN} \xrightarrow{f_C} R_C$ *where R_A, R_{SN},, $R_C \subset A$.*

We have used the above transformation functions in a specific order to compute the final result R_C, as a composition of functions: $f_C \circ (f_{SN} \circ f_{MAS})$. Nevertheless, there are different ways that we can order the transformation functions and create a different composition. For example, $L_1 : f_{SN} \circ (f_C \circ f_{MAS})$ is another composition where the context function f_C is applied before the semantic network function f_{SN}. It is interesting to examine whether we obtain the same results depending on the order of the transformation functions in the function composition. In general, the composition of functions is not a commutative operation, i.e., $f_C \circ (f_{SN} \circ f_{MAS}) \neq f_{SN} \circ (f_C \circ f_{MAS})$. In practice, we can apply the transformation functions in different orders depending on how we want the process to take place, we can even apply the same transformation function multiple times. For example, it makes sense to apply the context function f_C before *and* after the semantic net function f_{SN}, having a new lifecycle composition function $L_2 : f_C \circ (f_{SN} \circ (f_C \circ f_{MAS}))$. We discuss the different options (L, L_1, and L_2) in our next Section where we describe our experiments.

5 Experiments

We first describe the setup of our experiments in section 5.1. In Section 5.2 we present our results for the automatic creation of the semantic network. Next we discuss the experiments with context and without context and in Section 5.3 we present the results. We use recall and precision as our basic measures according to the definitions of [73] and [6]. We also describe our prototype system in the Appendix.

5.1 Setup of Experiments

We used two test-beds of two software projects each:

(1) The *Tactical Separation Assisted Flight Environment* (TSAFE) and the *Flight Management System* (FMS). These two software projects are based on a specification for Automated Air Traffic Control by NASA [21], implemented by MIT [19] and turned into a test-bed at Fraunhofer Center, Maryland [44]. This test-bed makes a good fit for the proposed research for two reasons. First, it contains two parallel threads of implementations of similar functionality. Second, historical design information exists for all variants and versions of TSAFE and FMS. There are as many as 38 different versions of each project, making the total number of artifacts more than 250, not counting the source code class files. The different versions of TSAFE and FMS are related, making reuse possible but not straightforward. Valuable design information can be retrieved; however, the different versions and amount of existing

data makes finding such design information difficult using the current full text search system.

(2) The second test bed consists of information from two different software projects; we selected 82 artifacts from DMGroup1 and 76 artifacts from LosGroup3. These two projects implemented similar functionality. We asked three domain experts to create an initial semantic network to use it as a baseline on these test beds. The relationship between two files in different projects (DMGroup1 and LosGroup3) receives a weight of 1 if these two files are deemed similar or a weight of 0 if they are dissimilar to each other. The metadata of each artifact includes the name of the artifact, the type of the artifact (requirement document, code, etc.), programming language, impact analysis (the impact of a specific software change to the various phases of software maintenance) , design pattern (a blueprint that can be applied to provide a solution to a commonly occurring problem), etc.

5.2 Automatic Semantic Network Generation

Creation of Feature Vectors

To create the feature vectors we used the second test bed with a set of 158 Java files from two different projects (82 files from DMGroup1 and 76 files from LosGroup3) as an input to a Java program. We used this test bed since we wanted to specifically evaluate the similarity among files across different projects. These files are compared with each other based on 4 main characteristics: java import statements (150), package names (120), class names (120) and method names (almost 300). A universal feature vector is automatically generated with a set of 680 attributes identifying characteristics which are unique across these 158 files. Next we compare each of these characteristics in every other file from two different projects to find whether they are similar or not. This similarity is captured in a similarity matrix which maps the similarity of each file with all other files across different projects. If two files from different projects are similar based on a Jaccard similarity coefficient and our weight computation then we mark the matrix location with a 1 otherwise with a 0. The similarity threshold for this task was set to 0.8. Different threshold values produce different results as described in Section 5.3.1. Source and edges along with weights are stored in an Oracle database which is subsequently used to build the tree structure.

Evaluation and Validation of the Automatically Created Network

The domain experts review the files from two different projects and label the similarity weights as 0 or 1. If they find similar files they give the weight 1, if the files are not similar they label them with a weight 0. Their concept of similarity is purely based on the manual evaluation of the artifacts and no specific features are considered. For the evaluation of the automatic network creation we consider this similarity provided by the domain experts as our labeled data where the domain expert provides a weight to the pairs of artifacts. Since the domain experts view is absolute numeric value of 0 or 1 we devised a method to check whether we did find similar files using our approach. We compared one artifact from one project with all the other artifacts in a different project and the one which has maximum similarity weight based on our approach was checked against the one provided by the domain experts as having the maximum similarity weight of 1.

	Probability from Rule Enforced Network		
Probability from Domain Experts		***Max similarity***	***No similarity***
	Max Similarity	144 (TP)	3 (FN)
	No similarity	10 (FP)	0 (TN)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad \text{.....} (1)$$

Fig. 5. Performance evaluation using Class labels

Based on this we validate against the labeled data and find the Accuracy of our method. Using the values from table in Fig. 5 we can compute the accuracy as shown in Equation 1. Thus, Accuracy = (145 + 3)/ (145+10+0+3) = 148/158 = 0.93*100 = 93%. From a set of 158 files taken from two projects (DMGroup1 and LOS Group3). Domain Experts found 148 files highly similar out of these 158 and our approach found 145 similar out of 158. Max Similarity is 145 i.e. artifacts which were found to be highly similar by our approach and the domain experts. Out of 158, there are 10 false positives where the domain experts found no similarity but our approach found some similarity, In addition we found 3 files which were highly similar which the domain experts did not identify.

5.3 Experiments with Context

For this set of experiments we used the data from the TSAFE/FMS test-bed. We collected eight queries from the domain experts. For each query, we also created eight different contexts by assuming a certain type of user (user context), a certain type of project (application context), a certain type of programming language (user context and/or application context), and a certain type of platform (environment context). Thus there are altogether 64 combinations of queries and contexts. The domain experts provided us with the correct answers to those queries. We compare the precision and recall of three search algorithms:

1. Using the normal full-text search algorithm without semantic network or context. We used Oracle's full-text search feature for this algorithm (referred to as No-Network in Fig. 6-9)
2. Using semantic network but not context (referred to as Network in Fig. 6-9)
3. Using both semantic network and context (referred to as Network+Context in Fig. 6-9)

5.3.1 Results

An important parameter in our approach is the threshold ***t*** for relevance score in the semantic network. We experimented using the default and also a varying threshold. Fig. 6 and 7 report the average recall and precision of all three algorithms using the default setting ***t*** = 0.8.

The x-axis identifies the queries, while the y-axis presents the value of recall (Fig. 6), and precision (Fig. 7), both averaged over the eight different contexts for each query. The results show that using a semantic network produces a much higher recall than not using a semantic network (see Fig. 6). This is expected because the semantic network returns artifacts that may not contain searched keywords, but are related to the artifacts containing those keywords.

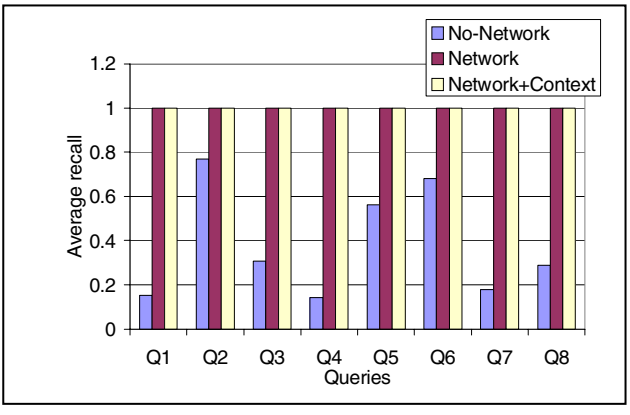

Fig. 6. Recall when threshold = 0.8

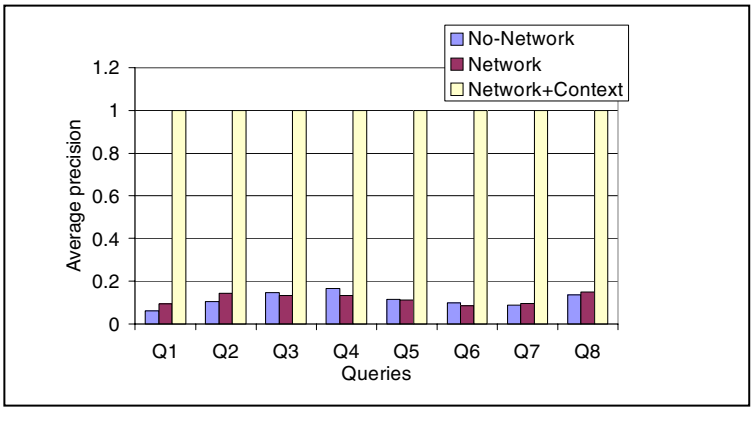

Fig. 7. Precision when threshold = 0.8

The results also show that the use of context increases precision because the context information is used to filter out results not relevant to the user (Fig. 7). In general, using both the semantic network and context leads to higher precision and recall for all eight queries (the recall and precision values at 1 occur due to the relatively small size of the data set).

Next we varied the relevance score threshold ***t*** in the semantic network. Fig. 8 and 9 report the average recall and precision over the eight queries when ***t*** varies from 1 down to 0.2. Note that No-Network does not use semantic network and it is interpreted as ***t*** being fixed at 1. Thus, the recall and precision of No-Network do not change with ***t***.

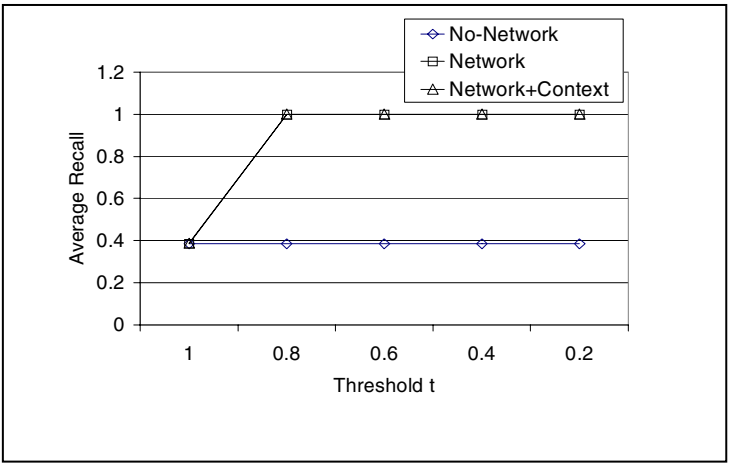

Fig. 8. Average recall with varying threshold

The results show that as the threshold decreases, the recall of Network and Network+Context increases (see Fig. 8). This occurs because it signifies that the user is willing to accept less relevant artifacts in the result set, leading to a higher number of results, when the semantic network is used. The recall of not using a semantic network is very low (about 0.4) compared to the other two methods because it only considers artifacts contained in the searched keywords. The recall values of using Network or Network+Context are always the same because Network+Context would filter out answers from the results created by using semantic networks. In practice, missing a relevant artifact means that the project team may miss the opportunity of reusing existing code; or come up with a wrong estimate of the cost of implementing a change request, which may be very costly. Thus these results clearly show the value of using semantic networks, as they bring additional relevant artifacts in the result set.

As the threshold decreases, the precision of both Network and Network+Context starts to decline (see Fig. 9) when threshold values are below 0.8. As threshold decreases below 1.0 but is still quite high (say, 0.8), artifacts which are very closely related to the answers in the full-text search are returned, and are considered as correct answers; thus, the precision remains high. However, as the threshold further decreases, artifacts that are not very closely related are returned. Thus, the precision starts to decline. This suggests that using a relatively high threshold (we use 0.8) would ensure both high precision and recall. Of course, if recall is very important (e.g., the cost of missing a relevant artifact is very high) a lower threshold can be used to ensure high recall, but with possibly lower precision.

The results also show that the use of context and semantic network always leads to higher precision than using semantic network alone, because context helps filter out irrelevant answers. Using Network+Context also has higher precision than No-Network over a wide range of threshold values (actually for all the threshold values we tested), and with a much higher recall as shown in the previous figures. It clearly displays the benefits of using a combined approach of semantic networks and context.

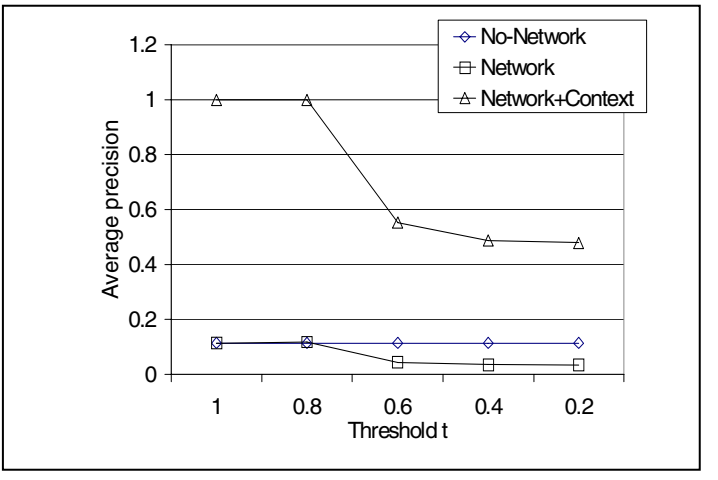

Fig. 9. Average precision with varying threshold

Experimenting with Different Lifecycle Composition Functions

The experiments we just described correspond to the lifecycle composition function L (see Section 4.5). We also performed another set of experiments using the alternate lifecycle composition functions L_1 (with context applied only before the semantic network $f_{SN} \circ (f_C \circ f_{MAS})$) and L_2 (with context applied before and after the semantic network $f_C \circ (f_{SN} \circ (f_C \circ f_{MAS}))$) and compared the results with those of L (with context applied only after the semantic network $f_C \circ (f_{SN} \circ f_{MAS})$). Fig. 10 and 11 report the precision and recall of these three composition functions.

The results show that L (using context after the semantic network) and L_2 (using context-based filtering both before and after using semantic network) produced the same results (both recall and precision) for six out of the eight queries. Query Q3 and Q7 were exceptions. For those two queries, when we applied context before the semantic network, it did not return any answer, resulting in the lowest recall and precision. The reason was that the direct hits (the results after full text search but without the semantic network) were actually "out-of-context". However, these direct hits were related to the correct answers that were in "in-context". Thus using L_2 the system filtered out the direct hits and did not return any correct answer. On the other hand, using L the system still used all the direct hits to find relevant artifacts through the semantic network, thus correct answers were still returned.

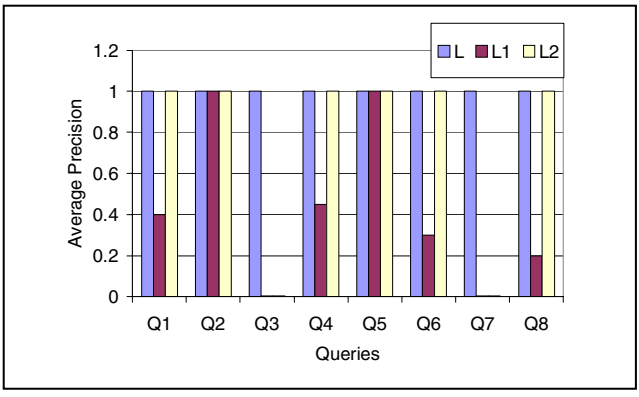

Fig. 10. Precision with varying composition functions

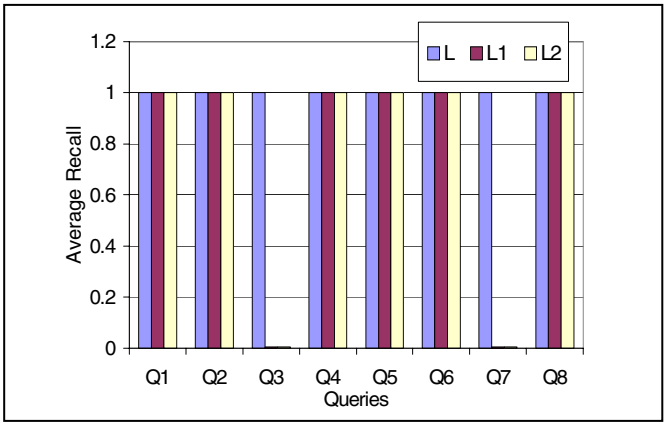

Fig. 11. Recall with varying composition functions

Both L_1 and L_2 apply context based filtering before the semantic network. However, L_2 applies context again after using the semantic network. Since using context after the semantic network does not eliminate any artifacts in the correct answer (i.e., matching the context), L_1 and L_2 have the same recall. The results also show that L_1 (using context-based filtering before semantic network) leads to lower precision than both L_2 and L. This is expected as the use of the semantic network augments the results with semantically related artifacts. But checking the context filtering condition before the use of the semantic network does not guarantee that the augmented results are "in context". For example, one of the contexts precludes source code for managers; still L_1 returns source code related to requirement documents which are in the correct answer. This also exemplifies the property of non-commutativity of the lifecycle composition function, $f_C \circ (f_{SN} \circ f_{MAS}) \neq f_{SN} \circ (f_C \circ f_{MAS})$.

In general, if the contexts transformation function f_C is used just as a filtering condition, then it is advisable to apply f_C after the semantic network transformation function f_{SN} has produced results. The reason is that this process will return artifacts that are connected to those initial artifacts that match the query, but do not match the context. Thus, in such cases the default composition function L should be used. Of course, if the contexts are used in other ways, e.g., to expand the query results or to modify a possible ranking function for displaying results, then it may make sense to use context transformation function f_C before the semantic network transformation function f_{SN}.

User Studies

We conducted two studies regarding the searching behavior of users. First, we carried out an experiment about search tools' performance [49], which confirmed our main hypothesis that retrieving information is difficult, especially for subjects unfamiliar in the domain of the search. Concepts, acronyms, and company lingo were information that most such subjects lacked in order to find the relevant information. In contrast, subjects with some company experience did not experience these problems. Based on these results, we developed and evaluated a prototype search tool that automatically manipulates the search query adding synonyms, acronyms, and abbreviations, increasing the relevance of the search results substantially.

Secondly, we performed another study [25], and we surveyed members of two small IT organizations, one in the US and another one in Germany regarding their search behavior. The results showed that more than 80% of the subjects used only one to four search keywords for their queries.

6 Conclusions

We created a set of tools and technologies which use metadata to assist software engineers in their search for software artifacts. Semantic networks capture the semantic relationships between software artifacts; these networks help return additional artifacts that are semantically relevant to the search, which would not have been included in the original search results using traditional database techniques. We provided an automated way to create the similarity based semantic network and described two algorithms towards its creation. Once the semantic network is built, it can be enhanced with semantic rules; subsequent user queries take advantage of the relationships that are represented in it. This technique produces an augmented result set of the user query, relevant to the original search, thus improving the recall of the search results. However, this augmented and relevant artifact set, may not be tailored to the appropriate contexts of the particular user. Therefore, we employ techniques to filter out "out-of-context" results, and return only "in-context" artifacts pertaining to the user. As a result, the precision is also improved.

We applied our techniques in a software engineering environment with software engineering projects. We performed experiments on real life software projects with the help of domain experts, measuring precision and recall, by comparing full-text, semantic network only, and a combined use of semantic networks and context methods. The

results demonstrated that our metadata techniques are promising and they produce a better recall and precision results compared to the other methods.

In the future, we plan to investigate under which circumstances it would be better to use different orderings of transformation functions, by creating various lifecycle composition functions. In addition, during the automatic creation of the semantic network, the size of the universal vector may explode quickly as each artifact may potentially contribute new features to the universal vector U. Thus, to control the high dimensionality of U we will need to assign weights to the features which we defer to future work. Additionally in the current work we use a Jaccard similarity coefficient; however, in the future we plan to investigate other similarity coefficients such as Matching, Tanimoto, Cosine and Dice coefficients [34, 40] and compare them to Jaccard similarity coefficient.

Acknowledgements. This work was partially supported by an award from the US National Science Foundation (SGER 0738898) and by a grant from Northrop-Grumman Corporation.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational Issues, Methodological Variations, and System Approaches. Artificial Intelligence Communications 7(1), 39–59 (1994)
2. Agrawal, R., Rantzau, R., Terzi, E.: Context-sensitive ranking. In: ACM SIGMOD International Conference on Management of data, Chicago, IL, USA, pp. 383–394 (2006)
3. Aleman-Meza, B., Halaschek-Wiener, C., Arpinar, I.B., Ramakrishnan, C., Sheth, A.P.: Ranking Complex Relationships on the Semantic Web. IEEE Internet Computing, 37–44 (2005)
4. Antoniou, G., Hermelen, F.v.: A Semantic Web Primer, p. 238. The MIT Press, Cambridge (2004)
5. Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: International Conference on Software Engineering (2006)
6. Baeza-Yates, R.A., Ribeiro-Neto, B.A.: Modern Information Retrieval. ACM Press / Addison-Wesley (1999)
7. Basili, V.R., Rombach, H.D.: Support for Comprehensive Reuse. IEEE Software Engineering Journal 6(5), 303–316 (1991)
8. Bazire, M., Brezillon, P.: Understanding Context Before Using It. In: Dey, A.K., Kokinov, B., Leake, D.B., Turner, R. (eds.) CONTEXT 2005. LNCS (LNAI), vol. 3554, pp. 29–40. Springer, Heidelberg (2005)
9. Bergmann, R., Göker, M.: Developing Industrial Case-Based Reasoning Applications Using the INRECA Methodology. In: Workshop at the International Joint Conference on Artificial Intelligence, IJCAI - Automating the Construction of Case Based Reasoners, Stockholm (1999)
10. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (2001)
11. Bohner, S.A., Arnold, R.S.: Software Change Impact Analysis. IEEE Computer Society Press, Los Alamitos (1996)
12. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing ontologies. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 164–179. Springer, Heidelberg (2003)

13. Brezillon, P.: Context in Human Machine Problem Solving: A Survey. LAFORIA (1996)
14. Canfora, G., Cerulo, L.: Impact analysis by mining software and change request repositories. In: International Software Metrics Symposium, METRICS 2005 (2005)
15. Canfora, G., Cerulo, L., Penta, M.D.: Relating software interventions through IR techniques. In: International Conference on Software Management (2006)
16. Chen, Z., Gangopadhyay, A., Holden, S., Karabatis, G., McGuire, M.: Semantic Integration of Government Data for Water Quality Management. Journal of Government Information Quarterly – Special Issue on Information Integration 24(4), 716–735 (2007)
17. Chen, Z., Gangopadhyay, A., Karabatis, G., McGuire, M., Welty, C.: Semantic Integration and Knowledge Discovery for Environmental Research. Journal of Database Management 18(1), 43–67 (2007)
18. Cubranic, D., Murphy, G.C.: Automatic bug triage using text categorization. In: International Conference on Software Engineering & Knowledge Engineering, Banff, Alberta, Canada, pp. 92–97 (2004)
19. Dennis, G.: TSAFE: Building a Trusted Computing Base for Air Traffic Control Software. MIT, Cambridge (2003)
20. Diligenti, M., Coetzee, F., Lawrence, S., Giles, C.L., Gori, M.: Focused Crawling Using Context Graphs. In: 26th International Conference on Very Large Data Bases, Cairo, Egypt, pp. 527–534 (2000)
21. Erzberger, H.: Transforming the NAS: The Next Generation Air Traffic Control System. In: 24th International Congress of the Aeronautical Sciences (2004)
22. Fankhauser, P., Kracker, M., Neuhold, E.J.: Semantic vs. Structural Resemblance of Classes. SIGMOD Record 20(4), 59–63 (1991)
23. Farquhar, A., Dappert, A., Fikes, R., Pratt, W.: Integrating Information Sources using Context Logic. In: AAAI Spring Symposium on Information Gathering from Distributed Heterogeneous Environments (1995)
24. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From Data Mining to Knowledge Discovery: An overview. In: Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press (1996)
25. Feldmann, R.L., Rech, J., Wenzler, A.J.: Experience Retrieval in LSOs: Do you find what you are looking for? In: 8th International Workshop on Learning Software Organizations (LSO 2006), Rio de Janeiro, Brazil (2006)
26. Fellbaum, C.: WordNet: An Electronic Lexical Database, p. 423. MIT Press, Cambridge (1998)
27. Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., Ruppin, E.: Placing Search in Context: the Concept Revisited. In: WWW, pp. 406–414 (2001)
28. Georgakopoulos, D., Karabatis, G., Gantimahapatruni, S.: Specification and Management of Interdependent Data in Operational Systems and Data Warehouses. Distributed and Parallel Databases 5(2), 121–166 (1997)
29. Glass, R.: Agile Versus Traditional: Make Love, Not War. Cutter IT Journal, 12–18 (2001)
30. Goh, C.H., Bressan, S., Madnick, S., Siegel, M.: Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. ACM Transactions on Information Systems 17(3), 270–293 (1999)
31. Gong, L., Riecken, D.: Constraining Model-Based Reasoning Using Contexts. In: IEEE International Conference on Web Intelligence, WI 2003 (2003)
32. Guha, R., McCool, R., Fikes, R.: Contexts for the Semantic Web. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 32–46. Springer, Heidelberg (2004)

33. Haney, F.M.: Module Connection Analysis - A Tool for Scheduling Software Debugging Activites. In: AFIPS Joint Computer Conference, pp. 173–179 (1972)
34. Haranczyk, M., Holliday, J.: Comparison of Similarity Coefficients for Clustering and Compound Selection. Journal of Chemical Information and Modeling 48(3), 498–508 (2008)
35. Joachims, T.: Optimizing search engines using clickthrough data. In: ACM SIGKDD international conference on Knowledge discovery and data mining, Edmonton, Alberta, Canada, pp. 133–142 (2002)
36. Johnson, J.H.: Micro Projects Cause Constant Change. In: 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering, pp. 132–135 (2001)
37. Karabatis, G.: Using Context in Semantic Data Integration. Journal of Interoperability in Business Information Systems 1(3), 9–21 (2006)
38. Karabatis, G., Rusinkiewicz, M., Sheth, A.: Interdependent Database Systems. In: Management of Heterogeneous and Autonomous Database Systems, pp. 217–252. Morgan-Kaufmann, San Francisco (1999)
39. Kashyap, V., Sheth, A.: Semantic and schematic similarities between database objects: a context-based approach. The VLDB Journal 5(4), 276–304 (1996)
40. Kaufman, L., Rousseeuw, P.J.: Finding Groups In Data: An Introduction To Cluster Analysis. Wiley-Interscience, Hoboken (2005)
41. Kraft, R., Maghoul, F., Chang, C.C.: Y!Q: Contextual Search at the Point of Inspiration. In: ACM International Conference on Information and Knowledge Management, Bremen, Germany, pp. 816–823 (2005)
42. Lieberman, H., Selker, T.: Out of context: Computer Systems that adapt to, and learn from, context. IBM Systems Journal 39(3&4), 617–632 (2000)
43. Lindvall, M., Feldmann, R.L., Karabatis, G., Chen, Z., Janeja, V.P.: Searching for Relevant Software Change Artifacts using Semantic Networks. In: 24th Annual ACM Symposium on Applied Computing SAC 2009, Honolulu, Hawaii, U.S.A., pp. 496–500 (2009)
44. Lindvall, M., Rus, I., Shull, F., Zelkowitz, M.V., Donzelli, P., Memon, A., Basili, V.R., Costa, P., Tvedt, R.T., Hochstein, L., Asgari, S., Ackermann, C., Pech, D.: An Evolutionary Testbed for Software Technology Evaluation. Innovations in Systems and Software Engineering - a NASA Journal 1(1), 3–11 (2005)
45. Lindvall, M., Sandahl, K.: How Well do Experienced Software Developers Predict Software Change? Journal of Systems and Software 43(1), 19–27 (1998)
46. Liu, H., Singh, P.: ConceptNet: A Practical Commonsense Reasoning Toolkit. BT Technology Journal 22(4), 211–226 (2004)
47. Lormans, M., Deursen, A.v.: Can LSI help Reconstructing Requirements Traceability in Design and Test? In: Conference on Software Maintenance and Reengineering, CSMR 2006 (2006)
48. Lucca, G.D., Penta, M.D., Gradara, S.: An approach to classify software maintenance requests. In: International Conference on Software Maintenance, Los Alamitos, CA (2002)
49. Lydie, Y.T.M.: Context-Based Information Retrieval -User Problems and Benefits of Potential Solutions, Technical Report. FC-MD (2006)
50. Marcus, A., Maletic, J.I.: Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing. In: 25th International Conference on Software Engineering, ICSE 2003 (2003)
51. Masterman, M.: Semantic message detection for machine translation, using an interlingua. NPL, 438–475 (1961)

52. McCarthy, J.: Notes on formalizing context. In: International Joint Conference on Artificial Intelligence (IJCAI), Chambéry, France, pp. 555–560 (1993)
53. McGuinness, D.L., Harmelen, F.v.: OWL Web Ontology Language Overview W3C (2004), http://www.w3.org/TR/owl-features/
54. Mockus, A., Herbsleb, J.D.: Expertise browser: a quantitative approach to identifying expertise. In: International Conference on Software Engineering, New York, NY, pp. 503–512 (2002)
55. Mylopoulos, J., Cohen, P., Borgida, A., Sugar, L.: Semantic Networks and the Generation of Context. In: International Joint Conference on Artificial Intelligence, Tiblisi, Georgia, pp. 134–142 (1975)
56. Ostertag, E., Hendler, J., Prieto-Diaz, R., Braun, C.: Computing similarity in a reuse library system: An AI-based approach. ACM Transactions on Software Engineering and Methodology 1(3), 205–228 (1992)
57. Ostertag, E.J.: A Classification System for Software Reuse, Ph.D. Dissertation. University of Maryland (1992)
58. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco (1988)
59. Pearl, J.: Causality: Models, Reasoning, and Inference. Cambridge University Press, Cambridge (2000)
60. Pomerol, J.-C., Brezillon, P.: Dynamics between Contextual Knowledge and Proceduralized Context. In: Bouquet, P., Serafini, L., Brézillon, P., Benercetti, M., Castellani, F. (eds.) CONTEXT 1999. LNCS (LNAI), vol. 1688, pp. 284–295. Springer, Heidelberg (1999)
61. Prieto-Diaz, R.: Implementing faceted classification for software reuse. Communications of the ACM 34(5), 89–97 (1991)
62. Prieto-Diaz, R.: Status report: Software reusability. IEEE Software 10(3), 61–66 (1993)
63. Prieto-Diaz, R., Freeman, P.: Classifying software for reusability. IEEE Software 4(1), 6–16 (1987)
64. Rice, J.A.: Mathematical Statistics and Data Analysis. Duxbury Press (1994)
65. Rusinkiewicz, M., Sheth, A., Karabatis, G.: Specifying Interdatabase Dependencies in a Multidatabase Environment. IEEE Computer 24(12), 46–53 (1991)
66. Shen, X., Tan, B., Zhai, C.: Context-Sensitive Information Retrieval using Implicit Feedback. In: 28th international ACM SIGIR conference on Research and development in information retrieval, Salvador, Brazil, pp. 43–50 (2005)
67. Sheth, A., Aleman-Meza, B., Arpinar, I.B., Bertram, C., Warke, Y., Ramakrishanan, C., Halaschek, C., Anyanwu, K., Avant, D., Arpinar, F.S., Kochut, K.: Semantic Association Identification and Knowledge Discovery for National Security Applications. Journal of Database Management 16(1) (2004)
68. Sheth, A., Karabatis, G.: Multidatabase Interdependencies in Industry. In: ACM SIGMOD International Conference on Management of Data, Washington, DC (1993)
69. Sowa, J.F.: Semantic Networks, http://www.jfsowa.com/pubs/semnet.htm
70. Sowa, J.F.: Semantic Networks. In: Shapiro, S.C. (ed.) Encyclopedia of Artificial Intelligence, pp. 1493–1511. Wiley, New York (1992)
71. Sowa, J.F.: Laws, Facts, and Contexts: Foundations for Multimodal Reasoning. In: Hendricks, V.F., Jorgensen, K.F., Pedersen, S.A. (eds.) Knowledge Contributors, pp. 145–184. Kluwer Academic Publishers, Dordrecht (2003)
72. Sugiyama, K., Hatano, K., Yoshikawa, M.: Adaptive web search based on user profile constructed without any effort from users. In: WWW (2004)

73. Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison-Wesley, Reading (2006)
74. TopicMap: XML Topic Maps (XTM) 1.0, http://www.topicmaps.org/xtm/
75. W3C: Semantic Web (2001), http://www.w3.org/2001/sw/
76. Wache, H., Stuckenschmidt, H.: Practical Context Transformation for Information System Interoperability. In: Akman, V., Bouquet, P., Thomason, R.H., Young, R.A. (eds.) CONTEXT 2001. LNCS (LNAI), vol. 2116, pp. 367–380. Springer, Heidelberg (2001)
77. Wu, H., Siegel, M., Ablay, S.: Sensor Fusion for Context Understanding. In: 19th IEEE Instrument and Measurement Technology Conference, Anchorage, AK, USA (2002)
78. Zimmermann, A., Lorenz, A., Oppermann, R.: An operational definition of context. In: Kokinov, B., Richardson, D.C., Roth-Berghofer, T.R., Vieu, L. (eds.) CONTEXT 2007. LNCS (LNAI), vol. 4635, pp. 558–571. Springer, Heidelberg (2007)