Requirement Traceability for Software Development Lifecycle

[1]Sharadha Murugappan, [2]Dr. D. Prabha,
[1]PG Student, [2]Associate Professor,
[1, 2]Department of Computer Science and Engineering,
Sri Krishna College of Engineering and Technology, Coimbatore

Abstract*—Traceability is the ability to verify the history, location, or application of an item by means of documented recorded identification. Traceability includes the capability and implementation of keeping track of a given set or type of information to a given degree, or the ability to chronologically interrelate uniquely identifiable entities in a way that is verifiable. In Software development, the term traceability or Requirements Traceability refers to the ability to link product requirements back to stakeholders' rationales and forward to corresponding design artifacts, code, and test cases. Traceability systems are constituted by univocal identification of units/batches or lots of every product components, information collection about time and location for every batch transfer/transformation, and a method to relate this kind of data. Traceability technique has been proposed based on the four elements in order to manage information on manufacture. The four elements are physical lot integrity, which determines the traceability resolution, collection of tracing and process data, product identification and process linking, and reporting/system data retrieval. Traceability is an attribute of any artifact in a software system. Traceability is referred as the potential for traces to be set up and used. "Requirement traceability (RT) is the most common concept and seems to be mentioned in most literature discussing issues related to traceability of a software system. Requirement traceability refers to the ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)". Requirements traceability is intended to ensure continued alignment between stakeholder requirements and various outputs of the system development process.Traceability architecture is sometimes inaccurate and is less efficient.

The proposed method provides an efficient way for requirement traceability with better accuracy by using information retrieval techniques like Boolean model, which uses logic expressions, Probabilistic model, which uses set theory and sample space, Vector space model which uses weights to represent queries, Inference network model, which uses nodes to connect the queries and represents queries as concepts or terms.

***Index Terms*—Automated Traceability, Information retrieval, Precision, Recall, Requirement Traceability, Traceability Links.**

I INTRODUCTION

In the software development lifecycle, SDLC, traceability primarily means the traceability of requirements throughout application development, ensuring that the delivered software fulfills all requirements and therefore prevent failures.

Traceability provides for a logical connection between artifacts of the software development process. In support of change management tasks, traceability delivers important information about the possible consequences of a changing requirement. For project management tasks, traceability supports the control of a project's progress and provides a way to demonstrate the realization of user requirements. Traceability is essential for numerous quality-oriented software

development practices such as these. Though widely accepted as beneficial, the costs associated with traceability can be high, so the return on investment remains debatable. Unless mandated, traceability is rarely used throughout all development stages, due firstly to the number of artifacts or elements therein that often need to be related to yield value, and due secondly to the need to maintain these relations each time a change occurs. Even where the set of relations is minimal, the maintenance of traceability demands effort. While attention has been directed toward approaches for establishing traceability initially among artifacts, less attention has been paid to ensuring this traceability remains correct over time. Traceability also indicates the ability to establish a predecessor-successor relationship between one work product and another. Traceability helps to minimize failures and helps to deliver the right software on schedule that meets business requirements. Requirements traceability is critically important element within the application development lifecycle assuring successful product development if used to best effect.

II REQUIREMENT TRACEABILITY

Requirements traceability is an explicittracing of requirements to other requirements, models, test requirements, and other traceabilityitems such as design and user documentation.According to Domges and Pohl (1998)[4], "If requirements traceability is notcustomized it can lead to an unwieldy mass of unstructured and unusable data that will hardlyever be used". Traceability in Software Development LifeCycle (SDLC) helps monitoring and controlling, proper requirements definition, checking if accepted requirements are broken down into development and test tasks that refer to each other, ensures that during development, source code is reviewed according to acceptance criteria, changes at any time during the development lifecycle are traced, collaboration is ensured, and testing is performed and released for deployment on-time.

Software industries managing the requirements for a project must be able to trace a requirement back to a need that is an essential component of the proposed project. By examining each need, traceability enables identification of missed requirements early on in the design or implementation process. Requirements traceability also allows spotting extra requirements that are not really needed. The achievement of traceability in software Engineering has received more attention and research efforts. The traditional traceability methods contained **gripes,** which include unnecessary creation of trace artefacts, the focus on upfront activities and comprehensive documentation which meant that the important task of writing code and delivering executable product was delayed and had a negative impact on production performance; creating an illusion that real work is being done while in fact time is being wasted developing the trace matrix; focusing on comprehensive documentation rather than the real deliverable of working software; creation of overhead to the change process itself which actually makes change more difficult to implement.

The maintenance of traceability relations is a multi-step activity. As changes occur to the artifacts of software development, it is essential to appreciate both where and how these artifacts play a role with respect to the current traceability, along with an understanding of the encompassing development activity that can characterize the nature of the change. It is then necessary to understand the impact of the development activity on the traceability and to carry out those activities that can re-establish the traceability, at least to the prior levels. These core tasks demand effective method and tool support. This

paper describes a novel approach for the maintenance of requirements traceability relations. The approach currently supports development models expressed in structural United Modeling Language (UML) diagrams and converts part of the manual effort necessary for traceability maintenance into computational effort. There are two important innovations with the approach: first is the automatic identification of development activities with impact on existing traceability relations (event-based development activity recognition); and second is the use of rules to describe development activities and the necessary updates in an abstract way (rule-based traceability maintenance). The approach is (semi-) automated as, depending on the nature of the change and the status of the existing traceability, the user may have to provide input to the process.

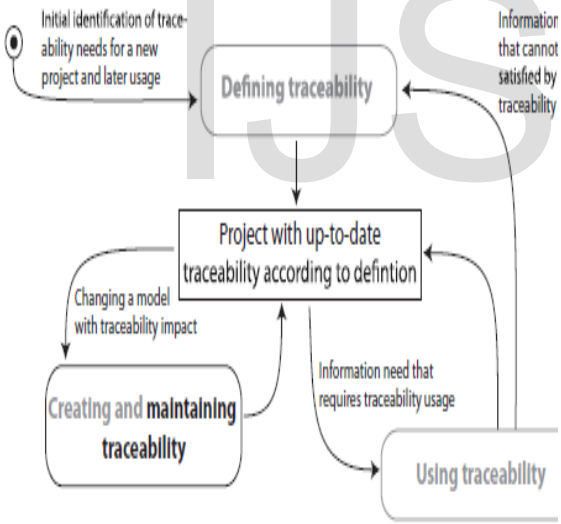

Figure 1

III AUTOMATED TRACEABILITY

As a result of these problems, a number of researchers have investigated the use of automated traceability methods using information retrieval methods such as the vector space model, semantic indexing, or probabilistic network models to dynamically generate traces at runtime. The effectiveness of automated traceability is measured using the standard metrics of recall and precision, where recall measures the number of correct links that are retrieved by the tool, and precision measures the number of correct links out of the total number of retrieved links. Numerous experiments, conducted using both experimental data sets as well as industry and government data sources, have consistently shown that when recall levels of 90-95% are targeted precision of 10-35% is generally obtainable. This means that automated traceability methods require a human analyst to manually evaluate the candidate links returned by the tool and to filter out the incorrect ones. Automated trace retrieval, while no silver-bullet, is increasingly recognized by industry as a potential traceability solution. Prototype tools such as Poirot and RETRO, are currently being used in industrial pilot studies. The new Center of Excellence in Traceability has been established specifically to address these issues.

Traceability of software artifacts is considered as an important factor in supporting various activities in the development process of a software system. In general, the objective of traceability is to improve the quality of software systems. More specifically, traceability information can be used to support some activities such as: the change impact analysis, software maintenance and evolution, the reuse of software artifacts by identifying and comparing requirements of the new system with those of the existing system.

Large-scale industrial projects often comprise of thousands of software development artifacts, for example: requirements documents, design documents, code, bug reports, test cases, and etc. The goal of software traceability is to discover relationships between these artifacts to facilitate the efficient retrieval of relevant information, which is necessary for many software engineering tasks. Traceability helps developers to control and manage the development and evolution of a software system. Ithas been

defined as the \ability to follow the life of a requirement in both a forward and backward direction"in order to understand the origins of the requirement and also to determine how a requirement has beenrealized in downstream work products such as design, code, and test cases .

IV IR TECHNIQUES

Promising results have been achieved using Information Retrieval (IR) techniques in automated traceability for traceability recovery. IR-based methods propose a list of candidate traceability links on the basis of the similarity between the texts contained in the software artefacts. IR methods provide a useful support to the identification of traceability links.

To build the sets of traceability links, we use the VSM (from the algebraic family of techniques) and JSM (from the probabilistic family of techniques) techniques. Vector Space Model and the Jensen-Shannon model outperform other IR techniques. In addition, these two techniques do not depend on any parameter. Thus, we use both JSM and VSM to recover traceability links and compare their results in isolation with those of Trustrace. These techniques both essentially use term-by-document matrices. Consequently, we choose the well-known TF=IDF measure for VSM and the normalized term frequency measure for JSM. These two measures and IR techniques arestate-of-the-art IR techniques.

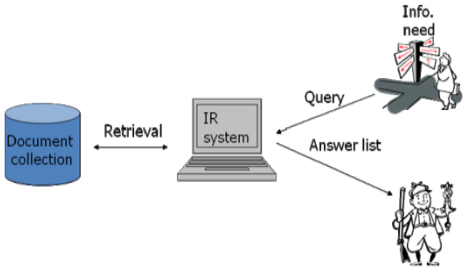

Figure 2

A document based IR system typically consists of three main subsystems: documentrepresentation, representation of users' requirements (queries), and the algorithms used tomatch user requirements (queries) with document representations.

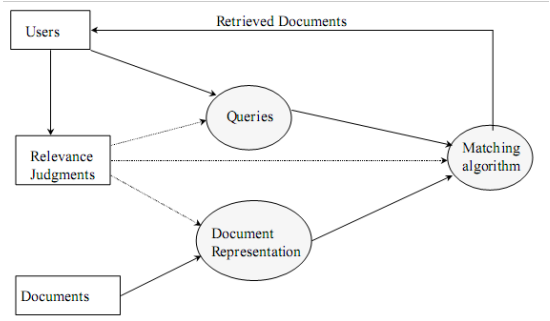

Figure 3

A document collection consists of many documents containing information about various subjects or topics of interests. Document contents are transformed into a document representation (either manually or automatically). Document representations are done in a way such that matching these with queries is easy. Another consideration in document representation is that such a representation should correctly reflect the author's intention. The primary concern in representation is how to select proper index terms. Typically representation proceeds by extracting keywords that are considered as content identifiers and organizing them into a given format.

Queries transform the user's information need into a form that correctly represents the user's underlying information requirement and is suitable for the matching process. Query formatting depends on the underlying model of retrieval used.

The user rates documents presented as either relevant or non-relevant to his/her information need. The basic problem facing any IR system is how to retrieve only the relevant documents for the user' s information requirements, while not

retrieving non- relevant ones. Various system performance criteria like precision and recall have been used to gauge the effectiveness of the system in meeting users' information requirements. Recall is the ratio of the number of relevant retrieved documents to the total number of relevant documents available in the document collection. Precision is defined as the ratio of the number of relevant retrieved documents to the total number of retrieved documents. Relevance feedback is typically used by the system to improve document descriptions or queries ,with the expectation that the overall performance of the system will improve after such a feedback.

4.1 LATENT SEMANTIC INDEXING

IR methods index the documents in a document space as well as the queries by extracting information about the occurrences of terms within them. This information is used to define similarity measures between queries and documents. In the case of traceability recovery, this similarity measure is used to identify that a traceability link might exist between two artifacts, one of whichis used as query.

4.1.1 VECTOR SPACE MODEL

In the Vector Space Model (VSM), documents and queries are represented as vectors of terms that occur within documents in a collection [Baeza-Yates and Ribeiro-Neto 1999; Harman 1992]. Therefore, a document space in VSM is described by a*m x n* matrix, where *m* is the number of terms, and *n* is the number of documents in the collection. Often this matrix is referred to as the *term-by document matrix*. A generic entry *ai, j* of this matrix denotes a measure of the weight of the *i*th term in the *j*th document. Different measures have been proposed for this weight [Salton and Buckley 1988]. In the simplest case, it is a boolean value, either 1 if the *i*th term occurs in the *j*th document, or 0 otherwise; in other cases, more complex measures are constructed based on the frequency of the terms in the documents. In particular, these measures apply both a local and global weightings to increase/decrease the importance of terms within or among documents. Specifically, we can write:

$$a_{i,j} = L(i,j) \cdot G(i)$$

where $L(i, j)$ is the local weight of the *i*th term in the *j*th document and $G(i)$ is the global weight of the *i*th term in the entire document space. In general, the local weight increases with the frequency of the *i*th term in the *j*th document, while the global weight decreases as much as the *i*th term is spread across the documents of the document space. Dumais [1991][12] has conducted a comparative study among different local and global weighting functions within experiments with Latent Semantic Indexing (LSI). The best results have been achieved by scaling the term frequency by a logarithmic factor for the local weight and using the entropy of the term within the document space for the global weight:

$$L(i,j) = \log(tf_{ij} + 1) \quad G(i) = 1 - \sum_{j=1}^{n} \frac{p_{ij} \log(p_{ij})}{\log(n)}$$

Where tf_{ij} is the frequency of the *i*th term in the *j* th document and p_{ij} is defined as:

$$p_{ij} = \frac{tf_{ij}}{\sum_{k=1}^{n} tf_{ik}}$$

We also use these two functions in our implementation of LSI. An advantage of using the entropy of a term to define its global weight is the fact that it takes into account the distribution of the term within the document space. From a geometric point of view, each document vector (columns of the term by-document matrix) represents a point in the *m*-space of the terms. Therefore, the similarity between two documents in this space is typically

measured by the cosine of the angle between the corresponding vectors, which increases as more terms are shared. In general, two documents are considered similar if their corresponding vectors point in the same (general) direction.

Many traceability recovery techniques use VSM as the base algorithm. In VSM, documents are represented as vector in the space of all the terms. Different term weighting schemes can be used to construct these vectors. We use the standard *TF/IDF* weighting scheme: A document is a vector of *TF/IDF* weights. *TF* is often called the local weight. The most frequent terms will have more weight in *TF*, but this by itself does not mean that they are important terms. The inverse document frequency, *IDF*, of a term is calculated to measure the global weight of a terms and is computed as $IDF=log_2(|D|/|d:ti \mathcal{E} d|)$. Then, *TF/IDF* is defined as

$$(TF/IDF)_{i,j} = n_{i,j}/\sum_k n_{k,j} \; X \; log_2(|D|/|\; d : ti \; \mathcal{E} \; d \;|),$$

where $n_{i,j}$ are the occurrences of a term t_i in document d_j, $\sum_k n_{k,j}$ is the sum of the occurrences of all the terms in document d_j, $|D|$ is the total number of documents d in the corpus, and $|d: t_i \; \mathcal{E} \; d|$ is the number of documents in which the term t_i appears. Once documents are represented as vectors of terms in a VSM, traceability links are created between every two documents with their own similarity value depending on each pair of documents, e.g., a requirement and a class. The similarity between two documents is measured by the positive cosine of the angle between their corresponding vectors (because the similarity between two documents cannot be negative). The ranked list of recovered links and a similarity threshold are used to divide links into a set of candidate links to be manually verified.

The vector space model can best be characterized by its attempt to rank documents by the similarity between the query and each document.In the Vector Space Model(VSM), documents and query are represent as a Vector and the angle between the two vectors are computed using the similarity cosine function. Similarity Cosine function can be defined as:

$$sim(d_j, q) = \frac{d_j \cdot q}{\|d_j\| \cdot \|q\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w^2{}_{i,j}} \sqrt{\sum_{i=1}^{N} w^2{}_{i,q}}}$$

Documents and queries are represented as vectors.

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$

$$q = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$$

Vector Space Model have been introduce term weight scheme known as if-idf weighting. These weights have a term frequency (t_f) factor measuring the frequency of occurrence of the terms in the document or query texts and an inverse document frequency (idf) factor measuring the inverse of the number of documents that contain a query or document term.

4.1.2 SINGULAR VALUE DECOMPOSITION

A common criticism of VSM is that it does not take into account relations between terms. For instance, having “automobile” in one document and “car” in another document does not contribute to the similarity measure between these two documents. LSI was developed to overcome the synonymy and polysemy problems, which occur with the VSM model. In LSI, the dependencies between terms and between documents, in addition to the associations between terms and documents, are explicitly taken into account. LSI assumes that there is some underlying or “latent structure” in word usage that is partially obscured by variability in word choice, and uses statistical techniques to estimate this latent structure. For example, both “car” and

"automobile" are likely to co-occur in different documents with related terms, such as "motor," "wheel," etc. LSI exploits information about co-occurrence of terms (latent structure) to automatically discover synonymy between different terms. LSI defines a term-by-document matrix *A* as well as VSM. Then it applies singular value decomposition (SVD) [Cullum and Willoughby, 1985] to decompose the term-by-document matrix into the product of three other matrices:

$$A = T_0 \cdot S_0 \cdot D_0,$$

where *T*0 is the *m*×*r* matrix of the terms containing the left singular vectors(rows of the matrix), *D*0 is the *r* ×*n* matrix of the documents containing the right singular vectors (columns of the matrix), *S*0 is an *r* ×*r* diagonal matrix of singular values, and *r* is the rank of *A*. *T*0 and *D*0 have orthogonal columns, such that:

$$T_0^T \cdot T_0 = D_0 \cdot D_0^T = I_r.$$

SVD can be viewed as a technique for deriving a set of uncorrelated indexing factors or concepts [Deerwester et al. 1990], whose number is given by the rank *r* of the matrix *A* and whose relevance is given by the singular values in the matrix*S*0. Concepts "represent extracted common meaning components of many different words and documents" [Deerwester et al. 1990]. In other words, concepts are a way to cluster related terms with respect to documents and related documents with respect to terms. Each term and document is represented by a vector in the *r*-space of concepts, using elements of the left or right singular vectors. The product *S*0 ·*D*0 (*T*0 ·*S*0, respectively) is a matrix whose columns (rows, respectively) are the document vectors (term vectors, respectively) in the *r*-space of the concepts. The cosine of the angle between two vectors in this space represents the similarity of the two documents (terms, respectively) with respect to the concepts they share. In this way, SVD captures the underlying structure in the association of terms and documents. Terms that occur in similar documents, for example, will be near each other in the *r*-space of concepts, even if they never co-occur in the same document. This also means that some documents that do not share any word, but share similar words may none the less be near in the *r*-space. SVD allows a simple strategy for optimal approximate fit using smaller matrices [Deerwester et al. 1990]. If the singular values in *S*0 are ordered by size, the first *k* largest values may be kept and the remaining smaller ones set to zero. Since zeros were introduced into *S*0, the representation can be simplified by deleting the zero rows and columns of *S*0 to obtain a new diagonal matrix *S*, and deleting the corresponding columns of *T*0 and rows of *D*0 to obtain *T* and *D* respectively. The result is a reduced model:

$$A \approx Ak = T \cdot S \cdot D,$$

where the matrix *Ak* is only approximately equal to *A* and is of rank *k* <*r*. The truncated SVD captures most of the important underlying structure in the association of terms and documents, yet at the same time it removes the noise or variability in word usage that plagues word-based retrieval methods. Intuitively, since the number of dimensions *k* is much smaller than the number of unique terms *m*, minor differences in terminology will be ignored. The choice of *k* is critical: ideally, we want a value of *k* that is large enough to fit all the real structure in the data, but small enough so that we do not also fit the sampling error or unimportant details. The proper way to make such a choice is an open issue in the factor analysis literature [Deerwester et al. 1990; Dumais 1992]. In the application of LSI to information retrieval, good performances have been achieved using about 100 concepts on a document space of about 1,000 documents and a vocabulary of about 6,000 terms [Deerwester et al.1990].

With much larger repositories (between 20,000 and 220,000 documents and between 40,000 and 80,000 terms), good results have been achieved using between 235 and 250 concepts [Dumais 1992].

4.2 JENSEN-SHANNON MODEL

JSM is driven by a probabilistic approach and hypothesis testing technique. JSM represents each document through a probability distribution, i.e., a normalized term-by-document matrix.

The probability distribution of a document is

$$p = n(\omega,d)/T_{d,}$$

where $n(\omega,d)$ is the number of times a word appears in a document d and T_d is the total number of words appearing in *d*. The empirical distribution can be modified to take into account the term's global weight, e.g., IDF. After considering the global weight, each document distribution must be normalized. Once the documents are represented as probability distribution, JSM computes the distance between two documents' probability distribution and returns a ranked list of links. JSM ranks target documents via the "distance" of their probability distributions to that of the source documents:

$$\boldsymbol{JSM(q,d) = H((p_q+p_d)/2)-((H(p_q)+H(p_d))/2),}$$

$$\boldsymbol{H(p) = \sum h(p(\omega)),}$$

$$\boldsymbol{h(x) = -xlogx,}$$

Where $H(p)$is the entropy of the probability distribution *p*, and p_q and p_d are the probability distributions of the two documents (a "query" and a "document"), respectively. By definition, h(0) ≡ 0. We compute the similarity between two documents using $1 - JSM(q,d)$. The similarity values are in [0,1].

Gerard Salton and his colleagues suggested a model based on Luhn's similarity criterion that has a stronger theoretical motivation (Salton and McGill 1983). They considered the index representations and the query as vectors embedded in a high dimensional Euclidean space, where each term is assigned a separate dimension.

V IR-BASED ARTIFACT QUALITY IMPROVEMENT

The similarity between software artifacts has been previously used to assess software quality. Lawrie propose an approach implemented in the Quality Assessment using Language Processing (QALP) tool. The QALP tool leverages identifiers and related comments to characterize the quality of a program. We share with Lawrie et al. the conjecture that the textual similarity between related software artifacts can positively contribute to quality and comprehensibility. The approach we propose aims at showing such a similarity to the developer to induce improvements in the quality of the source code lexicon. Poshyvanyk and Marcus propose an approach that uses traceability links to assess and maintain the quality of software documentation. The approach is based on the observation that the quality of the documentation should reflect the source code structure. In other words, elements of the documentation that link to strongly coupled elements of the source code should be related too. In particular, they use LSI to establish relationships between elements of the documentation and source code coupling measures to assess the strength of dependencies among source code artifacts. De Lucia use LSI to identify cases of low similarity between artifacts previously traced by software engineers. The lack of textual similarity might be an indicator of low quality

between the traced artifacts, in terms of poor text description in high-level artifacts, or of meaningless identifiers or poor comments in source code artifacts. These approaches use textual similarity to perform an offline quality assessment of both source code and documentation, with the objective of guiding a software quality review process. We use the similarity between source code and high-level artifacts—continuously recomputed while coding—to induce the developer to write source code with better identifiers and comments. Also, we propose the suggestion of identifiers obtained by extracting n-grams from high-level artifacts. IR methods have also been used to define new measures for source code quality assessment. Patel propose a cohesion metric, based on the vector space model, which highlights properties shared between members of a module. Such a cohesion measure can be considered as a measure of the information strength of a module. Marcus and Poshyvanyk propose a cohesion metric that exploits LSI to estimate the overlap of semantic information—computed as a textual similarity—among methods of a class. Conceptual Coupling of Classes (CoCCs) metric captures the coupling among classes based on semantic information obtained from source code identifiers and comments. They show through a case study that the conceptual measure captures new dimensions of coupling which are not captured by existing coupling measures. Etzkorn propose a new semantic metric for object-oriented systems called the Semantic Class Definition Entropy metric (SCDE),which examines the implementation domain content of a class to measure its complexity. The proposed metric allows us to measure other aspects of a class complexity which cannot be measured with existing structural metrics.

VI META MODEL FOR REQUIREMENT TRACEABILITY

We present the reference models resulting from our studies. We assume that our traceability reference models will be implemented in some trace repository (manual or computerized). It is widely accepted that such a repository will comprise at least three layers:

- the meta model defining the language in which traceability models can be defined;
- a set of reference traceability models which can be customized within the scope defined by the meta model; and
- a (possibly distributed) database of actual traces, recorded under the chosen models.

We adopt the convention that we denote node metaclasses(e.g., STAKEHOLDER) by small bold caps and their instances (e.g., CUSTOMER) by non bold small caps. Similarly, link metaclasses (e.g., TRACES-TO) are denoted by bold italics, specific link types by standard italics (e.g.,REFINES).The practitioners and focus groups in Phase I of the main study confirmed that the most essential aspects of traceability can be captured in the very simple meta model, shown in Fig. 1, which thus provides the basic language primitives for categorizing and describing traceability models in more detail. Each entity and link in the meta model can be specialized and instantiated to create organization or project specific traceability models. The meta model can be used to represent the following dimensions of traceability information (cf. Table 4):

1. What information is represented including salient attributes or characteristics of the information? In the model, **OBJECTS** represent the inputs and outputs of the system development process. Examples of various types of **OBJECTS** include **REQUIREMENTS, ASSUMPTIONS, DESIGNS, SYSTEM COMPONENTS, DECISIONS, RATIONALE, ALTERNATIVES, CRITICAL SUCCESS FACTORS,** etc. These represent the major conceptual

elements among which traceability is maintained during the various life cycle stages. **OBJECTS** are created by various organizational tasks. Examples of tasks include systems analysis and design activities. This information can be represented as an attribute of **OBJECTS**. Traceability across various **OBJECTS** is represented by the **TRACES-TO** links. For example, a **DEPENDS-ON** link between two objects (a **REQUIREMENT** and an **ASSUMPTION**) can be represented as a specialization of this **TRACES-TO** link.

2. Who are the **STAKEHOLDERS** that play different roles in the creation, maintenance and use of various **OBJECTS** and traceability links across them? In the model, **STAKEHOLDERS** represent the agents involved in the system development and maintenance life cycle activities. Examples of **STAKEHOLDERS** include the project managers, systems analysts, designer etc. These **STAKEHOLDERS** act in different **ROLES** or capacities in the establishment and use of the various conceptual **OBJECTS** and traceability links.

3. Where it is represented in terms of sources that document traceability information? All **OBJECTS** are documented by **SOURCES**, which may be physical media, such as documents or intangible things, such as references to people or undocumented policies and procedures. Examples of **SOURCES** include **REQUIREMENT SPECIFICATION DOCUMENTS, MEETING MINUTES, DESIGN DOCUMENTS, MEMORANDA, TELEPHONE CALLS** as well as references to various **STAKEHOLDERS** using their phone numbers, e-mail address etc., **STAKEHOLDERS** manage the **SOURCES**; i.e., they create, maintain, and use them.

4. How this information is represented both by formal and informal means and how it relates to other components of traceability?

The sources, as mentioned above, can be physical or intangible. Further, they can be represented at different levels of formality. Some sources such as requirements specifications may be text documents, whereas others design documents may be represented in multiple formats such as graphics and text.

5. Why a certain conceptual **OBJECT** was created, modified, or evolved?

The rationale behind the creation, modification and evolution of various conceptual **OBJECTS** can be represented as a specialization of the meta-class **OBJECT**. Then, it can be linked to the conceptual object (using a specialization of the traces-to link).More complex models of rationale, which include issues, alternatives and arguments supporting and opposing them can also be represented as specialization of the **OBJECT-TRACES-TO-OBJECT** relationship in our model.

6. When this information was captured, modified, and evolved?

Relevant temporal information about any of the entities or links in our model can be represented as their attributes. For example, the frequency or the time/duration at which a requirement or design was created, reviewed, modified, or justified by a specific rationale can be represented with this scheme. The three nodes of the meta model correspond roughly to the three dimensions of requirements engineering proposed by Pohl in that they cover the aspects of understanding (objects), agreement (stakeholders), and physical representation (sources). However, note that we are not discussing the requirements process per se, but the creation and usage of traces.

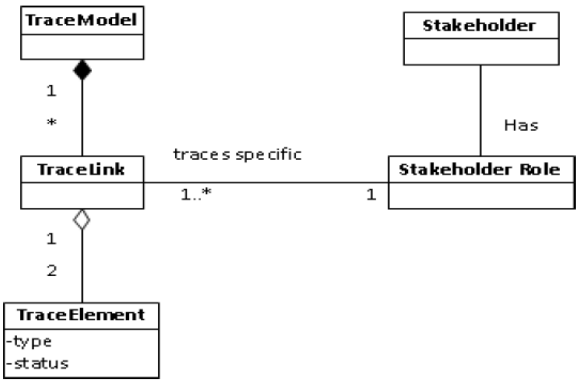

Figure 4

VII CONCLUSION

The literature showed that IR techniques are useful to recover traceability links between requirements and source code. However, IR techniques lack accuracy (precision and recall). In this paper, we conjectured that: we could consider heterogeneous sources of information to discard/rerank the traceability links provided by an IR technique to improve its accuracy.

REFERENCES

[1] A.D. Lucia, M.D. Penta, and R. Oliveto, "Improving Source CodeLexicon via Traceability and Information Retrieval," IEEE Trans.Software Eng., vol. 37, no. 2, pp. 205-227, Mar. 2011.

[2] A. Marcus and J.I. Maletic, "Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing,"Proc. 25th Int'l Conf. Software Eng., pp. 125-135, 2003.

[3] A. Abadi, M. Nisenson, and Y. Simionovici, "A TraceabilityTechnique for Specifications," Proc. 16th IEEE Int'l Conf. ProgramComprehension, pp. 103-112, June 2008.

[4] Domges, Ralf & Pohl, Klaus (1998, December). Adapting traceability environments toproject-specific needs.*Commun. ACM 41, 12*, 54-62.

[5] D. Poshyvanyk, Y.-G. Gue´he´neuc, A. Marcus, G. Antoniol, and V.Rajlich, "Feature Location Using Probabilistic Ranking of MethodsBased on Execution Scenarios and Information Retrieval," IEEETrans. Software Eng., vol. 33, no. 6, pp. 420-432, June 2007.

[6] Nasir Ali, Yann-Gae¨lGue´he´neuc, and GiulianoAntoniol, "Trustrace: Mining Software Repositoriesto Improve the Accuracy ofRequirement Traceability Links," IEEE Trans. Software Eng., vol. 39, no. 5, may 2013.

[7] KhaledJaber, Bonita , and Chang Liu, "A Study on the Effect of Traceability Links inSoftware Maintenance," IEEE Access, October 2013.

[8] AkramRoshdi, and AkramRoohparvar, "Information Retrieval Techniques and Applications ," International Journal of Computer Networks and Communications Securityvol. 3, no. 9, september 2015.

[9] A. De Lucia,"Recovering Traceability Links in SoftwareArtifact Management Systems usingInformation Retrieval Methods,"ACM Transactions on Software Engineering and Methodology, Vol. 16, No. 4, Article 13, Sept. 2007.

[10] Baeza-Yates, R. Andribeiro-Neto,B. 1999. "Modern Information Retrieval,".addison-wesley, reading, MA.

[11] Salton, G. And Buckley, C. 1988. "Term-Weighting Approaches In Automatic Text Retrieval,".*inf.process.manage. 24*, 5, 513–523.

[12] Dumais, S. T. 1991. improving the retrieval of information from external sources. *behav. res.meth. instrum. comput.23*, 229–236