A Text Retrieval Approach to Recover Links among E-Mails and Source Code Classes

Giuseppe Scanniello and Licio Mazzeo
Universitá della Basilicata, Macchia Romana,
Viale Dell'Ateneo, 85100, Potenza, ITALY,
Email: giuseppe.scanniello@unibas.it, licio.mazzeo@gmail.com

***Abstract*—During software development and evolution, the communication among stakeholders is one of the most important activities. Stakeholders communicate to discuss various topics, ranging from low-level concerns (e.g., refactoring) to high-level resolutions (e.g., design rationale). To support such a communication, e-mails are widely used in both commercial and open source software projects. Although several approaches have been proposed to recover links among software artifacts, very few are concerned with e-mails. Recovering links between e-mails and software artifacts discussed in these e-mails is a non trivial task. The main issue is related to the nature of the communication that is scarcely structured and mostly informal. Many of the proposed approaches are based on text search or text retrieval and reformulate the link recovery as a document retrieval problem. We refine and improve such solutions by leveraging the parts of which an e-mail is composed of: header, current message, and previous messages. The relevance of these parts is weighted by a probabilistic approach based on text retrieval. The results of an empirical study conducted on a public benchmark indicate that the new approach in many cases outperforms the baselines: text retrieval and lightweight text search approaches. This paper is built on [1].**

Keywords-Information retrieval, software maintenance; traceability recovery; BM25F

I. INTRODUCTION

SOFTWARE maintenance is one of the most expensive, time consuming, and challenging phase of the development process. In contrast with software development, that typically can last for 1-2 years, the maintenance phase typically lasts for many years. It has been estimated that the costs needed to perform maintenance operations range from 85% to 90% of the total cost of a software project [2]. This is due to the fact that maintenance starts after the delivery of the first version of a system and lasts until that system is dismissed [3].

A software system is continuously changed and enhanced during the maintenance phase. Maintenance operations are carried out for several reasons [4]. Corrective, perfective, and adaptive are typical examples [5]. Independently from the kind of maintenance operation, the greater part of the cost and effort to its execution is due to the comprehension of source code [6]. Pfleeger and Atlee [7] estimated that up to 60% of software maintenance is spent on the comprehension of source code. There are several reasons that make the source code comprehension even more costly and complex and range from the size of a subject software to its overall quality. Other reasons are related to the knowledge of a subject system that is implicitly expressed in software artifacts (i.e., models, documentation, source code, e-mails, and so on) [8]. This knowledge is very difficult to retrieve and it is very often enclosed in non-source artifacts [9].

Among non-source artifacts, free-form natural language artifacts (e.g., documentation, wikis, forums, e-mails) are intended to be read by stakeholders with different experience and knowledge (e.g., managers, developers, testers, and end-users). This kind of artifacts often implicitly or explicitly references to other forms of artifacts, such as source code [10]. Linking e-mails and source code could improve the comprehension of a system and could help to understand the justification behind decisions taken during design and development. Then, links between e-mails and source code could be worthwhile within the entire software lifecycle and in software maintenance, in particular [8], [11], [12]. However, linking free-form natural language artifacts to source code is a critical task [9], [13].

Although several approaches have been proposed to recover links among software artifacts (e.g., [13], [14], [15], [16]), a very few are concerned with e-mails [17], [18]. These approaches are based on text search or text retrieval and reformulate the problem of recovering links among e-mails and source code artifacts as a document retrieval problem. We refine and improve such solutions by leveraging the parts of which an e-mail is composed of, namely the header, the current message (body, from here on), and the sentences from previous messages (quote in the following). The relevance of these parts has been weighted by means of a text retrieval probabilistic model. In particular, we implemented our solution exploiting the BM25F model [19], [20]. To assess the validity of our proposal, we have conducted an empirical study on the public benchmark presented by Bacchelli *et al.* [17].

The work presented here is based on the paper presented in [1]. With respect to this paper, we provide hereafter the following new contributions:

1) The approach has been better described;
2) Related work has been discussed and differences with respect to our approach have been highlighted;
3) An extended data analysis to strengthen the achieved results has been provided;
4) An extended discussion of results and of their practical implications has been given;
5) A prototype of a supporting tool for our approach has been described.

Structure of the paper. In Section II, we discuss related work and motivations, while we illustrate our approach in

Section III. In Section IV, we present the design of the empirical evaluation, while we discuss the achieved results and possible threats to validity in Section V. Final remarks and future work conclude the paper.

II. Related Work and Motivation

Many standards include traceability as a recommended or legally required activity for software development (e.g., IEEE Std. 830-1998 [21]). Unfortunately, in projects where traceability management is not initially a systematic part of the development process, it is very difficult to establish traceability links among software artifacts [22]. Automated traceability recovery methods deal with these problems reducing the effort to construct and maintain traceability links among software artifacts (e.g., requirements and test cases) and source code [23].

In the following, we first discuss methods that use information retrieval (IR) techniques to retrieve links among any kind of software artifact and then we focus on approaches and methods to recover links between e-mails and source code.

A. IR-Based Traceability Recovery

IR-based traceability recovery approaches use IR techniques to compare a set of source artifacts with a set of target artifacts. All the possible pairs of target and source artifacts are ranked with respect to their textual/lexical similarity that is computed using an IR method. The pairs of software artifacts and their similarity form a ranked list, which is in turn analyzed by a software engineer to establish correct and false recovered traceability links. Different IR methods can be used to compute the similarity between two artifacts. The widely used methods are: vector space model (VSM) [24] and latent semantic indexing (LSI) [25]. For example, Antoniol *et al.* [16] apply VSM and a probabilistic model to trace source code onto software documentation. The first model calculates the cosine similarity of the angle between the vectors corresponding to the source and target artifacts. The probabilistic model computes a ranking score based on the probability that a document is related to a specific source code component. It is worth noting that this model is not a text retrieval probabilistic model. The authors compare these models on two small software systems. The obtained results show that the two models exhibit similar accuracy. This study is the first one in which IR methods are applied to the problem of recovering traceability links between software artifacts.

To recover traceability links between source code and documentation, Marcus and Maletic [26] use an extension of VSM, namely LSI. The validity of their approach is assessed on the same software as those in [16]. The results indicate that LSI performs at least as well as the probabilistic model and VSM and in some cases outperforms them.

To compute the similarity between software artifacts, Abadi *et al.* [27] propose the use of the Jensen-Shannon (JS) Divergence. The authors perform a comparison among IR methods (including VSM, LSI, and JS). The experimental results indicate that VSM and JS provide best results.

Capobianco *et al.* [28] propose an IR method based on numerical analysis principles. Artifacts are represented with interpolation curves. The distance between pairs of interpolation curves indicates the similarity between the artifacts represented with these curves. The authors also report an empirical evaluation, whose results suggest that their method outperforms VSM, LSI, and JS.

A work complementary to the ones highlighted above is presented by De Lucia*et al.* [29]. The authors investigate whether the use of smoothing filters improves the performances of existing traceability recovery techniques based on IR methods. The results suggest that the use of these filters significantly improves the performances of VSM and LSI.

De Lucia *et al.* [30] analyze incremental methods for traceability recovery, namely relevance feedbacks. An empirical evaluation is conducted to assess strengths and limitations of relevance feedbacks within a traceability recovery process. The achieved results suggest that even using feedbacks a part of correct links are not retrieved. However, feedbacks mostly improve retrieval performances and can be used within an incremental traceability recovery process.

An approach that combines different orthogonal IR techniques is proposed by Gethers et al. *et al.* [31]. The used techniques are those that produce dissimilar and complementary results: VSM, JS, and Relational Topic Modeling (RTM). An empirical study on six software systems is presented to assess whether the new approach outperforms stand-alone IR methods as well as any other combination of non-orthogonal methods. The results suggest that the retrieval performances are statistically better when using the new proposed method.

B. Recovery Links between E-Mails and Source Code

Although several approaches have been proposed to recover links among software artifacts (e.g., [13], [15], [16]), only a few are concerned with e-mails [17], [18], [1]. In general, these approaches can be classified as: rule-based and IR-based.

Rule-based. All these approaches are based on text search or text retrieval and reformulate this problem as a document retrieval problem. To detect latent links between emails and source code entities hand-code specific rules (i.e., sets of regular expressions) have to be specified. These rules are in turn triggered whenever they match with a portion of email text (e.g., [10]). For example, if the identifiers in the source code repository follows the CamelCase naming convention, we basically know that each identifier is either a single or a compound name (i.e., a sequence of unseparated single names). In the case of class names, all the single names start with a capital letter. Therefore, we can define a regular expression so that every time we find a string in an e-mail of the form `Foo`, `FooBar`, `FooBarXYZ`, etc., we can mark it as a link between the source code and the e-mail. This kind of approach is computationally lightweight for small/medium corpora (e.g., repositories with a small number of e-mails) and easy to implement. Conversely, they lack of flexibility since they are strictly programming-language-dependent. Even more, they do not provide any ranking score associated with the discovered link (i.e., information about a link is binary: a link is either present or not). For example, Bacchelli *et al.* [18] define and evaluate various lightweight methods for recovering

links between source code and e-mails on their benchmark. Characteristics and naming conventions of source code were exploited. The results suggest that lightweight methods can be successfully used.

IR-based. This kind of approaches (e.g., [1]) reformulate the problem as a particular instance of the more general document retrieval problem. They use IR techniques to compare a set of source artifacts (software entities) with a set of target artifacts (e-mails). Each source code entity (e.g., the class name) is used as the query to retrieve the set of most relevant e-mails. Candidate links are then devised by inspecting the ranked list of retrieved e-mails. Relevance between any pair of source and target artifacts (i.e., source code entity and email) can be determined by their textual/lexical similarity, which is computed by using a specific IR model in conjunction with a particular term-weighting score (e.g., cosine similarity using *tf-idf* vector space model) [24]. The main advantage of IR-based approaches is that they are more flexible and associate each discovered link with a ranking score. Nevertheless, the queries for retrieving relevant emails mostly consist of the strings identifying package and/or class names, which are parsed directly from source code. This leads to "semi-structured" queries, which are used to retrieve almost "unstructured" documents (e-mails), organized in a "quasi-unstructured" way. Inverted indices built on top of *free-text* documents have proven to work good when used to answer free-text queries as well (e.g., [13]). Therefore, using keywords derived from structured source code that highly likely do not appear in the email index may lead to poor retrieval results. For example, Bacchelli *et al.* [17] evaluate and compare lightweight methods based on regular expressions with LSI and VSM. This comparison is based on their benchmark. The effectiveness of the lightweight methods and LSI and VSM is evaluated on the basis of the measures: precision, recall, and F-measure. Differently from [27], the authors observe that LSI outperforms VSM on Java systems. The results achieved with these methods are close for software written in C, PHP, and ActionScript. Furthermore, the authors show that lightweight methods outperform LSI and VSM. The interpretation of the results is that: e-mails are often referred to by name, not synonyms, and source code is rare reported in. One of the concerns related to lightweight methods is that particular configurations are needed, which depend on the system understudy. This implies that on that system is required a specific knowledge to retrieve accurate links. For lightweight methods, scalability issues could be also present when the number of e-mails increases. To deal with these concerns, IR-based link recovery should to be preferred.

III. The Approach

IR-based traceability recovery approaches reformulate traceability recovery as a document retrieval problem. We refine and improve such solutions by leveraging the parts of which an e-mail is composed of: object, body, and quote. Our approach is composed by the following steps:

1) **Creating a Corpus**. A corpus is created, so that each e-mail of a subject system will have a corresponding document in the resulting corpus. Each document has three fields: header, body, and quote.
2) **Corpus Normalization**. The corpus is normalized using a different set of techniques (e.g., stop word removal and special token elimination).
3) **Corpus Indexing**. An IR engine is used to index the corpus. Different engines work in various ways. Most of them create a numerical index for each document in the corpus. In our case, the total number of terms for each field of an e-mail is determined. This information is used in our approach. In this work, we exploited VSM and BM25F.
4) **Query Formulation**. In a typical text retrieval problem, a software engineer writes a textual query and retrieves documents that are similar to that query (e.g., [32]). Differently, in IR-based traceability recovery a set of source artifacts (used as the query) are compared with set of target artifacts (even overlapping) [16]. In this work, software entities (i.e., source code) are used as the query. The textual query is normalized in the same way as the corpus and the boolean operator "AND" is used with the individual terms of that query. The number of queries is equal to the number of source code artifacts.
5) **Ranking Documents**. The index is exploited to determine similarity measures between the queries and the documents in the corpus (i.e., the e-mails). In particular, the queries are projected into the document space generated by the IR engine on the corpus. Then lexical similarities between each query and the e-mails are computed. The pairs of source and target artifacts are ranked in descending order with respect to their lexical similarity.
6) **Examining Results**. The software engineer investigates the ranked list and classifies the pairs of source and target artifacts as true or false links.

Although all these steps are part of a baseline IR-Based Traceability Recovery approaches (e.g., [13]), we sensibly change here the steps 3 and 5. In the remainder of this section, we describe how we instantiated all the steps above. Investigating alternative instances for these steps is the subject of our future work.

A. Creating a Corpus

Each e-mail results in one document in the corpus. Each document has three well defined fields: header, body, and quote. The header field contains the subject, while the body the sentences of the current message. All the sentences from previous messages are within the quote field. In particular, it includes a chain of messages (e.g., ideas, opinions, issues, or possible solutions) exchanged among stakeholders (mostly developers) linked in the sequence in which they espoused that discussion. We also consider the quote because IR approaches produce better results when a huge amount of lexical information is available [24]. Moreover, the body and the quote fields are separately considered since the lexical information within the body is on the current focus of a discussion, while the quote field includes the text that might provide useful information on the entire discussion thread.

B. Corpus Normalization

The corpus is normalized: *(i)* deleting non-textual tokens (i.e., operators, special symbols, numbers, etc.), *(ii)* splitting terms composed of two or more words (e.g., `first_name` and `firstName` are both turned into `first` and `name`), and *(iii)* eliminating all the terms within a stop word list (including the keywords of the programming languages: Java, C, ActionScript, and PHP) and with a length less than three characters. We applied these normalization rules because they have been widely applied in IR-based traceability recovery approaches (e.g., [16]).

Splitting identifiers could produce some noises in the corpus. For example, if the name of a class is `FileBuffer`, it is possible that a software engineer talks about FileBuffer in an e-mail rather than `File` and `Buffer`. However, if the identifiers are not split the things could go from bad to worse: the class name is not in the text of the e-mails (e.g., [17] and [18]), while that name is used as the query. To deal with this issue, we apply the same normalization process on both the corpus and the queries. In addition, the "AND" operator is used to formulate each query. That is, we use the "AND" operator between each pair of words in the query (e.g., `A Sample Query` is seen as `A` "AND" `Sample` "AND" `Query`).

Differently from the greater part of the traceability recovery approaches (e.g., [13], [16]), we did not apply any stemming technique [24] to reduce words to their root forms (e.g., the words `designing` and `designer` have `design` as the common radix). This is because we experimentally observed that the use of a Porter stemmer [33] led to worse results. Also, in [17] the stemming was not used for similar reasons. Investigating alternative normalization techniques and possible combinations of them is a future direction for our work.

C. Corpus Indexing

We adopt here a probabilistic IR-based model, namely BM25F [19]. This model extends BM25 [20] to handle semistructured documents from a corpus. The BM25 model was originally devised to pay attention to term frequency and document length, while not introducing a huge number of parameters to set [34]. BM25 showed very good performances [20] and then widely used specially in web document retrieval applications [35], [36]. BM25F was successively proposed to build a term weighting scheme considering the fact that documents from a corpus can be composed of fields (e.g., [35]). Each document is in the corpus and contains information on the contained fields. Then, the fields of a document differently contribute to its representation. We used BM25F because it has been successfully used on very large corpuses [36] in terms of both scalability and quality of the retrieved documents [37]. The use of other probabilistic models (e.g., the Expectation-Maximization algorithm [38]) could lead to different results. This point is subject of future work.

The difference between "vector space" and "probabilistic" IR methods is not that great. In both the cases, an information retrieval scheme is built for considering each document as a point in a multi-dimensional geometrical space. Therefore, BM25F is based on the bag-of-words model, where each document in the corpus is considered as a collection of words disregarding all information about their order, morphology, or syntactic structure. A word could appear in different fields of the same document. In this case, that word is differently considered according to the field in which it appears. Applying BM25F, each e-mail in the corpus is represented by an array of real numbers, where each element is associated to an item in a dictionary of terms. BM25F does not use a predefined vocabulary or grammar, so it can be easily applied to any kind of corpora.

BM25F works on the occurrence of each term in the fields of all the documents in the corpus. These occurrences are used to build a *term-by-document* matrix. In the current instantiation of this step we modified the original definition of BM25F to better handle the problem at the hand. In the model a generic entry of the table is computed as follows:

$$idf(t,d) = \log(\frac{N - df(t) + 0.5}{df(t) + 0.5} + 1) * weight(t,d) \quad (1)$$

where N is the total number of documents in the corpus, while df is the number of documents where the term t appears. The weight of the term t with respect to the document d is computed by $weight(t,d)$ as follows:

$$weight(t,d) = \sum_{c\ in\ d} \frac{occurs^d_{t,c} * boost_c}{((1-b_c) + b_c * \frac{l_c}{avl_c})} \quad (2)$$

l_c is the length of the field c in the document d; avl_c is the average length of the field c in the all documents; and b_c is a constant related to the field length; and $boost_c$ is the boost factor applied to the field c. $occurs^d_{t,c}$ is the number of terms t that occur in the field c of the document d. This equation is dependent on the field and document relevance and it is similar to a mapping probability. This is because BM25F is considered a probabilistic IR-based model. Regarding the constants of the equation (1), we chose 0.75 as the value for b_c, while 1 is the boost value applied to each field (i.e., header, body, and quote). These values were experimentally chosen and are customary in the IR field [37]. In the future, we plan to automate the choice of the boost values using supervised machine learning techniques [39]. For example, we could divide the benchmark in test and training sets and then the training set could be used to determine the best boost values.

In the original definition of BM25F [36], if a term occurs in over half the documents in the corpus, the model gives a negative weight to the term. This undesirable phenomena is well established in the literature [24]. It is rare in some applicative contexts, while it is common in others as for an example in the recovery of links between e-mails and source code. In such a context, in fact, e-mails quote sentences from previous messages and then the difference among e-mails (in the same discussion thread) is not that great with respect to the terms contained. To deal with this concern, we modified the computation of idf. The adopted solution is that shown in the equation (1), which is based on that suggested in [40]. The main difference with respect to the canonical computation of idf is that 1 is added to the argument of the logarithm.

D. Query Formulation

In the traceability recovery field, source artifacts are used as the query [13]. The number of queries is then equal to the number of source artifacts. In this work, we used source code entities as the source artifacts and applied the following two instantiations for Query Formulation: *(i)* class names and *(ii)* class and package names. In both the cases, the queries are normalized in the same way as the corpus. When the textual query is composed of more than one term (e.g., `ArgoStatusBar`), the boolean operator "AND" is used with the individual terms of that query (`Argo`, `Status`, and `Bar`). This implies that all the individual terms have also to exist anywhere in the text of a document.

E. Ranking Documents

For a probabilistic IR method, the similarity score between a query with the documents in the corpus is not computed by the cosine similarity and $tf - idf$ in a vector space [41], but by a different formula motivated by probability theory [24]. In this work, we used a formula based on a non-linear saturation to reduce the effect of term frequency. This means that the term weights do not grow linearly with term frequency but rather are saturated after a few occurrences:

$$score(q,d) = \sum_{t\ in\ q} idf(t) * \frac{weight(t,d)}{k_1 + weight(t,d)} \qquad (3)$$

where q is the textual query and d is a document in the corpus. The values for $idf(t)$ and $weight(t,d)$ are computed as shown in the equations (1) and (2), respectively. The parameter k_1 usually assumes values in the interval $[1.2, 2]$. We used 2 as the value because experiments suggested that it is a reasonable value [24] to maximize retrieval performances.

F. Examining Results

A set of source artifacts is compared with set of target artifacts (even overlapping). Then, all the possible pairs (candidate links) are reported in a ranked list (sorted in descending order). The software engineer investigates the ranked list of candidate links to classify them as actual or false links.

IV. Empirical Evaluation

The presentation of the study is based on the guideline suggested in [42].

A. Definition

Using the Goal Question Metrics (GQM) template [43], the goal of our empirical investigation can be defined as follows: "Analyze the adoption of our approach for the purpose of evaluating it with respect to the links between e-mails and source code from the point of view of the researcher in the context of open source systems and from the point of view of the project manager, who wants to evaluate the possibility of adopting that approach in his/her own company."

We have then formulated and investigated the following research question: *Does our proposal outperform baseline approaches based on text search or text retrieval methods?*

We considered the following baselines in our empirical investigation:

1) **BM25F with the "OR" operator**: We apply the BM25F model and the "OR" operator in the step Query Formulation. The Corpus Indexing step is executed by considering the e-mails as composed of header, body, and quote. The only difference with respect to our proposal is that the "OR" operator is used against the "AND" operator;

2) **BM25F considering body and quote together**: We apply the BM25F model and the operators "AND" and "OR". Furthermore, the Corpus Indexing step is performed considered two fields: *(i)* header and *(ii)* body and quote together;

3) **Lucene**[1] **with "AND" and "OR" operators**: In the Corpus Indexing step, we use Lucene. It uses a combination of VSM and the Boolean model to determine how relevant a document is to a query. We here apply both the operators "AND" and "OR". Since Lucene is based on VSM, more times a query term appears in a document relative to the number of times the term appears in all the documents in the corpus, the more relevant that document to the query is;

4) **VSM**: It represents the documents in the corpus as term vectors, whose size is the number of terms present in the vocabulary. Term vectors are aggregated and transposed to form a *term-document matrix*. To take into account the relevance of terms in each document and in all the corpus, many weighting schema are available. In our empirical evaluation, we employed the *tf-idf* (term frequency - inverse document frequency) weighting;

5) **LSI**: Even for a corpus of modest size, the term-document matrix is likely to have several tens of thousand of rows and columns, and a rank in the tens of thousands as well. LSI is an extension of VSM developed to overcome the synonymy and polysemy problems [25]. SVD (Singular Value Decomposition) is used to construct a low-rank approximation matrix to the term-document matrix [44]. In LSI there is no way to enforce Boolean conditions [24];

6) **Lightweight linking technique (LLT) - case sensitive (CS)**: To reference software entities from e-mails, the names of the software entities are used as text search queries. There exists a link between a software entity and an e-mail, when there is a case sensitive match on the entity name;

[1] lucene.apache.org

7) **LLT - mixed approach (MA)**: In case the name of software entities are compounded words, they are split (e.g., `ClassName` becomes `Class Name`). The compounded words are then used for the case sensitive match on the entity name, otherwise it is used a regular expression based on class and package name;

8) **LLT - MA with regular expression (RE)**: This approach is based on that above. A different regular expression is used to better handle non-Java systems. Further details about Lightweight linking techniques can be found in [17].

The baselines from 1 to 5 are different instantiations of the recovery process shown in Section III, while the others are lightweight approaches based on regular expressions. In all the IR-based baseline approaches, with the exception of the first and second one, the corpus was indexed considering together header, body, and quote. For the baselines from 4 to 8, we used the results published in [17]. For these baselines, there were available only the results, when using the class names as the queries.

B. Planning

1) Context: Many IR-based traceability recovery approaches depend on users' choices: the software engineer analyzes a subset of the ranked list to determine whether each traceability link has been correctly retrieved. It is the software engineer who makes the decision to conclude this process. The lower the number of false traceability links retrieved, the better the approach is. The best case scenario is that all the retrieved links are correct. IR-based traceability recovery methods are far from this desirable behavior [13]. In fact, IR-based traceability recovery approaches might retrieve links between source and target artifacts that do not coincide with correct ones: some are correct and others not. To remove erroneously recovered links from the candidate ones, a subset of top links in the ranked list (i.e., retrieved links) should be presented to the software engineer. This is possible by selecting a threshold to cut the ranked list (e.g., [16], [26]).

There are methods that do not take into account the similarity between source and target artifacts: *Constant Cut Point*, it imposes a threshold on the number of recovered links, and *Variable Cut Point*, it consists in specifying the percentage of the links of the ranked list to be considered correctly retrieved. Alternative possible strategies for threshold selection are based on the similarity between source and target artifacts: *Constant Threshold*, a constant threshold is chosen, *Scale Threshold*, a threshold is computed as the percentage of the best similarity value between two vectors, and *Variable Threshold*, all the links among those candidate are retrieved links whether their similarity values are in a fixed interval. In our experiment, we used the Constant Threshold method. This is the standard method used in the literature [13]. We applied this method employing thresholds assuming values between 0 and 1. The increment used was 0.01.

For each software entity, the Query Formulation step was instantiated using either the original class name or the concatenation of class and package names. The order with which class and package names were concatenated is indifferent. We opted for these query formulations because used in [17]. Other instantiations for the Query Formulation step are possible. This point is subject of future work.

2) Variable selection: The traceability links retrieved by applying both our approach and the baselines are analyzed in terms of *correctness* and *completeness*. Correctness reflects the fact that an approach is able to retrieve links that are correct. To estimate the correctness, we used (as customary) the *precision* measure. On the other hand, completeness reflects how much the set of retrieved links is complete with respect to the all actual links. The *recall* measure is used to estimate this aspect. We used here the following definitions:

$$precision = \frac{|TP|}{|TP| + |FP|} \quad recall = \frac{|TP|}{|TP| + |FN|} \tag{4}$$

where TP (true positives) is the set of links correctly retrieved. The set FN (false negatives) contains the correct links not retrieved, while FP (false positives) the links incorrectly presented as correct ones.

When the e-mails in the benchmark do not have any reference to source code artifacts, the union of TP and FN is empty (i.e., $|TP| + |FN| = 0$). In all these cases, we cannot calculate the values for the recall measure. The values for precision could not be computed in case the approach found no link between an e-mail and the source code. Similar to [17], we avoided these issues calculating the average of $|TP|$, $|FP|$, and $|FN|$, on the entire dataset. We then computed the average values for precision and recall. Precision and recall assume values in the interval $[0, 1]$. The higher the precision value, the more correct the approach is. Similarly, the higher the recall value, the better the approach is.

To get a trade-off between correctness and completeness, we applied the balanced F-measure (F_1):

$$F_1 = \frac{2 * precision * recall}{precision + recall} \tag{5}$$

We applied this formula because we would like to emphasize neither recall nor precision. F_1 was used to estimate the *accuracy* of the approach. This is the main criterion we considered in the study. This measure has values in the interval $[0, 1]$. When comparing two approaches, the one with higher F_1 value is considered the best, namely the most accurate.

3) Instrumentation: Regarding the baselines from 1 to 3, we implemented the underlying instances of the process in a prototype of a supporting software tool. This tool was intended as an Eclipse plug-in. It can be downloaded at www.scienzemfn.unisa.it/scanniello/LASCO/. We named that plug-in LASCO (Linking e-mAils and Source Code). In the following, we highlight this tool prototype. The interested reader can find more details on LASCO in [45]. It is worth mentioning that the baselines from 1 to 3 are different instantiations of our process shown in Section III. These instantiations have been implemented in our prototype, but they are not available in the current distribution of LASCO.

In Table I, we report the steps needed to recover links between e-mail and source code (steps 4.a and 4.b) with LASCO. The steps to search e-mails that are similar to a given

TABLE I. USAGE STEPS OF LASCO

Usage Step	Expected Output	Description
1. Selecting the e-mail repository (steps 1 and 2 of the approach)	The e-mail repository is loaded.	In this step, an e-mail repository is loaded and the corpus is created. The corpus is also normalized.
2. Indexing the corpus (step 3 of the approach)	An index of the corpus is created using an IR engine.	Most of IR methods create a numerical index for each document in the corpus. LASCO supports both Lucene and BM25F (two fields, namely header, body and quote together, and three fields, namely header, body and quote). The corpus is also normalized.
3.a Recovering the links (steps 4 and 5 of the approach)	The ranked list for the system/s is computed.	The class name or package and class names are are used as the query. These names are compared with set of target artifacts: the e-mails. The textual query is normalized in the same way as the corpus. The user can use the boolean operators AND and "OR". The ranked list is then computed.
3.b Formulating a textual query (steps 4 and 5 of the approach)	The user exploits LASCO to write a textual query.	The user writes a query and the system retrieves the e-mails that are more similar to that query. The textual query is normalized in the same way as the corpus. The boolean operators AND and "OR" can be used.
4.a The ranked list is shown (step 6 of the approach)	Links between e-mails and source code are shown.	The links between e-mails and software entities (e.g., class names) are reported in a ranked list. The e-mails more similar to the software entities are shown first.
4.b The list of e-mails is shown (step 6 of the approach)	The e-mails similar to the textual query are shown.	The e-mails are reported in a ranked list. The e-mails in that list are in decreasing order with respect to their similarity with the textual query.

textual query (steps 3.b and 4.b) are reported as well. The expected output for each step is mentioned together with its description. We made also clear the connection between each step of the approach shown in Section III and the usage steps of our tool.

Figure 1 shows the top of the ranked list attained on the system Freenet. Each link in that list is characterized by a source artifact (i.e., the class name in this case) and the target artifact (i.e., the e-mail). The similarity score between these two artifact is also reported. LASCO also shows the e-mail (i.e., header, body, and quote) associated to a link double clicking that link in the ranked list. On the other hand, Figure 2 shows the screenshot that allows a user to specify a textual query (step 3.b).

To estimate the correctness, the completeness, and the accuracy of our approach and to compare it with the baselines, we used the benchmark proposed in [18]. To build the benchmark, the authors manually analyzed the e-mails of six unrelated software systems written in four different languages: Java, ActionScript, PHP, and C. 99 versions for these systems were considered. The systems are all open-source and both the code and the e-mails are freely accessible on the web. The benchmark was built on development mailing lists and considered a reliable sample set from all the e-mails in these lists. Some information on the benchmark is shown in Table II. In particular, the first column shows the name of the open source

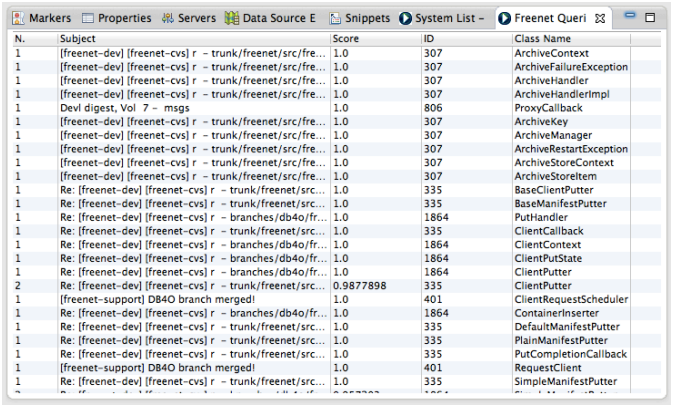

Figure 1. Ranked list attained on Freenet

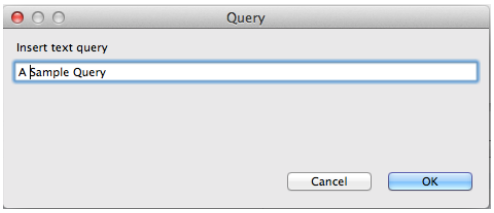

Figure 2. Formulating a textual query in LASCO

software system. A short summary of the functionality of the system and the programming language used to implement it are presented in the second and third columns, respectively. The total number of e-mails for each system are reported in the fourth column. Details on the sample are presented in the last three columns. In particular, these columns shows the size of the statistically significant sample set of e-mails, the number of e-mails in the sample with at least one reference to a software entity, and the total number of links from the e-mails in the sample. Further details can be found in [18].

For each system and all the threshold values, we computed the values of precision, recall, and F_1. To this end, we have implemented and used a tool to automatically collect TP, FP, and FN. To compare our approach with the baselines, we selected the constant threshold that produced the best accuracy. The values of precision, recall, and F_1 are computed in LASCO for our approach and the baselines from 1 to 3.

V. RESULTS AND DISCUSSION

In this section, we present and discuss the results and some lesson learned. The section concludes presenting possible threats that could affect the validity of the results.

TABLE II. SUMMARY INFORMATION ON THE USED BENCHMARK [18]

System	Language	E-mails	Sample Size	E-mails with a link	Total links
ArgoUML	Java	29,112	355	108	290
Freenet	Java	26,412	379	148	570
JMeter	Java	20,554	380	207	617
Away3D	Action Script 3	9,757	370	243	747
Habari	PHP 5	13,095	374	135	252
Augeas	C	2,219	281	140	273

TABLE III. BM25F RESULTS INDEXING THE CORPUS USING: *(i)* HEADER, *(ii)* BODY, AND *(iii)* QUOTE

	Class Name + "AND"			Class Name + "OR"			Class and Package Names + "AND"			Class and Package Names + "OR"		
System	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1
ArgoUML	0.32	0.53	0.40	0.05	0.55	0.10	0.41	0.72	0.52	0.04	0.48	0.07
Freenet	0.23	0.49	0.31	0.03	0.23	0.06	0.30	0.52	0.39	0.02	0.40	0.05
JMeter	0.32	0.41	0.36	0.10	0.41	0.16	0.49	0.62	0.55	0.06	0.43	0.10
Away3D	0.31	0.51	0.39	0.15	0.24	0.18	0.39	0.44	0.41	0.12	0.24	0.16
Habari	0.77	0.48	0.59	0.29	0.35	0.32	0.77	0.48	0.59	0.29	0.35	0.32
Augeas	0.12	0.27	0.16	0.04	0.32	0.08	0.12	0.26	0.16	0.04	0.32	0.08
Average value	0.35	0.45	0.37	0.11	0.35	0.15	0.41	0.51	0.44	0.10	0.37	0.13

TABLE IV. BM25F RESULTS INDEXING THE CORPUS USING: *(i)* HEADER AND *(ii)* BODY AND QUOTE TOGETHER

	Class Name + "AND"			Class Name + "OR"			Class and Package Names + "AND"			Class and Package Names + "OR"		
System	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1
ArgoUML	0.34	0.51	0.41	0.07	0.58	0.12	0.40	0.46	0.43	0.05	0.55	0.09
Freenet	0.22	0.54	0.31	0.08	0.45	0.14	0.29	0.62	0.40	0.07	0.5	0.13
JMeter	0.29	0.45	0.36	0.14	0.41	0.21	0.34	0.66	0.45	0.12	0.45	0.19
Away3D	0.29	0.76	0.42	0.21	0.24	0.22	0.37	0.44	0.40	0.16	0.23	0.19
Habari	0.74	0.52	0.61	0.46	0.45	0.46	0.74	0.52	0.61	0.46	0.45	0.46
Augeas	0.11	0.35	0.17	0.10	0.17	0.13	0.11	0.35	0.17	0.06	0.35	0.10
Average value	0.33	0.52	0.38	0.18	0.38	0.21	0.38	0.5	0.41	0.15	0.42	0.19

TABLE V. LUCENE RESULTS

	Class Name + "AND"			Class Name + "OR"			Class and Package Names + "AND"			Class and Package Names+ "OR"		
System	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1
ArgoUML	0.32	0.50	0.39	0.06	0.50	0.11	0.39	0.47	0.43	0.03	0.53	0.06
Freenet	0.20	0.59	0.30	0.07	0.47	0.11	0.27	0.64	0.38	0.05	0.56	0.10
Jmeter	0.27	0.46	0.34	0.10	0.36	0.15	0.34	0.70	0.46	0.07	0.49	0.13
Away3D	0.29	0.77	0.42	0.17	0.22	0.19	0.37	0.44	0.40	0.13	0.24	0.17
Habari	0.61	0.55	0.58	0.45	0.40	0.43	0.61	0.55	0.58	0.45	0.40	0.42
Augeas	0.10	0.27	0.15	0.05	0.21	0.08	0.10	0.27	0.15	0.05	0.20	0.08
Average value	0.30	0.52	0.36	0.15	0.36	0.18	0.35	0.51	0.40	0.13	0.40	0.16

TABLE VI. RESULTS BY BACCHELLI *et al.* [17]

	VSM with $tf - idf$			LSI			LLT - case sensitive			LLT - mixed approach			LLT - mixed approach with RE		
System	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1	$Precision$	$Recall$	F_1
ArgoUML	0.25	0.34	0.29	0.60	0.48	0.53	0.27	0.68	0.38	0.64	0.61	0.63	0.35	0.68	0.46
Freenet	0.15	0.25	0.19	0.62	0.43	0.51	0.17	0.70	0.27	0.59	0.59	0.59	0.27	0.69	0.39
JMeter	0.21	0.34	0.26	0.52	0.40	0.45	0.15	0.73	0.25	0.59	0.65	0.62	0.30	0.72	0.42
Away3D	0.35	0.31	0.33	0.35	0.33	0.34	0.32	0.74	0.44	0.40	0.54	0.46	0.41	0.72	0.52
Habari	0.34	0.39	0.36	0.36	0.41	0.38	0.40	0.41	0.41	0.83	0.09	0.17	0.49	0.38	0.43
Augeas	0.10	0.20	0.14	0.10	0.28	0.14	0.09	0.72	0.15	0.14	0.02	0.04	0.15	0.64	0.24
Average value	0.23	0.31	0.26	0.43	0.39	0.39	0.23	0.66	0.32	0.53	0.42	0.42	0.33	0.64	0.41

A. Results

The results achieved by applying our approach are summarized in Table III. The table also reports the results achieved by applying the "OR" operator. The results are grouped according to the two different instantiations of the step Query Formulation: *(i)* class name and *(ii)* class and package names. The last row reports the average values for each measure. Better average accuracy was achieved using class and package names and the "AND" operator (F_1 = 0.44). With respect to each individual system, we obtained the higher accuracy for Habari, namely the system implemented in PHP (F_1 = 0.59). On that system, the higher value of correctness was also obtained (precision = 0.77). It is worth mentioning that the results for that system are the same both using class name alone and class and package names together. This is because PHP 5 did not have packages. Namespaces (i.e., packages) where only introduced in PHP 5.3. The same held for Augeas (the C software system).

Table IV shows the results achieved by indexing the corpus using: *(i)* header and *(ii)* body and quote together. With respect to accuracy, better results were achieved using the operator "AND" and class and package names. The best average accuracy value was 0.41. Among the analyzed software systems, the best accuracy was obtained for Habari (F_1 = 0.61). Figure 3 summarizes the accuracy results achieved by indexing the corpus considering header, body, and quote (three fields) and header and body and quote together (two fields). This figure shows that: it is better to use the "AND" operator, the use of three fields produces better results, and formulating the query as class and package names is better.

The results achieved with Lucene are shown in Table V. The best average accuracy value was reached using the operator "AND" and class and package names (i.e., 0.40). The better accuracy was achieved for Habari (F_1 = 0.58).

These results are presented for each system in the benchmark and are grouped according to the operator used in the Query Formulation step. The average values are reported in the last row. As for BM25F, we obtained on average better results by applying the "AND" operator and using class name and

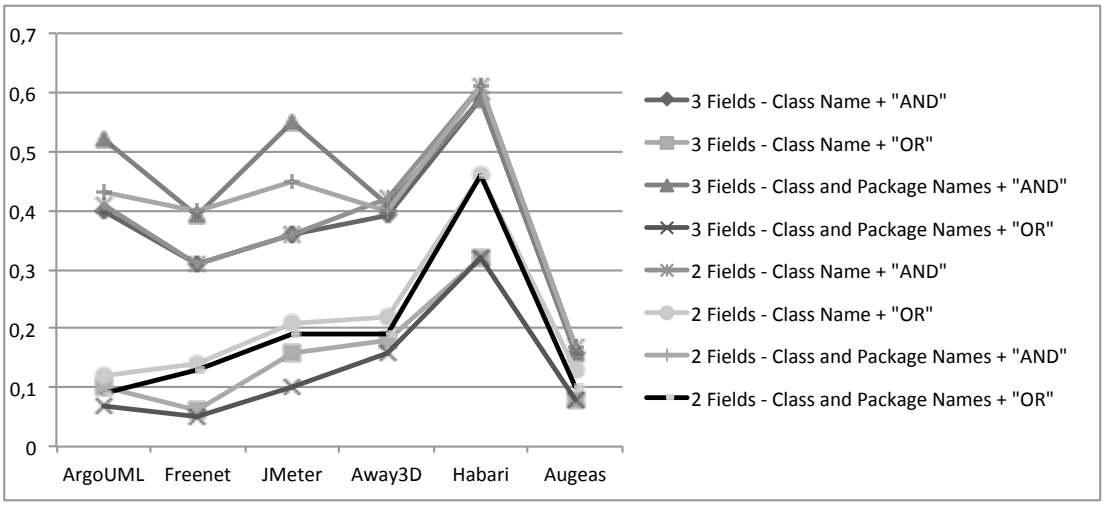

Figure 3. Accuracy results achieved with the "AND" and "OR" operators and indexing the corpus using: *(i)* header, body and quote and *(ii)* header and body and quote together

package. The comparison of these results with those achieved with our approach suggests that the use of BM25F improves the correctness and the accuracy of the retrieved traceability links. The main effect of the probabilistic model used is on the $precision$ values. The average improvement ranges from 3% to 5%. The improvement on the accuracy is on average 2% with class name and the "AND" operator, while is 1% class name and package and the "AND" operator. The use of that model does not improve completeness. The benefit deriving from the instantiation based on the probabilistic model are still better, when using the "OR" operator. The retrieved links are more precise, complete, and then accurate.

Table VI summarizes the results presented in [17], instantiating Query Formulation step with class name. As mentioned before, the results for class and package names together are not reported for VSM and LSI because the authors observed that better results were achieved using only class names. Table VI also shows the results for the lightweight linking techniques.

The results indicate that our proposal is more accurate than BM25F using two fields (header and body and quote together) on all the Java systems with the exception of Freenet (the F_1 values were 0.39 and 0.40, respectively). On the non-Java systems, the use of BM25F indexing the corpus with three or two fields did not produce remarkable differences in accuracy (see the Tables III and IV and Figure 3).

Our approach using class and package names as the queries is more accurate than VSM. Similar results were achieved for Lucene using both the operators and class name and class and package names together as the queries. Indeed, our proposal did not outperform Lucene only on Away3D when using the "AND" operator and class name as the query. The F_1 values were 0.41 and 0.42, respectively.

As far as LSI is concerned, our approach is more accurate on all the non-Java system and Jmeter. For ArgoUML the difference in favor of LSI was negligible (the F_1 values was 0.52 with respect to 0.53). A larger difference in accuracy was obtained for Freenet.

Regarding the lightweight approaches, our proposal outperformed LLT-CS in accuracy on all the systems with the exception of Away3D (the F_1 values were 0.44 and 0.41, respectively). BM25F with three fields was on average more accurate than LLT MA with and without RE (see the average values of F_1). With respect to LLT MA, we achieved better F_1 values results on Habari and Augeas (0.59 vs. 0.17 and 0.16 vs. 0.04, respectively). On the Java systems, LLT MA was more accurate than our approach. For LLT MA RE, we reached better results on the Java systems and Habari.

Regarding the correctness and completeness of the retrieved links, we can observe an interesting pattern in the data: our approach mostly allowed obtaining a more complete set of retrieved links that are correct. This result is desirable when you are interested in the recovery of links among software artifacts (e.g., [13]).

B. Discussion

Several aspects must be taken into account before drawing conclusions. We discuss IR-based approaches first and than the lightweight ones. The section concludes with our overall recommendation.

1) IR-Based recovery: For the Java systems, LSI outperformed other approaches based on IR techniques with respect to the accuracy of the retrieved links. A possible reasons is that each e-mail in the corpus quotes a large number of sentences from previous messages. This is the best scenario for using LSI [24]. In fact, this technique is used to disclose the latent semantic structure of the corpus and to recognize its topics, so dealing with synonymy and polysemy problems. Further, each document in the corpus has a large size as compared with the entities used as the queries. This might also represent another possible reason for having achieved better results for the Java systems. The considerations above and the fact that LSI outperformed our approach in terms of accuracy only on

Freenet (this difference was 0.04, while this difference was in favor of our approach on ArgoUML and JMeter and was 0.01 and 0.1, respectively) suggest that BM25F represents a viable alternative also when dealing with large documents in the corpus. It is also possible that differently tuning the parameters of our solution (e.g., the non-linear saturation) the difference with LSI could decrease on Freenet. This represents a possible future extension for our study.

In case the e-mails in the corpus quote a small number of sentences from previous e-mails, our approach outperformed other baseline approaches based on IR techniques. This happened for all the non-Java systems. For the Habari system, the e-mails were very short and then BM25F made the difference also considering the information in the body and quote together.

For the system implemented in C (i.e., Augeas), the application of the IR-Based approaches mostly produced worse results in terms of correctness, completeness, and accuracy. As also suggested in [17], a possible justification is related to the names of the entities. However, our approach outperformed the IR based baselines. Again, indexing the e-mails considering two or three fields did not produce remarkable differences.

The instantiation of the Query Formulation step with class and package names improved the correctness and completeness when our technique was used. Then, it is possible that the choice of the source artifact can make the difference in the accuracy of the links recovered. This point needs future and special conceived investigations.

The use of a stemming technique in the Normalization step produced worse results. Then, this technique seems useless in the recovery of links between source code and e-mails, when using BM25F (with two and three fields) and Lucene. On these instances, the use of the "AND" operator led to better results in terms of accuracy and correctness of the retrieved links with respect to the "OR" operator. This result held for all the systems. For completeness, the results achieved with the "AND" operator were mostly better than those achieved with the "OR" operator. Only in four cases the use of the "OR" operator led to better recall values.

The use of source code (program statements and/or source code comment) as the query was also analyzed. The results revealed that this kind of instantiation for the Query Formulation step led to worse results with respect to the other two kinds of queries considered here. This result is in line with that shown in [17] and has the following implication: it is better to use class name and class and package names as the queries.

We also performed an analysis to get indications on whether BM25F might introduce scalability issues. We used a laptop equipped by a processor Intel Core i7-2630QM with 4 GB of RAM and Windows Seven Home Premium SP-1 64bit as operating system. This analysis was performed on each system and the baseline processes implemented for our experiment (see Section IV-A). The results indicated that the time to build, normalize, and index the e-mails of the entire benchmark was twice when using three fields (i.e., 5033 milliseconds) with respect to the use of two fields (i.e., 2668 milliseconds). For Lucene, the average execution time on all the systems in the benchmark was 2660 milliseconds. For the Query Formulation step, nearly the same pattern was observed. Further details are not provided for space reason.

2) Lightweight Approaches: Regarding the accuracy of the retrieved links, LLT MA outperformed other lightweight techniques and our approach on the Java systems. On the non-Java system with the exception of Away 3D, LLT MA did not outperform our approach and the differences in the F_1 values were significant (0.59 vs. 0.17 and 0.16 vs. 0.04, respectively). The difference on Away3D was small (F_1 values were 0.41 and 0.44, respectively). Similarly, LLT MA did not outperform LLT MA RE on the non-Java systems. The achieved results suggest that our approach and LLT MA RE are more independent from the kind of documents in the corpus. Since our approach was more accurate, we can then conclude that it is the best and can be applied without making any assumption on the mailing list and the programming language of the understudy system. The same did not hold for lightweight techniques based on regular expressions because they heavily rely on common conventions and intrinsic syntactical characteristics of the corpus [17].

C. Lesson Learned

The accuracy of our approach increased when e-mails contain a huge amount of text and the entity names are carefully chosen and naming conventions are used. Furthermore, when e-mails did not contain a huge amount of text, the application of BM25F on two or three fields did not produce noteworthy differences. Then, BM25F on header, body, and quote with the operator "AND" is the best alternative (see Figure 3).

We experimentally observed that, in terms of accuracy, our approach outperformed on 5 out of 6 systems the lightweight technique that is more independent from the kind of e-mails in the corpus (i.e., LLT MA RE) [17]. To apply our approach, any assumption on the system understudy has to be made and any particular configuration setting is required. Therefore, our approach is easier to use than lightweight approaches and it is accurate enough to be worth the costs it may introduce in the corpus preprocessing and indexing phases. Furthermore, IR-based approaches, such as the one we introduce here, are more scalable. They are more efficient than lightweight techniques when the number of e-mails in the corpus increases. Finally, lightweight techniques return documents without any ranking: an e-mail either matches or not a regular expression. As a consequence, all the retrieved links have to be analyzed. To deal with this issue, text retrieval and text search techniques could be used in combination. This point could be the subject of future work.

1) Pieces of evidences: We distilled the achieved findings and lesson learned into the following pieces of evidence (PoE):

- PoE1. Accuracy increases when using class and package names as the queries;
- PoE2. Applying our approach on three fields (i.e., header, body, and quote) improves the results when the corpus contains e-mails with a huge amount of text and the entity names are carefully chosen by developers;
- PoE3. Using the "AND" operator leads to better results in terms of correctness, completeness, and accuracy;

PoE4. The corpus normalization by using stemming techniques reduces the accuracy of the recovered links;
PoE5. Our approach scales reasonable well also when the number of documents in the corpus increases;
PoE6. Our approach is more independent from the mailing list than lightweight approaches.

D. Threats to Validity

To comprehend the strengths and limitations of our study, we present here the threats that could affect the validity of the results and their generalizability. Although our efforts in mitigating as many threats as possible, some threats are unavoidable. For example, a possible threat to the validity of the results is related to the used benchmark. It is built on human judgement and then links in the benchmark could be wrong. To alleviate this threat to the construct validity, the authors of the benchmark applied several strategies (see Section 6.1 in [17]). Another threat related is that the researchers involved in the creation of the benchmark were unfamiliar with the systems and then they could have missed implicit references to software entities that an actual developer might understand. The use of a sample set of the entire dataset may also affect the validity of the results.

The use of open source software represents another threat to validity. To alleviate this threat, the benchmark was built on 6 different systems developed from separate communities and implemented in 4 programming languages based on two paradigms: object-oriented and procedural. Despite this effort, there are some differences between commercial and open source software systems. For example, open source software is usually developed outside companies mostly by volunteers and the development methodologies used are different from the ones commonly applied in the software industry. Although many large companies are using open source software in their own work or as a part of their marketed software, it will be worth replicating the study on real project. These replications will help us to confirm or contradict the achieved results.

The instantiation of the Query Formulation step represents another threat. We used class names or class and package names. The observed results suggest that this aspect influences the accuracy, correctness, and completeness of the results. In this work, we used the instantiations above to compare our approach with those in [17]. Empirical studies are needed to better assess how different kinds of queries affect the quality of the retrieved links.

Further threats concern the validity of the comparison between the results achieved with our approach and those obtained with the baselines. In the experiment presented in this paper, we could not perform statistical analyses because the results presented in [17] were not provided in an adequate form and replications were not possible (e.g., regular expressions were not described at an adequate level of detail). Although a comparison was possible between our approach and the baselines from 1 to 3, at this step of our research we preferred to propose a new approach and compare it with the lightweight approaches proposed in [17], namely the state of the art in the recovery of links between e-mails and source code.

VI. Conclusion

For software maintainers, who are unfamiliar with a software system, the recovery of links among free-form natural language software artifacts can be a laborious task if performed manually. We proposed, implemented, and evaluated an approach to recover links between e-mails and source code. The approach is based on text retrieval techniques combined with the BM25F probabilistic model. We used this model because it showed very good performances [20], [35], [36]. The defined approach is general and can be applied to software implemented with any programming language and to any kind of documents (i.e., e-mails) in the corpus. To assess the validity of our proposal, we conducted an empirical study using a public benchmark [18]. Based on this benchmark, we performed a comparison between our approach and 8 baselines. The results indicated that our approach in many cases outperformed the IR-based baseline approaches and the lightweight techniques proposed in [17].

Furthermore, our approach scales well when the number of e-mails increases and it does not require any assumption on the system understudy. Traditionally, probabilistic IR has had neat ideas but the methods have never won on performance [24]. This is possibly due to the severity of the modeling assumptions that makes achieving good performance difficult. Things changed when the BM25 weighting method was introduced. Our results confirm that in a different context.

Acknowledgment

The authors would like to thank the Michele Lanza and the Alberto Bacchelli for their support and for having made available the benchmark used in this work. Special thanks are due to the Anna Tolve and the Raffaele Branda, who developed some of the software modules of the prototype implementing the approach presented here.

References

[1] R. Branda, A. Tolve, L. Mazzeo, and G. Scanniello, "Linking e-mails and source code using bm25f," in *International Conference on Software Engineering Advances*, 2013, pp. 271–277.

[2] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Professional*, vol. 2, pp. 17–23, 2000.

[3] M. V. Zelkowitz, A. C. Shaw, and J. D. Gannon, "Principles of software engineering and design," 1979.

[4] M. M. Lehman, "Program evolution," vol. 19, no. 1, pp. 19–36, 1984.

[5] E. B. Swanson, "The dimensions of maintenance," in *Proc. of International Conference on Software Engineering*. IEEE CS Press, 1976, pp. 492–497.

[6] A. V. Mayrhauser, "Program comprehension during software maintenance and evolution," *IEEE Computer*, vol. 28, pp. 44–55, 1995.

[7] S. Pfleeger and J. Atlee, *Software Engineering - Theory and Practice*. Pearson, 2006.

[8] A. De Lucia, F. Fasano, C. Grieco, and G. Tortora, "Recovering design rationale from email repositories," in *Proc. of the International Conference on Software Maintenance*. IEEE, 2009, pp. 543–546.

[9] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Trans. Software Eng.*, vol. 32, no. 1, pp. 4–19, 2006.

[10] A. Bacchelli, M. Lanza, and M. D'Ambros, "Miler: a toolset for exploring email data," in *Proceedings of the International Conference on Software Engineering*. ACM, 2011, pp. 1025–1027.

[11] N. Bettenburg, B. Adams, A. E. Hassan, and M. Smidt, "A lightweight approach to uncover technical artifacts in unstructured data," *International Conference on Program Comprehension*, vol. 0, pp. 185–188, 2011.

[12] N. Bettenburg, S. W. Thomas, and A. E. Hassan, "Using code search to link code fragments in discussions and source code," in *CSMR '12: Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, IEEE. IEEE, 2012, pp. 319–329.

[13] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, 2007.

[14] H. Sultanov and J. H. Hayes, "Application of swarm techniques to requirements engineering: Requirements tracing," in *Proc. of IEEE International Requirements Engineering Conference*, ser. RE '10. IEEE CS Press, 2010, pp. 211–220.

[15] S. K. Sundaram, J. H. Hayes, A. Dekhtyar, and E. A. Holbrook, "Assessing traceability of software engineering artifacts," *Requir. Eng.*, vol. 15, no. 3, pp. 313–335, 2010.

[16] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. Software Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[17] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proc. of International Conference on Software Engineering*. ACM, May 2010, pp. 375–384.

[18] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes, "Benchmarking lightweight techniques to link e-mails and source code," in *Proc. of Working Conference on Reverse Engineering*. IEEE Computer Society, 2009, pp. 205–214.

[19] S. Robertson, H. Zaragoza, and M. Taylor, "Simple bm25 extension to multiple weighted fields," in *Proc. of International Conference on Information and Knowledge Management*, ser. CIKM '04. ACM, 2004, pp. 42–49.

[20] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Found. Trends Inf. Retr.*, vol. 3, pp. 333–389, April 2009. [Online]. Available: http://dx.doi.org/10.1561/1500000019

[21] IEEE, *IEEE Recommended Practice for Software Requirements Specifications*, Std., 1998. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=720574

[22] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *Computer*, vol. 40, pp. 27–35, June 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1271918.1271947

[23] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia, "Maintaining traceability links during object-oriented software evolution." *Softw., Pract. Exper.*, vol. 31, no. 4, pp. 331–355, 2001.

[24] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[25] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[26] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proc. of the International Conference on Software Engineering*. IEEE CS Press, 2003, pp. 125–137.

[27] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Proc. of the International Conference on Program Comprehension*. IEEE CS Press, 2008, pp. 103–112.

[28] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Traceability recovery using numerical analysis," in *Proc. of the International Working Conference on Reverse Engineering*. IEEE CS Press, 2009, pp. 195–204.

[29] A. De Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery using smoothing filters," in *Proc. of the International Conference on Program Comprehension*. IEEE CS Press, 2011, pp. 21–30.

[30] A. De Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: a silver bullet for traceability recovery," in *Proc. of the International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 299–309.

[31] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *Proceedings of the International Conference on Software Maintenance*. IEEE CS Press, 2011, pp. 133–142.

[32] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in *Proc. of the International Workshop on Program Comprehension.*, 2002, pp. 271–278.

[33] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[34] K. S. Jones, S. Walker, and S. E. Robertson, "A probabilistic model of information retrieval: development and comparative experiments," *Inf. Process. Manage.*, vol. 36, pp. 779–808, November 2000.

[35] K. Y. Itakura and C. L. Clarke, "A framework for bm25f-based xml retrieval," in *Proc. of International Conference on Research and Development in Information Retrieval*. ACM, 2010, pp. 843–844.

[36] J. R. Pérez-Agüera, J. Arroyo, J. Greenberg, J. P. Iglesias, and V. Fresno, "Using bm25f for semantic search," in *Proc. of the International Semantic Search Workshop*. ACM, 2010, pp. 2:1–2:8.

[37] J. Pérez-Iglesias, J. R. Pérez-Agüera, V. Fresno, and Y. Z. Feinstein, "Integrating the Probabilistic Models BM25/BM25F into Lucene," *CoRR*, vol. abs/0911.5046, 2009. [Online]. Available: http://arxiv.org/abs/0911.5046

[38] G. J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, 2nd ed. Wiley-Interscience, March 2008.

[39] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006. [Online]. Available: http://books.google.it/books?id=Clc7PwAACAAJ

[40] L. Dolamic and J. Savoy, "When stopword lists make the difference," *J. Am. Soc. Inf. Sci. Technol.*, vol. 61, no. 1, pp. 200–203, Jan. 2010. [Online]. Available: http://dx.doi.org/10.1002/asi.v61:1

[41] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[42] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering - An Introduction*. Kluwer, 2000.

[43] V. Basili, G. Caldiera, and D. H. Rombach, *The Goal Question Metric Paradigm, Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

[44] J. K. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 1*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[45] L. Mazzeo, A. Tolve, R. Branda, and G. Scanniello, "Linking e-mails and source code with lasco," in *Proc. of the European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2013.