Supporting Traceability through Affinity Mining

Vincenzo Gervasi*†, Didar Zowghi†
*Dipartimento di Informatica, Università di Pisa, Italy
†School of Software, University of Technology Sydney, Australia

***Abstract*—Traceability among requirements artifacts (and beyond, in certain cases all the way to actual implementation) has long been identified as a critical challenge in industrial practice.**

Manually establishing and maintaining such traces is a high-skill, labour-intensive job. It is often the case that the ideal person for the job also has other, highly critical tasks to take care of, so offering semi-automated support for the management of traces is an effective way of improving the efficiency of the whole development process.

In this paper, we present a technique to exploit the information contained in previously defined traces, in order to facilitate the creation and ongoing maintenance of traces, as the requirements evolve. A case study on a reference dataset is employed to measure the effectiveness of the technique, compared to other proposals from the literature.

I. Introduction

In a talk at RE'2003 about measuring the mismatches between areas of interest of researchers and practitioners in RE [1], Martin Feather lamented *requirements traceability* as a practically important yet under-served area of RE research. In the 10 years since, his call has been taken up by several groups of researchers, and numerous techniques to help provide automatic or semi-automatic support for the establishment and maintenance of traces between requirements artifacts have emerged.

Early proposals have drawn mostly from the repertoire of Information Retreival (IR) techniques (e.g., using Vector Space Model and TF-IDF approaches), essentially positing that *lexical* similarity between two documents (or fragments thereof, e.g. single requirements) is a strong indicator that the two should be linked in a traceability relation. These approaches, while useful and reasonably effective as an initial attempt, suffered from certain limitations, namely:

1) They ignore the different nature of different relationships between requirements. A trace can embody a technical refinement, a goal-means relationship, a make-break relationship, a requirement-test, etc., but the only criterion considered is the occurrence of the same terms in the two artifacts.
2) They ignore that artifacts of different nature can employ different sublanguages — for example, marketing terms on one side, technical jargon on the other. In such circumstances, relying on lexical similarity will yield unsatisfactory results.
3) They do not leverage the effort already put into establishing links. In effect, every new artifact is treated as a fresh new query into an IR system, with the results used as candidate for establishing traces, and with no memory of the past.

To overcome these limitations, more recent approaches have applied Machine Learning (ML) techniques instead of or in addition to classical IR techniques. In general terms, ML approaches seek to extract information from pre-existing traces, use the information to build a model of the previous linking patterns, and leverage the model in suggesting candidates for future traces. Such techniques help overcome limitations 1) and 3) mentioned above, in that the future suggestions tend to mimic as far as possible what already had been done manually in the past, thus leading to an efficient re-use of the effort and knowledge put into establishing an initial set of traces.

As an aside, it should be noted that the pre-existing traces are not necessarily the "perfect" ones. It is entirely possible that an automated technique can identify relevant traces that had been missed by the human analyst. However, given the current state of the art, human analysts tend to be significantly more effective at establishing traces than existing algorithms, when traceability is considered at all. Indeed, in common industrial practice – and especially in small-to-medium enterprises – the real problem seems to be having any traceability in place at all, rather than having a perfect set of traces.

In this work, we focus on limitation 2) above, i.e. on those cases where different terms, choice of wording, style and level of abstraction are used in diverse, but related, requirements documents that need to be linked. Typical examples include linking marketing requirements to technical specifications, or business goals to requirements and to use cases or scenarios. At the same time, we apply the general framework of ML approaches, thus addressing 1) and 3) as well. Results show that even simple techniques can be remarkably effective in *assisting* a human analyst in the task of establishing traces.

Our main contribution in this work is a simple technique based on the concept of *affinity*, which is a measure of how the occurrence of certain pairs of terms increases the likelihood that two requirement artifacts should be linked, based on historical data. We formally define the problem of incremental traces construction, propose a solution based on affinity, and validate the solution by means of two experiments on industrial datasets, comparing our results with others published in the literature.

The paper is structured as follows. In Section II we lay out our problem in formal terms, and establish needed notation. Section III describes the technique we are proposing for identifying candidate links, based on affinity mining; this is followed by an account of our experimental validation in Section IV. We discuss the applicability of the technique, and

978-1-4799-3033-3/14 © 2014 IEEE

RE 2014, Karlskrona, Sweden

Accepted for publication by IEEE. © 2014 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

possible extensions, in Section V. This is followed by a review of related work, and some conclusions.

II. Problem Formalization

We consider a scenario in which two sets of textual artifacts, each consisting of a single unit which we will call without loss of generality[1] a *requirement*, are to be linked with each other by means of *traces* of unspecified (but homogenous) semantics. A trace between two requirements means that the two of them are related according to the considered semantics.

Due to historic usage, we will call the two sets of requirements H (for: high-level) and L (for: low-level), with $H = \{h_1, \ldots, h_n\}$ and $L = \{l_1, \ldots, l_m\}$. The traces $E = \{e_1, \ldots, e_p\}$, where each $e_k = (h_{i_k}, l_{j_k})$ are thus the edges of a bipartite graph $G = (H \cup L, E)$.

The problem we want to address can then be formalized as follows: given a current (non-empty) set of traces E, and a new incoming requirement h', identify a set of requirements $L' \subseteq L$ such that the links (h', l'), where $l' \in L'$, represent the same relationship between h' and elements of L as the pre-existing links in E did for elements of H and elements of L.

A few observations should be noted. First, the problem is stated here in terms of a new incoming h', but its symmetrical version with a new l' is totally analogous. We prefer the formulation with an incoming high-level requirement because this corresponds to a frequently occurring situation in practice, i.e. when a new user-level requirement is presented, and the analyst wants to check which technical requirements are related to the novel user request.

Second, the problem is stated in terms of the *insertion* of a new requirement in H, but *deletions* are easily modeled (by dropping from E all the links from or to the removed requirement), and *edits* can be modeled as a deletion followed by an insertion[2]. We will discuss in Section V how such an extension could be exploited to support traceability maintenance.

Third, in this paper we consider the simplest case with traces between just two sets of artifacts, and simple edges between two requirements at a time. In certain scenarios, we could find multiple artifacts or hyperlinks instead (i.e., links between a subset of H and a subset of L, for example when a single marketing requirement is *implemented-by* a set of technical requirements, but not by any one of them alone), or different set of links, with different semantics (i.e., E is partitioned into subsets $E_1, \ldots E_z$, for example when certain edges represent *implemented-by* whereas others represent *conflicts-with*). For expository purposes, we will keep to the simplest case in illustrating the technique, and will discuss later in Section V how our approach could be extended to these more complex scenarios.

[1] I.e., we are not interested in the present work in their particular role as goals, marketing requirements, technical requirements, bug reports, etc.

[2] As a minor technical detail, in a real system the knowledge extracted from links that are dropped following a deletion should be preserved, whereas knowledge extracted from links that are explicitly deleted should be discarded, since in that case it is the trace itself that has been deemed invalid.

III. Proposed Technique

In this section we will first present the intuition behind our notion of *affinity* and the rationale for its application in supporting the establishment of traces between requirement artifacts. We will then provide a technical description of the operations to be performed during the *training* phase (when affinity values are mined from pre-existing traces) and during the *application* phase (when mined affinity values are used to suggest candidate links for a new incoming requirement).

A. Rationale

In most practical cases, the information that is carried by the establishment of a link between two requirements expresses the fact that the linked requirements predicates over the same or related domain objects, or concern identical or related actions to be performed, or they specify that the same event has to be detected to trigger some action, etc. Such relationships are often hinted at by *lexical* elements in the requirements: e.g., when two requirements talk about the same system component, identified by a specific name, they are probably related.

Approximating the desired relationship by means of lexical similarity has been found to be an effective technique in many real-world applications (see, among others, the study on linking marketing requirements to technical requirements in [2], [3]). The underlying theory is that if the same terms appear in each of the two requirements, they stand a good chance of being related. From that theory, numerous methods based on classical Information Retrieval techniques (such as TF-IDF [4]) have been proposed, validated via experiments, and ultimately implemented in tools.

Those techniques, however, are applicable only when the exact same terms are used in both H and L. In many cases, the language used in the two artifacts tend to differ, e.g. H could be users' requests for new features in the software (expressed in terms of the users' domain vocabulary), whereas L could be technical specifications of the internal structure of the software (expressed according to the developers' vocabulary).

In our earlier work [5], we advocated using *affinity* between terms from H and L (i.e., a measure of how frequently requirements in which certain pair of terms occurred, were linked in pre-existing data) instead of mere occurrence of the same terms, in order to estimate the likelihood that a given pair of requirements (one of them new) should be linked with each other.

More precisely, in [5] we studied how affinity data mined from pre-existing requirements links could provide information about domain glossaries. Among the findings, we demostrated on a case study that the technique found that almost 90% of the terms used in requirements had a positive affinity towards themselves. In other words, affinity agrees with techniques based on occurrences of the same terms on 90% of the cases. However, in the remaining 10% of the cases, the occurrences of identical terms provided a *negative* contribution to the likelihood of links between requirements. For those terms, techniques such as TF-IDF would lead to a wrong candidate being suggested. Additionally, affinity mining identified a

$r =$"The DPU-TMALI shall configure the ping-pong frame limit at startup as specified by TMALI_PP_LIMIT provided during initialization. The default value shall be M frames and shall be capable of being modified dynamically."

$T(r) = \{$TMALI_PP_LIMIT, VALU, M, LIMIT, DPU-TMAL, DEFAULT, MODIFI, FRAME, SPECIFI, BE, DYNAM, CONFIGUR, CAPABL, STARTUP, PROVID, INITI, PING-PONG $\}$

Fig. 1. An example for the term-extraction function T.

large number of pairs of different terms that had a positive contribution to the linking likelihood, and that would be ignored by techniques based on occurrences of identical terms. Thus, on an artificial subset of the requirements from a publicly available case study, selected such that the probability of a link between two requirements would be on average 50%, our affinity-based technique exhibited a remarkable 95.7% recall and precision, compared to 86.5% for the standard cosine correlation measure.

B. A Measure for Affinity

In the present work, we used a simplified version of affinity, computed from pre-existing links as follows.

Each of the two requirements documents, H and L, is pre-processed, and for each requirement r (either an h_i or a l_i) a set of *terms* $T(r) = \{t_1, \ldots, t_{p_r}\}$ is extracted by means of a standard POS-tagger[3]. We consider the stemmed form of nouns, adjectives, adverbs and verbs as terms, ignoring instead prepositions, conjunctions, disjunctions, pronouns, interjections, particles, determiners, modal verbs, numerals, and various punctuation. An example for T, applied to a real requirement from the case study described in Section IV, is shown in Figure 1.

Since we consider *sets* of terms, information about the number of occurrences is lost at this stage. More refined measures of affinity might potentially use such information. However, in our tests requirements were rather terse, so that rarely a term occurred more than once in a requirement. As a consequence, ignoring multiple occurrences does not have a significant impact for the kind of documents we are considering.

Once T is available, affinity α between terms is computed as follows:

$$\forall\ t_h \in \bigcup_{h \in H} T(h),\ t_l \in \bigcup_{l \in L} T(l),$$

$$\alpha(t_h, t_l) = \#\{(h', l') \in E \mid t_h \in T(h') \wedge t_l \in T(l')\}$$

In intuitive terms, $\alpha(t_h, t_l)$ is a count of how many links already exist between high-level requirements in which t_h occur, and low-level requirements in which t_l occurs.

It should be noted that the measure α is a rather crude approximation of our concept of affinity, since we simply count the number of links already established between requirements that include the given pair of terms. This definition, however, has the advantage of being easily understandable and intuitively sound, and particularly easy to apply in practice even where there is no sophisticated tool available for requirements traceability[4].

Another interesting observation is that α need not be scaled according to document size, since its definition is only significant in the context of two given requirement documents (H and L), and hence any normalization to try to make it independent from the the size of H and L would not provide any real advantage.

C. Learning

Armed with the definitions above, the process of learning affinity from pre-existing data can be simply stated. Given two requirements documents H and L, and a set of already established traces E, compute α for all terms appearing in $H \cup L$, according to its definition.

The learning phase needs not be repeated from scratch every time a change occurs in H, L or E. In fact, our definition for α supports easy differential updating without requiring massive recomputations. For example, if a requirement h is removed from H, causing the cascade removal of a certain sets of links $E' \subseteq E$, where $e = (t_h, t_l) \in E' \implies t_h \in T(h)$, it is sufficient to subtract 1 to the affinity scores of all pairs $\alpha(t_h, t_l)$. In the same way, the establishment of a new link corresponds to a simple increment of the corresponding $\alpha(t_h, t_l)$.

Such differential updates can be done in real-time. This is important, because the technique enables taking into account linking decisions by the human analyst immediately, e.g. when clicking to confirm a link candidate proposed by a requirements management tool, the sets of further candidates can be updated accordingly to incorporate the additional information provided by expert confirmation.

D. Application

The application scenario that we consider in this paper is having a tool to support a human analyst in taking linking decisions when a new requirement is considered for inclusion in a requirements document.

Let $G = (H \cup L, E)$ be the pre-existing set of traces, T be the term-extracting function based on linguistic processing as described above, and $\alpha_{G,T}(\cdot, \cdot)$ be the corresponding affinity measure.

When a new requirement, say h', is presented for inclusion in H, the analyst has to identify which pre-existing requirements in L should be linked with h'. Our proposal provides a ranked list of requirements that are candidate for linking, with the highest-likelihood candidates (according to affinity score) at top.

[3]As will be better described in Section IV, we used the OpenNLP suite for linguistic processing.

[4]In fact, in [5] a more sophisticated measure was defined, which included term frequency and inverse document frequency as part of the definition.

In detail, the rank of each $l \in L$ in the list of candidates for being linked to h' is determined according to its score $\rho_{h'}(l)$ defined as

$$\rho_{h'}(l) = \sum_{\substack{t_{h'} \in T(h') \\ t_l \in T(l)}} \alpha(t_{h'}, t_l)$$

The human analyst can then examine the ranked list, and confirm or reject linking suggestions according to his or her own judgement.

For presentation purposes, the list of ranked candidates can be limited to a certain threshold for $\rho_{h'}(l)$, say τ, or to a fixed number of elements, say k. Alternatively, the list could be returned in its entirety, and then the analyst could be instructed to stop perusing the list after a certain number of consecutive items are judged not suitable for linking. The most appropriate choice for limiting the list returned can vary based on the particular context of application, so we do not investigate here the effect of different policies (in the next section, we will use a variable threshold τ for measurement purposes).

IV. Experimental Evaluation

We provide an evaluation based on an experiment, with a comparison to two other techniques proposed in the literature for the same problem.

A. Method

Our main hypothesis is that *affinity*, as approximated by our measure α, is a good predictor for whether a new requirement should be linked to a set of pre-existing ones. The alternate (null) hypothesis is that affinity is no better than random chance in determining whether two requirements should be linked.

Additionally, we are interested in measuring how good our affinity-based method fares compared to two other methods: Vector Space model (weighted according to TF-IDF) [4], and Reinforcement Learning [6].

Our reference for a correct set of links will be the judgement of an experienced analyst, familiar with the problem. To remove any threat of researcher bias, we use a dataset (both H and L requirements, and a set of manually-established traces E) that were developed independently from the current research endeavour. To facilitate comparison with other techniques, we use a dataset that has been developed in an industrial context and that has been used by other researchers in other traceability-related studies.

All computations are performed by computer programs implementing the various formal definitions from Sections II and III, thus reducing the threat of clerical errors in processing the data.

Being based on a single experiment, our method could refute our main hypothesis in case of failure, but in case of success cannot prove the generalizability of our proposed technique to different datasets or contexts. Thus, results can serve as a first indication of effectiveness, but a larger, systematic study would be needed to ultimately prove its wider applicability.

B. Experimental Design

1) Dataset: We used a published dataset for CM-1, a scientific instrument (to be carried onboard a satellite) whose requirements were developed by NASA and made available to the scientific community as part of the NASA Metrics Data program.

The full dataset consists of numerous documents, including defect reports with links to code artifacts. For our study, we considered two textual documents: a Requirements document (our H) consisting of 235 requirements, mostly 1–3 paragraphs long, and a Design document (our L) consisting of 220 design description statements. A total of 361 links were manually established between these two documents. The full dataset has a very low link density: of 51700 possible pairs (h, l), only 361 (0.7%) were correct links.

For the purpose of comparing to the Reinforcement Learning method from [7], we measured the performances of our Affinity method on a subset of the full CM-1 requirements, the same subset used in [7] and there named CM1SUB. This reduced dataset consists of 22 high-level requirements (H) and 53 low-level design statements (L), linked by a total of 45 links (E). CM1SUB has a slightly higher link density (3.9%) compared to CM-1, yet the task of automatically identifying the 45 correct links among the 1166 possible pairs remains challenging.

2) Linguistic Processing: We used the OpenNLP toolset [8] for the needed preprocessing at the linguistic level. OpenNLP can be configured for specific applications by training it on a given corpus; however in our experiment we wanted to simulate the situation where no special knowledge about the requirements language, style or domain is needed beforehand. Hence, we used the standard models for English distributed with OpenNLP itself for *sentence detection* (used, for example, to distinguish when a dot is just part of an abbreviation, or when it indicates a full stop ending a sentence) and *tokenization* (used to identify lexically significant chunks of text inside a sentence). After sentence detection and tokenization, a requirement r (either from H or L) consisted of a sequence of sentences, each consisting in a sequence of lexical tokens.

Part-of-speech tagging was performed on each sentence by using OpenNLP's maximum entropy POS tagger, which classifies tokens according to a slightly enriched variation of Penn's Treebank set [9]. As we described in Section III-B, only tokens classified as nouns, adjective, adverbs and verbs were considered; tokens in other classes were ignored and discarded at this stage. Moreover, the filtered sequence of tokens was flattened into a set at this stage, eliminating duplicates.

As a last step, stemming was performed by applying the popular Porter's stemmer [10] to the set of tokens, thus completing the calculation of $T(r)$. See Figure 1 for an example of the results of the whole process.

3) Building the Affinity Model: The affinity model (i.e., the values for $\alpha(\cdot, \cdot)$) is obtained by a straightforward implementation of the definition. In particular, all pre-existing traces are considered in sequence. For each trace $e = (h, l) \in E$, the affinity value between all terms in $T(h)$ and all terms in $T(l)$ is incremented by 1 (starting at 0).

It may be observed that while a simple increment by 1 may seem simplistic, in practice what we obtain is a count of how many links exist in the given data set between requirements containing at least one occurrence of the given terms. Since the actual values of $\alpha(\cdot,\cdot)$ are only used for comparison purposes, i.e. in ranking the list of candidates, their value is immaterial, and any more complicated function that still preserves the ordering would have no net effect on the results. On the other hand, we would be very wary of any function that does not preserve the ordering, i.e. one that would rank a pair of terms appearing in fewer links as more tightly related than another pair that appear in more links. Thus, a principle of economy leads us to stick with the simpler formulation.

4) Identifying Candidate Links: Once $\alpha(\cdot,\cdot)$ is mined from the data, identifying candidates for linking is simply a matter of computing the cumulative affinity score for two requirements, the new one h^* (to be inserted in H), and the pre-existing ones $l \in L$.

Again, the process involves a straightforward implementation of the definition of $\rho_{h^*}(\cdot)$. This is obtained by first computing $T(h^*)$, then summing, for each l, the affinity measure of each pair of terms from $T(h^*)$ and $T(l)$.

In this case, too, we could perform a normalization step; however since values of $\rho_{h^*}(l)$ are only used to rank different ls, and only in the context of the same h^*, normalization is not needed.

In order to keep the sets of candidates manageable by the analyst, we used a parametric threshold τ: any pair (h^*, l) with $\rho_{h^*}(l) \leq \tau$ would not be included in the list of candidates. All pairs above the threshold would be included, with an understanding that pairs with higher ρ values would be presented in a more prominent role (e.g., at the top of a list) to the user for validation.

5) Measuring Results: In order to measure the quality of the candidate links, we resorted to the classical Information Retrieval measures of *precision* and *recall*, computed in a k-fold validation scheme. More precisely, we performed 22 measures; on each measure, a different requirement h^* was removed from H, and all the links (h^*, l), with $l \in L$, were removed from E.

The system was then trained on the remaining data, constituting a reduced dataset G^*, and the corresponding $\alpha_{G^*,T}(\cdot,\cdot)$ measure was computed for all remaining pairs. We then simulated the arrival of the "new" requirement h^*, computed the ranked list of suggested candidate links according to $\rho_{h^*}(\cdot)$, and measured precision and recall of that list compared to the correct set of links (those that were removed from E). The results were then averaged over the 22 instances.

In some practical cases, the quality of the topmost (i.e., highest ranking) candidates would be more relevant than those of the candidates further down in the list. This happens, for example, when the semantics of links is something like *duplicate-of*; simply finding that a user request (or bug report) is a duplicate of a pre-existing (and already accepted) feature request, is sufficient to decide whether the new requirement should be considered for inclusion or not. In other cases, though, the aim is to find *all* significant links, e.g. when the semantics is, say, *contributes-to-goal* or *blocks-implementation-of*.

In our experiments, we adhered to the second (and more taxing) model, i.e. the aim was to find all relevant links, not just one or a few of the "best" ones. Accordingly, we measured precision and recall comparing the full set of candidates (truncated based on a threshold τ) to the full set of manually established links for each requirement h^*. Moreover, to study how the choice of τ affects results, we considered different values for τ, evenly spaced in 20 steps between the minimum and maximum value of ρ returned for a given h^*.

Let us call $E^c_\tau(h^*)$ the list of candidates (h^*, l) (with $\rho_{h^*}(l) > \tau$) returned by our method, and $E^g(h^*)$ the set of manually-established links in the original E (mnemonic: superscript c for *candidate*, g for *gold set*).

For each value of τ, we considered a *true positive* any link that was suggested as a candidate by our technique and was also manually established in the original data, and a *true negative* any link that was neither suggested as a candidate by our technique, nor present in the manually-established links. Conversely, a link suggested by our technique but not manually established in the original data would be considered a *false positive*, and a link that was manually established, but not returned as a candidate, would be a *false negative*.

These four sets can be defined formally as follows:

$$\begin{aligned} TP &= E^c_\tau(h^*) \cap E^g(h^*) \\ TN &= \{(h^*, l) \mid l \in L \wedge (h^*, l) \notin E^c_\tau(h^*) \cup E^g(h^*)\} \\ FP &= \{(h^*, l) \mid l \in L \wedge (h^*, l) \in E^c_\tau(h^*) \setminus E^g(h^*)\} \\ FN &= \{(h^*, l) \mid l \in L \wedge (h^*, l) \in E^g(h^*) \setminus E^c_\tau(h^*)\} \end{aligned}$$

Precision and recall (at threshold τ) are thus defined as usual; moreover since we are essentially conducting a binary test, we can also measure *accuracy*, e.g. how many of our classifications (true positives or true negatives) are correct, as a proportion of all possible tests. Another useful measure that combines precision and recall by computing their harmonic mean is the *F-score*[5]; this can be useful as a synthetic indicator of overall effectiveness of a classification method.

Formally, these measures are given by

$$\begin{aligned} Prec &= \frac{\#TP}{\#TP + \#FP} \\ Rec &= \frac{\#TP}{\#TP + \#FN} \\ Acc &= \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN} \\ F &= 2 \cdot \frac{Prec \cdot Rec}{Prec + Rec} \end{aligned}$$

C. Results

We can finally present the results from our experiment on the CM1SUB dataset, providing a comparison with other

[5]The F-score can be tailored to assign different weight to either precision or recall. Here we used the balanced F-score, also called $F1$, that weights them equally.

TABLE I
RESULTS FROM THE EXPERIMENT ON THE CM1SUB DATASET, AT VARYING THRESHOLDS.

τ	$Prec$	Rec	Acc	F	$\#P$
0%	0.051	1.000	0.055	0.10	40.0
5%	0.078	1.000	0.396	0.14	26.3
10%	0.108	0.956	0.594	0.19	18.2
15%	0.174	0.911	0.775	0.29	10.7
20%	0.255	0.867	0.864	0.39	7.0
25%	0.358	0.867	0.914	0.51	5.0
30%	0.515	0.778	0.951	0.62	3.1
35%	0.611	0.733	0.963	0.67	2.5
40%	0.727	0.711	0.972	0.72	2.0
45%	0.853	0.644	0.976	0.73	1.6
50%	0.880	0.489	0.971	0.63	1.1
55%	0.882	0.333	0.964	0.48	0.8
60%	0.857	0.267	0.960	0.41	0.6
65%	1.000	0.178	0.958	0.30	0.4
70%	1.000	0.089	0.954	0.16	0.2
75%	1.000	0.067	0.952	0.13	0.1
80%	1.000	0.044	0.951	0.09	0.1
85%	1.000	0.044	0.951	0.09	0.1
90%	1.000	0.044	0.951	0.09	0.1
95%	1.000	0.022	0.950	0.04	0.0
100%	1.000	0.022	0.950	0.04	0.0

approaches, and on the full CM-1 dataset, which more faithfully represent a larger traceability context

1) CM1SUB and Comparison: Table I presents the values for our four measures at varying settings of τ (which, we recall, is expressed in percentiles over the range of ρ, and measured at 5% intervals).

The results are quite satisfying, with accuracy peaking at $\tau = 45\%$ on a value of 0.976 (that is: 97.6% of all classification decisions are correct). On a highly skewed data set as our CM1SUB, accuracy alone does not provide a full assessment: in fact, a trivial classifier that would simply return no candidates at all for any new incoming requirement, always answering "no" so to say, would *still* be right 96.1% of the times on CM1SUB (and a full 99.7% of the times on the entire CM-1 dataset!). However, our affinity-based technique also exhibits an F score of 0.73 at $\tau = 45\%$, with 85% precision and 64% recall, whereas the trivial classifier mentioned above would return no true positives, and hence would have 0% precision and recall.

Another positive feature that can be observed in Table I is that precision and recall are not overly sensitive to the exact value of τ, and thus the exact value of the parameter is not critical. Any value of τ between 40% and 60% would yield approximately 75%-85% precision, and any value between 20% and 40% would yield approximately 70%-85% recall. As usual, increasing precision values tend to decrease recall, and vice versa.

To give an idea of the cognitive burden placed on the analyst that has to browse the list of candidate links, in order to judge which ones are worthy of establishing, the last column in Table I reports the total number of positives returned ($\#P = \#TP + \#FP$), i.e. the size of the list of candidates presented on average for each new incoming requirement.

At $\tau = 0\%$, there is of course very little selection, and the technique identifies 40 of 53 low-level requirements as candidates for linking. Needless to say, presenting 75% of all possible link targets is not particularly useful: it might be easier to just consider all 53 low-level requirements, and at least rely on some guarantee of completeness.

But already at $\tau = 15\%$, only around 10 requirements are presented, which on average include over 90% of the real links. Roughly speaking, since on average each high-level requirement in CM1SUB is manually linked to just above 2 low-level requirements, and we have a recall of 0.91, this means that the analyst will have to distinguish among the 10 candidates between 2 real links, to be confirmed, and 8 false positives, to be discarded. Only in less than 10% of the cases the list of 10 candidates will not include a relevant link. For non-critical applications, this seems a good compromise between effort spent in confirming link candidates, and level of completeness of the recovered links: the technique saves 75% of the effort, at the cost of a risk of not identifying 10% of the links that would be established (at four times the cost) in the case of a complete analysis of all low-level requirements for each new incoming high-level requirement.

The exact way precision and recall are related can be visualised by plotting them on a precision-recall graph (see Figure 2), which we also use to compare the effectiveness of our affinity-based proposal to TF-IDF and Reinforcement Learning (RL).

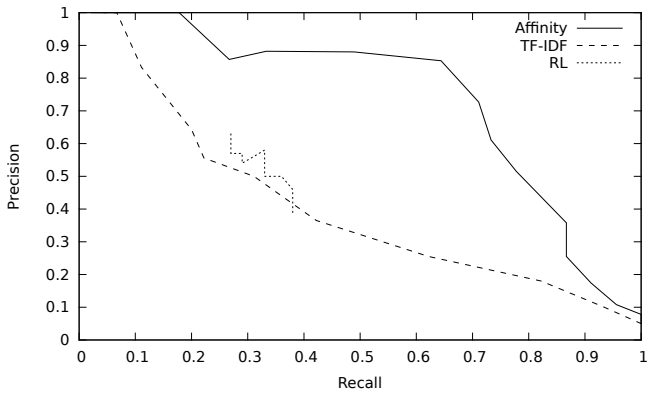

Fig. 2. Precision-Recall curve for Affinity, compared to TF-IDF and Reinforcement Learning, CM1SUB dataset.

The Affinity graph shows how precision is essentially stable for our technique for a large range of recall values (from 0.3 to 0.65), with a optimal spot around recall 65% and precison 0.85%. This corresponds to the optimal $\tau = 45\%$ that we had already identified in Table I.

More interestingly, it appears that Affinity substantially outperforms both TF-IDF and Reinforcement Learning[6].

While reinforcement learning is still a relatively novel approach in traceability research, and thus its effectiveness is

[6]For the latter two methods, numerical data was obtained from [11]; the resulting diagram is essentially the same as the one shown in Figure 6 in [7].

not yet well established, it is somewhat surprising that Affinity performs so much better than TF-IDF. However, we should consider that Affinity implicitly identifies and incorporates the relationship between occurrences of identical terms that is at the hearth of TF-IDF, while discarding those terms that present no evidence of being related in pre-established traces, despite occurring in multiple requirements, and adding those terms that are not identical, but frequently occur in linked pairs of requirements.

For example, Affinity identifies that high-level requirements containing the word BUFFER are often linked to low-level requirements containing BUFFER, just as TF-IDF would suggest. But in addition, it also identifies that high-level requirements containing COMMAND are even more frequently (actually, two times more frequently) linked to low-level requirements containing TASK, an association that TF-IDF would be unable to make. In a similar vein, Affinity is able to mine from pre-existing data that ERROR is often related to ENQUEU (the stem for *to enqueue*), which derives from the specific way errors are handled in the CM-1 system, and could not be derived by purely lexical means without looking at pre-established traces.

2) CM-1 and Scalability: The CM-1 dataset, with its larger size and lower link density, poses additional difficulties for an automated selection of candidate traces.

CM-1 has six times more requirements compared to CM1SUB. What is worse, since the number of potential links grows quadratically with the number of requirements, there are 44 times more potential links, and only 8 times more correct links. It is thus to be expected that the performance of any technique aiming to support traceability would drop considerably on larger datasets. Indeed, as Figure 3 shows, Affinity on CM-1 performs worse than on CM1SUB: but not dramatically worse. The optimal compromise here is slightly lower: the best value for F is found at $\tau = 50\%$ (pleasantly close to the optimum for CM1SUB, which was found at $\tau = 45\%$), with 77% precision and 63% recall.

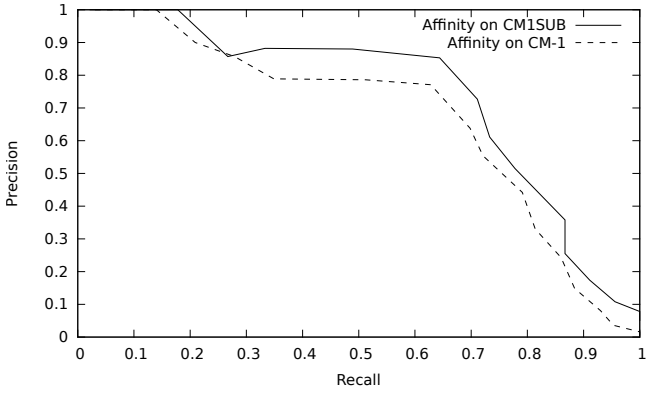

Fig. 3. Precision-Recall curve for Affinity on two different datasets: CM1SUB and CM-1.

Apparently these values are comparable to the 85% precision and 64% recall we had found for CM1SUB. But we have to remember that these numbers represent slightly lower percentages of a much larger base value,and that the amount of effort that can be put in selection does not scale linearly (or, in other words: the human analyst has a limited capacity for attention and time that can be spent in examining trace candidates). As an example, for CM1SUB we hypothesized that 90% recall could be a target (i.e., the risk of missing 10% of the links was acceptable), and that lead to 17% precision, meaning that on average, the analyst had to find the 2 true links amidst 10 candidates. For CM-1, the same level of recall would lead to a precision of 8%, half what we had for CM1SUB. In other terms, the analyst has to find 1 true link amidst 10 candidates (a task that could be deemed twice as difficult to get right).

In most contexts, scalability would thus be an issue: while the theoretical performances of the approach are still good, and the computational cost is negligible (see Table II), in practice the attention span of the analyst can be a limiting factor.

TABLE II
COMPUTATION TIMES FOR VARIOUS STAGES OF THE AFFINITY TECHNIQUE, ON A COMMODITY PC.

Operation	Time (secs)
Initializing OpenNLP components	2.7
Loading and stemming specifications	5.5
Building affinity model	2.1
Testing and computing measures	0.6

V. DISCUSSION AND APPLICABILITY

A. Generalizability and Tuning

The experimental results reported above support our main hypothesis, that Affinity can be an effective technique to identify candidate links in an ongoing requirements traceability process.

Of course, it would be quite daring to generalize from those experiments, only two, limited in size, and in the context of the same project, to all possible practical scenarios. The CM-1 dataset in fact exhibits certain peculiar properties (e.g.: highly stylized language, abundance of acronyms, rigorous naming conventions, high degree of textual polish of the requirements) that are all but universal in requirements documents. Hence, we do not claim generalizability beyond what the experiments have demonstrated.

Still, we see these results as a clear indication that relaxing the hypothesis that artefacts to link are homogeneous in language and vocabulary, can not only not be detrimental, but even improve the effectiveness of certain automatic techniques. Classical IR techniques developed for a scenario where queries were expressed in the same language as the documents to be retrieved, do not transition gracefully to software development scenarios, where links span vastly different categories of artefacts (e.g., from test cases to code). Even among requirements documents, authors and purpose can be so different that linked elements have often little lexical content in common.

The affinity concept can be applied in different ways, and extended to accommodate different needs. For example, links

representing different semantic relationships could be treated separately, and a special affinity score computed for each category. This view also accommodates bidirectional linking (e.g., traces between multiple versions of the same requirement in time, with forward traces meaning *evolved-into* and backward traces meaning *originates-from*).

Similarly, the exact way a requirement artefact is mapped to a set of terms is largely parametric. In our experiments, the mapping function T was based on standard linguistic processing. But if one were to apply affinity to program code, T could be implemented as a function to transform a unit of source code such as a function, class, or module, into a set of tokens (say: names of local variable, methods, etc.). The basic idea of mining previous knowledge encoded in pre-existing traces would still be applicable.

A number of different parameters and tools could be tuned, including the details of the linguistic processing steps (our T), the exact operational definition of a measure for affinity between terms (our α), or the way affinity between terms is combined to establish a ranking between candidate links (our ρ). In the present work, we have chosen the most basic options for each of these, in order to avoid complicating the presentation. However, in experimenting with alternative options (e.g.: using non-linear combinations for ρ instad of simple summation, or considering multiplicity in the set returned by T instead of a flat set of terms, or only considering affinity for values of α above a certain threshold, or adding a penalty for identical terms that were *not* linked, etc.), we have observed that the technique tends to be remarkably stable with respect to such changes. As we have already observed, the deciding factor is the relative ranking of candidate links, and the value of the threshold τ. The first is preserved by most reasonable alternatives, and the second was found experimentally to be not particularly critical in a large range of values. We thus anticipate that our results would survive small changes to the parameters or source data.

B. Other Application Scenarios

For expository purposes, we have framed our reference problem (Section II) in terms of an incoming new high-level requirement that has to be linked to pre-existing low-level requirements. This, however, is just one of the many potential application scenarios. In a more general view, affinity mining can be seen as a tool to be employed, among others, in the wider traceability maintenance picture.

In maintaining a set of traces, the various sets of artifacts (and the traces themselves) are assumed to evolve independently; the challenge is then to keep the traces up-to-date with the changes, spending as little human effort as possible. In this more complex setting, the knowledge extracted from established (and confirmed) traces needs not be discarded when the artifacts they used to link evolve. Once a confirmed link has established evidence that, say, "COMMAND" has high affinity to "TASK" (see Section IV-C1), this affinity score might well survive the removal of a requirement about one such command. Hence, in traceability maintainance the knowledge about affinity has to be maintained independently from its sources; the approach would be more focused towards glossary construction and maintenance, with the affinity glossary being an independent repository, rather than a derived property of the pre-existing traces themselves.

We have not explored the maintenance setting in this work, choosing instead to focus on the simpler case with a stream of incoming new requirements. The basic case also lends itself well to investigation of application scenarios where affinity data is used, interactively, to support the writing of new requirements. For example, we could envision a scenario where upon arrival of a new high-level requirement, after checking that it is not a duplicate or already implemented, a corresponding new low-level requirement has to be written. An editing tool could then subtly suggest (e.g., as part of a word-completion functionality) those low-level terms that are already known to have high affinity with terms appearing in the text of the new high-level requirement.

In general, affinity could be used as a substitute or additional measure of relatedness between artifacts in a variety of contexts, beside the main one we discussed in this paper. We plan to investigate its effectiveness in those different roles as part of our future work in this area.

C. Threats to Validity

We have designed our experiments with a view to eliminate or mitigate most threats to validity; yet there are a number of threats that we could not eliminate.

Among the internal validity threats that we tried to address we can cite selection bias (the data set was selected based on its availability and previous usage by other researchers, and not based on the peculiar needs of our technique) and time-related effects such as memory, maturation, and repeated testing. In our design, the only human intervention happened well before the technique was developed, namely when the reference set of traces was manually defined by a human expert, and there could be no influence by the researchers on the original authors of the data set.

External validity is more problematic, since we only have experimental results from a single project, in two different data sets. Further experiences, and possibly replication studies, are needed before any claim of generalizability can be made.

There is also a threat about content validity, in that our measures (precision, recall, etc.) may not be able to capture those features that are really relevant for the analyst. Other factors (e.g., ease of use of an implemented tool) may play an important role, in that an analyst may be unwilling to spend much time working with an awkward tool, or be ineffective at link selection because of some user interface problem. However, those measures that we have used have been proven historically to correlate well with user's effectiveness, and moreover provide a convenient way to compare different techniques (each of them, a different treatment) all else being equal. In this sense, our comparison to TF-IDF and Reinforcement Learning in Section IV provides solid evidence that Affinity is, among the three treatments we considered, the one providing the best performances on the same data.

For a single experiment, there is no issue of statistical validity; hence we have not endeavored to perform statistical significance tests on our results. In a future study, measuring performance differences on a significant number of different datasets, it may be needed to consider statistical validity before claiming generalizability.

VI. Related Work

Among the vast scientific literature related to requirements tracing, we consider first those works that purpose to generate a set of candidate links for unlinked requirements.

Many early works advocated a direct application of classical Information Retrieval techniques, such as Vector Space Model (VSM) weighted by text frequency - inverse document frequency (TF-IDF) (among others, [12], [13], [2]). These techniques assumed to start with a clean slate, that is, from artifacts that were not linked at all; the problem was then to generate an initial set of links for the whole set of documents. Since no information other than the documents themselves was available, only lexical similarity was used in generating candidate links, and their applicability was restricted to artifacts that used the same vocabulary. In contrast, our proposal mines pre-existing links for context-dependent information, that is then used to relax the restriction that all artifacts should use the same vocabulary.

The limitations stemming from sole reliance on lexical content was quite rapidly felt. A successive stream of research tried to use Latent Semantic Analysis (in a static context as in [14] or in an evolutionary perspective as in [15]), or explicit dictionaries, thesauri or glossaries prepared for traceability purposes by a domain expert; the value of such a document was highlighted by several researchers as an important supplement to VSM-based techinques, e.g. [13]. In a sense, our approach is closer to this second strain, except that the vocabulary (relating the language used in one artifact, to a potentially different one used in another artifact) is inferred from pre-existing traces, rather than prepared by a domain expert. A similar idea has been recently advocated in [16], where an automated thesuarus builder is used in support to link generation.

Another idea for improving over the basic VSM model that has been proposed concerns using the location of terms, either to define a sort of *local neighbourhood* [17], or to assume n-grams as the unit of lexical reference, and sliding windows as the unit of relevance for linking purposes [18]. We have not addressed location of terms in our proposal: in fact, we explicitly discard any positional information in the computation of T. While positional information could potentially be useful, not just in terms of location, but also in terms of syntactic roles of the various terms, we have not considered the issue in this work. On the other hand, n-grams could be easily (and maybe more effectively) modeled by using even a basic nominal grammar in T, for example to assemble noun phrases, instead of considering each term on its own. We intend to investigate the issue in future studies.

The limitation of purely lexical approaches have been identified by several researchers. As a consequence, in more recent times a systematic exploration of the applicability of Machine Learning techniques to the requirements traceability problem has been initiated.

The various ML approaches are so diverse that we cannot conduct a full survey here; we will simply mention a few of the most relevant ones. Already in 2004, ML was advocated as an alternative to lexical based techniques [19]. More recently, various ML techniques such as swarm intelligence [20] and reinforcement learning [7] have been applied; these are the research efforts that are closest in spirit to our own, and the ones we used for a comparison in Section IV. Reinforcement learning in particular was applied first to recover traces from normative codes (i.e.: standards, laws, regulations) to requirements in [21], where the technique was found to be particularly effective for the specific context. This might be in part due to the stable nature of one of the two sets of artifacts (the regulations), which are supposed to be the same over long periods of time and across different projects, thus providing good opportunities for reinforcement learning. In contrast, our affinity based approach is completely symmetric, and it is not expected that variability will occur on one class of artifacts only.

While all of the ML approaches assume either a given "golden" dataset to train a classifier of sort, or a given "oracle" to serve as a utility function (e.g., in approaches based on genetic algorithms, such as [22]), not all of them consider the ongoing evolution of multiple sets of artifacts as a source for traceability information, as our approach does. Traceability in an evolving context is thus the second major area of research which we briefly survey.

An adaptation of static IR techniques to the evolving context is presented in [23] (and more extensively in [24]), where different strategies are introduced and evaluated on a set of cases. A more thorough treatment, focusing on modeling the problems involved in trace evolution in addition to proposing a technical solution, is given in [25] in the context of traceability between requirements and design elements. Interestingly, [25] explicitly disclaims applicability to the initial construction of a set of traces, suggesting instead to use IR and data mining techinques for that. In contrast, we propose using IR and data mining techniques as part of ongoing evolution, and assume that the *bootstrap* is given by manually-established links (or, by traditional lexically-based IR techniques).

A fuller view of traceability activities as parts of a larger process is provided in [26]. In fact, our affinity-based approach could be configured as a part of a more complex evolution workflow, and possibly integrated with other techniques. Indeed, a recent trend is emerging towards composing traceability environments out of distinct components, and optimizing the various parameters for the task at hand. In its most sophisticate incarnation, customization of tools and parameters is obtained by applying genetic algorithms to a configuration, and letting the population of settings evolve until a reasonable optimum is obtained [27]. Affinity mining could then be one of the techniques and measures used in a rich toolbox of complementary approaches.

VII. Conclusions

We have presented a technique to exploit the information contained in previously defined traces among a set of software requirements, in order to facilitate the creation and ongoing maintenance of the traces as the requirements evolve, new requirements are defined, or old ones are edited or deleted.

The results from our experiments show that the proposed technique, based on a measure of *affinity* between pair of terms that embodies the likelihood that the terms will appear in requirements that are linked with each other, is substantially more effective than TF-IDF, in particular in cases where the artifacts to be linked employ different languages, jargon, or domain vocabularies. This is a frequently occurring scenario in many software development processes, e.g. when there is a need to link users' requests for new features (expressed in the users' jargon) to technical requirements (as written by analysts and developers) that implement them.

We also compared the effectiveness of our approach with a ML technique based on Reinforcement Learning, again finding substantial improvement.

While no claim of generalizability can be made at this stage, we believe that further studies will draw a large scope for affinity-based techniques, in specific contexts.

This work has revolved around proving the viability of the basic technique. As part of future work we intend to investigate the improvements that can be obtained by more refined linguistic processing, as well as testing the technique on a wider variety of scenarios.

Our validation has been entirely technical so far. This is both an advantage (in that the experiments are fully replicable, and there is no human variability involved), and a disadvantage (in that we have not considered human factors that might impact the practical usability of the technique). Hence, an avenue for further improvement is certainly deploying our technique in the context of a live industrial project, in order to gather first-hand users' feedback.

References

[1] M. S. Feather, T. Menzies, and J. R. Connelly, "Relating practitioner needs to research activities," in *Proc. of the 11th IEEE Int. Requirements Engineering Conf.* Los Alamitos, CA: IEEE CS Press, Sep. 2003, p. 352.

[2] J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell, "Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering," in *Proc. of the 12th IEEE International Requirements Engineering Conference*, Kyoto, Japan, Sep. 2004.

[3] ——, "A linguistic engineering approach to large-scale requirements management," *IEEE Software*, vol. 22, no. 1, Jan./Feb. 2005.

[4] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[5] V. Gervasi and D. Zowghi, "Mining requirements links," in *Proceedings of the 17th International Conference on Requirements Engineering: Foundation for Software Quality*. Essen, Germany: Springer-Verlag, 2011.

[6] A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[7] H. Sultanov and J. H. Hayes, "Application of reinforcement learning to requirements engineering: requirements tracing," in *Proc. of the 21st IEEE Int. Requirements Engineering Conf.*, Jul. 2013, pp. 52–61.

[8] Apache Foundation, "Apache OpenNLP," https://opennlp.apache.org/, Oct. 2013.

[9] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Comput. Linguist.*, vol. 19, no. 2, pp. 313–330, Jun. 1993.

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

[11] H. Sultanov, "Application of swarm and reinforcement learning techniques to requirements tracing," Ph.D. dissertation, University of Kentucky, 2013.

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

[13] J. Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," in *Proc. of the 11th IEEE Int. Requirements Engineering Conf.*, Sep. 2003, pp. 138–147.

[14] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using Latent Semantic Indexing," in *Proc. of the 25th Int. Conf. on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 125–135.

[15] H. yi Jiang, T. Nguyen, I.-X. Chen, H. Jaygarl, and C. Chang, "Incremental latent semantic indexing for automatic traceability link evolution management," in *Proc. of the 23rd IEEE/ACM Int. Conf. on Automated Software Engineering*, Sep. 2008, pp. 59–68.

[16] S. Pandanaboyana, S. Sridharan, J. Yannelli, and J. Hayes, "Requirements tracing on target (retro) enhanced with an automated thesaurus builder: An empirical study," in *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*, May 2013, pp. 61–67.

[17] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving automated requirements trace retrieval: a study of term-based enhancement methods," *Empirical Software Engineering*, vol. 15, no. 2, pp. 119–146, 2010.

[18] N. Niu and S. Easterbrook, "Extracting and modeling product line functional requirements," in *Proc. of the 16th IEEE Int. Requirements Engineering Conf.*, Sep. 2008, pp. 155–164.

[19] G. Spanoudakis, A. Zisman, E. Prez-miana, and P. Krause, "Rule-based generation of requirements traceability relations," *Journal of Systems and Software*, vol. 72, pp. 105–127, 2004.

[20] H. Sultanov, J. H. Hayes, and W.-K. Kong, "Application of swarm techniques to requirements tracing," *Requir. Eng.*, vol. 16, no. 3, pp. 209–226, Sep. 2011.

[21] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proc. of the 32nd ACM/IEEE Int. Conf. on Software Engineering*, vol. 1, New York, NY, USA, 2010, pp. 155–164.

[22] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *Proc. of the 2013 Int. Conf. on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 522–531. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486857

[23] S. Winkler, "Trace retrieval for evolving artifacts," in *Proc. of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, ser. TEFSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 49–56. [Online]. Available: http://dx.doi.org/10.1109/TEFSE.2009.5069583

[24] ——, *EvoTrace: Evolution-Aware Trace Retrieval*. Hamburg, Germany: Verlag Dr. Kovac, 2011.

[25] P. Mäder and O. Gotel, "Towards automated traceability maintenance," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2205 – 2227, 2012, automated Software Evolution. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121211002779

[26] J. Cleland-Huang, P. Mader, M. Mirakhorli, and S. Amornborvornwong, "Breaking the big-bang practice of traceability: Pushing timely trace recommendations to project stakeholders," in *Proc. of the 20th IEEE Int. Requirements Engineering Conf. (RE)*, Sept 2012, pp. 231–240.

[27] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang, "Improving trace accuracy through data-driven configuration and composition of tracing features," in *Proc. of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 378–388. [Online]. Available: http://doi.acm.org/10.1145/2491411.2491432