Improving the Effectiveness of Traceability Link Recovery using Hierarchical Bayesian Networks

Kevin Moran
William & Mary
Williamsburg, VA, USA
kpmoran@cs.wm.edu

David N. Palacio
William & Mary
Williamsburg, VA, USA
dnaderp@cs.wm.edu

Carlos Bernal-Cárdenas
William & Mary
Williamsburg, VA, USA
cebernal@cs.wm.edu

Daniel McCrystal
William & Mary
Williamsburg, VA, USA
dmc@cs.wm.edu

Denys Poshyvanyk
William & Mary
Williamsburg, VA, USA
denys@cs.wm.edu

Chris Shenefiel
Cisco Advanced Security Research Group
Morrisville, NC, USA
cshenefi@cisco.com

Jeff Johnson
Cisco Security and Trust Engineering
Morrisville, NC, USA
johnsonj@cisco.com

ABSTRACT

Traceability is a fundamental component of the modern software development process that helps to ensure properly functioning, secure programs. Due to the high cost of manually establishing trace links, researchers have developed automated approaches that draw relationships between pairs of textual software artifacts using similarity measures. However, the effectiveness of such techniques are often limited as they only utilize a single measure of artifact similarity and cannot simultaneously model (implicit and explicit) relationships across groups of diverse development artifacts.

In this paper, we illustrate how these limitations can be overcome through the use of a tailored *probabilistic model*. To this end, we design and implement a Hierarchi**C**al Pr**O**babilistic **M**odel for Softwar**E** **T**raceability (Comet) that is able to infer candidate trace links. Comet is capable of modeling relationships between artifacts by combining the complementary observational prowess of multiple measures of textual similarity. Additionally, our model can holistically incorporate information from a diverse set of sources, including developer feedback and transitive (often implicit) relationships among groups of software artifacts, to improve inference accuracy. We conduct a comprehensive empirical evaluation of Comet that illustrates an improvement over a set of *optimally configured* baselines of ≈14% in the best case and ≈5% across all subjects in terms of average precision. The comparative effectiveness of Comet *in practice*, where optimal configuration is typically not possible, is likely to be higher. Finally, we illustrate Comet's potential for practical applicability in a survey with developers from Cisco Systems who used a prototype Comet Jenkins plugin.

CCS CONCEPTS

• **Software and its engineering** → **Software development process management**; **Software development methods**.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380418

KEYWORDS

Software Traceability, Probabilistic Modeling, Information Retrieval

ACM Reference Format:
Kevin Moran, David N. Palacio, Carlos Bernal-Cárdenas, Daniel McCrystal, Denys Poshyvanyk, Chris Shenefiel, and Jeff Johnson. 2020. Improving the Effectiveness of Traceability Link Recovery using Hierarchical Bayesian Networks. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380418

1 INTRODUCTION & MOTIVATION

The importance of traceability in modern software systems cannot be overstated. Traceability links that connect "high-level" artifacts such as requirements and use cases to "low-level" artifacts written in code help to facilitate crucial components of the software development and maintenance cycle. For instance, linking requirements to code provides visibility into a system by enumerating what has been implemented, whereas linking requirements to test cases helps to provide an indication that the software is functioning as expected. Additionally, the establishment of trace links aids in facilitating a broad set of developer activities including code comprehension, change impact analysis, and compliance validation [14]. In certain software domains, such as those involving safety critical systems, traceability is necessarily *mandated* by regulatory bodies in order to properly demonstrate the safe functioning of a system [13, 36, 44, 52]. Furthermore, traceability is increasingly used to help ensure the *security* of a given system [46]. For example, our industrial partners at Cisco Systems, Inc. require that security-critical requirements are verified by a dedicated group of analysts to avoid software threats and ensure best practices.

Unfortunately, despite its importance, software traceability is, by its nature, an inherently difficult and error prone task [15, 36, 37]. This difficulty primarily stems from the need to bridge a logical abstraction gap that exists between different software artifacts, such as requirements written in natural language and code written in "lower-level" programming languages. Given the effort required to establish and evolve effective trace links, it is often too costly to manually establish them outside of regulated domains, and in practice the quality of mandated links are often questionable [16].

The inherent difficulty in establishing trace links has lead to research on automated techniques for modeling, establishing, and evolving trace links that primarily rely upon information retrieval (IR) [5, 7, 19–21, 25, 30, 32, 35, 37, 39–41] and machine learning (ML) [7, 22, 27, 38, 53] techniques which retrieve or predict trace links based upon textual similarity metrics. However, in large part, current automated approaches for traceability often trade precision for completeness and vice versa, making them difficult to adopt in practice. We observe three major shortcomings of current automated techniques that contribute to their limited effectiveness:

1) Limited Measures of Artifact Similarity: Existing techniques for trace link recovery tend to use a single textual similarity metric to draw relationships between artifacts. This is problematic for several reasons. Perhaps most importantly, it is often difficult or impossible to determine how well a technique that uses a given similarity measure will function on artifacts from a new project without any pre-existing trace links. This so-called "cold-start" problem is due to the fact that existing IR/ML techniques for measuring textual similarity often need to be calibrated on a subset of "ground-truth" artifact pairs with pre-existing links. This makes performance of these techniques difficult to predict when applied to new datasets. Furthermore, in practice industrial projects often lack pre-existing trace links, as confirmed by our partners at Cisco. Thus, while certain techniques have been shown to perform well on research benchmarks, the efficacy of a similarity measure is often tightly coupled to the underlying semantics of software artifact text [9, 26, 34], and to the configuration of the corresponding IR/ML technique [48].

Using only a single textual similarity metric also needlessly restricts the predictive power of a traceability technique. Past work has illustrated the orthogonality of different similarity measures [48], suggesting that combining *several* different measures could lead to more accurate and robust techniques that function *consistently* well when applied to new projects without pre-existing links.

2) Inability to Effectively Capture Developer Feedback: The rapid pace of modern agile development practices often results in crucial knowledge about a software system being siloed within the expertise of individual developers. Thus, one unstructured development artifact that has gone underutilized by past techniques is *developer feedback*. When an automated traceability model is uncertain about particular trace link pairs, developers can provide critical feedback to help improve trace link inference.

3) Limited View of Interactions Between Artifacts Existing automated traceability approaches are typically tailored to establish relationships between pairs of specific types of artifacts (e.g., user stories and class files). However, information pertaining to the relationship of one type of artifact pair may be contained within other related artifacts. For example, if a piece of source code is linked to a given requirement through textual similarities, and this source code is also intrinsically linked to test code via method calls, then it is likely the requirement is also linked to the test code. However, in this situation, it may be difficult for a textual similarity metric to link the requirements and test code, due to limited test documentation, for example. Thus, in this way, established relationships between certain artifacts may influence the probability of other artifact relationships. In this paper we refer to these phenomena as *transitive links*. Existing techniques generally cannot model such interactions between artifacts.

The limitations discussed above stem from both technical and practical limitations of existing traceability techniques, and surfaced during our development of an automated traceability approach in close collaboration with Cisco Systems. In this paper we introduce a novel technique that overcomes these limitations by constructing a Hierarchical Bayesian Network for inferring a set of candidate trace links. The model that underlies our approach is capable of deriving the probability that a trace link exists between two given artifacts by combining information from multiple measures of textual similarity, while simultaneously modeling transitive relationships and accounting for developer expertise. We implemented our approach, called Comet (Hierarchi**C**al Pr**O**babilistic **M**odel for Softwar**E** **T**raceability), in both an extensible Python library and as a plugin for the popular Jenkins CI/CD system. In an extensive set of empirical experiments we illustrate that Comet is able to outperform the median precision of *optimally configured* baseline techniques by ≈5% across subjects and ≈14% in the best case. Given that optimal configuration is typically not possible in practice, this illustrates that given a project with no pre-existing trace links, Comet is likely to perform significantly better than most existing IR/ML techniques. Additionally, we show Comet's potential for integration into the workflows of development teams at Cisco. In summary, this paper's contributions are as follows:

- The derivation of a Hierarchical Bayesian Network (HBN) for inferring a candidate set of trace links;
- An implementation of this model, called Comet, as both an extensible Python library and a Jenkins plugin that has been deployed for testing with our industrial partners at Cisco;
- An extensive evaluation of Comet on both open source projects and two industrial datasets from one industrial software project, including feedback from professional developers at a major telecommunication software company;
- An open source, commercial-grade traceability benchmark, developed in coordination with our industrial partner, for the benefit of the research community;
- An online appendix, including our open source implementation of Comet and evaluation data for reproducibility [2].

2 RELATED WORK

We focus our discussion of related work on prior techniques that have, in limited contexts, (i) considered novel or hybird textual similarity measures, (ii) modeled the effects of multiple types of artifacts, or (iii) incorporated developer expertise. We then conclude with a statement distilling Comet's novelty.

Novel/Hybird Textual Similarity Measures: Guo et al. [26] proposed an approach for candidate trace link prediction that uses a semantically enhanced similarity measure based on Deep Learning (DL) techniques. However, unlike Comet, this technique requires pre-existing trace links in order to train the DL classifier. In contrast, Comet does not require known links for the projects it is applied to, but rather requires a project to serve as a tuning set. We show that Comet performs well when tuned and tested on different datasets, outperforming Guo et al.'s DL-based approach when it is trained in a similar manner. Gethers et al. [25], implemented an approach that is capable of combining information from canonical IR techniques (i.e., VSM, Jensen-Shannon) with Topic Modeling

techniques. However, their approach can only combine two IR/ML techniques, whereas COMET can combine and leverage the observations from several IR/ML techniques, and combine this with other information such as expert feedback and transitive links.

Modeling of Multiple Artifacts: Rath et al. [50] recently explored linking nontraditional information including issues and commits, and Cleland-Huang et al. [13] have investigated linking regulatory codes to product level requirements. COMET's model has the potential to improve trace link recovery in these scenarios both through its more robust modeling of textual similarity, and through incorporation of transitive link information. Furtado et al. [24], explored traceability in the context of agile development, and Nishikawa et al. [47] first explored the use of transitive links in a deterministic traceability model. Additionally, Kuang et al. used the closeness of code dependencies, to help improve IR-based traceability recovery [32]. However, none of these approaches is capable of incorporating transitive links while also considering combined textual similarity metrics and developer feedback.

Incorporation of Developer Expertise: De Lucia et al. [18] and Hayes et al. [28] analyzed approaches that use relevance feedback to improve trace link recovery. However, these approaches are either tied to a particular type of model (such as TF-IDF [18]), or require knowledge of the underlying model to function optimally. In contrast, COMET implements a lightweight, likert-based feedback collection mechanism that we illustrate can improve link accuracy even when only a small amount of feedback is collected.

Summary of Advancement over Prior Work: COMET's features facilitate its application to projects without any pre-existing trace links, and as our evaluation illustrates, allow it to perform consistently well across datasets. COMET is able to combine information from transitive links with both robust textual similarity measures and lightweight developer feedback for improved accuracy. While some aspects of COMET's approach have been considered in limited contexts in prior work – such as developer feedback [18, 28] and restricted combinations of IR/ML techniques [25] – there has never been a framework capable of combining all these aspects in a holistic approach. Our evaluation illustrates that COMET's holistic HBN is able to outperform baseline techniques on average.

3 BACKGROUND

3.1 Problem Definition

Our goal is to design a model that captures meaningful information regarding logical relationships between software artifacts, and then use this model to infer a set of candidate trace links. More specifically, given a set of source artifacts S (e.g., requirements, use cases) such that $S = \{S_1, S_2, \ldots S_n\}$ and a set of target artifacts T (e.g., source code files, test cases) such that $T = \{T_1, T_2, \ldots T_n\}$, we aim to infer whether a trace link L exists between all possible pairs of artifacts in S and T such that $L = \{(s,t) | s \in S, t \in T, s \leftrightarrow t\}$ where each pair of artifacts s and t are said to be logical trace links.

3.2 Defining a Probabilistic View of Traceability

3.2.1 ***The Probabilistic Nature of Software Traceability.*** The process of building software is not inherently deterministic, and is instead the result of decisions made by engineers over prolonged periods of time that may be hard to predict. Developer decisions related to nearly every observable phenomenon in modern software development are influenced by a combination of multiple factors. For instance, the presence of a functional bug may be influenced by the quality of related requirements, implementation constraints imposed by a given programming language [51], or the change-proneness of underlying APIs [33]. Given that such factors are often hard to predict, there is a clear sense of randomness inherent to the software development process. Similarly, the existence of trace links among software artifacts is also likely to be influenced by several different effectively *random* factors.

These factors could include textual similarities between artifacts, programmatic associations between pieces of code, or even abstract notions of similarity held by expert developers. For example, the textual quality of requirements or identifiers in code are typically a function of several factors such as the fluency and writing style of the author and the familiarity of key phrases chosen for identifiers [17]. This may lead to variable names that may be perfectly clear to one engineer being indecipherable to another. *From this view point, the existence of trace links between software artifacts can be thought of as an inherently a probabilistic phenomenon.*

3.2.2 ***Traceability as a Bayesian Inference Problem.*** Hence, in order to effectively model trace links among software artifacts, it is necessary to model a collection of random factors that influence *the probability that a trace link exists*. Thus, the process of deriving trace links can be modeled as a *bayesian inference* problem, wherein a probability distribution representing the existence of a trace link between two artifacts can be inferred. As we illustrate, by modeling the trace link recovery problem in a probabilistic manner, we are able to construct an an automated approach that largely overcomes the typical drawbacks discussed in Sec. 1. To understand this context, let us consider the general definition of Bayes' Theorem:

$$P(H|O) = \frac{P(O|H) \cdot P(H)}{P(O)} \tag{1}$$

where H is a hypothesis regarding some phenomenon, O is a set of observations that provide some information about the hypothesis, and where our goal is to infer or estimate the probability that our hypothesis is true $P(H|O)$, which is called the *posterior probability distribution*, or more simply the *posterior*. However, a given hypothesis is rarely made in a vacuum, and one typically holds some *prior belief* as to the probability that is being inferred. This prior belief is modeled as a probability distribution $P(H)$, which we will simply refer to as the *prior*, and can be influenced by a number of factors. In order for the posterior to be inferred from a set of observations, these must be modeled in a probabilistic manner. This is the purpose of the *likelihood* $P(O|H)$, which is a probability distribution that is derived purely from observed data. Thus, in Bayesian inference initial beliefs are represented as the prior, observations are modeled as the likelihood and the final beliefs are represented by the posterior. This posterior probability distribution can be *inferred* via one of several existing statistical inference techniques. In framing the problem of inferring trace links as a Bayesian problem, we consider our hypothesis to be whether a given trace link exists between a single source artifact S_x and a single target artifact T_y. Given the nature of trace links (e.g., a link either does or does not exist) we can model our prior as a distribution on the interval $[0, 1]$, where 1 indicates the presence of a link and 0 indicates an absence.

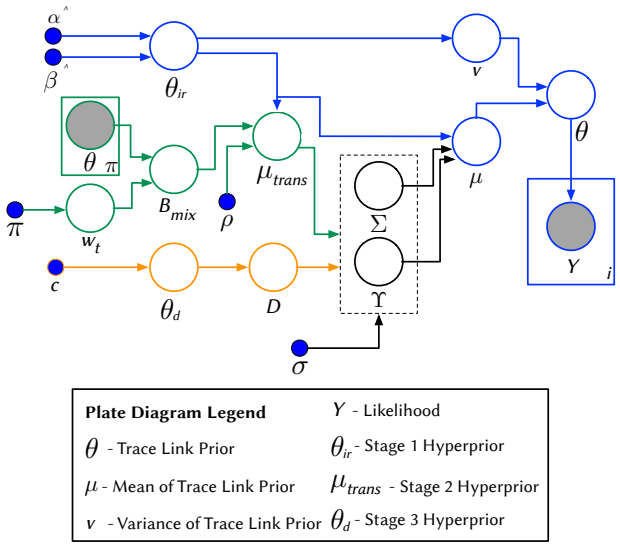


Figure 1: Plate Diagram of COMET's HBN

3.2.3 ***A Hierarchical Bayesian Network for Traceability.*** In the context of this paper, we will consider our *likelihood* (observations) to be the binary indication that a link exists according to a set of textual similarity measures and an empirically derived threshold value. However, given that we aim to model multiple factors that might influence traceability, our model employs multiple *priors*, called *hyperpriors*, forming a Hierarchical Bayesian Network (HBN). In this work, we consider three priors corresponding to the three factors we wish to model: (i) a normalized set of diverse textual similarity measures, (ii) developer expertise, and (iii) transitive trace links. We assign each of these priors an initial probability distribution, which is then influenced and estimated based upon observable data (e.g. a developer confirming or denying a trace link). Once this network is established, the *posterior* can be computed via one of several estimation techniques. By modeling these three information sources, our technique is able to largely overcome the limitations enumerated in Sec. 1. HBNs are also highly extensible via adjustments to the prior(s). Thus our defined model be capable of adapting to advancements in textual similarity measures or considering new development artifacts from future development workflows.

4 INFERRING TRACE LINKS WITH A HIERARCHICAL BAYESIAN NETWORK

In this section, we provide a formal description of COMET's probabilistic model introduced at a high level in Sec. 3.2. To help aid in the comprehension of COMET's underlying model, we provide a graphical representation using plate notation [43] in Fig. 1, which we use to guide our introduction and discussion. The model in Fig. 1 is computed on a *per link* basis, that is between all potential links between a set of source (S) and target artifacts (T). In this section we will use S_x and T_y to refer to a single source and target artifact of interest respectively. COMET's probabilistic model is formally structured as an HBN, centered upon a *trace link prior* θ which represents the model's prior belief about the probability that S_x and T_y are linked. Our model is hierarchical, as the trace link prior is influenced by a number of *hyperpriors*, which are constructed in accordance with a set of *hyper-parameters* that are either derived empirically, or fixed. In Fig. 1, hyperpriors are represented as empty nodes, and hyper-parameters are represented as shaded blue nodes. In general, empty nodes represent latent, or hidden, variables whereas shaded nodes represent variables that are known or empirically observed quantities. The rectangles, or "plates" in the diagram are used to group together variables that repeat.

To make our model easier to comprehend, we have broken it down into four major configurations, which we call *stages*, indicated by different colors in Fig. 1. The first stage of our model (shown in blue at top) unifies the collective knowledge of textual similarity metrics computed by IR/ML techniques. The second stage (shown in orange at the bottom) reconciles expert feedback to improve the accuracy of inferred trace links. The third stage (shown in green in the middle) accounts for transitive relationships among development artifacts, and the fourth stage combines each of the underlying stages. It should be noted that the first stage of our model can be taken as the "base case" upon which the other complexities build and is always required to infer the existence of a trace link. The order of calculation starts with the first stage and proceeds sequentially. The design and parameterization of our model presented in this section is not arbitrary, but instead based on the well-founded theory of *conjugate priors* [49] which aids in defining appropriate distributions and hyperparameters for a given prior. We center the description of our model first upon the likelihood estimation and then around the estimation of the prior probability distribution as defined by the four stages. After defining the hyperpriors for each of the four stages we briefly discuss the inference techniques we employ to estimate the posterior probability distribution of our model and thus the probability of whether a given link exists. While this section provides an overview of our model, we discuss its instantiation (including utilized IR/ML techniques) in Sec. 5.1.

4.1 Estimating the Likelihood

The likelihood function in our HBN models observed data so that these observations can be reconciled with our estimated prior probability distribution to infer a posterior probability. The likelihood is shown as the observed variable Y (Fig. 1). The variable i represents the number of observations made. In the context of traceability, we express the likelihood as a discrete Bernoulli distribution, as two artifacts can either be "linked" or "not-linked":

$$Y = p(l_i|\theta_i) = Bern(l_i|\theta_i) \tag{2}$$

where l_i is an observable data point 0, 1 for i number of observations. We define an observation as a function of the textual similarity score generated by an IR technique between S_x and T_y and some threshold k_i where any similarity above the threshold is considered an observed link, and any similarity value below this threshold is considered a non-link. The number of IR techniques or configurations utilized corresponds to the number of observations i. Ideally, to capture the most accurate trace link observations from IR techniques, the threshold k_i should be chosen to maximize the chance that each IR technique correctly establishes whether two artifacts are linked. In other words, k_i should be chosen for each IR technique such that the precision and recall of the technique is maximal across the entire set of considered source and target artifacts S and T. However, this information is not available a-priori without the consultation

of a ground truth set of trace links. As we illustrate in Sec. 5 this threshold can often be estimated with surprising accuracy by analyzing the distribution of similarity values an IR technique produces for a given set of artifacts.

4.2 Stage 1 - Unifying Textual Similarities between Development Artifacts

The first "base" stage of our model informs the trace link prior, represented as a probability distribution $p(\theta)$, according to the textual similarity measurements of a set of IR techniques. However, converse to the likelihood estimation, the actual textual similarity values of IR techniques are directly used to estimate a Beta distribution (the conjugate prior of the likelihood's Bernoulli distribution). This Beta distribution is represented as follows:

$$\theta \sim B(\mu, \nu) \tag{3}$$

where μ and ν are parameters of the Beta distribution representing its mean and variance. This prior, and its two parameters are illustrated in the right-most part of the blue segment (Fig. 1). To inform this Beta distribution, the textual similarity of values of a given number i of IR/ML techniques are normalized according to a sigmoid function centered upon the median of the distribution of similarity values across all S and T in a given dataset. Then a logistic regression is performed upon the normalized similarity values to infer the values $\hat{\alpha}$ and $\hat{\beta}$ which define a hyperprior beta distribution θ_{IR}. This hyperprior with parameters are shown on the left of the blue segment (Fig. 1). The mean and the variance of this hyperprior distribution then inform μ and ν of the base prior θ:

$$\nu = Var[\theta_{IR}] \quad \mu = Mean[\theta_{IR}] \tag{4}$$

Thus, by considering the textual similarity values of a set of IR techniques, our model can effectively reconcile the collective knowledge to ultimately make an informed prediction.

4.3 Stage 2 - Incorporating Developer Feedback

The second stage of our model is capable of leveraging human feedback by influencing the prior distribution introduced in the first stage of our model. To model expert feedback, we estimate hyperpriors D and θ_d, shown in orange in Fig. 1. To perform this estimation, our model accepts from a developer or analyst, their confidence that a given link exists as a value between $[0, 1]$. In Section 4.7 we illustrate how such feedback can be collected from developers in a lightweight manner. This confidence value serves as a parameter for estimating the distribution of the first hyperprior:

$$\theta_d \sim B(\mu_d = c, sd = 0.01) \tag{5}$$

where θ_d is a Beta distribution parameterized by its mean μ_d set to the confidence value provided by a developer, and standard deviation sd which we set to 0.01 signaling a low variance in the derived Beta distribution. This distribution then parameterizes the second hyperprior D, modeled as a Bernoulli distribution. Now that we have derived a distribution representing developer feedback, we must define how this distribution affects the prior probability of the first stage of our model. To do this, we define reward and penalty functions Υ and Σ that are influenced by σ which represents a specified *belief factor* between $[0, 1]$ that controls the extent to which the feedback influences the trace link prior. The reward function is defined as $\Upsilon = \sigma * D$, whereas the penalty function is defined as $\Sigma = \sigma * (D - 1)$. These factors impact the first stage prior Beta distribution by affecting its mean μ:

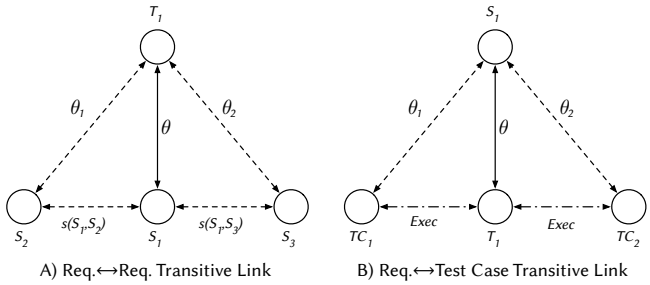


Figure 2: Illustration of Transitive Links

$$\mu \sim N(\mu_n = \Sigma + \Upsilon, sd = 0.01) \tag{6}$$

where the mean of the first stage prior is represented as a normal distribution parameterized by μ_n set to the sum of Σ and Υ, and a standard deviation set to 0.01. Thus, in this manner, expert feedback is utilized to influence the prior distribution that a given trace link exists. The structure of the Stage 2 hyperpriors allows Comet's HBN to effectively consider feedback from multiple developers.

4.4 Stage 3 - Leveraging Transitive Links

As discussed earlier, the probability that a trace link exists between a source and target artifact can be influenced by *transitive* relationships among varying software development artifacts. The third stage of Comet's HBN is able to utilize these transitive links to improve the accuracy of its inferred trace link. However, before we describe how our model reconciles this information in a probabilistic manner, it is first important to understand the phenomena of transitive links. At a high level, a transitive link is an inherent relationship between two software artifacts (A_1,A_2) that may influence the existence of a trace link between either A_1 and any other artifact or A_2 and any other artifact. Comet is currently capable of leveraging two types of transitive links, one based on textual-similarities (req. ↔ req.) and one based on dynamic execution information (req. ↔ test case). However, Comet could also be extended to model transitive relationships between other types of artifacts, such as commit messages or issues. Fig. 2 provides an illustration of both transitive link types, which we detail below. Note that for execution traces, a relationship is considered *strong* between a test method and source method if the test executes the method, and *weak* otherwise. For req↔req relationships, it is *strong* if the textual similarity is above the threshold τ, and *weak* if it is below the threshold.

4.4.1 ***Req. ↔ Req. Links.*** Consider S_1, S_2, S_3 as three source artifacts representing three discrete requirements documents and T_1 is a potential target document (i.e., source code file), where the target relationship being inferred is $S_1 \rightarrow T_1$, indicated by the solid line. The connections between nodes denote relationships among the artifacts. Consider the scenario in which the relationship between S_1 and T_1 is weak, but S_1 is highly similar to the other two requirements S_2 and S_3. Given that we know the three source artifacts are highly related, the relationships between $S_2 \rightarrow T_1$ and $S_3 \rightarrow T_1$ have a *transitive* influence on the target relationship between S_1 and T_1. For instance, if $S_2 \rightarrow T_1$ and $S_3 \rightarrow T_1$ both indicate strong probabilities, then likewise the probability of the target link $S_1 \rightarrow T_1$ should be increased to account for these transitive relationships.

4.4.2 ***Req.↔ Test Case Links.*** Consider S_1 to be a source artifact representing a requirement document, T_1 to be a potential target source code file, and TC_1, TC_2 to be test cases, where the target relationship being inferred is $S_1 \rightarrow T_1$, indicated by the solid line. Consider again the scenario in which the relationship between S_1 and T_1 is weak, whereas the relationship between S_1 and TC_1, TC_2 are stronger. If we observe that TC_1 and TC_2 are related to T_1 by execution information (e.g., TC_1 and TC_2 both exercise T_1, *Exec* in Fig. 2) then, this transitive relationship should influence the probability that a trace link exists between S_1 and T_1.

4.4.3 ***Incorporating Transitive Links into Comet's HBN.*** In order for our HBN to incorporate transitive req. ↔ req. links, it must first derive the set of requirements that are related to a given target requirement S_x. To accomplish this, one of several IR techniques can be used to compute textual similarity, or the first stage of our model can be used to derive the relationships, illustrated as $s(S_1, S_2)$ & $s(S_1, S_3)$ in Fig. 2. To incorporate information from req. ↔ test case links, dynamic information must be collected that provides the $Exec_1$ & $Exec_2$ relationships illustrated in Fig. 2. In either case, a specified threshold τ signals whether a pair of requirements is related, and the total number of related requirements or test cases is specified by the hyper-parameter π. Once the related requirements have been derived, our HBN estimates three hyperpriors, w_t, B_{mix} and μ_{trans}. First w_t is formulated as a Dirichlet distribution according to the number of related transitive requirements. Then to estimate B_{mix}, the first stage of our HBN is computed between each related requirement and a given target artifact T_y. The inferred values for each transitive link, and w_t are used to form a mixture model:

$$B_{mix} \sim Mix(w_t, \theta_\pi) \tag{7}$$

where B_{mix} is a Beta mixture model parameterized by the 1st stage inference of each transitive link and π weights modeled as a Dirichlet distribution parameterized by π. This Dirichlet distribution is then used to derive a meditated normal distribution μ_{trans}:

$$\mu_{trans} \sim \rho * B_{mix} + (1 - \rho) * Mean[\theta_{IR}] \tag{8}$$

where $Mean[\theta_{IR}]$ represents the mean of the probability distribution of IR similarity values (from stage 1) on the trace link prior and ρ is represents the *belief factor* of the transitive links (e.g., the degree to which the transitive relationships should affect overall prior trace link probability). μ_{trans} can then be utilized to derive the reward and penalty functions introduced earlier where $\Upsilon = \sigma * (1 - \mu_{trans})$ whereas $\Sigma = \sigma * \mu_{trans}$. The reward and penalty functions can then in turn be used to influence the mean of trace link prior μ:

$$\mu \sim N(\mu_n = \mu_{trans} + \Sigma + \Upsilon, sd = 0.01) \tag{9}$$

in the same manner as introduced in Eq. 6. In this way, our model is capable of incorporating information from transitive links, increasing the overall prior probability if transitive links are strongly connected to the target artifact T_y and decreasing it if they are not strongly connected.

4.5 Stage 4 - The Holistic Model

The holistic model combines all three underlying stages. To accomplish this, the calculations of the reward and penalty functions for affecting the mean μ of the overall prior are modified to incorporate information from both expert feedback and transitive links:

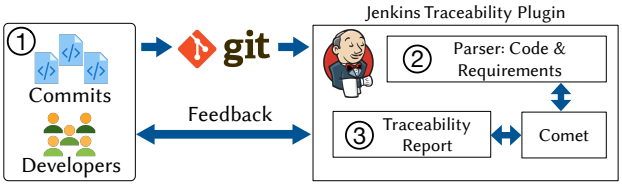


Figure 3: Comet Jenkins Plugin Flow

$$\begin{aligned} \Upsilon &\sim (1 - \mu_{trans}) * \sigma * D \\ \Sigma &\sim \mu_{trans} * \sigma * (D - 1) \end{aligned} \tag{10}$$

Then Eq. 9 can be used to derive the new mean for the overall prior probability distribution of the model.

4.6 Inferring the Posterior

In order to reason about the probability that a trace link exists, we must estimate the posterior probability distribution of our hierarchical model $p(\Theta|L)$ according to the observable data L and prior knowledge of the link $p(\Theta)$. Here $p(\Theta)$ encompasses the trace link prior and all constituent hyperpriors depending upon the stage of the model. Once the posterior has been estimated, Comet utilizes the *mean* of the distribution as the general probability that a link exists. We can represent the general calculation of the posterior for our model using using Bayes Theorem as follows:

$$p(\Theta|L) = \frac{p(\Theta)p(L|\Theta)}{\int p(\Theta)p(L|\Theta)d\Theta} \propto p(\Theta)\prod_{i=1}^{n} p(L_i|\Theta_i) \tag{11}$$

where n represents the total number of observations (i.e., the number of underlying IR techniques and configurations). Comet's HBN is non-trivial, and thus the posterior $p(\Theta|L)$ cannot be computed analytically. Therefore, we turn to approximation techniques for estimating the posterior probability distribution. Comet can currently utilize three different techniques including (i) Maximum a Posteriori (MAP) estimation [8], a Markov Chain Monte Carlo (MCMC) technique via the No-U-Turn sampling (NUTS) process [29], and a machine learning-based technique called Variational Inference (VI) [10]. We provide experimental results in Sec. 6 for all techniques for Stage 1 of Comet's model, and NUTS/MAP for Stages 2-4, as VI cannot be applied to more complex stages of the model.

4.7 The Comet Python Library & Jenkins Plugin

We implemented the four stages of Comet's HBN in an extensible traceability library written in `python3`. In order to explore the practical applicability of Comet, we implemented the first two stages of Comet's HBN as a Jenkins [3] plugin. We did not implement the final two stages due to time limitations, but are actively working on this in partnership with Cisco. This plugin was developed by one of the authors during an internship with Cisco Systems in close collaboration with researchers, engineers and analysts. The code and extensive documentation for the Comet library and plugin is available in our online appendix [2].

4.7.1 ***An Illustrated Use Case of the Comet Plugin.*** Fig. 3 provides a general overview of the Comet plugin architecture. To illustrate the *utility* of the plugin, we describe its workflow from the viewpoint of a developer. As depicted in Fig. 3-① the plugin is triggered when developers commit changes to a given project

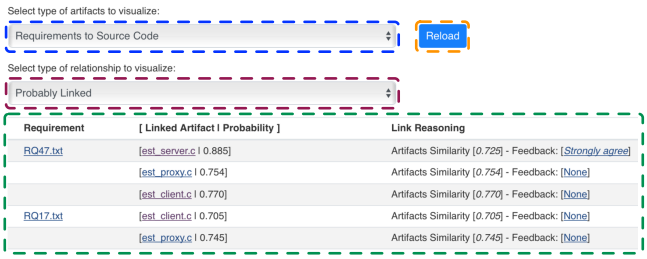


Figure 4: Comet Jenkins Plugin Traceability Report

configured to utilize our traceability plugin in Jenkins. The commit triggers a job that checks out, compiles, and executes the project code according to our industrial partner's existing CI pipeline. As illustrated in Fig. 3-② our plugin parses and preprocesses the source code, test code, and requirement text to be analyzed with Comet once the normal CI's build flow finishes. The preprocessed corpora of requirements, code, and tests are then passed to the Comet library where the *first stage* of the HBN (configured according to the tuning datasets described in Sec. 5) is run to establish an initial set of trace links among artifacts. Note that our plugin can be configured to run according to varying intervals (e.g., every minor or major release). Given that our plugin supports the first two stages of Comet's HBN, it is capable of collecting feedback from developers to improve trace links. To do this, developers select an option from a dropdown likert scale, with each option representing a potential value of μ_d for stage 2 of Comet's HBN (Strongly agree=1.0, Agree=0.75, Unsure=0.5, Disagree=0.25, & Strongly Disagree=0.0). To ensure a responsive feedback mechanism to reflect developer feedback, the Stage 2 link probabilities are precomputed.

Once the plugin job finishes it generates an interactive *traceability page* (Fig. 4), that allows a developer or analyst to view inferred trace links and provide feedback. As shown in Fig. 4 the blue rectangle highlights a drop-down menu that allows a developer to filter inferred trace links by type (i.e., req↔src, req↔test cases, src↔test cases, and artifacts not linked). The drop-down menu highlighted in red allows developers to filter trace links according to the inferred probability ranges of θ in Comet's HBN including: *probably linked* (i.e., $\theta = [0.7, 1)$), *probably not linked* (i.e., $\theta = [0, 0.4)$), and *unsure* representing links where the model is unable to make a confident inference (i.e., $\theta = [0.4, 0.7)$). The area highlighted in green lists the types of selected source and target artifacts allowing the user to easily inspect candidate trace links as well as provide feedback on the inferences. When a developer clicks on the `feedback` link for a given pair of artifacts, a popup window allows them to select one of the likert options delineated earlier. Finally, the traceability page also allows developers to view artifacts that are not linked to any other artifacts, and thus may be suspicious. Additional screenshots detailing the workflow of the Comet plugin are available in our online appendix [2].

4.7.2 ***Comet's Complexity & Scalability.*** We have designed the Comet plugin and library to facilitate easy integration into modern automated CI/CD systems via a set of higher level APIs that abstracts much of the complexity of the model for most users, while still providing mechanisms for advanced users to tweak parameters. Furthermore, Comet was able to be successfully deployed into a commercial CI pipeline for a pilot project with our industrial partner. We have also designed the Comet plugin and library to be highly scalable. Trace link probabilities between each pair of artifacts can be calculated independently, making the process highly parallelizable. Thus, we implemented parallel process management using `Theano` [4] into the Comet library, allowing computation to scale across modern multi-core machines. To further optimize performance, the Comet plugin can makex use of change analysis in git to only recompute trace link values for artifacts that have been altered since the last computation.

Table 1: Datasets used for Comet's evaluation. Rq = Requirement, Src = Source code, UC = Use Case

Project	Language	Size (LoC)	# Source Artifacts	# Target Artifacts	#Pairs/ #Links	Artifact Type
Tuning Projects						
Albergate	Java	10,464	55	17	935 / 53	Rq→Src
EBT	Java	1,747	40	25	1000 / 51	Rq→Tests
Experimental Projects						
LibEST	C	70,977	59	11	649 / 204	Rq→Src
			59	18	1,062 / 352	Rq→Tests
eTour	Java	23,065	58	116	6,728 / 308	UC→Src
EBT	Java	1,747	40	50	2,000 / 98	Rq→Src
SMOS	Java	9,019	67	100	6,700 / 1,044	UC→Src
iTrust	Java, JSP	38,087	131	367	48,077 / 399	Rq→Src

5 DESIGN OF THE EXPERIMENTS

To evaluate Comet, we perform an extensive empirical evaluation with two major *goals*: (i) evaluate the effectiveness of the four stages of Comet's HBN in terms of their ability to effectively infer trace links, and (ii) examine whether Comet is applicable in industrial workflows. The *quality focus* of our study is Comet's effectiveness, in terms of generating an accurate and complete set of trace links, and practical applicability. We formulate the following set of RQs:

- **RQ$_1$**: *How effective is* Comet *at inferring candidate trace links using combined information from IR/ML techniques?*
- **RQ$_2$**: *To what extent does expert feedback impact the accuracy of the candidate trace links of* Comet*?*
- **RQ$_3$**: *To what extent does information from transitive links improve* Comet's *trace link inference accuracy?*
- **RQ$_4$**: *How effective is the holistic* Comet *model in terms of inferring candidate trace links?*
- **RQ$_5$**: *Do professional developers and security analysts find our implementation of the* Comet *Jenkins plugin useful?*

5.1 Experimental Context

5.1.1 ***Subject Datasets.*** The *context* of this empirical study includes the eight datasets shown in Table 1. Six of these are taken from the open source CoEST community datasets [1]. These datasets represent a set of benchmarks created by the research community and widely used as an effective assessment tool for automated traceability techniques [6, 12, 25, 31]. In order to maintain the quality of our experimental subjects, we do not use all available projects in the CoEST repository, as we limited our studied systems to those that: (i) included trace links from requirements or use cases written in natural language to some form of code artifact, (ii) were written in English and/or included English translations, and (iii) had at least 1k LoC. We utilize two datasets to investigate and tune the hyper-parameters of Comet's HBN, Albergate, and the Rq→Tests dataset of the EBT project. We utilize the other six datasets for our

Table 2: IR/ML Techniques used in the Construction of Comet's HBN

IR Technique	Tag	Treshold Technique
Vector Space Model	VSM	Link-Est
Latent Semantic Indexing	LSI	Link-Est
Jensen-Shannon Divergence	JS	Min-Max
Latent Dirichlet Allocation	LDA	Min-Max
NonNegative Matrix Factorization	NMF	Median
Combined VSM + LDA	VSM+LDA	Link-Est
Combined JS+LDA	JS+LDA	Link-Est
Combined VSM+NMF	VSM+NMF	Link-Est
Combined JS+NMF	JS+NMF	Link-Est
Combined VSM+JS	VSM+JS	Min-Max

empirical evaluation. The subject system called "LibEST" is an open source networking related software project, which was created and is actively maintained by engineers at Cisco as an implementation of RFC-7030 "Enrollment over Secure Transport". We derived the ground truth set of trace links between Rq→Src and Rq→Tests for this dataset in close collaboration with our industrial partner. First, one of the authors carefully created an initial set of trace links. Then, an engineer working on the project reviewed the links and confirmed or denied a subset, based on their availability. The author then revised the links using the engineer's feedback, and this process continued over several months until the ground truth was established. The "LibEST" dataset is available along with all of our experimental data to facilitate reproducibility [2].

5.1.2 ***Studied IR Techniques.*** The "base" first stage of Comet's HBN is able to utilize and unify information regarding the textual similarity of development artifacts as computed by a set of IR/ML techniques. While there is technically no limit to the number of IR/ML techniques that can be utilized, we parameterized our experiments using ten IR-techniques enumerated in Table 2. The first five techniques are standalone techniques, whereas the second five are combined techniques utilizing the methodology introduced by Gethers et al. [25]. This combined approach normalizes the similarity measures of two IR techniques and combines the similarity measures using a weighted sum. We set the weighting factor λ for each technique equal to 0.5, as this was the best performing configuration reported in the prior work [25]. We explain the differences between the technique employed by Gethers et. al. and Comet in Sec. 2. The other parameters for each of the techniques were derived by performing a series of experiments on the two tuning datasets, and using the optimal values from these experiments. For all IR techniques, we preprocessed the text by removing non-alphabetic characters and stop words, stemming, and splitting camelCase. Text was vectorized using tf-idf vectors. We explored using word2vec to encode documents but found the performance to be considerably worse than tf-idf vectors. We performed 30 trials for each technique involving LDA, and chose the number of topics that led to optimal performance on our tuning projects. To aid in experimental reproducibility, complete configurations for each technique are listed in our online appendix [2].

5.2 RQ_1: Comet Performance w/ Combined IR/ML Techniques

To answer RQ_1, we ran the first stage of Comet's HBN on our six evaluation datasets using the ten IR/ML techniques enumerated in Table 2. However, as explained in Section 4.2, in order to accurately estimate the likelihood function Y we need to choose a threshold k_i for each IR technique that maximizes the precision and recall of the trace links according to the computed textual similarity values. To derive the best method for determining the threshold for each IR technique, we performed a meta evaluation on our two tuning datasets. We examined five different threshold estimation techniques: (i) using the mean of all similarity measures for a given dataset, (ii) using the median of all similarity measures across a given dataset, (iii) using a Min-Max estimation, (iv) a sigmoid estimation, and (v) link estimation (Link-Est), where an estimation of the number of confirmed links for a dataset is made based on the number of artifacts, and a threshold derived to ensure that the estimated number of links is above that threshold. We performed each of these threshold estimation techniques for all studied IR techniques across our two tuning datasets, and compared each estimation to the known optimal threshold. We used the optimal technique across our two tuning datasets, as reported in Table 2. To aid in reproducibility, we provide a detailed account of these experiments in our online appendix [2].

To provide a comparative baseline against which we can measure Comet's performance, we report results for the best-performing and median of the studied IR/ML techniques, optimally configured for each dataset. We chose to optimally configure the baseline techniques, even though such configurations would not be possible in practice due to the absence of a ground truth, in order to illustrate how close Comet can come to the "best-case baseline scenario".

To provide a comprehensive comparison of Comet to a state of the art technique for candidate trace link generation, we re-implemented the DL-based approach proposed by Guo et al. [26]. However, it should be noted that the intended purpose of this DL approach and Comet differ. The DL technique proposed by Guo et al. was intended to be both trained and evaluated on a single project that contains a set of *pre-existing* trace links the model can be trained upon, and was quite effective in improving the accuracy of trace links in this scenario. However, as pre-existing trace links may not always exist Comet *does not* require them for analysis. Instead, our experiments aim to illustrate that Comet can accurately infer trace links when tuned on one small set of projects, and applied to others. Therefore, we design an experimental setup where both techniques are applied on projects without pre-existing trace links. Thus, we train the DL approach on our two tuning projects, using the optimal parameters reported in [26]. Our main goal in comparing with this DL technique is to illustrate the performance of a recent ML-based technique applied to Comet's intended "cold-start" use case.

In order to measure the performance of our studied techniques for inferring trace links, we utilize three main metrics, Precision, Recall, and Average Precision (AP), similar to prior work that evaluates automated traceability techniques [25, 26]. Given that candidate link generation techniques infer a probability or similarity that a trace link exists, a threshold similarity or probability value must be chosen to make the final inference. In order to summarize the performance of our studied techniques, we calculate the Average Precision as a weighted mean of precisions per threshold: $AP = \Sigma_n(R_n - R_{n-1})P_n$ where P_n and R_n are the Precision and Recall at the nth threshold. Thus, the AP provides a metric by which we can quantitatively compare the performance of different

approaches. For the results of COMET, we report the highest AP achieved by the posterior estimation techniques outlined in Sec. 4.6. In addition to AP, we also provide Precision/Recall (P/R) curves to illustrate the trade-off between precision and recall at different threshold values. Curves further away from the origin of the graph indicate better performance. In lieu of a non-parametric statistical test as suggested by recent work [23], we perform a confidence interval analysis [45] between our baseline techniques and Stage 1 of COMET by calculating the standard error across different threshold values, applying bootstrapping where necessary. Thus, if one technique outperforms another within the bounds of our calculated error, it serves as a strong indication of statistical significance.

5.3 RQ_2: COMET Performance w/Expert Feedback

Collecting *actual* developer feedback on trace links for each of our test datasets was not possible given the time constraints on developers from our industrial partner, and we did not have access to the developers of the other projects. Thus, in order to evaluate Stage 2 of COMET's HBN, we simulated developer feedback by randomly sampling 10% of the artifact pairs from each studied subject, and used the ground truth to provide a confidence level for each of the sampled links. To accomplish this, we provided the model with a confidence value c of 0.9 if a link existed in the ground truth, and 0.1, if the link did not exist. However, even trace links derived from experts can be error-prone. Hence, we performed three types of experiments to simulate imperfect links being suggested to our model. That is, for the set of randomly sampled links, we intentionally reversed the confidence values according to the ground truth, for 25% and 50% of the sampled links respectively to simulate varying degrees of human error in providing link feedback. In other words, we sampled a small number of trace links from the ground truth, and then used these links to confirm/deny links predicted by COMET (i.e., if a ground truth link existed, and COMET predicted it, then it was confirmed). Because developers may not be correct all of the time, we simulated this by randomly flipping the sampled ground truth, which has a similar effect to a developer incorrectly classifying certain predicted links.

We set the value for the *belief factor* of the developer feedback $\sigma = 0.5$. For these experiments we illustrate the impact of developer feedback on AP and P/R curves for *only* the sampled links. In addition to the baseline IR techniques described in the procedure for RQ_1, we also compare our results from Stage 2 of the model to Stage 1, to illustrate the relative improvement.

5.4 RQ_3: COMET Performance w/Transitive Links

To measure the impact that transitive links have on the trace link inference performance of Stage 3 of COMET's HBN, we examined the impact of transitive links between requirements as described in Sec. 4.4. We utilize transitive requirement links rather than transitive links established by execution traces, as only one of our datasets (LibEST) had executable test cases. To derive the transitive relationships between artifacts, we computed the VSM similarity among all source documents for each dataset (e.g., requirements, use cases) and explored two values for the threshold τ, 0.65, and 0.5. We derived these thresholds by examining the total number of transitively linked requirements in our tuning datasets to achieve a balance between too many and too few requirements being linked. We set the *belief factor* ρ for Stage 3 of the HBN equal to 0.5. We report results for these experiments for only those requirements where transitive links impacted COMET's performance.

Table 3: AP Results from Stages 1 & 4 of COMET. The given p values from the Wilcoxon test measure the significance of performance variations between Stage 1, and Stage 4 of COMET's model compared to the median (Med.) baseline of IR/ML techniques. "I=Net" signifies the "Industry-Net" dataset.

Subject	Best Base.	Med. Base.	Std. Err	DL	Stage 1	Std. Err	Stage 4
LibEST (Rq→Src)	0.69	0.55	±0.008	0.28	0.63	±0.006	0.64
LibEST (Rq→Tests)	0.42	0.36	±0.001	0.32	0.38	±0.002	0.42
eTour	0.40	0.30	±0.011	0.05	0.38	± 0.002	0.36
EBT	0.17	0.14	±0.005	0.07	0.15	±0.001	0.17
SMOS	0.29	0.25	±0.003	0.16	0.25	± 0.001	0.27
iTrust	0.17	0.13	±0.006	0.01	0.17	±0	0.17

5.5 RQ_4: Holistic COMET Performance

To evaluate the overall performance of COMET's holistic model, we combined our experimental settings for RQ_2 & RQ_3. That is, we randomly sampled 10% of the links from each dataset and simulated developer feedback with a 25% error rate. Additionally, we incorporated transitive links between requirements using the same procedure outlined for RQ_3. For the transitive links, we set τ to 0.65, and we set the σ and ρ hyper-parameters both equal to 0.5. For this research question, we report results across all links.

5.6 RQ_5: COMET Industrial Case Study

Given that the ultimate goal of designing COMET is for the approach to automate trace link recovery within industry, we perform a case study with our industrial partner. This case study consisted of two major parts. First, we conducted a feedback session with six experienced developers who have been contributing to the LibEST subject program. This session consisted of a roughly 15 minute presentation introducing the COMET Jenkins plugin. Then the developers were asked to use the plugin, which had been configured for LibEST, and evaluate the links and non links for which the model was most confident (i.e., the highest and lowest inferred probabilities). Then after using the tool, they were asked a set of likert-based user experience (UX) questions derived from the SUS usability scale by Brooke [11]. Additionally, participants were asked free-response user preference questions based on the honeycomb originally introduced by Morville [42]. Second, we conducted semi-structured interviews with two groups consisting of roughly 15 engineering managers who specialize in auditing software for security assurance. During these interviews, a video illustrating the COMET plugin was shown, and a discussion was conducted with the questions illustrated in Fig. 6. We report results from both studies.

6 EMPIRICAL RESULTS

This section presents the results for our five proposed RQs. We highlight two P/R curves and focus our discussion on the AP results. However, all P/R curves and confidence interval graphs are currently available in our appendix alongside all experimental data [2].

6.1 RQ_1 Results: COMET Stage 1 Performance

The AP values for Stage 1 of COMET's HBN are provided in Table 3 alongside the p values for the Wilcoxon test between Comet and the median IR/ML baseline. The P/R curves for the iTrust dataset

Table 4: AP Results from Stage 2 of Comet with simulated expert feedback with error rates of 25% and 50%. The Baseline AP reported in this table is the median of the IR/ML techniques for the sampled links affected by feedback.

Subject	Baseline	St.1	St.2 (25%E)	Baseline	St.1	St.2 (50%E)
LibEST (Rq→Src)	0.52	0.65	0.96	0.52	0.65	0.64
LibEST (Rq→Tests)	0.28	0.32	0.80	0.28	0.32	0.44
eTour	0.48	0.60	0.66	0.48	0.60	0.39
EBT	0.20	0.22	0.38	0.20	0.22	0.24
SMOS	0.18	0.17	0.39	0.18	0.17	0.17
iTrust	0.12	0.15	0.25	0.12	0.15	0.10

Table 5: AP Results from Stage 3 of Comet for transitive links between requirements with τ=0.55 and τ=0.65. The baseline reported in this table is the median of the IR/ML techniques for links affected by transitive relationships.

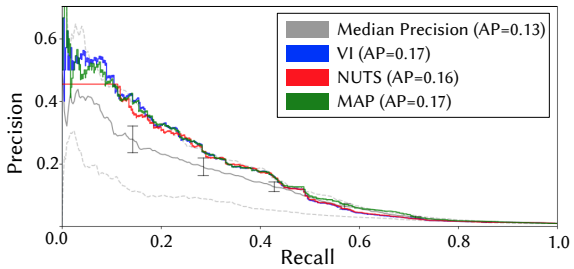
Subject	Baseline	St. 1	St.3 (τ=.55)	Baseline	St. 1	St.3 (τ=.65)
LibEST (Rq→Src)	0.53	0.6	0.59	0.39	0.67	0.44
LibEST (Rq→Tests)	0.38	0.4	0.38	0.18	0.19	0.22
eTour	0.33	0.4	0.42	0.37	0.48	0.48
EBT	0.24	0.26	0.24	0.02	0.03	0.06
SMOS	0.19	0.20	0.19	0.24	0.23	0.24
iTrust	0.11	0.14	0.15	–	–	–

are illustrated in Fig. 5a. As Table 3 indicates, Stage 1 of Comet outperforms the median IR/ML baseline across all subjects, to a statistically significant degree according to the confidence intervals. In some cases, such as for iTrust, LibEST, and eTour, Stage 1 of Comet *significantly* outperforms the median IR/ML baseline, and approaches the performance of the *best* IR/ML baseline. Fig. 5a illustrates the P/R curve for the iTrust project, with performance that outpaces the best IR/ML technique, particularly for lower recall values. Comet also outperforms the state of the art DL approach across all subjects, likely because the DL approach had difficulty generalizing semantic relationships across datasets.

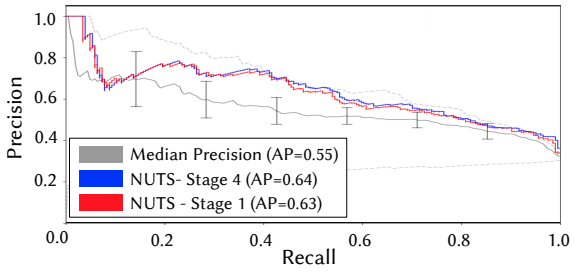
These results signal remarkably strong performance for Comet's Stage 1 model. Recall that, the Stage 1 model *only* utilizes observations taken from the set of ten IR/ML techniques introduced in Sec. 5.2, thus the fact that the Stage 1 model was able to consistently outperform the median IR/ML baselines and in some cases, nearly match the best IR/ML baseline. This indicates that Comet's HBN is capable of effectively combining the observations from the underlying IR/ML techniques for improved inference power. This is significant, as currently practitioners cannot know a-priori which IR/ML technique for traceability will perform best on a given project without pre-existing trace links. Thus, by combining the collective information of several IR techniques Comet's first stage HBN is able to perform ***consistently well***, achieving reasonably high performance ***across projects***, lending to the credibility of using Comet for projects that do not contain preexisting links.

6.2 RQ_2 Results: Comet Stage 2 Performance

The AP for for Stage 2 of Comet across all subject programs for both 25% and 50% error rates is given in Table 4.The results indicate that Stage 2 of Comet's HBN is able to effectively incorporate expert feedback to improve the accuracy of its trace link inferences, as the Stage 2 model dramatically outperforms the median (and best) IR/ML techniques as well as the first stage of the model, with a simulated error rate of 25%. Even for the larger error rate of 50%, we see Stage 2 outperform Stage 1 for LibEST (Rq→Src), LibEST (Rq→Tests) and EBT, while it slightly underperforms the Stage 1 model for the other subjects. These results illustrate that Stage 2 of Comet's HBN is able to effectively utilize expert feedback to improve its inferences, even in the presence of significant noise.



(a) P/R Curve for iTrust for Stage 1.



(b) P/R Curve for I-Net (Req→Src) for Stage 4

Figure 5: Selected P/R Curves for Stage 1 and Stage 4 of Comet. Solid grey line is median of the baseline IR/ML techniques, dotted grey lines are best and worst performing IR/ML techniques respectively.

6.3 RQ_3 Results: Comet Stage 3 Performance

The AP results for Stage 3 of Comet, which incorporates transitive relationships between requirements, for both $\tau = 0.55$ and $\tau = 0.65$ are given in Table 5 (There were no transitive links in iTrust for $\tau = 0.65$). This table also includes the median of the baseline IR/ML techniques, as well as the Comet Stage 1 model AP results, for the set of links affected by transitive relationships (hence the differing Stage 1 columns). The results show that, in general, for $\tau = 0.65$ for Comet's Stage 3 model, the accuracy of Comet's inferred trace links improve, with four of the six datasets showing improvements. For $\tau = 0.55$ the results generally exhibit similar or slightly worse performance compared to Stage 1. The fact that the higher value of τ led to better performance improvements is not surprising, as this parameter essentially controls the *degree of relatedness* required to consider transitive relationships. Thus, a higher value of τ means that only highly similar transitive requirement relationships are considered by Comet's model. Using a lower value for this parameter might introduce noise by incorporating transitive relationships between artifacts that don't have as high a degree of similarity.

The LibEST (Rq→Src) dataset exhibited decreased performance for $\tau = 0.65$, however this is likely because the requirements for this industrial dataset are based on formal format from the Internet Engineering Task Force (IETF). The somewhat repetitive nature of the language used in these requirements could lead to non-related requirements being transitively linked, leading to a decrease in performance. This suggests leveraging transitive relationships between requirements leads to larger performance gains for more unique language. Overall, our results indicate that Comet's Stage 3 model improves the accuracy of links for a majority of subjects.

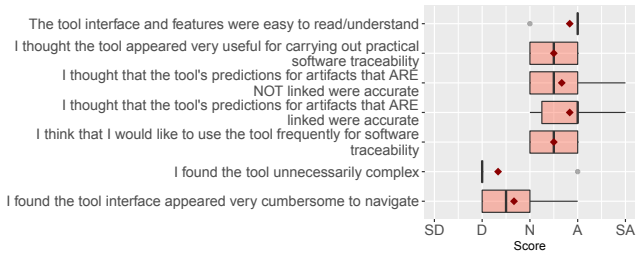


Figure 6: Results for LibEST Case Study UX Questions

6.4 RQ_4 Results: Comet Holistic Performance

The AP results for the the holistic Comet (Stage 4) model are given in Table 3. These results show that Comet's holistic model outperforms the baseline median IR/ML techniques, and Stage 1 for all subject programs. For three subjects (LibEST Req→Src, EBT, and iTrust), Comet's holistic model matches or outperforms the best baseline IR/ML technique. Fig. 5b illustrates the P/R curve for the LibEST (Req→Src) dataset, which shows that the performance gains in inference precision extend for a large range of recall values. The results of these experiments demonstrate that Comet's holistic model is able to effectively combine information from multiple sources to improve its trace link inference accuracy.

6.5 RQ_5 Results: Industrial Case Study

Fig. 6 provides the responses to the likert-based UX questions from the six developers who work on the LibEST project after interacting with the Comet plugin. Overall, the responses from these developers were quite positive. They generally agreed the Comet plugin easy to use and understand, but more importantly, generally found the accuracy of the inferred links and non-links to be accurate. Additionally, we highlight representative responses to the user experience questions in this section, and provide the survey questions with response summaries in our online appendix [2], in accordance with the NDA established with our industrial partner. Overall the developer responses were encouraging, indicating the practical need for approaches like Comet. For instance, one developer stated their need for such a tool, *"I really want a tool that could look at test cases and requirements and tell me the coverage. That way the team can know whether we are missing functionality or not."* Another developer explained the need for a feature that incorporates developer feedback, stating the importance of the *"ability to describe or explain how the code matches up with the code for future reference. Discussion/comments about such explanation as different developers might see links that others don't"*, whereas another developer stated, *"Being able to provide feedback is useful and seeing it update the percentage immediately was nice."* This indicates that the support for developer feedback and responsiveness of the Comet plugin inherently useful. Developers also found the traceability report to be useful, with most criticism suggesting practical UI improvements. For instance, developers appreciated *"The fact that there were the three different options for viewing the traceability between different [artifacts]"*, and *"The ability to bring up the specific requirement quickly in the same window."*. These responses illustrate the utility that developers saw in the Comet plugin. Given that these developers had little automated support for traceability tasks, they appreciated any automated assistance.

We also collected feedback that validated the importance of the practical use cases that the Comet plugin enabled. In these interviews, the teams generally stated that Comet would be very useful for code auditing, as one manager stated that it would *"allow compliance analysts to [inspect] links, look at the code and validate [the links]"*. Furthermore, a team responsible for security audits of systems found an interesting use case for Comet that is often overlooked in traceability analysis. That is, they were interested in code and requirements that are *not linked to any other artifact*, as such artifacts are likely to be suspicious and should be inspected further. In this case, Comet's inferences of non-links would be just as important as the inferences of links. Overall, the interviewed teams saw great promise in Comet, and expressed interest in adoption.

7 THREATS TO VALIDITY

Similar to past work on automated trace link recovery approaches [26], our work exhibits two main threats to validity. The first of these threats is to external validity. We utilize a limited number of systems to carry out our evaluation of Comet, and thus it is possible that our results may not generalize to other systems. However, the systems utilized in our evaluation have been widely used in past work, and are of varying sizes and domains. We also examine one industrial grade open source project developed by our partner.

The second threat to validity that affects experimental evaluation concerns construct validity, and more specifically, the accuracy of the ground truth for our subjects, and our implementation of the DL approach by Guo et al. [26]. We cannot guarantee that the ground truth links for all of subjects are perfectly accurate. However, the ground truth sets for the CoEST datasets have been accepted by several pieces of prior work [6, 12, 25, 31]. The ground truth for LibEST was derived by a team of the authors, and was validated with the help of industrial developers working on the project (see Sec. 5.1). We re-implemented Guo et al.'s DL approach closely following the details of the paper, although a full replication was not possible due to previous use of proprietary industrial dataset. We will release our code [2] for this approach to aid in reproducibility. Another potential threat to validity is that our simulation of developer feedback in answering RQ_2 is not representative of real feedback. Due to constraints on developers time, we could not use real feedback, however, we believe simulating a small number of links using the ground truth, complete with error rates, represents a reasonable approximation.

8 CONCLUSION & FUTURE WORK

We have presented Comet, which takes a probabilistic view of the traceability problem and models the existence of traceability links using a Hierarchical Bayesian Network. We have shown that Comet performs more consistently than IR/ML techniques, and has promising industrial applicability. We plan to adapt Comet to new information sources, investigate tailoring Comet's analysis to infer security-related links, and further deploy the Comet plugin with our industrial collaborators for feedback.

ACKNOWLEDGMENTS

This work is supported in part by Cisco Systems Inc. and the NSF CCF-1927679 grant. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

REFERENCES

[1] 2020. CoEST Traceability Datasets http://sarec.nd.edu/coest/datasets.html.

[2] 2020. Comet Online Appendix https://semeru-code-public.gitlab.io/Project-Websites/comet-website.

[3] 2020. The Jenkins Automation Framework https://jenkins.io.

[4] 2020. Theano Library http://deeplearning.net/software/theano/.

[5] Antoniol, Canfora, Casazza, and De Lucia. 2000. Information retrieval models for recovering traceability links between code and documentation. In *Proceedings of the International Conference on Software Maintenance (ICSE'00)*. 40–49.

[6] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. 2002. Recovering Traceability Links between Code and Documentation. *IEEE Transactions on Software Engineering* 28, 10 (2002), 970–983.

[7] Hazeline U. Asuncion, Arthur U. Asuncion, and Richard N. Taylor. 2010. Software Traceability with Topic Modeling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE '10)*. 95–104.

[8] Robert Bassett and Julio Deride. 2018. Maximum a posteriori estimators as a limit of Bayes estimators.

[9] Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas E. Webster. 1994. Program Understanding and the Concept Assignment Problem. *Commun. ACM* 37, 5 (May 1994), 72–82.

[10] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc.

[11] J. Brooke. 1996. SUS: A quick and dirty usability scale. In *Usability evaluation in industry*, P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland (Eds.). Taylor and Francis.

[12] Jane Cleland-Huang, Carl K. Chang, and Mark Christensen. 2003. Event-Based Traceability for Managing Evolutionary Change. *IEEE Trans. Softw. Eng.* 29, 9 (Sept. 2003), 15.

[13] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. 2010. A Machine Learning Approach for Tracing Regulatory Codes to Product Specific Requirements. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10)*. ACM, 155–164.

[14] Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman. 2012. *Software and Systems Traceability*. Springer Publishing Company, Incorporated.

[15] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. 2014. Software Traceability: Trends and Future Directions. In *Proceedings of the on Future of Software Engineering (FOSE'14)*. ACM, 55–69.

[16] Jane Cleland-Huang, Mona Rahimi, and Patrick Mäder. 2014. Achieving Lightweight Trustworthy Traceability. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*. 849–852.

[17] Tathagata Dasgupta, Mark Grechanik, Evan Moritz, Bogdan Dit, and Denys Poshyvanyk. 2013. Enhancing software traceability by automatically expanding corpora with relevant documentation. In *2013 IEEE International Conference on Software Maintenance*. IEEE, 320–329.

[18] Andrea De Lucia, Rocco Oliveto, and Paola Sgueglia. 2006. Incremental approach and user feedbacks: a silver bullet for traceability recovery. In *Proceedings of the International Conference on Software Maintenance (ICSM'06)*. 299–309.

[19] Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. 2008. IR-based traceability recovery processes: An empirical comparison of one-shot and incremental processes. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 39–48.

[20] Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. 2009. Assessing IR-based traceability recovery tools through controlled experiments. *Empirical Software Engineering* 14, 1 (2009), 57–92.

[21] Alex Dekhtyar, Jane Huffman Hayes, Senthil Karthikeyan Sundaram, Elizabeth Ashlee Holbrook, and Olga Dekhtyar. 2007. Technique Integration for Requirements Assessment. *Proceedings of the 15th IEEE International Requirements Engineering Conference* (2007), 141–150.

[22] Davide Falessi, Massimiliano Di Penta, Gerardo Canfora, and Giovanni Cantone. 2017. Estimating the number of remaining links in traceability recovery. *Empirical Software Engineering* 22, 3 (2017), 996–1027.

[23] Carlo A. Furia, Robert Feldt, and Richard Torkar. 2019. Bayesian Data Analysis in Empirical Software Engineering Research. *IEEE Transactions on Software Engineering* abs/1811.05422 (2019).

[24] Felipe Furtado and Andrea Zisman. 2016. Trace++: A traceability approach to support transitioning to agile software engineering. In *Requirements Engineering Conference (RE), 2016 IEEE 24th International*. IEEE, 66–75.

[25] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia. 2011. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Proceedings of the International Conference on Software Maintenance (ICSM'11)*. 133–142.

[26] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically Enhanced Software Traceability Using Deep Learning Techniques. In *Proceedings of the 39th International Conference on Software Engineering (ICSE'17)*. IEEE Press, 3–14.

[27] Jin Guo, Mona Rahimi, Jane Cleland-Huang, Alexander Rasin, Jane Huffman Hayes, and Michael Vierhauser. 2016. Cold-start Software Analytics. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR'16)*. ACM, 142–153.

[28] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. 2006. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering* 32, 1 (2006), 4.

[29] Matthew D. Hoffman and Andrew Gelman. 2011. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. (2011).

[30] Hsin-Yi Jiang, Tien N Nguyen, Xiang Chen, Hojun Jaygarl, and Carl K Chang. 2008. Incremental latent semantic indexing for automatic traceability link evolution management. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. IEEE Computer Society, 59–68.

[31] Samuel Klock, Malcom Gethers, Bogdan Dit, and Denys Poshyvanyk. 2011. Traceclipse: an eclipse plug-in for traceability link recovery and management. In *Proceedings of the international workshop on traceability in emerging forms of software engineering (TEFSE'11)*. 24–30.

[32] Hongyu Kuang, Jia Nie, Hao Hu, Patrick Rempel, Jian Lü, Alexander Egyed, and Patrick Mäder. 2017. Analyzing closeness of code dependencies for improving IR-based Traceability Recovery. In *Proceedings of the 24th International Conference on Software Analysis, Evolution and Reengineering (SANER'17)*. 68–78.

[33] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2013. API Change and Fault Proneness: A Threat to the Success of Android Apps. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE'13)*. 477–487.

[34] Sugandha Lohar, Sorawit Amornborvornwong, Andrea Zisman, and Jane Cleland-Huang. 2013. Improving Trace Accuracy Through Data-driven Configuration and Composition of Tracing Features. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013)*. ACM, 378–388.

[35] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. 2004. Enhancing an artefact management system with traceability recovery features. In *Proceedings of the 20th IEEE International Conference on Software Maintenance, 2004. Proceedings. (ICSM'04)*. 306–315.

[36] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. 2013. Strategic Traceability for Safety-Critical Projects. *IEEE Software* 30, 3 (May 2013), 58–66.

[37] A. Mahmoud, N. Niu, and S. Xu. 2012. A semantic relatedness approach for traceability link recovery. In *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC'12)*. 183–192.

[38] Anas Mahmoud and Grant Williams. 2016. Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering* 21, 3 (01 Sep 2016), 357–381.

[39] Andrian Marcus and Jonathan I. Maletic. 2003. Recovering Documentation-to-source-code Traceability Links Using Latent Semantic Indexing. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*. IEEE Computer Society, Washington, DC, USA, 125–135.

[40] Collin McMillan, Denys Poshyvanyk, and Meghan Revelle. 2009. Combining Textual and Structural Analysis of Software Artifacts for Traceability Link Recovery. In *Proceedings of the ICSE Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '09)*. IEEE Computer Society, 41–48.

[41] Chris Mills, Javier Escobar-Avila, and Sonia Haiduc. 2018. Automatic Traceability Maintenance via Machine Learning Classification. In *Proceedings of the 34th IEEE International Conference on Software Maintenance and Evolution (ICSME'18)*. ACM, Madrid, Spain, 369–380.

[42] Peter Morville. 2020. User Experience Design. http://semanticstudios.com/user_experience_design/.

[43] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective*. The MIT Press.

[44] Shiva Nejati, Mehrdad Sabetzadeh, Davide Falessi, Lionel Briand, and Thierry Coq. 2012. A SysML-based Approach to Traceability Management and Design Slicing in Support of Safety Certification: Framework, Tool Support, and Case Studies. *Inf. Softw. Technol.* 54, 6 (June 2012), 569–590.

[45] J. Neyman. 1937. *Outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability*. Royal Society. https://books.google.com/books?id=jqBtGwAACAAJ

[46] Armstrong Nhlabatsi, Yijun Yu, Andrea Zisman, Thein Tun, Niamul Khan, Arosha Bandara, Khaled M. Khan, and Bashar Nuseibeh. 2015. Managing Security Control Assumptions Using Causal Traceability. In *Proceedings of the 8th International Symposium on Software and Systems Traceability (SST '15)*. IEEE Press, 43–49.

[47] Kazuki Nishikawa, Hironori Washizaki, Yoshiaki Fukazawa, Keishi Oshima, and Ryota Mibe. 2015. Recovering transitive traceability links among software artifacts. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'15)*. IEEE, 576–580.

[48] Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. 2010. On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In *Proceedings of the 18th International Conference on Program Comprehension (ICPC '10)*. 68–71.

[49] H. Raïffa and R. Schlaifer. 1961. *Applied statistical decision theory*. Division of Research, Graduate School of Business Adminitration, Harvard University. https://books.google.com/books?id=wPBLAAAAMAAJ

[50] Michael Rath, Jacob Rendall, Jin LC Guo, Jane Cleland-Huang, and Patrick Mäder. 2018. Traceability in the wild: automatically augmenting incomplete trace links. In *Proceedings of the 40th International Conference on Software Engineering (ICSE'18)*.

ACM, 834–845.

[51] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar T. Devanbu. 2014. A large scale study of programming languages and code quality in github. *Commun. ACM* 60 (2014), 91–100.

[52] Patrick Rempel, Patrick Mäder, Tobias Kuschke, and Jane Cleland-Huang. 2014. Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*. ACM, 943–954.

[53] George Spanoudakis, Artur S d'Avila Garcez, and Andrea Zisman. 2003. Revising Rules to Capture Requirements Traceability Relations: A Machine Learning Approach.. In *SEKE (SEKE'03)*. 570–577.