Trust-based Requirements Traceability

Nasir Ali[1,2], Yann-Gaël Guéhéneuc[1], and Giuliano Antoniol[2]
[1] *Ptidej Team, DGIGL, École Polytechnique de Montréal, Canada*
[2] *SOCCER Lab, DGIGL, École Polytechnique de Montréal, Canada*
E-mail: {nasir.ali,yann-gael.gueheneuc}@polymtl.ca,antoniol@ieee.org

***Abstract*—Information retrieval (IR) approaches have proven useful in recovering traceability links between free-text documentation and source code. IR-based traceability recovery approaches produce ranked lists of traceability links between pieces of documentation and source code. These traceability links are then pruned using various strategies and, finally, validated by human experts. In this paper we propose two contributions to improve the precision and recall of traceability links and, thus, reduces the required human experts' manual validation effort. First, we propose a novel approach, Trustrace, inspired by Web trust models to improve the precision and recall of traceability links: Trustrace uses any traceability recovery approach to obtain a set of traceability links, which rankings are then re-evaluated using a set of other traceability recovery approaches. Second, we propose a novel traceability recovery approach, Histrace, to identify traceability links between requirements and source code through CVS/SVN change logs using a Vector Space Model (VSM). We combine a traditional recovery traceability approach with Histrace to build Trustrace$^{\text{VSM, Histrace}}$ in which we use Histrace as one expert adding knowledge to the traceability links extractted from CVS/SVN change logs. We apply Trustrace$^{\text{VSM, Histrace}}$ on two case studies to compare its traceability links with those recovered using only the VSM-based approach, in terms of precision and recall. We show that Trustrace$^{\text{VSM, Histrace}}$ improves with statistical significance the precision of the traceability links while also improving recall but without statistical significance.**

***Keywords*—Traceability, requirements, source code, experts, trust-based model.**

I. Introduction

Preliminary to any software evolution task, a developer must understand the project landscape [1] and, in particular, the system architecture, design, implementation, and the relations between the various artifacts and code sections using any available documentation. Traceability links between the documentation associated with the development and maintenance cycle of the system and its source code are helpful in reducing program comprehension effort.

Existing comprehension models share the idea that program comprehension occurs in a bottom-up manner [2], [3], a top-down manner [4], [5], or some combination thereof [6], [7], [8]. They also agree that developers use different types of knowledge during program comprehension, ranging from domain specific knowledge to general programming knowledge [4], [9], [10]. Traceability links between code sections and related sections of the documentation aid both top-down and bottom-up comprehension [11].

Information Retrieval (IR) approaches [12] have proven useful in recovering traceability links and various approaches have been applied to recover links, from Vector Space Models (VSM) and probabilistic models [13], to Latent Semantic Indexing (LSI) [14], and to Latent Dirichelet Allocation (LDA) [15], with various success [16]. Traceability recovery approaches assume the availability of high-level free-text documentation and infer links between pieces of this documentation and source code entities using an indexing process producing a ranked list of traceability links [17].

Unfortunately, as the source code evolves, its accompanying documentation is often not updated and maintaining the traceability links between evolving software artifacts and the consistency of existing links is tedious and error-prone [18] and, thus, frequently neglected by developers due to the pressure to reduce the time to market and to move on to the next change. Consequently, traceability links become less relevant and should be recomputed and revalidated manually, which is costly and time-consuming.

We propose two contributions to improve the precision and recall of traceability links. First, we propose a novel approach, Trustrace, inspired by Web trust models [19], [20], [21], [22] to improve precision and recall of traceability links: Trustrace uses any traceability recovery approach as the basis on which it applies various experts' opinions [23] to remove and–or adjust the rankings of the traceability links. The experts can be human experts or other traceability recovery approaches.

Second, we show how Trustrace improves traceability recovery accuracy using Histrace, an expert supporting the identification of traceability links between requirements and source code through CVS/SVN change logs and a Vector Space Model (VSM). In any organization developing software, there are many heterogeneous sources of information available, including CVS/SVN repositories, bug-tracking systems, mailing lists, forums, and blogs. Histrace uses CVS/SVN change logs to build traceability links between high-level documentation and source code entities, observing that log messages are tied to changed entities and, thus, can be used to infer traceability links.

We combine a traditional recovery traceability approach with Histrace to build Trustrace$^{\text{VSM, Histrace}}$ in which we use Histrace as one expert commenting the traceability

links recovered using the VSM-based approach. We apply Trustrace$^{\text{VSM, Histrace}}$ on two case studies to compare its traceability links with those recovered using only the VSM-based approach, in terms of precision and recall. The objects of our case studies are Pooka v2.0 and SIP Communicator v1.0-draft, whose requirement have been previously documented and manually traced to source code entities. We show that Trustrace$^{\text{VSM, Histrace}}$ improves with statistical significance the precision of the traceability links while also improving recall but without statistical significance. We thus show that our trust-based approach indeed improves precision and recall and also that CVS/SVN change logs are useful in the traceability recovery process.

The remainder of the paper is organized as follows. Section II discusses related. Section III presents Trustrace, our trust-based approach while Section IV describes Histrace, our traceability recovery approach based on CVS/SVN change logs. Section V sketches our implementation of Trustrace and Histrace. Section VI presents the two case studies while Section VII reports and discusses their results. Finally, Section IX concludes with future work.

II. Related work

Traceability recovery, concept and feature location as well as trust models are related to our research work.

Static analyses [24], execution traces [25], and IR techniques [13] have been used by researchers since the early works on traceability recovery [13], feature location [25], and concept recognition [26] or location [24]. Often, IR-based approaches [13], [14], [23], use vector space models, probabilistic rankings, or a vector space model transformed using latent semantic indexing. Whenever available dynamic data [25], [23] proved to be complementary and useful for traceability recovery by reducing the search space. Recently, high-level documentation was mapped into source code features using a variety of information sources [27] and the CERBERUS tool was applied to the Rhino Java ECMA script implementation and favorably compared to a variety of other approaches, *e.g.*, [23] and [28].

The precision and the recall [13] of the links recovered during traceability analyses are influenced by a variety of factors, including the conceptual distance between high-level documentation and low-level artifacts, the way in which queries are formulated, and the applied IR technique. Comparisons have been made between different IR techniques, *e.g.*, [17] and [29], with inconclusive results. On certain data set, the vector space model performs favorably in comparison to more complex techniques, such as Jensen inequality or probabilistic latent semantic semantic analyses [29]. Yet, the vector space model [13] and latent semantic indexing [14], [17] with $tf - idf$ weighting schema [30] are a reference baseline for both feature location [23], [31] and traceability recovery [13], [17].

Kagdi et al. [32] presented a heuristic-based approach to recover traceability links between software artifacts using the software system version history. Their approach assumes that, if two or more files co-change [33] in the system history, then there is a possibility that they have a link between them. However, it is quite possible that two files are co-changing but that they do not have any semantic relationship. It is also likely that some documents evolve outside the system version control repository and, in such a case, their approach cannot find link from/to these documents, *e.g.*, requirement specifications. In addition, their approach does not analyse contents of the CVS/SVN logs and files that were committed in CVS/SVN. More importantly, in co-change-based traceability [32], [33], [34], if two or more files have a link but they were not co-changed then these approaches fail to find a link. Our proposed novel approach is not dependent on co-changes and overcomes these limitations.

Our proposed novel approach is influenced by the Web trust model [19], [20], [21], [22]. There are two type of trust in e-commerce: first a customer's initial trust [22] when she interacts with a Web site for the first time and, second, the Web site reputation trust [35] that develops over time and after repeated experiences. When customers hesitate to buy things online, they may ask their friends, family, and other buyers to make sure that they can trust a Web site. Many researchers investigated [19], [35], [36], [21], [22] the problem of increasing customers' trust in a Web site. Some researchers [19], [21] suggested that using other sources of information can increase the trust in a Web site. Palmer et al. [21] presented an empirical investigation of how firms can improve customers' trust by exploring and using Trusted Third Parties (TTPs) and privacy statements. Their exploratory data shows that the use of TTPs and privacy statements increase a customer's trust in a Web site. Wang et al. [37] presented a novel content trust learning algorithm that use the content of a Web site as a trust point to distinguish trustable Web contents and spam contents.

Our proposed approach uses traceability from requirement to source code as initial trust and then uses CVS/SVN change logs and could also use bug reports, mailing lists, temporal information and so on, as reputation trust for a traceability link. As the reputation of a link increases, the trust in this link also increases.

III. Trustrace: Trust-based Traceability

We introduce our novel approach, Trustrace, to improve the trust in traceability links and thus improve the precision and recall of any traceability recovery process by combining experts' opinions. Figure 1 shows the high-level architecture of Trustrace, whose conceptual steps are detailed below.

A. Definitions

In Trustrace, we represent a traceability link as a triple {source document, target document, and similarity}. Let $R = \{r_1, \ldots, r_N\}$ be a set of requirements or high-level

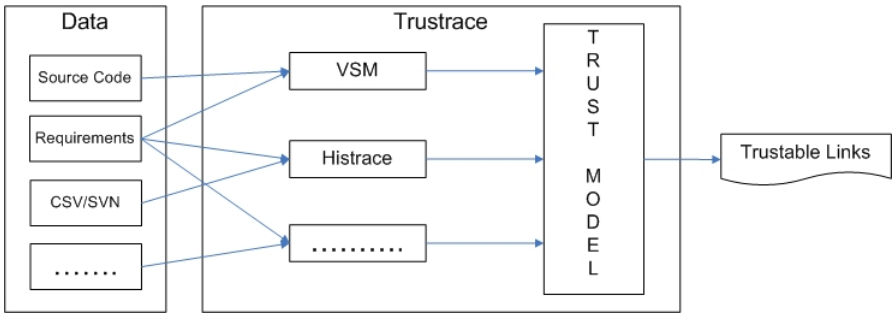

Figure 1. Trust-based requirement traceability process

documents, $C = \{c_1, \ldots, c_M\}$ be a set of implementing classes. Let $\mathcal{T} = \{T_1, \ldots, T_P\}$ be a collection of sets where each set $T_i = \{t_1, \ldots, t_{N_i}\}$ is a set of homogeneous pieces of information, for example the set of all CVS/SVN commits or of all bug-tracking reports, and so on.

Then, let us assume that, for each $T_i \in \mathcal{T}$, it is possible to define a function δ_{T_i} mapping one element of T_i into a subset of C or the empty set. For example, if T_i is a set of bug reports, then, for a given bug report t_k, $\delta_{T_i}(t_k)$ returns the set of classes affected by t_k, with $\delta_{T_i}(t_k) \subseteq C$. We describe in details such a δ function for the set T_i of CVS/SVN commits in the following Section IV.

Consequently, we can define $R2C$ as the set of traceability links recovered between R and C and, for each set $T_i \in \mathcal{T}$, a set $R2CT_i$ generated by each piece of information t_k:

$$R2CT_i(r_j, t_k) = \{(r_j, c_s, \sigma_{j,k}) | c_s \in \delta_{T_i}(t_k) \ \& \ t_k \in T_i\}$$

Finally, let us define two functions α and ϕ: the first function, $\alpha(r_i, c_j, \sigma_{i,j})$, returns the pair of involved documents (or the set of pairs), thus (r_i, c_j) while, the second function, $\phi(r_i, c_j, \sigma_{i,j})$, returns the similarity score $\sigma_{i,j}$.

B. Model

Trustrace uses the set of candidate links $l_{rc} = (r_j, c_s, \sigma_{j,s})$ with $j \in [1, \ldots, N]$ and $s \in [1, \ldots, M]$. It also uses the sets of candidate links $l_{rt} = (r_j, t_k, \sigma'_{j,k})$ with $j \in [1, \ldots, N]$ and $k \in [1, \ldots, N_i]$ generated from each set of some other pieces of information T_i and for each requirement $r_j \in R$.

Indeed, for each set $T_i \in \mathcal{T}$, Trustrace builds a set of trustable links Tr_i as follows:

$$\begin{aligned} Tr_i = \{(r_j, c_s, \sigma_{j,k}) \quad & | \\ \exists\ t_k \in T_i \quad & : \quad (r_j, c_s) \in \alpha(R2CT_i(r_j, t_k)) \\ & \& \quad (r_j, c_s) \in \alpha(R2C)\} \end{aligned} \tag{1}$$

The last rule $(r_j, c_s) \in \alpha(R2C)$ may over-constrain the problem as it imposes that a link be present in the base-line set $R2C$. We plan to lessen this rule in future version of Trustrace. Because pair of documents (r_j, c_s) may appear in several links in $R2CT_i$, Trustrace re-assigns scores to document pairs in Tr_i according to the frequency of the pairs. Let $TC_i(r_j, c_s)$ be the restriction of Tr_i on (r_j, c_s), *i.e.*, the set $\{(r_j, c_s, \sigma'_{j,s}) \in Tr_i\}$, then Trustrace re-assigns to $TC_i(r_j, c_s)$ elements a new similarity $\sigma^*_{j,s}$ computed as:

$$\sigma^*_{j,s} = \frac{\sigma_{j,s} + \sum_{l \in TC_i(r_j, c_s)} \phi(l)}{|TC_i(r_j, c_s)| + 1} \tag{2}$$

where $\sigma_{j,s}$ is the similarity between the requirement r_j and the class c_s as computed in $R2C$ and $\phi(l)$ is the similarity of elements in the link l of $TC_i(r_j, c_s)$, *i.e.*, $\sigma'_{j,s}$. The higher the evidence (*i.e.*, $\sum_{l \in TC_i(r_j, c_s)} \phi(l)$) provided by links in TC_i, the higher the new similarity $\sigma^*_{j,s}$; in the contrary, little evidence decreases $\sigma^*_{j,s}$ relatively to other similarities.

Then, Trustrace assigns a trust level to each link in Tr_i using the ψ function defined as:

$$\psi_{j,s}(Tr_i) = \lambda_i \sigma^*_{j,s} + (1 - \lambda_p) \frac{|TC(r_j, c_s)|}{\max_{n,m} |TC_i(r_n, c_m)|} \tag{3}$$

where $\lambda_p \in [0, 1]$. In essence, with this ψ function, the more often a pair (r_j, c_s) exists in $R2CT_i$, the more we can trust this link (if such a link is also present in $R2C$). Trustrace model is thus similar to the Web model of users' trust: the more users buy from a Web merchant, the higher the users' trust of this merchant.

Finally, Trustrace merges the trust levels of each Tr_i into a global level of trustworthiness:

$$\begin{aligned} \psi^*_{j,s} \quad = \quad & \sum_p \beta_p \ \psi_{j,s}(Tr_p) \\ \text{with} \quad & \sum_p \beta_p = 1 \\ \text{and} \quad & 0 \leq \beta_p \leq 1 \end{aligned} \tag{4}$$

and where the β_p coefficient states our general level of trust in each source of information. For example, we may trust more CVS/SVN commits over bug reports and, thus, the β_p coefficient of the former will be higher than that of the latter.

IV. HISTRACE: CVS/SVN CHANGE LOGS-BASED TRACEABILITY RECOVERY

In this paper, we apply our model to a single source of information, *i.e.*, the set T_0 of CVS/SVN commits, and, thus, $\psi^*_{j,s} = \psi_{j,s}$ with $\beta = 1$.

Thus, we now describe our novel traceability recovery approach using the sets of requirements (*i.e.*, R), classes (*i.e.*, C) and, as supplementary source of information, the CVS/SVN commits. We consider each requirement's textual description, SVN/CVS commit message, and class source code as a separate document. We implement the processing of CVS/SVN commits and the computation of the δ_{T_0} function in the Histrace approach, which uses historical data to recover traceability links.

A. Document Pre-processing

Depending on the input information source (*i.e.*, R, high-level documents, C, source code, or T_0 CVS/SVN commit transactions), we perform specific pre-processing steps to remove irrelevant details, (*e.g.*, CVS/SVN commit number, source code punctuation or language keywords), split identifiers, and, finally, normalize the text using stop words removal and stemming.

1) Requirements and Source Code: Source code files are first processed to extract all the identifiers and comments in each class. Trustrace then uses underscore and the Camel Case convention to split identifiers into terms, thus producing for each class a separate document.

Trustrace then performs the following steps to normalize documents: (i) convert all upper-case letters into lower-case and remove punctuation; (ii) remove all stop words (such as articles, numbers, and so on); and, (iii) perform word stemming using the Porter Stemmer bringing back inflected forms to their morphemes. We retain all processed identifiers and comments of source code to create traceability links.

2) CVS/SVN Commits: To build T_0, Histrace extracts CVS/SVN commits and discards those that (i) are tagged as "delete", (ii) does not concern source code (*e.g.*, changed manual pages or documentation only), and (iii) have messages of length shorter or equal to two words. Histrace then analyzes remaining CVS/SVN commits to extract the messages as well the classes that (i) are still part of the source code and (ii) have been part of the commits. Histrace relies on the CVS/SVN `diff` mechanism and source code parsing to identify modified classes. Formally, T_0 is the set of all CVS/SVN commit messages and, for any $t_k \in T_0$, we have a function δ_{T_0} that returns a subset of classes $C'_k \subset C = \{c_1, \ldots, c_M\}$ modified in the transactions; the log messages are fed into a VSM model to compute $\sigma'_{j,k}$. Using δ_{T_0}, we can build $R2CT_0$, see previous section.

B. Traceability Recovery Process

Trustrace traceability recovery process uses as base algorithm the standard VSM [12], [13], [17]. Documents are represented by elements in a vectors space. Different term weighting schemes can be used to construct these vectors. Trustrace uses the $tf - idf$ weighting schema [12]: a document is a vector of $tf - idf$ weights. More precisely, a vector entry represents a term with a weight product of a local weight (*i.e.*, the number of times a term appears in a document) and the logarithm of the term inverse document frequency (*i.e.*, one divided by the number of document containing the term).

Once documents (*i.e.*, requirement descriptions, classes source code, and SVN/CVS commit messages) are represented as element of the VSM, links are recovered assigning a similarity value to each pair of documents (*e.g.*, a requirement and a class) using the cosine similarity between document vector representations. Cosine values are in $[-1, 1]$ but negative values are discarded and a link has thus a value in $[0, 1]$. Finally, as in previous work [13], a ranked list of recovered links and a similarity threshold are used to divide links into a set of candidate links to be manually verified and discarded links.

In summary, we build the set $R2C$ from R and C using VSM and Histrace builds the set Tr_0 using CVS/SVN commits by implementing δ_{T_o}. We use the simplified version of Equation (4), because we use just one set of homogeneous pieces of information, to re-assign similarities to the links in $R2C$. Indeed, when introducing the new set T_0 in Equations (1), (2), (3), and (4), we obtain a new set of traceability links, whose similarities have been revised using our trust model.

V. TOOL SUPPORT

FacTrace [1], for arteFACT TRACEability, provides several modules that help from traceability recovery to traceability links verification. FacTrace aids software engineers in different tasks, namely, requirement elicitation, requirement analysis, artefact traceability, and most importantly for Trust-based traceability. FacTrace has a graphical interface to perform different tasks. Following sections give you a brief description of different features of FacTrace:

1) Traceability Management: FacTrace aids in recovering traceability links between different software artefacts. For example, traceability links among requirements, source code, and CVS/SVN change logs. FacTrace allows experts to create new manual traceability links as well. It supports different level of granularity for creating traceability links. Expert can write description for each link and other identifiers specifications as well.

2) Traceability Links Verification: To avoid bias when recovering traceability links, FacTrace supports a voting system for each link. It supports up to five experts voting for each link. If three or more than three expert accepts a link, then a link will be considered as a valid link by FacTrace. Experts can change their voting option at any time.

[1] http://www.factrace.net

All other experts' voting is hidden to avoid bias. Experts can see source code files in the source code viewer of FacTrace to verify each link.

VI. Empirical Studies

We perform two case studies to assess the precision and recall of our proposed approaches for requirement traceability, Trustrace and Histrace. These two case studies provide data to assess the improvement over a "traditional" VSM-based approach and, consequently, the reduction of the experts' effort brought to the maintainer when tracing requirements and validating traceability links to source code.

A. Goal

The *goal* of our case studies is to evaluate the effectiveness of our novel approaches, combined into Trustrace$^{\text{VSM, Histrace}}$, in recovering traceability links against a traditional VSM-based approach, using requirements, source code, and–or CVS/SVN change logs as sources of information. The *quality focus* is the ability of Trustrace$^{\text{VSM, Histrace}}$ to recover traceability links between high-level documents and source code in terms of precision and recall [30]. Both measures have values in $(0, 1)$.

Precision is defined as the number of relevant documents retrieved divided by the total number of retrieved documents by an approach. If the value is 1 for precision then all the recovered documents are correct.

Recall is defined as the relevant documents retrieved divided by the total number of relevant documents. It is ratio between the number of documents that are successfully retrieved and the number of documents that are relevant. If the value is 1 for recall, then all relevant documents have been retrieved by an approach.

The *perspective* is that of practitioners and researchers, interested in recovering traceability links with greater precision and recall values than that of currently-available traceability recovery approaches based on IR techniques. The *objects* of our case studies are two open-source systems, *Pooka* and *SIP* whose requirements, source code, and SVN change logs are available on-line[2].

B. Research Question

The research question that our case studies address is:

RQ*: *How do the precision and recall values of the traceability links recovered by Trustrace$^{VSM,\ Histrace}$ compare with those of a traditional VSM-based approach?

To answer this research question, we assess the precision and recall of Trustrace$^{\text{VSM, Histrace}}$ when identifying correct traceability links between requirements and source code on the one hand and between requirements and source code using SVN change logs on the other. We thus apply Trustrace$^{\text{VSM, Histrace}}$ and a VSM-based approach on the two systems seeking to reject the two null hypotheses:

H_{01}: *There is no difference in the precision of the recovered traceability links when using Trustrace$^{VSM,\ Histrace}$ or a VSM-based approach.*

H_{02}: *There is no difference in the recall of the recovered traceability links when using Trustrace$^{VSM,\ Histrace}$ or VSM-based approach.*

Table I
Statistics describing Pooka and SIP

	Pooka	SIP
Version	2.0	1.0
Number of Classes	298	1,771
Number of Methods	20,868	31,502
Source Code Sizes	244,870 LOCs	486,966 LOCs
	5.39 MB	27.3 MB

C. Variables

We use precision and recall as independent variable and the approach, either a "traditional" VSM-based approach or Trustrace$^{\text{VSM, Histrace}}$, as dependent variables to empirically attempt rejecting the null hypotheses. The independent variable corresponding to Trustrace$^{\text{VSM, Histrace}}$ also includes varying values of λ as analyzed and discussed below.

D. Objects

We select the two open-source systems, *Pooka* and *SIP*, because they satisfy several criteria. First, we select open-source systems, so that other researchers can replicate our experiment. Second, we avoid small systems that do not represent systems handled by most developers. Yet, both systems were small enough so that we were able to recover and validate their requirements manually in a previous work. Finally, their source code was freely available in their respective SVN repositories. Table I provides some descriptive statistics of the two systems.

Pooka[3] is an email client written in Java using the JavaMail API. It supports reading email through the IMAP and POP3 protocols. Outgoing emails are sent using SMTP. It supports folder search, filters, context-sensitive colors, and so on. SIP[4] is an audio/video Internet phone and instant messenger that supports some of the most popular instant messaging and telephony protocols, such as SIP, Jabber, AIM/ICQ, MSN, Yahoo! Messenger, Bonjour, IRC, RSS.

E. Gathering and Pre-processing Requirements, Source Code, and SVN Change Logs

We now details how we gather and prepare the input data necessary to our case studies.

[2]See also for convenience at http://www.ptidej.net/downloads/experiments/icpc11b/

[3]http://www.suberic.net/pooka/

[4]http://www.jitsi.org/

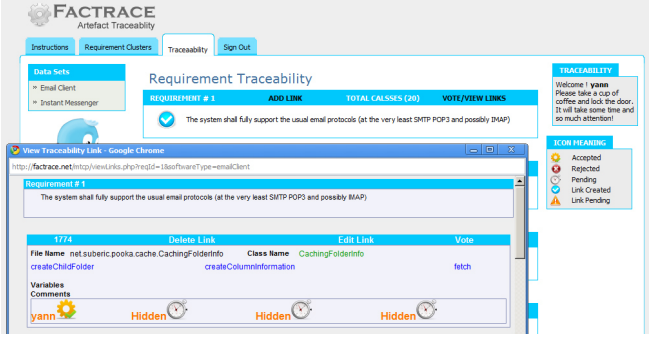

Figure 2. Excerpt of FacTrace UI

```
<logentry revision="1741">
    <author>akp</author>
    <date>2008-10-18T16:14:08.529138Z</date>
    <paths>
        <path kind="" action="M">/trunk/pooka/todo</path>
        <path kind="" action="M">/trunk/pooka/src/net/
            suberic/pooka/gui/NewMessageDisplayPanel.java
        </path>
        <path kind="" action="M">/trunk/pooka/src/net/
            suberic/pooka/NewMessageInfo.java
        </path>
    </paths>
    <msg>
        fixing a bug where the cc: on reply-all doesn't
        get cleared out if the cc field is set to empty.
    </msg>
</logentry>
```

Figure 3. Excerpt of Pooka SVN Log

Requirements: In a previous work [38], we used Prequir [39] to recover requirements for Pooka and SIP. We recovered 90 and 82 functional requirements for both systems respectively.

We used these previously-built requirements to create manually traceability links between requirements and source code. Two of the authors created traceability links and the third author verified all the links to accept or reject them. We used FacTrace to create/verify the manually-built traceability links, which form two oracles, $\text{Oracle}_{\text{Pooka}}$ and $\text{Oracle}_{\text{SIP}}$, of respectively 546 and 949 traceability links for Pooka and SIP and which we use to compute the precisions and recalls of $\text{Trustrace}^{\text{VSM, Histrace}}$ and of the VSM-based approach.

Source Code: We downloaded the source code of Pooka v2.0 and SIP v1.0-draft from their respective SVN repositories. Table I provides descriptive statistics of the two set of source code. We made sure that we could compile and run both systems by setting up the appropriate environments and downloading the relevant libraries before building traceability links.

SVN Change Logs: Figure 3 shows an excerpt of a log message of Pooka. There are $1,743$ and $8,079$ SVN change logs for Pooka and SIP respectively. We perform the data pre-processing steps on both systems' SVN commits with the help of FacTrace. After performing the pre-processing steps, we obtained $1,453$ and $6,289$ SVN commits for Pooka and SIP respectively, because there were many SVN commit messages that did not concern source code files. For example, revision 1604 in Pooka points only to HTML files but for one Java file, `FolderInternalFrame.java`. Therefore, we only kept the Java file and removed any reference to the HTML files. We stored all filtered SVN commit messages and related files in a database.

Document Pre-processing: We extracted all the identifiers from Pooka and SIP requirements, source code, and filtered SVN commit messages using FacTrace. The output of this step are the three corpora that we use for creating traceability links as explained in Sections III and IV.

F. Building Sets of Traceability Links

First, we use a VSM-based approach to create traceability links between requirements and source code, which creates $11,056$ and $79,422$ links for Pooka and SIP, respectively.

Second, we process Pooka and SIP SVN commit messages and requirements to create traceability links between requirements and source through the change logs using our approach Histrace. For each requirement and each filtered SVN commit messages, we extract all the source code files that were committed in SVN with current commit message and then link all recovered source code files directly to the requirement through the SVN commit message.

For example, we trace Pooka requirement "*it should have spam filter option*" to the SVN commit message "*adding prelim support for spam filters*", SVN commit revision number 1133. Then, we recover all the source code files related to this commit, *i.e.*, `SpamSearchTerm.java` and `SpamFilter.java`. Finally, we create a direct traceability link to the files `SpamSearchTerm.java` and `SpamFilter.java` to the requirement "*it should have spam filter option*".

Third, we apply our approach $\text{Trustrace}^{\text{VSM, Histrace}}$ using every traceability links between requirements and source code, both built in the first and second steps. Essentially, if the same traceability link exist between requirement and source code through some SVN commit message, then we marked this link as trustable and counted the number of occurrences of that link. This last step returns $6,329$ and $39,445$ trustable links for Pooka and SIP respectively.

G. Analysis Method

We performed the following analysis on the recovered trustable links to answer our research questions and attempt rejecting our null hypotheses.

We compute the F-measure of the trustable links in comparison to $\text{Oracle}_{\text{Pooka}}$ and $\text{Oracle}_{\text{SIP}}$ to find the optimal λ values for the Equation (3) in the Trustrace model in Section III for Pooka and SIP. Figure 4 shows the graph with different λ values in which the shaded area shows that λ values between 0.9 and 0.5 yield close F-measures. We

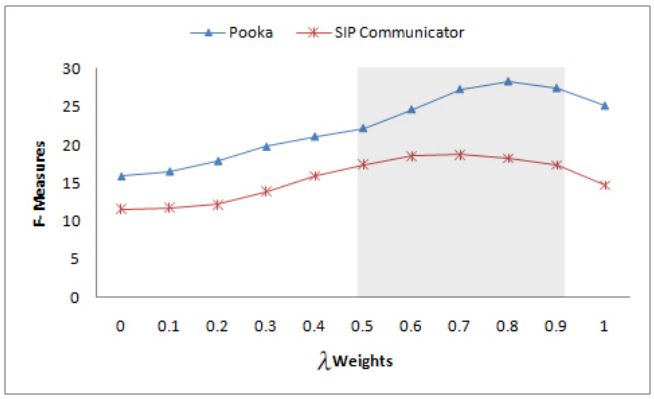

Figure 4. F-measure in function of λ values

favor precision over recall and therefore we select $\lambda = 0.9$ and $\lambda = 0.7$ for Pooka and SIP, respectively.

We use $\text{Oracle}_{\text{Pooka}}$ and $\text{Oracle}_{\text{SIP}}$ to compute the precision and recall values of the VSM and $\text{Trustrace}^{\text{VSM, Histrace}}$ approaches. The VSM approach assigns a similarity value to each and every traceability links, whereas Trustrace uses its model (see Section III-A) to assign trust value to each link. We use a threshold t to prune the set of traceability links, keeping only links whose similarities or trust values are greater than or equal to $t \in [0, 1]$. We use different values of t from 0.01 to 1 per steps of 0.01 to obtain different sets of traceability links with varying precision and recall values, for both approaches. We use these different sets to assess which approach provides better precision and recall values. Then, we use the Mann-Whitney test to assess whether the differences in precision and recall values, , in function of t, are statistically significant between the VSM and $\text{Trustrace}^{\text{VSM, Histrace}}$ approaches. Mann-Whitney is a non-parametric test; therefore, it does not make any assumption about the distribution of the data.

VII. Results

Figure 5 shows the precision and recall values of VSM and $\text{Trustrace}^{\text{VSM, Histrace}}$. It shows that $\text{Trustrace}^{\text{VSM, Histrace}}$ provide better precision values while recall values follow closely the values of the VSM approach. It thus shows that Trustrace improves over VSM results.

Table II shows the average precision, recall, and p-values calculated by comparing the differences between the VSM and $\text{Trustrace}^{\text{VSM, Histrace}}$ approaches. Average precision and recall values for both approaches show that Trustrace increase precision without impacting dramatically recall.

We have statistically significant evidence to reject the H_{01} hypothesis. Table II shows that the p-values for the precision values are below the standard significant value, $a = 0.05$. Trustrace increases up to 12% precision and 3% recall. However, p-values for the differences in recall values are not statistically significant, thus we cannot reject H_{02}.

Thus, we answer the RQ as follow: Trustrace helps to recover more correct links than traditional IR-based approach. Our proposed approaches do not influence the recall of the recovered links statistically.

VIII. Discussion

We now discuss lesson learned from applying our approaches on the case studies and our preliminary experience in using $\text{Trustrace}^{\text{LSI, Histrace}}$ instead of $\text{Trustrace}^{\text{VSM, Histrace}}$.

A. Lesson Learnt

The case studies support our claim that Trustrace combined with IR-based approaches is effective in increasing the precision of requirement traceability while improving slightly recall. Our novel approach performs better than a single IR-based approach.

Automated traceability link recovery approaches use thresholds based on similarity values to accept or reject a link. It is quite possible that, with only similarity values, an expert accepts a false link and–or that she misses a true link with a similarity value below the threshold value. Trustrace uses a trust model to re-assigns similarity values to each link, similarity values based on the trust that the expert puts into each source of information and that helps her to recover links with a better precision.

Thus, Trustrace dramatically reduces the total number of recovered links (see Section VI-F), which allow an expert to verify less traceability links. Our case studies with Pooka and SIP show that as the size of the change log increases, $\text{Trustrace}^{\text{VSM, Histrace}}$ provides better precision (see Table II).

While creating $\text{Oracle}_{\text{Pooka}}$ and $\text{Oracle}_{\text{SIP}}$, we tagged some requirements as unsupported features. While we ran the experiment, we found that Hisrace produced some links to those unsupported features. We manually verified these links and found that the source code related to those features indeed exists. This observation shows that Histrace not only helps to recover traceability links but also may help in evolving traceability links: if an expert creates traceability links and, after some years, wants to update traceability links, then she does not need to create/verify all the links again. She could run Hisrace and–or $\text{Trustrace}^{\text{VSM, Histrace}}$ to obtain possible missing links that she can finally verify.

For example, in Pooka, we declared the drag and drop feature as an unsupported feature. Yet, Histrace produced some traceability links to this requirement. We manually verified and found that developers implemented the drag and drop feature partially but named it “dnd” while writing “this is drag and drop feature” in some SVN commit messages. We looked for all such cases in Pooka and SIP and updated $\text{Oracle}_{\text{Pooka}}$ and $\text{Oracle}_{\text{SIP}}$ and used these oracles in our case studies. This example shows that Trustrace can indeed help experts in recovering from human mistakes and the limitations of single approaches.

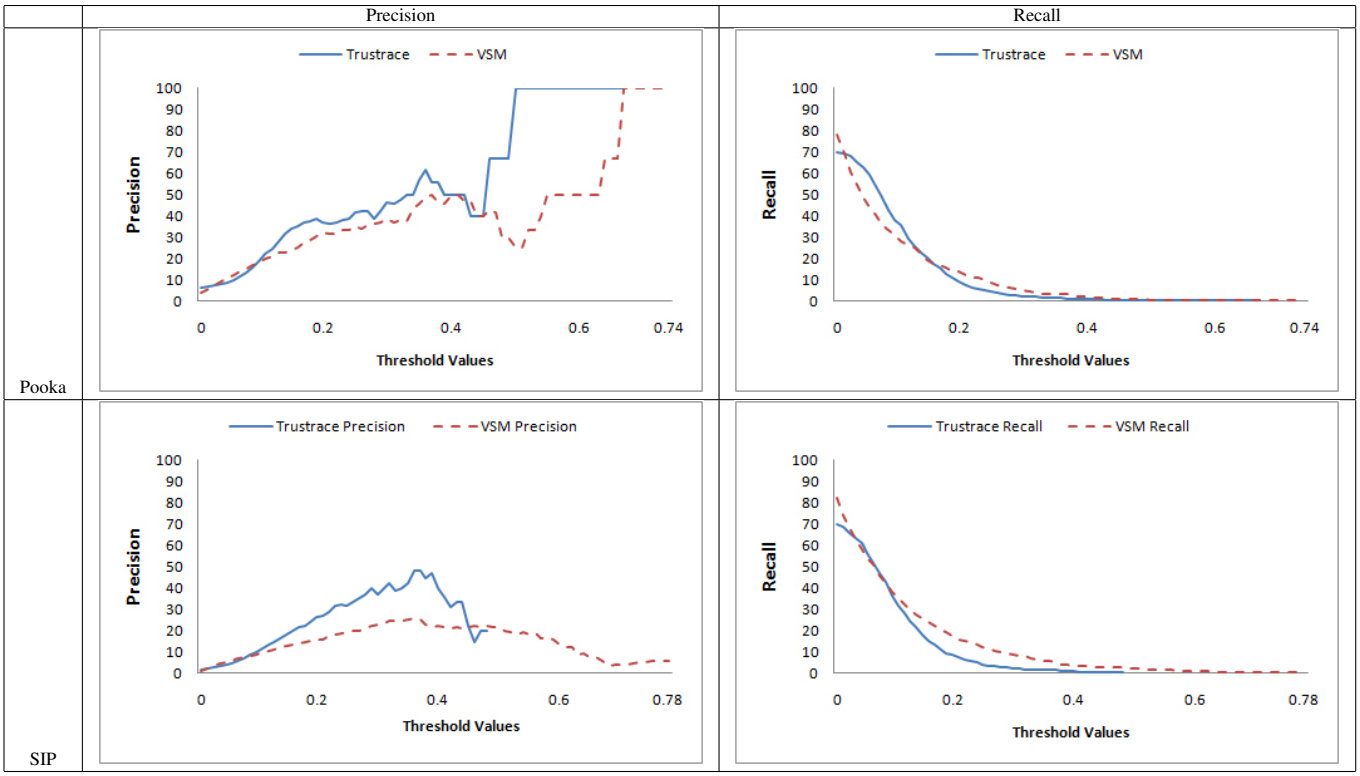

Figure 5. Precision and recall values, from 0.01 to 1 by step 0.01

Table II
POOKA AND SIP RESULTS

	Precision			Recall		
	VSM	**Trustrace**	**p-value**	**VSM**	**Trustrace**	**p-value**
Pooka	42.27864472	54.34682496	0.01821	11.13503614	12.59909245	0.707
SIP Communicator	14.23119197	25.12689933	2.64E-05	13.4203345	16.63373727	0.4894

B. *Trustrace*$^{LSI,\ Histrace}$ *vs. Trustrace*$^{VSM,\ Histrace}$

We also used LSI to create traceability links between requirements and source code. We used $K = 50$ and $K = 200$ to reduce the LSI dimensions for Pooka and SIP, respectively. We then used Trustrace with LSI approach to create trustable links, through Trustrace$^{\text{LSI, Histrace}}$, and found that Trustrace again produced traceability links with better precision and recall values than the LSI approach alone.

However, using different K values for LSI can produce different results. Thus, in future work, we will perform a detailed empirical study based on different K values for LSI and compare them with Trustrace$^{\text{LSI, Histrace}}$ and Trustrace$^{\text{VSM, Histrace}}$.

Trustrace works as an extension to any existing IR-based approach. We cannot claim that Trustrace will perform the same as with the VSM approach: it may increase or decrease precision and recall values. We plan to perform more experiments using various IR-based approaches to study whether Trustrace is effective with all these approaches.

C. Threats to Validity

Several threats potentially impact the validity of our experimental results.

Construct validity: Construct validity concerns the relation between theory and observations. The degree of imprecision of automatic requirement traceability was quantified by means of a manual validation of the precision and recall of the approach, using manually-built oracles. First, two authors created manual traceability oracles and then the third author verified their content to avoid imprecision in the measurements. Moreover, we improved the oracles after applying Trustrace$^{\text{VSM, Histrace}}$ and discovering missing links.

Internal Validity: The internal validity of a study is the extent to which a treatment effects change in the dependent variable. The internal validity of our empirical study could only be threatened by our choice of the λ value: other values could lead to different results. We mitigated this threat by studying the impact of λ on the F-measure of our approach in Section VI-G, see in particular Figure 4.

External Validity: The external validity of a study relates to the extent to which we can generalize its results.

Our case studies are limited to two systems, Pooka and SIP. Yet, our approach is applicable to any other systems. However, we cannot claim that the same results would be achieved with other systems. Different systems with different SVN change logs, SVN commit messages, requirements, and source code may lead to different results. However, the two selected systems have different SVN change logs, SVN commit messages, requirements, and source code quality. Our choice reduces the threat to the external validity.

Conclusion validity: Conclusion validity threats deals with the relation between the treatment and the outcome. The appropriate non-parametric test, Mann-Whitney, was performed to statistically reject the null-hypotheses, which does not make any assumption on the data distribution.

IX. Conclusion and Future Work

Information Retrieval approaches [12] have proven useful in recovering traceability links. We proposed two contributions to improve the precision and recall of traceability links.

First, we proposed a novel approach, Trustrace, inspired by Web trust models [19], [20], [21], [22] to improve precision and recall of traceability links: Trustrace uses any traceability recovery approach as the basis on which it applies various experts' opinions [23] to remove and–or adjust the rankings of the traceability links. The experts can be human experts or other traceability recovery approaches.

Second, we proposed Histrace, an expert supporting the identification of traceability links between requirements and source code through CVS/SVN change logs, using a VSM. Histrace uses CVS/SVN commit messages to build traceability links between high-level documentation and source code entities, observing that messages are tied to changed entities and, thus, can be used to infer traceability links between high-level documentation and source code entities.

We combined a VSM-based approach with Histrace to build Trustrace$^{\text{VSM, Histrace}}$ in which we use Histrace as one expert commenting the traceability links recovered using the VSM-based approach. We applied Trustrace$^{\text{VSM, Histrace}}$ on Pooka and SIP to compare its traceability links with those recovered using only the VSM-based approach, in terms of precision and recall. We showed that Trustrace$^{\text{VSM, Histrace}}$ improves with statistical significance the precision of the traceability links without decreasing recall.

We thus showed that our trust-based approach indeed improves precision and recall and also that CVS/SVN change logs are useful in the traceability recovery process. In future work, we plan more experiments with other combination of IR-based approaches to further improve the precision and recall values. We will also perform more experiments on heterogeneous software artefacts to measure the usefulness of these other artefacts for a traceability recovery process. In particular, we are currently building new traceability approaches using bug reports and mailing lists.

X. Acknowledgment

This work has been partially supported by the NSERC Research Chairs on Software Cost-effective Change and Evolution and on Software Patterns and Patterns of Software.

References

[1] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vries, "Moving into a new software project landscape," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 275–284.

[2] N. Pennington, *Comprehension Strategies in Programming. In: Empirical Studies of Programmers: Second Workshop. G.M. Olsen S. Sheppard S. Soloway eds.* Englewood Cliffs, NJ: Ablex Publisher Nordwood NJ, 1987.

[3] ——, "Stimulus structures and mental representations in expert comprehension of computer programs," *Cognitive Psychology*, vol. 19, pp. 295–341, 1987.

[4] R. Brooks, "Towards a theory of the comprehension of computer programs," *International Journal of Man-Machine Studies*, vol. 18, pp. 543–554, 1983.

[5] E. Soloway and K. Ehrlich, "Empirical studies of programming knowledge," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, pp. 595–609, 1994.

[6] A. V. Mayrhauser and A. Vans, "From program comprehension to tool requirements for an industrial environment," in *Proceedings of IEEE Workshop on Program Comprehension*. Capri Italy: IEEE Comp. Soc. Press, 1993, pp. 78–86.

[7] ——, "Dynamic code cognition behaviours for large scale code," in *Proceedings of IEEE Workshop on Program Comprehension*. Washington DC USA: IEEE Comp. Soc. Press, 1994, pp. 74–81.

[8] A. V. Mayrhauser and A. M. Vans, "Identification of dynamic comprehension processes during large scale maintenance," *IEEE Transactions on Software Engineering*, vol. 22, no. 6, pp. 424–437, 1996.

[9] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behaviour: A model and experimental results," *IJCIS*, vol. 8, no. 3, pp. 219–238, Mar 1979.

[10] I. Vessey, "Expertise in debbugging computer programs: A process analysis," *IJMMS*, vol. 23, pp. 459–494, 1985.

[11] A. De Lucia, M. Di Penta, R. Oliveto, and F. Zurolo, "Improving comprehensibility of source code via traceability information: a controlled experiment," in *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on*, 2006, pp. 317 –326.

[12] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

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

[14] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of 25th International Conference on Software Engineering*. Portland Oregon USA: IEEE CS Press, 2003, pp. 125–135.

[15] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, March 2003.

[16] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. D. Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," in *ICPC*, 2010, pp. 68–71.

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

[18] J. H. Hayes, A. Dekhtyar, S. Sundaram, and S. Howard, "Helping analysts trace requirements: An objective look," in *Proceedings of IEEE Requirements Engineering Congerence (RE) 2004*, Kyoto Japan, Sept 2004, pp. 249–261.

[19] R. Berg and J. M. L. Van, "Finding symbolons for cyberspace: addressing the issues of trust in electronic commerce," *Production Planning and Control*, vol. 12, pp. 514–524(11), 2001.

[20] D. H. McKnight, V. C., and C. K., "The impact of initial consumer trust on intentions to transact with a web site: a trust building model," *The Journal of Strategic Information Systems*, vol. 11, no. 3-4, pp. 297 – 323, 2002.

[21] J. W. Palmer, J. P. Bailey, and S. Faraj, "The role of intermediaries in the development of trust on the www: The use and prominence of trusted third parties and privacy statements," *Journal of Computer-Mediated Communication*, vol. 5, no. 3, 2000.

[22] K. M. and H. William, "The development of initial trust in an online company by new customers," *Information & Management*, vol. 41, no. 3, pp. 377 – 397, 2004.

[23] D. Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 420–432, 2007.

[24] K. Kontogiannis, R. D. Mori, R. Bernstein, M. Galler, and E. Merlo, "Pattern matching for design concept localization," in *Second Working Conference on Reverse Engineering*, Toronto Canada, July 1995.

[25] N. Wilde and C. Casey, "Early field experience with software reconnaissance technique for program comprehension," in *Proceedings of IEEE Working Conference on Reverse Engineering*, 1996.

[26] V. Kozaczynski, J. Q. Ning, and A. Engberts, "Program concept recognition and transformation," *IEEE Transactions on Software Engineering*, vol. 18, no. 12, pp. 1065–1075, Dec 1992.

[27] M. Eaddy, A. Aho, G. Antoniol, and Y.-G. Guéhéneuc, "Cerberus: Tracing requirements to source code using information retrieval dynamic analysis and program analysis," in *ICPC '08: Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*. Washington DC USA: IEEE Computer Society, 2008, pp. 53–62.

[28] L. Dapeng, M. Andrian, P. Denys, and R. Vaclav, "Feature location via information retrieval based filtering of a single scenario execution trace," in *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York NY USA: ACM, 2007, pp. 234–243.

[29] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, 2008, pp. 103 –112.

[30] W. B. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[31] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang, "Sniafl: Towards a static noninteractive approach to feature location," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, pp. 195–226, April 2006.

[32] H. Kagdi, J. Maletic, and B. Sharif, "Mining software repositories for traceability links," in *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on*, June 2007, pp. 145 –154.

[33] H. Kagdi and J. Maletic, "Software repositories: A source for traceability links," in *International Workshop on Traceability in Emerging Forms of Software Engineering (GCT/TEFSEâ07)*, 2007, pp. 32 –39.

[34] H. Kagdi, S. Yusuf, and J. I. Maletic, "Mining sequences of changed-files from version histories," in *Proceedings of the 2006 international workshop on Mining software repositories*, ser. MSR '06, New York, NY, USA, 2006, pp. 47–53.

[35] D. Artza and Y. Gil, "A survey of trust in computer science and the semantic web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 58 – 71, 2007.

[36] T. Grandison and M. Sloman, "A survey of trust in internet applications," *Communications Surveys Tutorials, IEEE*, vol. 3, no. 4, pp. 2 –16, 2000.

[37] W. Wanga, G. Zenga, and D. Tang, "Using evidence based content trust model for spam detection," *Expert Systems with Applications*, vol. 37, no. 8, pp. 5599 – 5606, 2010.

[38] N. Ali, W. Wu, G. Antoniol, M. D. Penta, Y.-G. Guéhéneuc, and J. H. Hayes, "A novel process and its implementation for the multi-objective miniaturization of software," Ecole Polytechnique de Montreal, Tech. Rep. EPM-RT-2010-04, 2010, technical Report.

[39] J. H. Hayes, G. Antoniol, and Y.-G. Guéhéneuc, "Prereqir: Recovering pre-requirements via cluster analysis," in *Reverse Engineering, 2008. WCRE '08. 15th Working Conference on*, Oct 2008, pp. 165 –174.