# Traceability Recovery by Modeling Programmer Behavior

Giuliano Antoniol*, Gerardo Casazza**, Aniello Cimitile*
antoniol@ieee.org gec@unisannio.it cimitile@unisannio.it

(*)University of Sannio, Faculty of Engineering - Piazza Roma, I-82100 Benevento, Italy
(**)University of Naples "Federico II", DIS - Via Claudio 21, I-80125 Naples, Italy

**Keywords: redocumentation, traceability, program comprehension, object orientation**

## Abstract

*When a system evolves, while the source code is changed documentation and traceability links are barely ever updated: maintaining traceability information between software artifacts is a costly and tedious activity frequently sacrificed during development and maintenance due to market pressure.*

*This paper presents a new method to recovery traceability links between high level and low level artifacts. The method is based on the partial knowledge of a subset of traceabilty links. It can be fully automated and the human intervention is only required to confirm or confute recovered traceability links.*

*The method has been applied to software written in Java, to trace classes onto functional requirements, experimental result demonstrate the superiority of the novel method over the previously published results on the same system.*

## 1. Introduction

It is widely recognized that a program must evolve to meet ever changing user needs [9]. Several evolution driving factors may be identified: new functionalities added, lack of software quality, lack of overall system performance or software portability (on new software and hardware platforms), market opportunities are just a few examples.

Quite often, while source code evolves documentation and traceability links are not updated: maintaining consistency and traceability information between software artifacts is a costly and tedious activity frequently sacrificed due to market pressure. Furthermore, since outsourcing has been widely adopted as common practice, people who wrote or maintained a system may no longer be available. Hence, the only reliable source of information about a software system is often the system itself.

The research reported in this paper addresses the problem of recovering traceability links between the free text documentation associated with the development and maintenance life cycle of a software system and its source code.

Traceability links recovery has not been extensively investigated, very few contributions were published in the past 20 years. A cornerstone paper was written in 91 by Maarek and others [10], the paper relies on information retrieval techniques for automatically assembling software libraries based on a free text indexing scheme. This approach uses attributes automatically extracted from natural language IBM RISC System/6000 AIX 3 documentation to build a browsing hierarchy which accepts queries expressed in natural language. Other approaches such as TOOR [13], IBIS [6], and REMAP [14] are few examples of software development tools able to build and maintain traceability links among various software artifacts.

The method proposed in this paper shares with the previous work [10] the general framework based on information retrieval. More commonalities can be found with [1, 2, 3] where to recover traceability links probabilistic modeling was coupled with computational linguistic tools such as stopper and stemmer, i.e. morphological analyzers. The key idea of those approaches is that the application-domain knowledge processed by programmers when writing the code is captured by mnemonic identifiers. Albeit the conjecture is strongly supported by the published results, available knowledge (e.g., on known traceability links) is disregarded; furthermore, a certain degree of manual intervention is required: quite often program item names are obtained with complex rules from application-domain concepts e.g., concatenation of two or more words with vowel deletions or substring interposition.

This paper overcomes the limitation outlined above: a new method to recover traceability links is presented. The method can be fully automated and human intervention is only required to confirm or confute automatically recovered traceability links. The underlying assumption is that programmers tend to process application-domain knowledge in

Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)
1095-1350/00 $10.00 © 2000 IEEE

a consistent way when writing code: program item names of different code regions related to a given text document are likely to be the same or at least very similar. Under the above assumption, the knowledge of existing traceability links can be exploited; once programmer behavior can be modeled, no matter where the knowledge comes from, few links suffice to allow the method bootstraping itself.

Stochastic modeling and Bayesian classification were exploited to model programmers as stochastic communication channels: programmers process high level documentation and domain concepts producing low level artifacts (e.g. program item names). Once a subset of traceability links is available (i.e couples high level/low level artifacts), for any given link the joint probability distribution of the text document (words) and a set of linked areas of code (program item names) is estimated together with the document words marginal probability distributions. The estimated probability distributions are used in a Bayesian classifier to score sequence of mnemonics extracted from a not yet *classified* area of code (i.e., area of code not belonging to the subset of known traceability links). Higher scores suggest the existence of links between the area of code from which a particular sequence of mnemonics is extracted and the document that generated the marginal probability distribution.

The method was applied to a hotel management system, named ALBERGATE, to recover traceability links between the functional requirements, as specified in the requirement document, and the Java source code. The goal was to bind each functional requirement to the classes where it is implemented. Results demonstrate the advantage of exploiting available traceability links knowledge even with no morphological analysis and human intervention. While previously presented approaches on the same system requires morphological analysis and human intervention to obtain a 50 % best match recall, the new method gives a figure of 60 % once stop-words are automatically removed. If morphological analysis is applied an even better figure is obtained (i.e., 64 %), however, not surprisingly, while in [2, 3] morphological analysis greatly improves the results, it only slightly improves the new method results.

The paper novelties and contributions can thus be summarized in a novel method and the related equations presented; the method takes advantage of the existing knowledge and could be fully automated. Experimental evidence is provided that the method works fine with the available data, and it overcomes previously published results on the same system.

The remaining of the paper is organized as follows: Section 2 introduces the mathematical model. The validation and assessment methodology is discussed in Section 3. Section 4 presents the tools developed to implement the method. While Section 5 is devoted to experimental results. Lesson learned issues are addressed in Section 6. Finally, Section 7 summarizes the work and gives some concluding remarks.

## 2. Mapping Equations

Stochastic models are used extensively in several areas: automatic speech recognition, machine translation, spelling correction, text compression, etc [5, 8, 15].

In the software development/maintenance life cycle programmers may be thought of as stochastic channels transforming a high level text document onto *observations* i.e., chunks of code. More precisely, development/maintenance activity is modeled by a stochastic channel where the sequence of words (an high level text document) $W_k$, generated by a source $S$ with the *a-priori* probability $Pr(W_k)$, is transmitted through a channel and transformed into the sequence of observations O with probability $Pr(O \mid W_k)$. In the traditional pattern recognition applications, O could represent the acoustic signal produced by uttering $W_k$, the translation of $W_k$ from Italian to English, a typewritten version of $W_k$ with possible mis-spellings. In the software engineering domain, to model development/maintenance activities $W_k$ may represent a functional requirement or a maintenance change request while $O$ may correspond to the program item names i.e., the mnemonics for code identifiers chosen by developers/maintainers. Notice that any word of the $W_k$ document (as well as of the other documents) will, possibly, influence the entire observation $O$.

The method's underlying assumption is that programmers tend to build code identifiers (either variable of function name) processing, with a set of unknown rules, domain and problem knowledge. Those unknown rules are not expected to change sharply over the time and from one programmer to the other in that rules depend on the corporate culture coding standard and the programmer skill. Hence the program item names of different code regions are likely to share semantic meaning with the higher level concepts, or at least they have been created by applying consistently the set of unknown rules. Thus program item names of different code regions related to a given high level text document are likely to be the same or at least very similar.

The problem is decoding the observation O, e.g., program item names, into the original high level document $W_k$: that is, finding $\hat{W}_k$ that maximizes the *a-posteriori* probability $Pr(W_k \mid O)$. Applying the Baye's rule, the following identity is obtained:

$$Pr(W_k \mid O) = \frac{Pr(O \mid W_k) Pr(W_k)}{Pr(O)} \qquad (1)$$

The probability $Pr(W_k)$ is the high level document *a-priori* probability, since there is no reason to believe that an high level text document is more likely than any other (i.e.,

there is no *a-priori* information on the document distribution), it can be safely assumed that documents are equally likely; furthermore, in the above equation, $Pr(O)$ is a constant with respect to $k$ and could be eliminated. Hence, decoding is equivalent to find:

$$\hat{W_k} = arg \max_{W_k} Pr(O \mid W_k) \quad (2)$$

Let us assume an high level text document be represented as $W_k = \bigcup_{i=1}^{n} \{w_{k,i}\}$; the document is composed of n words: $w_{k,1}, \ldots, w_{k,n}$ assumed to be independent events thus:

$$Pr(w_{k,1}, \ldots, w_{k,n}) = \prod_{i=1}^{n} Pr(w_{k,i}) \quad (3)$$

The high level document $W_k$ is transformed by programmers into an observation $O = \bigcap_{j=1}^{m} \{o_j\}$ corresponding to a low level artifact (e.g., the identifier collection extracted from a chunk of source code). Having observed $\{o_1 \bigcap o_2 \ldots \bigcap o_m\}$ low level artifact representation, traced into $\{w_{k,1} \bigcup w_{k,2} \ldots \bigcup w_{k,n}\}$ high level document, it has to be computed:

$$Pr(O|W_k) = Pr(\bigcap_{j=1}^{m} o_j | \bigcup_{i=1}^{n} w_{k,i}) \quad (4)$$

Further assume $\{o_1\} \bigcap \{o_2\} \ldots \bigcap \{o_m\}$ conditionally independent from $W_k$ i.e., dependencies between $\{o_p\}, \{o_q\}$ are modeled by the conditioned probability of events $\{o_p|W_k\}, \{o_q|W_k\}$ the above equation can be rewritten as:

$$Pr(O|W_k) = \prod_{j=1}^{m} Pr(o_j|W_k) \quad (5)$$

In other words each $o_j$ depends on the entity $W_k$. The conditioned probability $Pr(o_j|W_k)$ can be factored as:

$$Pr(o_j|W_k) = \frac{Pr(o_j W_k)}{Pr(W_k)} = \frac{Pr(o_j \bigcap (\bigcup_{i=1}^{n} w_{k,i}))}{Pr(W_k)}$$

The event $o_j \bigcap (\bigcup_{i=1}^{n} w_{k,i})$ can be rewritten as $\bigcup_{i=1}^{n} (o_j w_{k,i})$ and since $w_{k,i}$ are assumed independent events:

$$Pr(o_j|W_k) = \sum_{i=1}^{n} \frac{Pr(o_j|w_i) Pr(w_{k,i})}{Pr(W_k)} \quad (6)$$

As in equation (1) the probability $Pr(W_k)$ corresponds to the *a-priori* high level document probability and can be dropped. By substituting equation (6) into equation (5) the following expression of $Pr(O|W_k)$ is obtained:

$$Pr(O|W_k) = \prod_{j=1}^{m} \sum_{i=1}^{n} Pr(o_j|w_{k,i}) Pr(w_{k,i}) \quad (7)$$

Equations (2) and (7) are central to the method, they provide a means to effectively recover traceability links based on the *a-priori* knowledge of a subset of traceability links.

The involved probabilities $Pr(o_j|w_{k,i})$ and $Pr(w_{k,i})$ need to be estimated on a *labeled* training corpus i.e., a subset of high level document and code fragments known to belong to treaceability relations. Given the labeled training corpus the probabilities may be approximated with frequencies:

$$Pr(o_j|w_{k,i}) \simeq \frac{c(o_j w_{k,i})}{c(w_{k,i})} \; i = 1 \ldots n \; j = 1 \ldots m \quad (8)$$

$$Pr(w_{k,i}) \simeq \frac{c(w_{k,i})}{|W_k|} \quad (9)$$

Where $c(h)$ is the number of times that the $h$ word appears in the texts, in the same way $c(hl)$ is the number of times that the couple $h, l$ appears ($h$ in the observation $O$ and $l$ in the document $W_k$) while $|W_k|$ is the number of words in $W_k$ (i.e. $|\ |$) gives the set cardinality).

## 3. Method Validation

Feasibility study and method assessment were performed by recovering traceability links between requirements and source code of a software system developed following the Boehm waterfall model [4]. For this system all the prescribed documents were available (e.g., requirement documents, design documents, test cases, etc.).

The software system, namely ALBERGATE, was developed in Java by a team of final year students at University of Verona (Italy). ALBERGATE is a software system designed to implement all the operations required to administrate and manage a small/medium size hotel (room reservation, bill calculation, etc.). The system has been developed from scratch on the basis of 16 functional requirements specified (as well as all the other system documentation) in Italian language. ALBERGATE exploits a relational database for most of the operations, therefore the system size is relatively low (about 20 KLOC with 95 classes). Non functional requirements as well as other documentation such as user manuals, UML use cases, architecture and detailed design were not used in the present study. The recovery process was focused on the 60 classes implementing the user interface of the software system.

Equation (2) was used in a ranked retrieval fashion; ranked retrieval returns a ranked list of documents: documents are ranked according to the probability of being relevant to the user query. ALBERGATE functional requirements, in the present experiment, play the role of high level documents, equation (2) term $W_k$, while the set of identifiers extracted from a given class was considered to be the observation $O$ i.e., the query.

### 3.1. Model generation

Probabilities estimation was based on the term frequency: $Pr(w_{k,i})$ is estimated from $c(w_{k,i})$, the number of times in which each word appears in a given document. Using the simple term frequency would turn the product $\prod_{j=1}^{m} Pr(o_j|W_k)$ to zero, whenever an observation $o_j$ was not present in the training set of the document $W_k$.

This problem, known as the zero-frequency problem [15], can be avoided using different approaches (see [11]). Basically, it has to be decided whether or not the *vocabulary* is closed. Closed vocabulary means that all the words in the text document collection (set of $W_k$) and in all possible observations are known. On the contrary, open dictionary means that a *dummy* word, accounting for all unforeseen events, is introduced.

While in other fields such as in spoken dialogue with computers open vocabulary approaches are extremely appealing, closed vocabulary does not constitute a limitation in software engineering. The adopted approach consists of a closed vocabulary where probabilities were smoothed according the the shift-$\beta$ smoothing techniques. Consider the equation (7), $Pr(w_{k,i})$ term it may be estimated with:

$$Pr(w_{k,i}) = \begin{cases} \frac{c_i - \beta}{|W_k|} + \lambda & \text{if } w_i \text{ occurs in } W_k \\ \lambda & \text{otherwise} \end{cases}$$

where $c_i$ is the number of occurrences of words $w_i$ in the document $W_k$. Being $w_i$ a generic word of the closed vocabulary it may happen that it never shows up in $W_k$ such in a case $Pr(w_{k,i})$ is set to $\lambda$. The interpolation term $\lambda$ is:

$$\lambda = \frac{n}{|W_k| * V}\beta$$

where $V$ is the size of the vocabulary and $n$ is the number of different words of the vocabulary occurring in the document $W_k$. The value of the parameter $\beta$ was chosen according to [12] as follows:

$$\beta = \frac{n(1)}{n(1) + 2 * n(2)}$$

where $n(j)$ is the number of words occurring $j$ times in the document $W_k$. More details and a complete overview on different smoothing techniques and probability estimations can be found in [11].

### 3.2. Model assessment

As already stated the method was conceived to take advantage of the already available knowledge in that

| Requirement | #Classes | Overlap |
|---|---|---|
| Req. 1 | 10 | Req. 9 (4) |
| Req. 2 | 4 | None |
| Req. 3 | 2 | None |
| Req. 4 | 4 | None |
| Req. 5 | 6 | None |
| Req. 6 | 7 | None |
| Req. 7 | 2 | None |
| Req. 8 | 3 | None |
| Req. 9 | 4 | Req. 1 (4) |
| Req. 10 | 2 | None |
| Req. 11 | 7 | None |
| Req. 12 | 2 | Req. 13 (1) |
| Req. 13 | 2 | Req. 12 (1) |
| Req. 14 | 1 | None |
| Req. 15 | 1 | None |
| Req. 16 | 1 | None |

**Table 1. ALBERGATE traceability matrix summary.**

$Pr(o_j|w_{k,i})$ encompasses the information on the known subset of traceability links. Table 1 summarizes the ALBERGATE traceability matrix as compiled by the original developers: three requirements are implemented by a single class; all the other functional requirements are traced into two or more classes. Notice that Req. 1, Req. 9, Req. 12 and Req. 13 implementing classes have no null intersection with other requirement implementation (e.g., 4 classes are common between Req. 1 and Req. 9 and 1 class is common between Req. 12 and Req. 13). Thus, classes implementing a given requirement can be divided in two sets: a training set and a test set.

Experiments were designed in such a way that training and test sets do not overlap, this means that for any given requirement the set classes implementing it has to be divided into two distinct sets. In particular, the following sets were defined

- $CS_i$ the set of software ClaSses implementing the $i-th$ requirement;
- $TR_i$ the Training Set related to the $i-th$ requirement, $TR_i$ *classes* were used to estimate the probabilities expressed by (8) and (9), notice that $TR_i \subseteq CS_i$;
- $TS_i$ the Test Set related to the $i-th$ requirement ($TR_i \cap TS_i = \emptyset$), $TS_i$ *classes* were used as queries to evaluate the (2), notice that $TS_i \subset CS_i$;
- $ES_l$ the set of couples $(TR_{il}, TS_{il})$ related to the $l-th$ experiment.

An exhaustive cross-validation on the ALBERGATE software system was not feasible: there are millions of different and consistent $ES_l s$. For this reason two classes of experiments were designed. The first class of experiments was aimed to assess the peak performance: all classes but one (used as query) were used as training set. The probabilities (8) and (9) were thus estimated using the largest possible amount of information. In the second class of experiments, $ES_l s$ have been chosen mimicking incremental reconstruction from scratch of traceability links. More precisely each $ES_l$ in the first class of experiments has been chosen according to the constraints:

$$\begin{cases} |TS_{jl}| = 1 \quad TR_{jl} = CS_{jl} - TS_{jl} \quad j = 1 \dots 16 \\ TS_{il} = \emptyset \quad\quad TR_{il} = CS_{il} - TS_{jl} \quad \forall i \neq j \;\; i = 1 \dots 16 \end{cases}$$

Given the traceability matrix compiled by the original developers there are 50 different $ES_l$ verifying the above constraints: $|TR_{il}| \neq 0$ and $|TS_{il}| = 1$.
In the second family of experiment, the $l-th\ ES_l$ has been chosen according to the constraints:

$$\begin{cases} |TR_{jl}| = min(s, |CS_{jl}|) \quad j = 1 \dots 16 \quad s = 1 \dots 5 \\ TS_{jl} = CS_{jl} - TR_{jl} - \bigcup_{i=1 \wedge i \neq j}^{16} TR_{il} \end{cases}$$

and given $ES_m$ and $ES_n$ with $m \neq n$:

$$TR_{jm} \neq TR_{jn} \quad \wedge \quad TS_{jm} \neq TS_{jn}$$

Once the size of the training set is fixed (e.g., $s$ parameter) the size of the test set is derived according to the above equations. Notice that the number of queries vary with $s$, since there is a variable number of couples $(TR_{il}, TS_{il})$ such that $|TS_{il}| > 0$.

*Precision* and *recall* [7] were used to assess the method results and compare them with previous works [2, 3]. *Recall* is the ratio of relevant documents retrieved for a given query over the number of relevant documents for that query in the "database". *Precision* is the ratio of the number of relevant documents retrieved over the total number of documents retrieved. When applying equation (2) the most relevant index is precision: a fixed number of models were trained and thus a fixed number of document retrieved.

## 4. Tool Support

A number of tools have been developed to automate the method. A top-down recursive parser was developed to analyze Java source code. Once the parse tree is available it is traversed and the required information is stored in support files. For each class, comments, identifiers of attributes, methods as well as method parameters are stored in separate files. For the present study comments were disregarded: the entire traceability link recovery process relies on the attributes, methods and parameters of each class.

A minimal tool suite to assist textual information processing for the Italian language was also developed. In particular, text processing is performed in two steps. The first step is completely automated, it splits in two or more words strings containing the underscore, capital letters are downcased and stop words (e.g., articles, language feature) are removed. The second step is semi-automatic: a tool with spelling facilities prompts the software engineer with the words that might be separated, leaving the final decision to the user. Moreover, a semi-automatic stemmer, based on thesaurus facilities, has been implemented that helps the software engineer to reconduct flexed words to their roots (e.g. verbs are reconducted to their infinite form [1].

Finally, we have implemented the estimation of probabilities $Pr(o_j|w_{k,i})$, $Pr(w_{k,i})$ with the shift-$\beta$ smoothing and closed vocabulary.

| | $DS1$ | | $DS2$ | |
|---|---|---|---|---|
| | Rec. | Prec. | Rec. | Prec. |
| Best 1 | 60% | 54% | 64% | 58% |
| Best 2 | 74% | 33% | 74% | 33% |
| Best 3 | 84% | 25% | 84% | 23% |

**Table 2. Peak Method Performance.**

## 5. Experimental Results

The experiments were carried out using two different data sets: $DS1$ and $DS2$. The data sets have been obtained by applying the same linguistic processing to the requirements and to the program items (i.e. class identifiers). Linguistic processing was divided in two steps as described in the previous Section: in the first step stop word removal and minor text normalization were automatically performed, this leads to $DS1$. Then morphological analysis, supported by human intervention, produced $DS2$ (i.e. all plurals were reconducted into singular and all flexed verbs were transformed into infinitive form).

The simplest way to trace software classes onto requirements is to use the `grep` UNIX command; the search can be done at least in two ways. In the first approach each class identifier is used as the string to be searched into the files of requirements; this give rise to about 4800 queries 94% of which give empty result. The second approach considers the *or* of the class identifiers, in such a case 94% of the classes were traced onto 10 or more requirements. Even worse the `grep` approach did not offer any way to rank the retrieved requirements. From a practical point of view this

[1] It is worth noting that this step is much more difficult when applied to languages such as Italian, as in the case of the documentation of the case study presented in this paper.

| | $DS2$ | |
|---|---|---|
| | Rec. | Prec. |
| Best 1 | 50% | 54% |
| Best 2 | 70% | 38% |
| Best 3 | 77% | 28% |

**Table 3. Previously published results on the same data set.**

means that the maintainer has to examine a large number of candidates with the same priority. `grep` baseline results were thus judged quite unsatisfactory.

| | Best 1 | | Best 2 | | Best 3 | |
|---|---|---|---|---|---|---|
| $s$ | Rec. | Var. | Rec. | Var. | Rec. | Var. |
| 1 | 33% | 0.011 | 47% | 0.007 | 56% | 0.007 |
| 2 | 50% | 0.006 | 63% | 0.006 | 70% | 0.005 |
| 3 | 52% | 0.008 | 64% | 0.013 | 70% | 0.011 |
| 4 | 52% | 0.012 | 64% | 0.016 | 70% | 0.015 |
| 5 | 52% | 0.043 | 68% | 0.043 | 73% | 0.029 |

**Table 4. Results obtained with the current model on $DS1$**

Table 2 summarizes the results of the first class of experiments aimed to evaluate the peak method performance. The row labeled with *Best 1* refers to recall and precision achieved by the top ranked requirement; the row labeled with *Best 2* refers to recall and precision achieved considering both the first and the second best scored requirements, and so on. 50 different queries were performed, thus the number of top ranked correctly associated requirements (over 50) give raise to the recall figure, while precision denominator is 55. $Best2$ and $Best3$ figures were computed accordingly.

For sake of comparison the previous results on the same data set published in [2] were summarized in Table 3, by comparing those results with the new result of Table 2 the advantages of the new method are evident. The current results significatively improve the recall: the recall is improved on both the data sets *DS1* and *DS2*. This represents an important achievement since *DS1* is automatically produced. In other words, the proposed method overwhelmed the previously obtained best results, best results obtained with a certain degree of human intervention. The analysis of precision is more difficult: the precision obtained considering the first candidate ($Best1$) on $DS1$ did not decrease, on the other hand the precision obtained considering both the first and the second candidates ($Best2$) and the one considering the first, the second and the third candidates ($Best3$) slightly decrease. Notice that these results are globally positive; in a foreseeable application of the presented method a maintainer will focus on the first candidate, once the top ranked traceability link is confirmed or confuted probabilities will be re-estimated and a new ranked list proposed to the maintainer. Moreover the precision decrease is restrained and is balanced by the recall increase.

Tables 4, 5, 6 and 7 show the results related to the second class of experiments, with these experiments the incremental reconstruction of the traceability links was simulated. The tables report the average values of precision and recall and the variances (computed over 100 randomly generated experiments) at three different levels of retained top ranked candidates.

| | Best 1 | | Best 2 | | Best 3 | |
|---|---|---|---|---|---|---|
| $s$ | Rec. | Var. | Rec. | Var. | Rec. | Var. |
| 1 | 33% | 0.081 | 45% | 0.005 | 53% | 0.006 |
| 2 | 46% | 0.005 | 60% | 0.007 | 68% | 0.007 |
| 3 | 51% | 0.009 | 65% | 0.008 | 72% | 0.006 |
| 4 | 52% | 0.018 | 64% | 0.017 | 73% | 0.010 |
| 5 | 55% | 0.027 | 67% | 0.024 | 76% | 0.017 |

**Table 5. Results obtained with the current model on $DS2$**

| | Best 1 | | Best 2 | | Best 3 | |
|---|---|---|---|---|---|---|
| $s$ | Prec. | Var. | Prec. | Var. | Prec. | Var. |
| 1 | 35% | 0.011 | 23% | 0.019 | 18% | 0.0008 |
| 2 | 50% | 0.006 | 31% | 0.001 | 23% | 0.0005 |
| 3 | 52% | 0.088 | 32% | 0.003 | 23% | 0.001 |
| 4 | 52% | 0.012 | 32% | 0.004 | 23% | 0.001 |
| 5 | 52% | 0.021 | 34% | 0.006 | 24% | 0.002 |

**Table 6. Results obtained with the current model on $DS1$**

The most relevant result of this second class of experiments is the empirical evidence that the system learns. For both data sets *DS1* and *DS2*, as the $s$ value increases (i.e. more training material is used) the recall and the precision increase as well. This trend clearly indicates that augmenting the training information improves the traceability recovery process. This supports the hypothesis that for this software system programmers tend to process application-domain knowledge in a consistent way when writing code. If program names were randomly chosen, adding new training material will not improve Tables 4, 5, 6, and 7. This is a key result: the major limitation of previous experiment [2] was the fact that it can satisfactory work if and only if

there is a significant intersection between the words used in the requirements and the source code identifiers i.e., if the input and the output vocabulary has no null intersection. For this reason the results presented in Tables 4, 5, 6 and 7 are encouraging even if the recall and precision average values are not always greater than [2].

| | Best 1 | | Best 2 | | Best 3 | |
|---|---|---|---|---|---|---|
| $s$ | Prec. | Var. | Prec. | Var. | Prec. | Var. |
| 1 | 33% | 0.008 | 22% | 0.001 | 17% | 0.0007 |
| 2 | 46% | 0.005 | 30% | 0.001 | 22% | 0.0008 |
| 3 | 52% | 0.009 | 32% | 0.002 | 24% | 0.0007 |
| 4 | 52% | 0.018 | 32% | 0.004 | 24% | 0.001 |
| 5 | 55% | 0.027 | 34% | 0.006 | 24% | 0.001 |

**Table 7. Results obtained with the current model on $DS2$**

The analysis of Tables 4, 5, 6 and 7 highlights that when $s$ varies from 1 to 3 the results obtained on $DS1$ are better then the ones obtained on $DS2$. This apparently contrasts with some our previous results [2]. However, this phenomenon may be explained taking into account that the morphological analyzer reduces the number of different words and consequently reduces the the dimension of the training corpus as well. This is the reason why when the cardinality of $TS_{ik}$ is lower or equal to 3 the results on $DS1$ are better. Increasing the value of s (e.g. $s = 4, 5$) adds enough material to the training set, thus the normalization effect of the morphological analyzer adds an appreciable contribution.

## 6. Lessons Learned

While implementing the equations, training the models and carrying out the experiments experience was gained, experience summarized in this lesson learned section.

The Bayesian classifier adopted to recover traceability links needs the estimation of joint and marginal probability distributions, unfortunately, in the software engineering area barely ever abundant training material is available. As a consequence, in the authors' experience, the probability distributions suffer due to zero-frequency and data sparseness. Several probability estimation techniques were investigated among which floor discount, shift-$\beta$, and a naive approach in which a constant (low) value is assigned to events not present in a given training set. The best results were obtained with the shift-$\beta$; this is in agreement with previous works on ALBERGATE [2] and on a C++ library of foundation classes [3]. However, for this new method the choice of the probability estimation technique is much more critical: naive and floor discount estimation produce poor results. This is not surprisingly in that the method bootstraps itself starting with a single link.

The joint probability distribution between program item names and high level documents model the programmers as communication channels with different input/output vocabularies. Coding standards represents a practice that soundly support an *a-posteriori* process of traceability link recovery. However, a coding standard *per-se* will not suffice. The coding standard has to adopted as a common practice, moreover, it should be complemented by a set of rules used to create program item names from high level documents and/or domain concepts.

The use of a morphological analyzer on the available data required particular care. The results obtained showed that a profitable use of the morphological analysis requires a suitable quantity of training material, in general such a quantity is greater than the one required when the morphological analysis is not performed. In the experiment presented in this paper the break-point related to the method performance with or without morphological analysis was $s = 4$. Other researchers on different software system may obtain different values, further work is required to draw a conclusion and identify rules establishing when the morphological analysis add a tangible contribution.

## 7. Conclusions

A new method to recover traceability links between high level documentation, such as functional requirements or use cases, and low level artifacts i.e., detailed design or source code and the method equations have been presented.

Central to the method is the assumption that programmers tend to process application-domain knowledge in an uniform way, applying a set of unknown rules, when writing code, more precisely when choosing identifiers. Thus program item names of different code chunks, related to the same high level documents and/or concepts, are likely to be the same or very similar. The method infers in a probabilistic form program behavior, i.e the rules adopted by the programmers choosing identifiers, those rules are implicitly represented by the joint probability distributions estimated on training sets.

The method was applied to a system written in Java: Java source code (i.e., classes) were traced into the functional requirements and results compared with the traceability matrix compiled by the system developers.

In the opinion of the authors the literature does not comprise of any method or approach similar to the one presented in this paper apart some previous work on the same Java system. The newly obtained results favorably compare with the previous results, even if the comparison is not straightforward and figures seem somehow contradictory.

Most noticeably, as the training set increase, the method

performance improved; in other words, on the available data the method learns i.e., the joint probability distributions seems to effectively capture the rules applied when creating identifiers.

Future works will be devoted to further validate the method on different systems, and to evaluate other probability estimation procedures.

## References


[1] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia. Identifying the starting impact set of a maintenance request. *Proc. of the Conference on Software Maintenance and Reengineering*, pages 227–230, March 2000.

[2] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Tracing object oriented code into functional requirement. *Proc. of the International Workshop on Program Comprehension*, To appear.

[3] G. Antoniol, G. Canfora, A. D. Lucia, and E. Merlo. Recovering code to documentation links in oo systems. *Proc. of the Working Conference on Reverse Engineering*, pages 136–144, Oct 1999.

[4] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[5] P. F. Brown, S. F. Chen, S. A. D. Pietra, V. J. D. Pietra, A. S. Kehler, and R. L. Mercer. Automatic speech recognition in machine-aided translation. *Computer Speech and Language*, 8:177–187, August 1994.

[6] J. Concling and M. Bergen. Ibis: a hypertext tool for exploratory policy discussion. *ACM Transaction on Office Information Systems*, pages 303–331, April 1988.

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

[8] K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[9] M. M. Lehman. Programs, life cycles and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, September 1980.

[10] Y. Maarek, D. Berry, and G. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*, 17:800–813, 8 1991.

[11] R. D. Mori. *Spoken dialogues with computers*. Academic Press, Inc., Orlando, Florida 32887, 1998.

[12] H. New and U. Essen. On smoothing techniques for bigram-bases natural language modelling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume S12.11, pages 825–828, Toronto, Canada, 1991.

[13] F. A. C. Pinhero and A. G. J. An object-oriented tool for tracing requirements. *IEEE Software, IEEE Computer Society Press*, pages 52–64, March 1996.

[14] B. Ramesh and V. Dhar. Supporting systems development using knowledge captured during requirements engineering. *IEEE Transactions on Software Engineering*, pages 498–510, June 1992.

[15] I. H. Witten and T. C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. Inform. Theory*, IT-37(4):1085–1094, 1991.