Enhancing IBM Requisite Pro with IR-based traceability recovery features

Andrea De Lucia, Raffaele Landi, Rocco Oliveto, Genoveffa Tortora

Department of Mathematics and Informatics
University of Salerno, 84084 Fisciano (SA), Italy
adelucia@unisa.it, raffaele.landi81@gmail.com, roliveto@unisa.it,
tortora@unisa.it

Abstract. The potential benefits of traceability are well known, as well as the impracticability of maintaining traceability links manually. Recently, Information Retrieval (IR) techniques have been proposed in order to support the software engineer during the traceability link identification process. Clearly, a research method/tool has more change to be transferred to practitioner if its usefulness is investigated through empirical user studies and it can be integrated within a commercial and widely used CASE tool. In this paper we try to achieve this result showing how IBM Requisite Pro can be enriched with IR-based traceability recovery features.

1 Introduction

Traceability refers to the ability to define, capture and follow the traces left by requirements on other software artefacts and the traces left by those artefacts on requirements [1] [2]. Thus, traceability information helps software engineers to understand the relationships and dependencies among various software artefacts. For this reason, it provides important insights during software development and evolution of software systems helping in program comprehension, maintenance, impact analysis, and reuse of existing software [3].

1.1 Motivation

The potential benefits of traceability are well known, as well as the impracticability of maintaining traceability links manually [4]. Thus, a tool support is needed in order to effectively manage traceability information. However, the support for traceability in contemporary software engineering environments and tools is not satisfactory. Although several research and commercial tools are available that support traceability between artefacts (see for instance [5] and [6]) the main drawback of these tools is the lack of automatic or semi-automatic traceability link generation and maintenance [4].

The need to provide the software engineer with methods and tools supporting traceability recovery has been widely recognised in the last years. Recently,

Table 1. Summary of IR-based traceability recovery tools

Tool name	IR method	Enhancing strategies	Architecture
ADAMS Re-Trace [17]	LSI	None	Web-based and Eclipse plug-in
Poirot:TraceMaker [18]	Probabilistic model	Hierarchical modelling	Web-based
ReqAnalyst [12]	LSI	None	Web-based
RETRO [11]	VSM and LSI	User feedback	Standalone
TraceViz [19]	LSI	None	Eclipse plug-in

Information Retrieval (IR) [7] [8] techniques have been proposed to resolve the problem of recovering traceability links between artefacts of different types [3] [9] [10] [11] [12] [13] [14]. IR-based tools recover traceability links on the basis of the similarity between the text contained in the software artefacts. The main assumption behind such a kind of tools is that most of the software documentation is text based or contains textual descriptions and that programmers use meaningful domain terms to define source code identifiers. Based on the traceability recovery methods proposed in the literature, several tools have also been implemented. Table 1 classifies them according to the IR method used to recover the links, enhancing strategies of the recovery technique, and the architecture of the tool[1].

Although the performance of these tools seem promising, only one IR-based traceability recovery tool has been actually integrated within an artefact management system. In particular, in [10] we presented the IR-based traceability recovery tool of ADAMS, an advanced artefact management system. The tool, called ADAMS Re-Trace [10], provides the software engineer with the set of links not traced yet by the software engineer and retrieved by the tool using the Latent Semantic Indexing (LSI) [8], an advanced IR technique. We also performed a set of user studies aiming at analysing how the tracing accuracy of the software engineer are affected by the use of an IR-based traceability recovery tool [10] [16]. The achieved results showed that a traceability recovery tool is required during traceability recovery task in order to reduce the time spent to complete the task as well as to reduce the number of links erroneously traced [10] [16].

Clearly, if the usefulness of a research tool is investigated through empirical user studies it has more chances to be transferred to practitioners. However, regardless its proved usefulness the ADAMS traceability recovery tool has still an important weakness in order to be transferred from academia to industry: it has been integrated in a research artefact management system. Clearly, the technology transfer from academia to industry is facilitated if a research tool can be easily integrated within a commercial and widely used CASE tool.

1.2 Contribution

In this paper we try to bring the gap between academia and industry. In particular, basing on our experience in designing and implementing ADAMS Re-Trace, we have enriched IBM Requisite Pro [6] with IR-based traceability recovery

[1] A complete survey on all these methods and tools can be found in [10] [15].

features. Requisite Pro is a requirements management tool that also provides support for traceability. However, traceability information has to be manually managed by software engineer. In order to better support the traceability identification process, we have developed a plug-in for IBM Modeler[2], that allows the software engineer to recover traceability links between the requirements stored in a Requisite Pro project. The reason why we decided to integrate the tool in IBM Modeler is two-folds:

- it is built on top of the open and extensible Eclipse[3] platform and provides a high level of extensibility;
- it provides API allowing the communication with IBM Requisite Pro.

1.3 Paper organisation

The paper is organised as follows. Section 2 describes the IR-based traceability recovery process, while Section 3 presents the IR-based traceability recovery tool integrated in IBM Modeler. Finally, Section 4 gives concluding remarks and directions for future work.

2 IR-based traceability recovery

Software artefacts having a high textual similarity probably share several concepts, so they are likely good candidates to be traced from one another. An IR-based traceability recovery tool is founded on such a conjecture and uses an IR technique to compute a textual similarity score between artefacts as a function of the frequency of co-occurrence of individual terms that are contained in both artefacts. Clearly, higher similarity scores indicate a possible trace between the two artefacts.

Figure 1 shows the phases of a traceability recovery process. The process starts by extracting information about the occurrences of terms (words) within the source and target artefacts. The extraction of the terms is preceded by a text normalisation phase that (i) prunes out white spaces and most non-textual tokens from the text (i.e., operators, special symbols, some numbers, etc.) and (ii) splits into separate words source code identifiers composed of two or more words (i.e., TelephoneNumber and telephone_number are split into the words telephone and number). Moreover, a stop word list is also used to discard common words (i.e., articles, adverbs, etc) that are not useful to capture the semantics of the artefact content. Sometimes, a more complicated artefact pre-processing is also applied. In particular, a morphological analysis, like stemming [20], can be used to remove suffixes of words to extract their stems.

[2] IBM Modeler (http://www-01.ibm.com/software/awdtools/modeler/swmodeler) is a collaborative platform for visual modeling and design that integrates several products of the IBM Rational Suite. It is built on top of the Eclipse platform.

[3] http://www.eclipse.org.

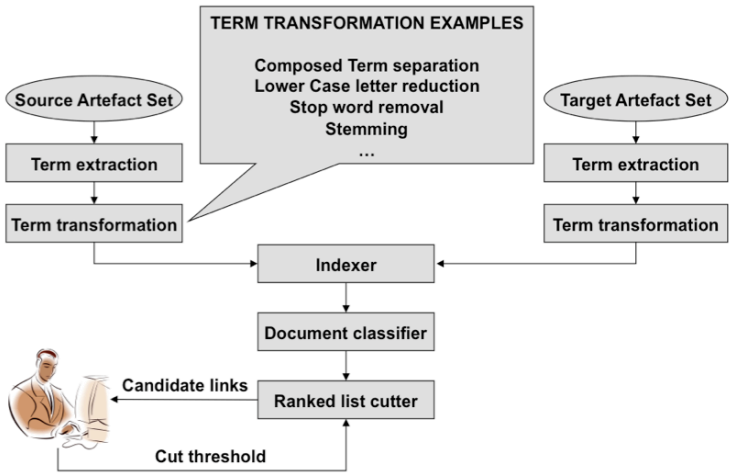

Fig. 1. An IR-based traceability recovery process

The output of the indexing process is a m x n matrix, where m is the number of all terms that occur in all the artefacts, and n is the number of artefacts in the repository. This matrix is referred to as the term-by-document matrix because a generic entry $a_{i,j}$ of this matrix denotes a measure of the weight (i.e., the importance) of the i^{th} term in the j^{th} document. It is worth noting that each column of the term-by-document matrix can be considered as a vector in the m-space of the terms. This is the rationale behind the Vector Space Model (VSM) [7], where artefacts are represented as vectors of terms that occur within artefacts in a repository (column of the term-by-document matrix) and the similarity between two artefacts is measured by the cosine of the angle between the corresponding vectors, which increases as more terms are shared.

Using the IR technique, a traceability recovery tool calculates the similarity between a set of source artefacts (used as a query) against another (even overlapping) set of target artefacts and rank the similarity of all possible pairs of artefacts. Because the number of all possible links contained in the ranked list can be very high, such a tool use some method to cut the ranked list, presenting the software engineer only the subset of top links in the ranked list (retrieved links). Unfortunately, the set of links retrieved by the tool does not in general coincide with the set of correct links between the artefacts in the repository. Indeed, the tool will fail to retrieve some of the correct links, while on the other hand it will also retrieve links that are not correct (false positives). This means that an IR-based traceability recovery process cannot be completely automated and requires the interaction of the software engineer. In particular, the software engineer analyses the list of candidate links tracing correct links and discarding false positives.

It is worth noting that the lower the similarity threshold used, the higher the number of correct links as well as the number of false positives retrieved. Indeed,

an IR-based tool is able to suggest correct links with good precision only in the upper part of the ranked list, where the density of the correct links is higher. In the lower part of the list the density of correct links is very low and there is a great predominance of false positives [10]. Therefore, identifying correct links in the lower part of the ranked list is similar to the case of deleting (unclassified) spam messages from the incoming e-mail box: besides being very tedious, the risk to also delete correct messages is likely to increase with the number of spam messages.

For this reason, we proposed an incremental approach to the traceability recovery problem, so that the links suggested by the tool can be analysed and classified step-by-step [10]. The process should start with a high threshold that is decreased at each iteration. In this way, the software engineer is able to monitor the recovering accuracy of the tool and decides to stop the process when he/she has the perception that the effort to discard false positives is becoming too high. Clearly, using such an approach the software engineer probably does not recover all correct links. Indeed, IR-based traceability recovery tools might be used to retrieve as many correct links as possible (keeping low the number of false positives to discard) but they have to be necessarily combined with manual tracing activities to build a complete traceability matrix [10] [16].

3 ReqTracer Pro overview

IBM Rational Requisite Pro is a requirements management tool used during software development. In Requisite Pro the requirements can be created and updated directly in the Microsoft Word application. Each requirement is composed of a name, a text and other attributes such as hierarchy and versions. Requisite Pro also allows the software engineer to store traceability links between requirements. Traceability information in Requisite Pro are useful for impact analysis and change management during software evolution, as well as for requirements coverage.

Unfortunately, Requisite Pro does not provide any support during the traceability link identification process. Indeed, traceability information has to be manually managed by the software engineer. Thus, when the number of requirements grows up, traceability link management become a difficult task. For this reason, we have enriched Requisite Pro with traceability recovery features. In particular, we have developed a plug-in for IBM Rational Software Modeler, called ReqTracer Pro, supporting the software engineer during traceability recovery task. In particular, the tool uses an Information Retrieval technique, namely VSM [7], in order to recover candidate links between requirements store in Requisite Pro.

3.1 Traceability recovery functionality

The traceability recovery functionality can be activated through a wizard consisting of three steps. In particular, in the first step the software engineer selects the type of requirements he/she is interested in tracing. In the second step the

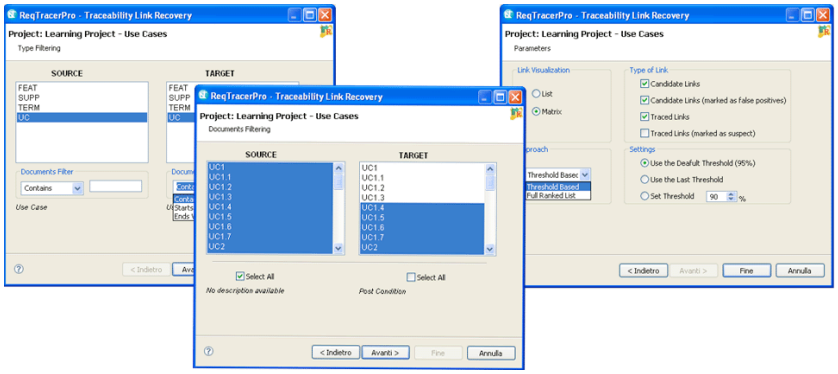

Fig. 2. ReqTracerPro: traceability recovery wizard

software engineer has the possibility to further filter the requirements to trace. In particular, he/she selects the sets of source and target requirements belonging to the requirement categories selected in the first step. Finally, in the third and final step, the software engineer defines the approach to use to show the list of candidate links:

- *Threshold based*: a threshold is used to cut the list of candidate links and to consider as candidate traceability links only the pairs of requirements with similarity above such a threshold. Three different thresholds can be used:
  - default threshold: the default threshold, i.e. 95%;
  - lowest threshold: the tool maintains for each pair of requirement types the lowest threshold used by the software engineer to trace some suggested links. Such a threshold, rather than the default threshold is proposed at the first iteration of the next traceability recovery session on the same pair of requirement types, thus reducing the time required for tuning;
  - last threshold: the last threshold used in the previous traceability recovery session on the same pair of requirement types.
- *Full ranked list*: the full list of candidate links is shown. It is worth noting that the same behaviour of the tool can be achieved using 0% as threshold to cut the ranked list.

In the final step the software engineer also selects how to visualise the candidate links, as well as the types of links to visualise. In particular, two different types of visualisation can be used:

- *List*: the ranked list of candidate links is visualised. In this case only the candidate links are visualised;
- *Matrix*: a candidate traceability matrix is visualised. Using such a visualisation - other than candidate links - the software engineer has the possibility to visualise links previously classified as false positives, as well as links previously traced and stored in Requisite Pro.

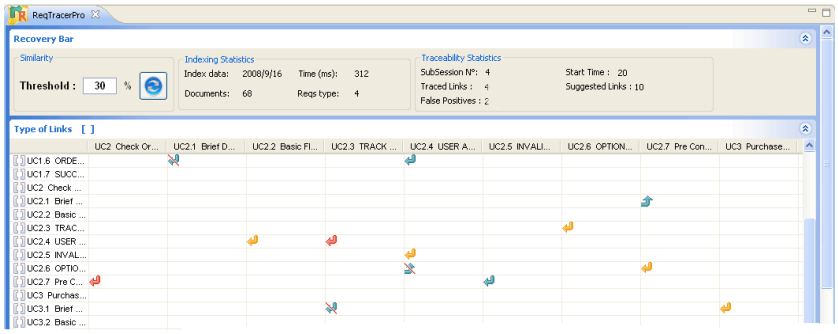

Fig. 3. ReqTracerPro: analysis of candidate links (matrix view)

Table 2. Description of link types and operations allowed on each type of link

Icon	Description	Allowed operation
	Links previously traced and stored in Requisite Pro	Set the link as suspect, change link direction, and remove link
	Links previously marked as suspect	Remove suspect, change link direction, and remove link
	Candidate link proposed by ReqTracerPro	Trace link and store it in Requisite Pro, set the proposed link as false positive
	Link previously classified as false positive	Remove false positive

Once all the inputs have been defined, the software engineer can start a traceability recovery session. The tool compares the links retrieved by using the VSM with the links traced by the software engineer and stored in Requisite Pro. In this way the tool is able to show the software engineer only retrieved links that have not been traced yet.

Figure 3 shows the list of links suggested by the tool using the matrix view. The software engineer can analyse such a list, trace a candidate links or classify it as false positive. Using such a view the software engineer has also the possibility to remove a link previously traced, as well as a link previously classified as false positive. Table 2 summarises the graphical representation and the relative description of the different types of links showed in the matrix view. The table also reports the description of all the operations allowed on each type of link.

It is important to note that using the matrix view the software engineer can also manually trace a link between two requirements. In particular, he/she can click on an empty entry of the matrix and trace a new link. This operation is not provided in Modeler.

As shown in Figure 3 the tool supports the incremental traceability recovery maintaining information concerning the previous recovery iterations (with a higher similarity threshold) in terms of number of suggested links with respect to the number of links classified as correct links or false positives. In this way, the software engineer can decide to stop the recovery process when he/she has the perception that the cost of discarding false positives is becoming to high compared with the benefits of identifying new correct links.

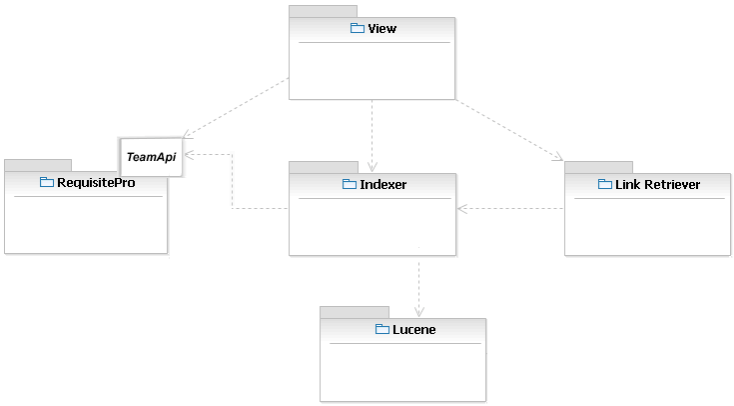

Fig. 4. ReqTracerPro architecture

3.2 Architecture

The tool has been developed as plug-in for IBM Rational Modeler. The plug-in is decomposed in three modules, namely *View*, *Indexer*, and *Link Retriever* (see Figure 4). The module *View* represents the presentation layer of the plug-in. In particular, it is in charge to visualise the wizard for the traceability recovery, as well as the candidate links in both the visualisation (i.e., list- and matrix-based).

The module *Indexer* is in charge of extracting the terms from the requirements stored in Requisite Pro and to build the document corpus (i.e., term-by-document matrix). In particular, the module uses the TeamAPI provided by IBM in order to access to the information stored in each requirements and Apache Lucene[4] in order to build the document corpus. This module is also in charge of building the ranked lists of similarity values for all pairs of requirement types by using VSM. The document corpus is built indexing the text of the requirement and its description, the description of the package, and the text of the discussions made on the requirement. These ranked lists are stored in a database and used by the module *Link Retriever*. The *Indexer* is the most expensive module from a computation time point of view. For this reason, it is implemented as a separate thread to avoid slowing-down the work of the software engineer. It is important to note that even if the *Indexer* is directly invoked by the software engineer, this module also checks if the document corpus is out-of-date. In particular, the document corpus can be out-of-date since requirements can be removed/changed or new requirements can be added in Requisite Pro. For this reason, each time the traceability recovery functionality is activated the *Indexer* checks if the document corpus is up-to-date and suggests to re-build it in case it is out-of-date.

[4] Apache Lucene (http://lucene.apache.org/java/docs/index.html) is Information Retrieval library written entirely in Java. It is available as open source project.

Finally, the module *Link Retriever* builds the list of candidate links using the information stored by *Indexer*. This module is also in charge of maintaining for each pair of requirement types both the last and the lowest thresholds used in the previous traceability recovery sessions, as well as the statistics of previous sessions.

4 Conclusion and future work

In the last decade, several IR-based traceability recovery tools have been proposed. Although the performance of such tools seems promising, only one IR-based traceability recovery tool has been actually integrated within an artefact management system. The tool, called ADAMS Re-Trace [10], has been integrated in a research artefact management system and it has been evaluated thought a set of user studies. Even if the usefulness of the tool has been investigated through empirical user studies it has still an important weakness in order to be transferred from academia to industry: it has been integrated in a research artefact management system. Clearly, the technology transfer from academia to industry is facilitated if a research tool can be easily integrated within a commercial and widely used CASE tool. For this reason, we have developed a plug-in for IBM Modeler, namely ReqTracer Pro, that allows the software engineer to recover traceability links between the requirements stored in a Requisite Pro project.

To the best of our knowledge ReqTracer Pro is the first traceability recovery tool that has been integrated in a widely used CASE tool. In this way the tool tries to address one of the main challenges in traceability [21]: stimulate developers to maintain traceability links up-to-date during software development. In this way we try to reduce the chances of errors that would appear if the links are recovered at the end of the development.

Future work will be devoted to further experiment and assess the tool in larger software projects, in particular projects of industrial partners interested in using ReqTracer Pro. We also plan to use the VSM in order to identify traced requirements with a low level of similarity in order to support the identification of suspect links.

References

1. Gotel, O., Finkelstein, A.: An analysis of the requirements traceability problem. In: Proceedings of 1st International Conference on Requirements Engineering, Colorado Springs, Colorado, USA, IEEE CS Press (1994) 94–101
2. Pinhero, F.A.C., Goguen, J.A.: An object-oriented tool for tracing requirements. IEEE Software **13**(2) (1996) 52–64
3. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering **28**(10) (2002) 970–983
4. Leffingwell, D.: Calculating your return on investment from more effective requirements management. Technical report, Rational Software Corporation (1997)

5. Ramesh, B., Dhar, V.: Supporting systems development using knowledge captured during requirements engineering. IEEE Transactions on Software Engineering **9**(2) (1992) 498–510
6. RequisitePro: http://www-306.ibm.com/software/awdtools/reqpro/ (2006)
7. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley (1999)
8. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science **41**(6) (1990) 391–407
9. Cleland-Huang, J., Settimi, R., Duan, C., Zou, X.: Utilizing supporting evidence to improve dynamic requirements traceability. In: Proceedings of 13th IEEE International Requirements Engineering Conference, Paris, France, IEEE CS Press (2005) 135–144
10. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artefact management systems using information retrieval methods. ACM Transactions on Software Engineering and Methodology **16**(4) (2007)
11. Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing candidate link generation for requirements tracing: The study of methods. IEEE Transactions on Software Engineering **32**(1) (2006) 4–19
12. Lormans, M., van Deursen, A.: Can LSI help reconstructing requirements traceability in design and test? In: Proceedings of 10th European Conference on Software Maintenance and Reengineering, Bari, Italy, IEEE CS Press (2006) 45–54
13. Marcus, A., Maletic, J.I.: 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) 125–135
14. Settimi, R., Cleland-Huang, J., Ben Khadra, O., Mody, J., Lukasik, W., De Palma, C.: Supporting software evolution through dynamically retrieving traces to UML artifacts. In: Proceedings of 7th IEEE International Workshop on Principles of Software Evolution, Kyoto, Japan, IEEE CS Press (2004) 49–54
15. Oliveto, R.: Traceability Management meets Information Retrieval Methods: Strengths and Limitations. PhD thesis, University of Salerno (March 2008)
16. De Lucia, A., Oliveto, R., Tortora, G.: Assessing ir-based traceability recovery tools through controlled experiments. Empirical Software Engineering (2008)
17. De Lucia, A., Oliveto, R., Tortora, G.: ADAMS Re-Trace: Traceability link recovery via latent semantic indexing. In: Proceedings of 30th IEEE/ACM International Conference on Software Engineering, ACM Press (2008) 839–842
18. Lin, J., Lin, C.C., Cleland-Huang, J., Settimi, R., Amaya, J., Bedford, G., Berenbach, B., Khadra, O.B., Duan, C., Zou, X.: Poirot: A distributed tool supporting enterprise-wide automated traceability. In: Proceedings of 14th IEEE International Requirements Engineering Conference, Minneapolis, Minnesota, USA, IEEE CS Press (2006) 356–357
19. Marcus, A., Xie, X., Poshyvanyk, D.: When and how to visualize traceability links? In: Proceedings of 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, Long Beach California, USA, ACM Press (2005) 56–61
20. Porter, M.F.: An algorithm for suffix stripping. Program **14**(3) (1980) 130–137
21. Oliveto, R., Antoniol, G., Marcus, A., Hayes, J.: Software artefact traceability: the never-ending challenge. In: Proceedings of 23rd IEEE International Conference on Software Maintenance, Paris, France, IEEE Press (2007) 485–488