TraceME: Traceability Management in Eclipse

Gabriele Bavota[1], Luigi Colangelo[1], Andrea De Lucia[1], Sabato Fusco[1], Rocco Oliveto[2], Annibale Panichella[1]
[1]*University of Salerno, Fisciano (SA), Italy*
[2]*University of Molise, Pesche (IS), Italy*
gbavota@unisa.it, luigicolbn@hotmail.it, adelucia@unisa.it
sabafusco@gmail.com, rocco.oliveto@unimol.it, apanichella@unisa.it

***Abstract*—In this demo we present TraceME (Traceability Management in Eclipse), an Eclipse plug-in, that supports the software engineer in capturing and maintaining traceability links between different types of artifacts. A comparative analysis of the functionalities of the tools supporting traceability recovery highlights that TraceME is the more comprehensive tool for supporting such a critical activity during software development.**

***Keywords*-Traceability Management; Information Retrieval**

I. Introduction

Traceability is the activity that allows to create links between and within software artifacts. Such activity is widely recognized as an important factor for program comprehension, software maintenance, impact analysis, and reuse of existing code components [1]. The importance of maintaining traceability links is confirmed on one side by the support provided by many CASE tools (see for instance Rational Requisite Pro[1]) and on the other side by the methods and tools presented in the literature to capture traceability links. Indeed, maintaining traceability information up-to-date during software development is generally impracticable due to (i) the huge number of dependencies existing between artifacts; and (ii) the evolutionary nature of software systems. This has pushed researchers to define methods and techniques to help the software engineer during the identification of traceability links. Promising results have been achieved using Information Retrieval (IR) [2] techniques. These techniques compare a set of source artifacts (used as a query) against a set of target artifacts and ranks the textual similarity of all possible pairs of artifacts (the output is a ranked list of candidate traceability links). The conjecture is that artefacts having a high textual similarity probably share several concepts, so they are likely good candidates to be traced from one to another [1].

Several IR-based tools supporting the capturing of traceability links have also been proposed (see e.g., [3], [4], [5], [6]). However, none of these tools provide a comprehensive support for capturing and managing traceability information during software development and maintenance.

This limitation has pushed us to develop TraceME, an Eclipse plug-in targeted at providing complete support for effectively capturing and managing traceability links during software development. TraceME allows the software engineer to:

- Define different artifacts categories, depending on the types of artifacts the software engineer is interested in tracing (e.g., use cases, classes).
- Capture traceability links between the defined artifacts categories by using the Lucene IR engine[2]. TraceME also provides two enhancing strategies (i.e., used feedbacks [7] and coverage analysis [8]) to improve the accuracy of the IR engine aiming at reducing the effort for the software engineer during traceability capturing.
- Manage the traceability information storing it in XML files and allowing its modification and deletion. Moreover, given one or more artefacts, TraceME shows their traceability dependency graph with respect to the other software artefacts. This feature of TraceME is particularly useful to support impact analysis.

A comparative analysis of the functionalities of the existing tools supporting traceability capture and management highlights that TraceME provides the most comprehensive support for such critical activities. Our tool is available online together with a demonstration video[3].

II. TraceME in Action

In this section we will explore the TraceME functionalities through an usage scenario. Jim is a software developer working on a software system called SMOS. His project is becoming increasingly large day by day, due to the implementation of new requirements. Thus, Jim was asked by his project manager, Tom, to create and store traceability links between the SMOS's artefacts in order to ease the software maintenance activities. Since Jim uses Eclipse as IDE, he chooses to use TraceME to support the traceability recovery process and starts by recovering links between use cases and classes of the source code.

A. Artefacts' Categories Creation

To start using TraceME on the SMOS Eclipse project, Jim has to define the categories of artefacts he is interested to trace. Selecting SMOS in the Eclipse workspace and clicking

[1]http://www-01.ibm.com/software/awdtools/reqpro/

[2]http://lucene.apache.org/core/

[3]http://www.distat.unimol.it/reports/traceme/

978-1-4673-2312-3/12/$31.00

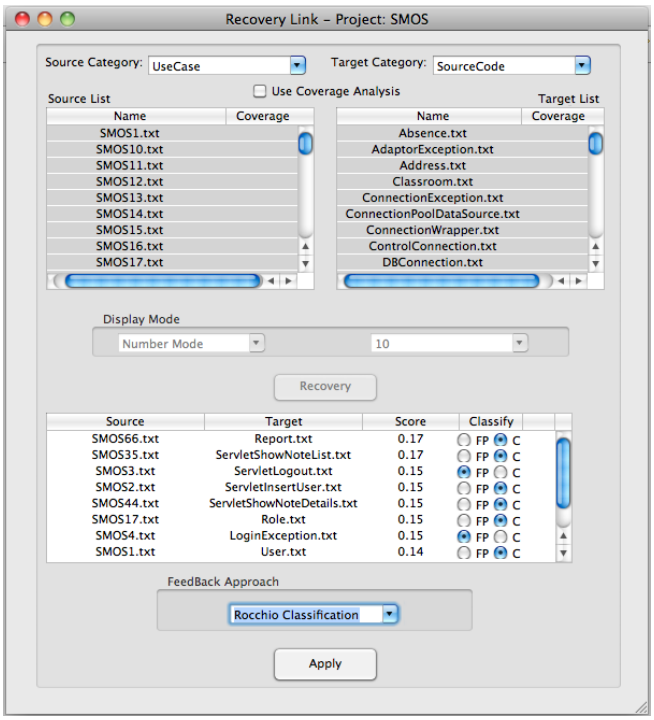

Figure 1. The Traceability Recovery View

on the *Add Artefact Category* menu item in the TraceME menu, Jim can add a new artefact category by simply defining its name (e.g., use cases) and selecting the folder containing artefacts of that category. TraceME will associate to the defined category all files in the selected folder (and subfolders) having one of the supported formats, i.e., txt, odt, pdf, and all those belonging to the Microsoft Office suite. Moreover, when running for the first time on a project, TraceME automatically creates a category *SourceCode* and associates to it all artefacts contained in the *src* folder of the selected Eclipse's project. Thus, to perform traceability recovery between use cases and source code classes, Jim has to define only the use cases category. Clearly, Jim can always delete or modify the created categories.

B. Traceability Recovery

Once the artefact categories have been created, Jim can use TraceME to recover traceability links between them. TraceME supports IR-based traceability recovery using the IR engine Lucene[4], an implementation of the Vector Space Model (VSM). Figure 1 shows the TraceME's traceability recovery view.

To start the recovery process, Jim has to select the source and target artefact categories he is interested in (*UseCase* and *SourceCode* in Figure 1). TraceME will show in the *Source List* and *Target List* the artefacts belonging to the two selected categories.

[4]http://lucene.apache.org/

Starting from these two lists, it is possible to perform traceability recovery in different ways. In particular, selecting all artefacts from the two categories the software engineer can perform a massive traceability recovery, where the IR engine computes the similarity between all possible pairs of artefacts. On the other hand, the software engineer can perform focused traceability recovery sessions by selecting a single source artefact and looking for links between it and the set of target artefacts. Clearly, TraceME also allows the selection of any subset of artefacts in the source and/or target category on which the software engineer wants to focus his/her attention. In this case, since Jim has to perform traceability recovery from scratch, he chooses to start with a massive approach, retrieving links between all source and target artefacts.

The next step is to choose a method to cut the ranked list of candidate links generated by the IR engine. Empirical studies have indicated that the list of candidate links contains a higher density of correct traceability links in the upper part of the list and a much lower density of such links in the bottom part of the list [9]. This means that in the lower part of the ranked list the effort required to discard false positives becomes much higher than the effort to validate correct links and thus, cutting the ranked list at some point is recommendable, especially when performing a massive traceability recovery process. In TraceME, the software engineer has three possible choices: (i) visualizing the full ranked list, (ii) visualizing the top n candidate links (i.e., the n pairs of artefacts with the higher textual similarity, and (iii) visualizing the candidate links having a similarity value higher than a defined threshold t. In this case, Jim chooses to adopt an incremental traceability recovery process [9], and thus to visualize and classify the top n candidate links at a time. In this way, the process can be stopped when the effort to discard false positives is becoming much higher than the effort to identify new correct links [9]. Jim sets $n = 10$ visualizing the 10 top pairs of artefacts at each iteration of the incremental process (see Figure 1).

Then, Jim clicks on the *Recovery* button asking the IR-engine (i.e., Lucene) to compute the ranked list of candidate links. Jim has to classify the links proposed by TraceME as *correct* or *false positives* using the provided radio buttons (see Figure 1). The classified links (both corrects and false positives) are stored by TraceME into XML files. Jim can also choose to provide his classification as input to the IR engine to allow it to learn from the user feedback and change the rank of the suggested links based on this. The learning process is based on the standard Rocchio [7] that is the most used relevance feedback algorithm for traditional IR tasks. Previous studies show as this kind of feedback mechanism can be useful to improve the performances of IR-engines during traceability recovery [10]. As shown in Figure 1, in this case Jim chooses to enable the feedback mechanism.

After some iterations and the classification of several

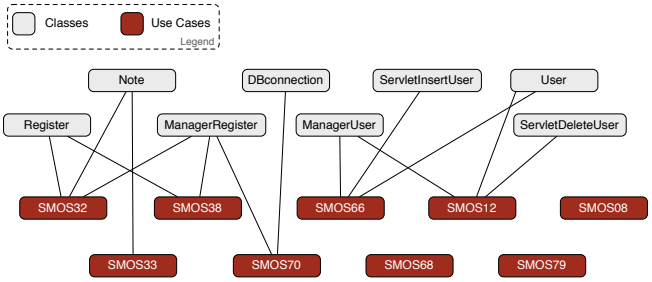

Figure 2. A snapshot of the TraceME's traceability graph

candidate links, Jim observes as TraceME is only proposing false positives and does not provide any further help in the identification of new correct links. As said before, this is typical of the IR-based traceability recovery, since in the lower part of the ranked list there is a great predominance of false positives [9]. Thus, he chooses to stop the recovery process and click on the *Traceability Graph* button to see the traced links. The obtained traceability graph is shown in Figure 2 (we only reported a snapshot of the graph for sake of readability). Jim noticed how some use cases (e.g., *SMOS68*) are not traced on any source code class. However, Jim also knows that all the functionalities described in the use cases have been implemented in SMOS. Thus, he goes back to the traceability recovery view, selects again use cases and source code as artefact categories, and checks the *Coverage Analysis* checkbox (see Figure 1). In this way, TraceME will sort the source artefacts in the *Source List* based on their coverage index with the target artefacts. In other words, use cases that have been poorly traced on the source code classes will appear on top of the *Source List*. The use of *Coverage Analysis* information during IR-based traceability recovery has been demonstrated to be worthwhile to enrich the set of correct links traced by the software engineer [8]. Jim found on top of the *Source List* the use case named *SMOS68*. Thus, he selects this artefact from the list and performs a focused traceability recovery session. This time TraceME will propose as candidate links only those involving the selected use case (i.e., *SMOS68*). Clearly, as Jim traces new links involving the *SMOS68* use case, the latter will be moved down in the *Source List* since its coverage index is increasing.

Jim focuses his attention on all source artefacts highlighted by the coverage analysis as poorly traced, until he feels that the traceability recovery task is completed.

III. Architecture

The plug-in is decomposed into six modules, namely *View*, *IR-Traceability Engine*, *Feedback Engine*, *Artefact Coverage Manager*, and *XML Manager* (see Figure 3). The module *View* implements the presentation layer of the plug-in. This view extends the *ViewPart* view defined in Eclipse.

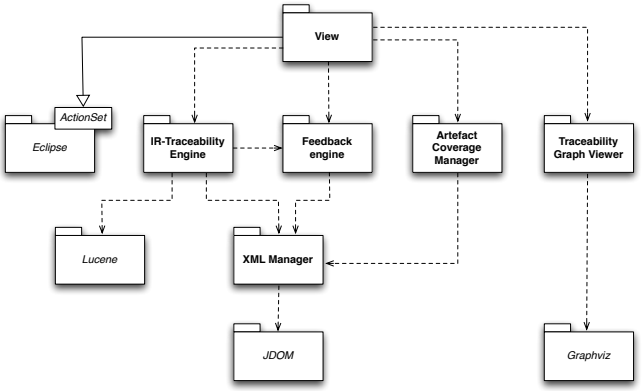

Figure 3. TraceME architecture

The *IR-Traceability Engine* module uses Lucene and is in charge of providing the list of candidate links for the source and target artefact categories provided as input. Note that, before running Lucene on the document corpus (i.e., the selected source and target artefacts), this module performs a text normalization phase. In particular, two steps are performed:

- *Text pre-processing*: white spaces and most non-textual tokens (e.g., special symbols) are pruned out from the text and all capital letters are transformed into lower case letters. Moreover, code identifiers composed of two or more words separated by using the under score or camel case separators are split into separate words, e.g., *getName* is split into *get* and *name*;
- *Word extraction and filtering*: a stop word function prunes out all the words having a length less than 3, while a stop word list cut-off all the words that are not useful to characterize the semantics of the document content (e.g., Java keywords, articles, etc.).

The *IR-Traceability Engine* module uses the *Feedback Engine* modules when the user selects to use the learning mechanism based on the standard Rocchio (see Section II). The links classified by the user are stored in XML files using the *XML Manager* module that is in charge of managing all the operations performed on the produced XML files.

The *Artefact Coverage Manager* computes the information needed when performing coverage analysis during the traceability recovery process. It uses the information stored in the XML files to compute the coverage indexes of all source artefacts.

Finally, the *Traceability Graph Viewer* uses the Graphviz[5] visualization framework to visualize the traceability graph (see Figure 2).

IV. Demo Remarks

In this demonstration we presented TraceME, an Eclipse plug-in supporting the software engineer in the recovery

[5]http://www.graphviz.org/

Table I
SUMMARY OF TRACEABILITY RECOVERY TOOLS

Tool name	Traceability Recovery	Artefact Category Management	Traceability Link Visualization	Enhancing Strategies	Coverage Analysis	Support for Link Evolution	Architecture
ADAMS Re-Trace [11]	LSI	Yes	No	None	Yes	No	Eclipse plug-in
Asuncion *et al.* [12]	Manual	No	No	None	No	Yes	Standalone
Poirot [5]	Probabilistic Model	Yes	No	Hierarchical Modeling	No	No	web-based
ReqAnalyst [4]	LSI	No	No	None	No	No	web-based
RETRO [3]	LSI and VSM	No	No	User Feedback	No	No	Standalone
Traceclipse [6]	VSM	Yes	No	None	No	No	Eclipse plug-in
TraceME	*VSM*	*Yes*	*Yes*	*User Feedback*	*Yes*	*No*	*Eclipse plug-in*
TraceViz [13]	LSI	Yes	Yes	None	No	No	Eclipse plug-in

and management of traceability links. Table I compares the characteristics of TraceME with the other tools presented in literature. Note that, among the seven reported tools, the one by Asuncion *et al.* [12] is the only one without IR-based traceability recovery. Thus, characteristics like "enhancing strategies" or "coverage analysis" simply do not make sense for this tool. However, it is worth noting that this is the only tool providing support for the evolution of the (manually) traced links during software maintenance.

Among the IR-based tools, TraceME is the one providing the largest set of features. As an example, tools as ReqAnalyst [4] or RETRO [3] do not allow the software engineer to create his/her own *categories of artefacts* bounding he/she in a set of pre-defined categories. This clearly negativly influence the flexibility of these tools.

TraceME and TraceViz [13] are the only tools providing a *Traceability Graph View* useful to quickly analyze the traced links. For example, this feature can be particularly useful while performing impact analysis.

As for the supported *enhancing strategies*, besides Poirot [5] supporting the Hierarchical Modeling (an enhancing strategy fitted for the IR probabilistic model), only TraceME and RETRO [3] provides a self-learning engine based on the standard Rocchio [7] that changes the ranks of candidate links on the basis of the link classification feedback provided by the user. The use of this kind of feedback mechanism has been demonstrated to be worthwhile during traceability recovery [10].

Finally, the coverage analysis [8] is implemented only by ADAMS [8], [11] and TraceME despite its simplicity and the benefits provided during traceability recovery [8].

Given the current state of TraceMe, it is clear that future work should be focused on the implementation of supports focused on the evolution of traceability links, e.g., provide alerts to the developer when a link might be no more valid.

REFERENCES

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

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

[3] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods." *IEEE TSE*, vol. 32, no. 1, pp. 4–19, 2006.

[4] M. Lormans and A. van Deursen, "Can LSI help reconstructing requirements traceability in design and test?" in *Proc. of 10th CSMR*. , 2006, pp. 45–54.

[5] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou, "Poirot: A distributed tool supporting enterprise-wide automated traceability," in *Proc. of 14th IEEE RE*. , 2006, pp. 356–357.

[6] S. Klock, M. Gethers, B. Dit, and D. Poshyvanyk, "Traceclipse: an eclipse plug-in for traceability link recovery and management," in *Proc. of the 6th TEFSE*, 2011, pp. 24–30.

[7] J. J. Rocchio, *The SMART Retrieval System – Experiments in Automatic Document Processing*. Prentice Hall, Inc., 1971, ch. Relevance feedback in information retrieval, pp. 313–323.

[8] A. De Lucia, R. Oliveto, and G. Tortora, "The role of the coverage analysis in traceability recovery process: a controlled experiment," in *Proc. of 25th ICSM*. , 2009.

[9] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM TOSEM*, vol. 16, no. 4, p. 13, 2007.

[10] A. De Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: a silver bullet for traceability recovery," in *ICSM '06: Proc. of the 22nd ICSM*. , 2006, pp. 299–309.

[11] A. De Lucia, R. Oliveto, and G. Tortora, "ADAMS Re-Trace: Traceability link recovery via latent semantic indexing," in *Proc. of 30th ICSE*. , 2008, pp. 839–842.

[12] H. U. Asuncion, F. François, and R. N. Taylor, "An end-to-end industrial software traceability tool," in *Proc. of the the 6th ESEC-FSE*, 2007, pp. 115–124.

[13] A. Marcus, X. Xie, and D. Poshyvanyk, "When and how to visualize traceability links?" in *Proc. of 3rd TEFSE*. 2005, pp. 56–61.