COCONUT: COde COmprehension Nurturant Using Traceability

Conference Paper · October 2006

DOI: 10.1109/ICSM.2006.19 · Source: IEEE Xplore

CITATIONS
8

READS
96

4 authors, including:

Massimiliano Di Penta
Università degli Studi del Sannio
384 PUBLICATIONS **15,212** CITATIONS
SEE PROFILE

Rocco Oliveto
Università degli Studi del Molise
218 PUBLICATIONS **9,520** CITATIONS
SEE PROFILE

COCONUT: COde COmprehension Nurturant Using Traceability

Andrea De Lucia[1], Massimiliano Di Penta[2], Rocco Oliveto[1], and Francesco Zurolo[1]

[1] *Department of Mathematics and Informatics, University of Salerno, Fisciano, Italy*
[2]*RCOST – Research Centre on Software Technology, University of Sannio, Benevento, Italy*
adelucia@unisa.it, dipenta@unisannio.it, roliveto@unisa.it, frazur@gmail.com

Abstract

In this demonstration we present an Eclipse plug-in, called COCONUT (COde COmprehension Nurturant Using Traceability), that shows the similarity level between the source code under development and high-level artefacts the source code should be traced onto. Also, the plug-in suggests candidate source code identifiers according to the domain terms contained into the corresponding high-level artefacts. Experiments showed that the plug-in helps to produce source code easier to be understood.

1. Introduction

Software artefact traceability is widely recognised as an important factor for program comprehension, software maintenance, impact analysis, and reuse of existing code components. Since traceability links are often missing or lost, several techniques (based on Information Retrieval or machine learning) have been developed to reconstruct them [1], [4]. In this demonstration, we show how the same ideas can be used during the development process: the similarity measure computed by traceability recovery approaches can be used to guide programmers to write more understandable code. In particular, we present COCONUT, an Eclipse plug-in that shows to the developer the similarity level between the source code being written and high-level artefacts the code should be traced onto. Such a similarity provides insights about the consistency between source code identifiers and high-level artefacts, suggesting the developer that the code is (or is not) properly traced to the related artefacts. In the latter case, the developer can try to make source code identifiers more consistent and/or he/she can better comment source code. This would help to produce code that contains better identifiers and comments, improving comprehensibility and maintainability.

To help the developer in improving the identifiers, the plug-in also extracts (using an n-gram extractor) a list of candidate source code identifiers from high-level artefacts. In this way, when a programmer writes code, the tool automatically suggests identifiers that are more appropriate to the particular context.

A controlled experiment we performed showed that the tool helps to produce code with better identifiers and comments [3].

2. COCONUT overview

COCONUT works with the Java Development Tool (JDT), although it can be easily adapted for other Eclipse tools (e.g., UML modellers). The plug-in contributes a new view to the Eclipse workbench organised in two different tabs, namely *Similarity* and *Identifiers*. In particular, in the tab *Similarity* information about the similarity between the source code under development and related artefacts are visualised, while in the tab *Identifiers* suggestions of source code identifiers are available.

The COCONUT project is hosted by SourceForge at the URL *http://sourceforge.net/projects/coconut*.

2.1. Visualisation of the similarity level

The *Similarity* tab in the COCONUT view shows a sorted list of all indexed artefacts as a table (see Figure 1). The first column of the table contains check boxes allowing the developer to indicate whether the artefact is traced onto the source code under development. The second column contains the artefact description, and the third column shows the similarity value between the artefact and the source code under development. Such a similarity is computed using Latent Semantic Indexing (LSI) as described in [2], [4]. The plug-in preferences allow the user to create a new artefact space, or to load an existing one, stored in a XML file containing the term-by-document matrix.

22nd IEEE International Conference on Software Maintenance (ICSM'06)
0-7695-2354-4/06 $20.00 © 2006 IEEE

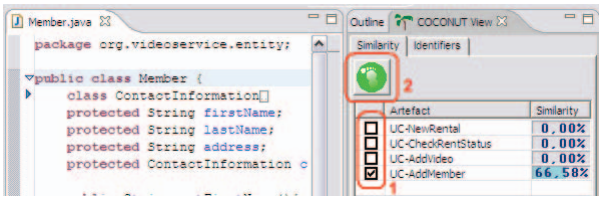

Figure 1 – Visualisation of similarity level

Figure 1 shows COCONUT at work in a scenario where the developer is coding the Java class *Member*. The developer relates one or more high-level artefacts to the class *Member* - in our example the use case *UC-AddMember* - by selecting their check-boxes (step 1). Then, he/she clicks on the button in the top of the plug-in view (step 2). COCONUT visualises the similarity between the class under development and selected high-level artefacts. This indicates to the developer how the source code identifiers and comments are consistent with the related high-level artefacts.

2.2. Suggestion of source code identifiers

The *Identifiers* tab in the COCONUT view shows a sorted list of candidate identifiers, obtained by extracting n-grams from the related high-level artefacts (selecting in the section *Similarity*) and containing a given substring (see Figure 2). The COCONUT n-gram extractor can be customised by specifying the min and max sizes of n-grams and the strategy to compose multi-word identifiers, namely using camel case (e.g., *telephoneNumber*) or an underscore separator (e.g., *telephone_number*).

Figure 2 shows a scenario where the developer is coding the Java class *Member* and he/she uses COCONUT to identify an appropriate name for the class modelling the member's contact information. To this aim, he/she starts writing the class name (see Figure 2) and then he/she selects the menu item *Get suggestions* from the pop-up menu activated on the selected substring *con*. Alternatively, it is possible to get suggestions by writing the substring in an appropriate field of the *Identifiers* tab, and then clicking on the button *Suggest*. As shown in Figure 2, COCONUT proposes different identifiers containing the selected substring. The developer can then select the most appropriate one by double clicking on it.

2.3 Preliminary empirical evaluation

A controlled experiment [3] was performed to assess the usefulness of the plug-in. The experiment compared groups of developers using the plug-in with control groups not using it. Results indicated that the

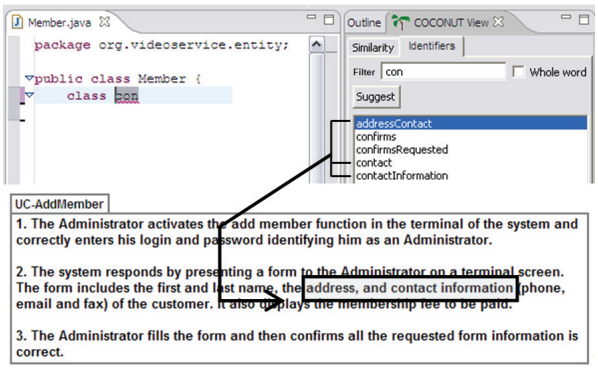

Figure 2 – List of candidate identifiers

plug-in significantly helps to improve the similarity between code and related high level artefacts.

Such a similarity improvement was especially due to better comments developers using the plug-in added, but also to more consistent identifiers, as confirmed by inspecting the code produced during the experiment.

3. Demo remarks

In this demonstration we presented an Eclipse plug-in, called COCONUT, which helps programmers to improve the comprehensibility of the code produced, by showing them the similarity between the source code under development and high-level artefacts. If the code is poorly traced with relevant requirements, the developers will react by changing identifiers and/or adding comments. This task is supported by a plug-in feature that suggests to the developer candidate identifiers.

References

[1]. G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. "Recovering Traceability Links between Code and Documentation", *IEEE Transactions on Software Engineering*, vol. 28, no. 10, 2002, pp. 970-983.

[2]. A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "ADAMS Re-Trace: a Traceability Recovery Tool", *Proceedings of 9th IEEE European Conference on Software Maintenance and Reengineering*, Manchester, UK, 2005, pp. 32-41.

[3]. A. De Lucia, M. Di Penta, R. Oliveto, and F. Zurolo, "Improving Comprehensibility of Source Code via Traceability Information: a Controlled Experiment", *Proceedings of 14th International Conference on Program Comprehension*, Athens, Greece, 2006, pp. 317-326.

[4]. A. Marcus and J. I. Maletic, "Recovering Documentation to Source Code Traceability Links using Latent Semantic Indexing", *Proceedings of the International Conference on Software Engineering*, Portland, Oregon, USA, 2003, pp. 125-135.