International Journal of Modern Engineering Research (IJMER)

www.ijmer.com Vol.2, Issue.3, May-June 2012 pp-1014-1016 ISSN: 2249-6645

Reducing Source Code Complexity For Software development using Code Comprehension Nurturant using Traceability

[1]N. Rajasekhar Reddy [2] B.Anand kumar

[1] Associate Professor, Department of CSE, Madanapalli Institute of Technology and Science,Madanapalli,Andhra Pradesh, India

[2]Research Scholar, Department of CSE, Madanapalli Institute of Technology and Science, Madanapalli, Andhra Pradesh, India.

Abstract- Nowadays, many papers are developing to improve the software quality control. In our paper we are going to help the developers to maintain the source code and identifiers and we will show the textual similarity between source code and related high level faults. The developers are improving the source code library. So, if the software development environment provides similarities between the source code and the high level problems then it will be quite easier for the developers to keep the software quality ahead. In our proposing system the candidate identifiers needs to implement in eclipse IDE for that we need a plug-in called **CO**de **CO**mprehension **N**urturant **U**sing **T**raceability (COCONUT). This paper also reports on two controlled experiments performed with master's and bachelor's students. The quality of identifiers, comments in the produced source code with or without coconut. The approach presented in this paper relates to approaches aimed at applying IR techniques for traceability recovery and for quality improvement/assessment. So that quality of the source code lexicon will be improved. Thus the usefulness of the coconut is taken as a feature of software development environments.

Keywords: Software traceability, source code comprehensibility, source code identifier quality, information retrieval, software development environments, empirical software engineering.

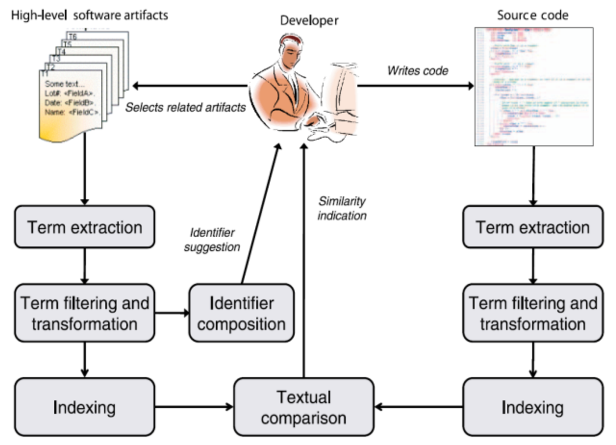
Introduction:

This paper proposes the use of traceability information combined with IR techniques from a different perspective. When a programmer writes the source code, a plug-in incorporated in the development environment shows the similarity between the source code under development and high-level artefacts on which the source code should be traced. Such a similarity provides information 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 second case, the developer can act in various ways. This chapter presents the concept of requirements tracing and discusses several aspects related to traceability. Particular importance is given to the informal aspects of requirements tracing and to the non-functional nature of requirements traceability. To give further support to the developer, the proposed approach also recommends candidate identifiers built from high-level artifacts related to the source code. The paper also describes an Eclipse1 plug-in, named COde COmprehension Nurturant Using Traceability (COCONUT), which implements the proposed approach. Its evaluation has been carried out through two controlled experiments involving master's and bachelor's students, where we asked students to perform development tasks with and without the availability of COCONUT features. The analysis of the achieved results confirms our conjecture that providing the developers with the similarity between code and high-level artifacts helps to significantly improve the quality of source code identifiers and comments, which also further increases when developers receive suggestions about candidate identifiers.

Related Work:

Recently, the artefact traceability support has been introduced in some process support systems where the traceability layer has been combined with event based notifications mainly to make users aware of artefact modifications. All these tools have a common drawback: they require a manual maintenance of the traceability layer while the system changes and evolves. The traceability recovery problem is widely tackled in the literature and several techniques are applied to support the process of traceability link recovery. Some of them deal with recovering traceability links between design and implementation. The proposed approaches represent source code and high-level models using a common language and use regular expressions, maximum match algorithm or more tolerant string matching to map source code in the high-level models. Other approaches consider text documents written in natural language, such as requirements documents. Automate the generation of traceability relations between textual requirement artefacts and object models using heuristic rules. we also evaluated the different magnitude using the Cohen d effect size. A major threat could be related to the applicability of ANOVA [2] when data deviate from normality and for crossover designs, although ANOVA is generally pretty robust to deviations from normality and the carryover effect is limited, as discussed. In addition, to confirm the results obtained by ANOVA, we performed multiple Mann-Whitney tests, with threshold p-values adjusted by means of the correction.

A novel approach of Proposed system:



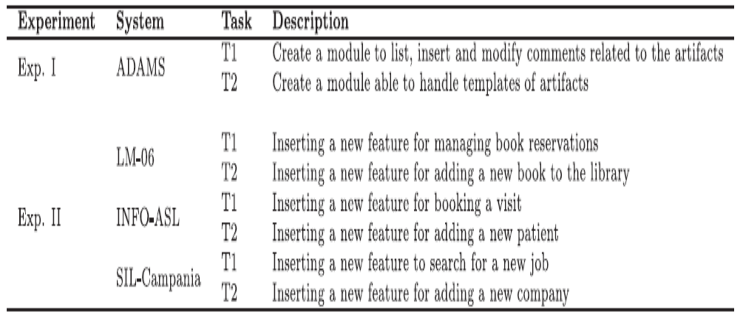
The proposed approach is based on the assumption that developers are induced to make the source code identifiers more consistent with domain terms or to better comment the source code if the software development environment provides information about the textual similarity between the source code being written and the related high-level artifacts. Clearly, the proposed approach is based on the assumption that high-level documentation requirements, use cases, and module specifications is available during the development process. The flow of information between a developer and the Integrated Development Environment (IDE) in the proposed approach. . The textual similarity between the source code and the related high-level artifacts is computed by using an IR-based approach. In general, an IR method compares a given query against all the documents in a collection by computing the textual similarity between these documents and the query.

1. Removing non textual tokens, e.g., numbers and punctuation for the high-level artifacts, and operators, special symbols, and programming language keywords from the source code;
2. Splitting into separate words source code identifiers composed of two or more words separated by using the underscore or camel case separators.
3. Removing stop words using a stop word removal function which removes words having a length less than a fixed threshold and also removing words belonging to a stop word list.

Hypothesis Formulation and Variable Selection:
As we wanted to investigate the usefulness of both the COCONUT similarity feature and the identifier suggestion feature, the experiments foresaw three possible treatments:

1. No plug-in (NOPL): Subjects performed their tasks without using COCONUT.
2. With simple plug-in (PL): Subjects performed their tasks with the COCONUT similarity feature available only.
3. With fully featured plug-in (PLP): Subjects performed their tasks with both features provided by COCONUT, i.e., the similarity and the identifier suggestion features.

Experimental task:



Experiment	System	Task	Description
Exp. I	ADAMS	T1	Create a module to list, insert and modify comments related to the artifacts
		T2	Create a module able to handle templates of artifacts
Exp. II	LM-06	T1	Inserting a new feature for managing book reservations
		T2	Inserting a new feature for adding a new book to the library
	INFO-ASL	T1	Inserting a new feature for booking a visit
		T2	Inserting a new feature for adding a new patient
	SIL-Campania	T1	Inserting a new feature to search for a new job
		T2	Inserting a new feature for adding a new company

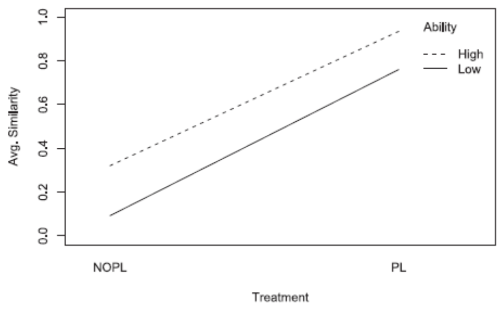
we defined two dependent variables,
Sim with Comments and Sim no Comments, measuring the average similarity between the source code and the related high level artifact(s), including source code comments given the set of raw similarity (RSim with Comments) computed for a given combination of projects and tasks, we compute Sim with Comments as follows:

$$Sim_{withComments} = \frac{RSim_{withComments} - min(RSim_{withComments})}{max(RSim_{withComments}) - min(RSim_{withComments})},$$

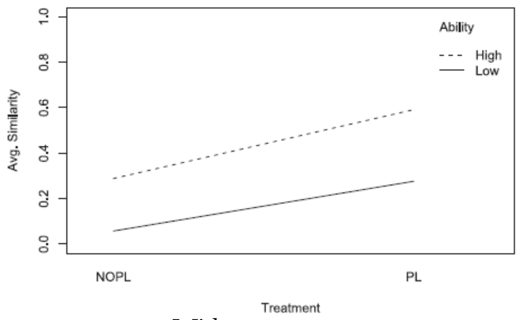
The first three null hypotheses the controlled experiments aimed at testing are:

1. H01 : The use of the COCONUT similarity feature does not significantly improve the similarity between the commented source code and the related high-level artifacts (compared with the use of Eclipse without the COCONUT plug-in).
2. H02 : The use of COCONUT similarity feature does not significantly improve the similarity between the uncommented source code and the related high-level artifacts (compared with the use of Eclipse without the COCONUT plug-in).
3. H03 (tested in Exp II only): The comprehensive use of COCONUT (similarity and identifier suggestion features) does not significantly improve the similarity between the uncommented source code and the related high-level artifacts (compared with the use of the COCONUT similarity feature only).

Comparison :
Effect of treatment:
Ability:



With comments



Without comments

Conclusion:

The proposed approach is to intimate the developers about the current developing project with the existing one which is similar to the current project. the main terms used as identifiers or present in comments. In particular, our approach

1) Computes and shows to developers the textual similarity between source code and related high-level artifacts.

2) Recommends candidate identifiers built from high-level artifacts related to the source code under development.

References:

[1] V.R. Basili, L.C. Briand, and W.L. Melo, "**A Validation of Object- Oriented Design Metrics as Quality Indicators**," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751-761, Oct. 1996.

[2]M. Lormans, A. Deursen, and H.-G. Gross, "**An Industrial Case Study in Reconstructing Requirements Views**," Empirical Software Eng., vol. 13, no. 6, pp. 727-760, 2008.

[3] T. Zimmermann, R. Premraj, and A. Zeller, "**Predicting Defects for Eclipse**," Proc. Third ICSE Int'l Workshop Predictor Models in Software Eng., 2007.

[4]G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "**Traceability Recovery Using Numerical Analysis**," Proc. 16th Working Conf. Reverse Eng., 2009.

AUTHOR'S DESCRIPTION

N.Rajasekhar reddy was born in Madanapalli, February 28.He was received Bachelor's degree in Computer Science in S.V University and M.Tech degree from Satyabama University respectively. After working as a research assistant and an assistant professor in the Dept. of Computer Science and Engineering, Madanapalli Institute of Technology and Science, Andhra Pradesh, India. His research interest includes Software Engineering, Software Quality Assurance and Testing. He was published 4 international journal papers and 5 National journal papers in Software Engineering. He is a member of SCIE, ISTE, and IEEE

B.Anand kumar was born in Theni, January 05. He was received Bachelor's degree in Information technology in Anna University and M.Tech degree pursuing from J.N.T University Anatapur respectively. His research interest includes Software Engineering, Software Quality Assurance and Testing. Software methodologies.I was done my project under the esteemed guidance of Mr.N.RajaSekharReddy.