Tracing Requirements in Object-Oriented Software Engineering

Ali S. Dowa. faculty of Information Technology, Azawia Zawia University
Amrou S. Dhunnis , faculty of Information Technology Zawia University
Mahmoud J. Abdullah , Computer Science Department, faculty of science , Zawia University.

Abstract:

Software system maintenance is a fact that plays a major role in software development life cycle. Software does change and evolve during maintenance or creation of another version or adding up new requirements or features onto existing software. Capturing the traceability relations between software requirement and design allows developers to check whether the design meets the requirement and to analyze the impact of requirement changes on the design .

This paper aims at describing an approach to generate relations between different types of software artifacts through object-oriented software development life cycle.. A prototype software has been developed to demonstrate the feasibility of the approach. Encouraging results of the feasibility and consistency of the software is obtained.

1. Introduction:

During their usage, software system requirements have to be changed, not only after releasing the product but also along the iterative software development process. Tracing requirements change during software system development is an important factor to ease software development, maintenance, evolution and maintain high quality software system`[4].

Requirement traceability shows how one artifact is related to another by following the life of its requirements during each stage of the software development process in both forward and backward direction [6] [page no].

In general, requirements traceability helps software developers to find the impact of changes in requirement statement on other artifacts in subsequent phases in software development process [7, 9].

At each stage of the object-oriented software development process, different models are produced. Engineering community adapted the Unified Modeling Language (UML) as its standard means for representing and documenting software system models and related artifacts [5,10]. The basic models produced during the object oriented software development process are:

- System requirements: define functional and non-functional requirements of a software system that are generated during the requirements engineering process, and it is expressed in natural language.
- Analysis model: Concerned with developing software engineering requirements and specifications that expressed as a system's object model, which is composed of a population of interacting objects .The analysis model contains the analysis classes and any associated artifacts.
- Design model: The design model can be thought of in two phases. The first, called high-level design which deals with the decomposition of the system into large and complex objects. The second phase is called low-level design. In this phase, attributes and methods are specified at the level of individual objects.
- Implementation model: defines components that representing source code (translate the solution domain model into source code)

These artifacts can be related and traced to each other via some relationships among them [6,8,10,11].

In this work an approach is proposed and described to support automatic generation of traceability relations between functional requirements (expressed in natural language) and other artifacts produced during the analysis model in the object oriented software development process (that expressed in UML diagrams).

2. Overview of a Prototype Software :

The main objective is to build a software tool that automatically generates the relations between software artifacts produced during object-oriented system development, and visually presenting them to software developer. The input of this tool is the software system documents which include the requirements and analysis models. The output is a visual representation of relations of software artifacts

3. Approach:

In the software development process, the object-oriented analysis and design methodologies usually use nouns and verbs to indicate either classes or actions [3].

In case of the analysis stage of object-oriented system development, nouns are found in a requirement statement model as either class names or attributes of a class, and verbs model as operations of a class, use-case name [1,2].

Therefore, those concepts (nouns and verbs) are used by the proposed software to automatically generate the relations among a requirement statements and other artifact in subsequent phases of the object-oriented system development process.

Therefore both concepts which are used by software to establish relations among different artifacts of software devilment process can be summarized as follows:

Concept-1:

If there is a matching between nouns in a requirement statement with an artifact (class-name) in the analysis model then a relation is generated between them.

Concept-2:

If there is a matching between a verbs in a requirement statement with an artifact (use-case-name) in the use cases model then a relation is generated between them.

In accordance with the first concept, when a noun is located in a requirement statement, our software tool will search for a match between the noun and class names in analysis models.

According to the second concept, when a verb is located in a requirement statement, the software tool searches for a match between that verb and the names of use-case in the use-case model.

In both cases, when a match is found a relationship is established between a requirement statement and a matched artifact.

4. The Analysis of Prototype Software:

The use case model shown in Figure-1 describes the main functionality of the software.

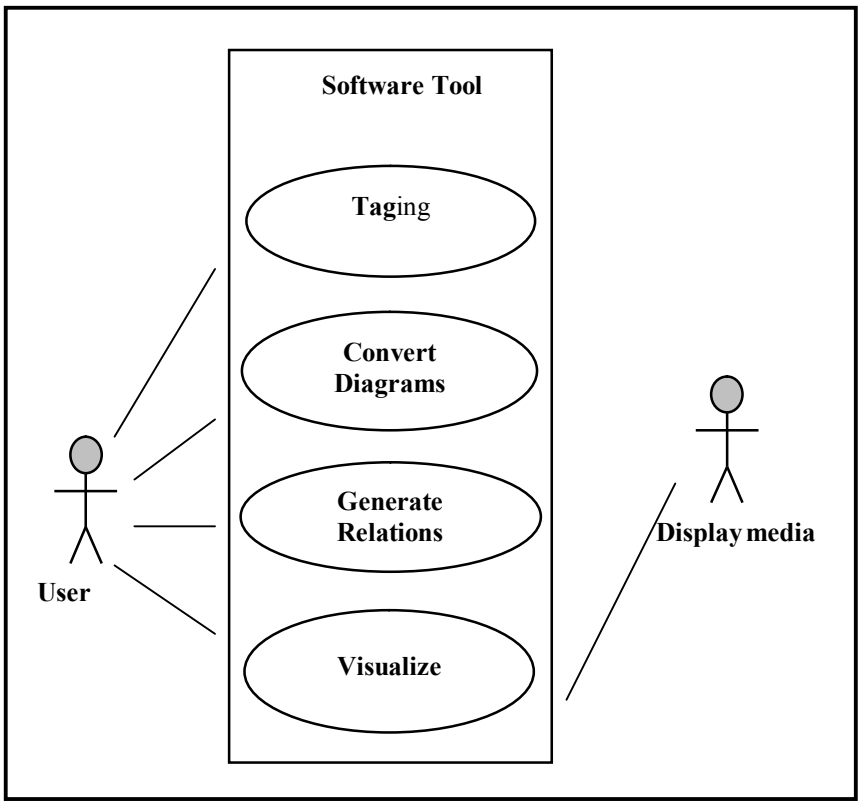

Figure 1: Use-Case Model of Software Prototype

4.1 The Tagging Use-Case:

This use case is executed as follows:

- Reading text file of requirement statements.
- Using grammatical tagger to produce tagged form of the text file.
- Analyzing the tagged form of the text file to extract needed information from the requirement statement such as, (nouns, verbs).
- Organizing and storing the extracted information in a repository.

4.2 The Convert Diagrams Use-Case:

This use case is executed as follows:

- Reading UML files, which include several diagrams such as:
  - ✓ Use-case diagram.
  - ✓ Class diagram.
- Using existing commercial exporter to convert the UML diagram to textual information.
- Storing the textual information in a storage repository.

4.3 The Generate Relations Use-Case:

This use case executed as follow:

- Read extracted information (nouns and verbs) from repository.
- Read converted textual information of UML models.
- Match each noun with class names in analysis models and class names and attributes of classes in the design model.
- Match each verb with names of use-cases in the analysis model, class operations, and names of sequence diagrams in the design model.
- Establish a relation when a match is found.
- Store established relations in the storage repository.

4.4 The Visualize Use-Case:

In this use-case when a requirement statement is selected, the use-case retrieves information from the repository and displays the relations

between a requirement statement and other artifacts resulted from subsequent phases of the object-oriented system development process.

5. The Packages of Prototype Software:

Figure-2 shows how the tool is organized in different components (Analyzer, Extractor, Repository, Relations Generator and Visualizer) that work together to achieve the functionality of the tool.

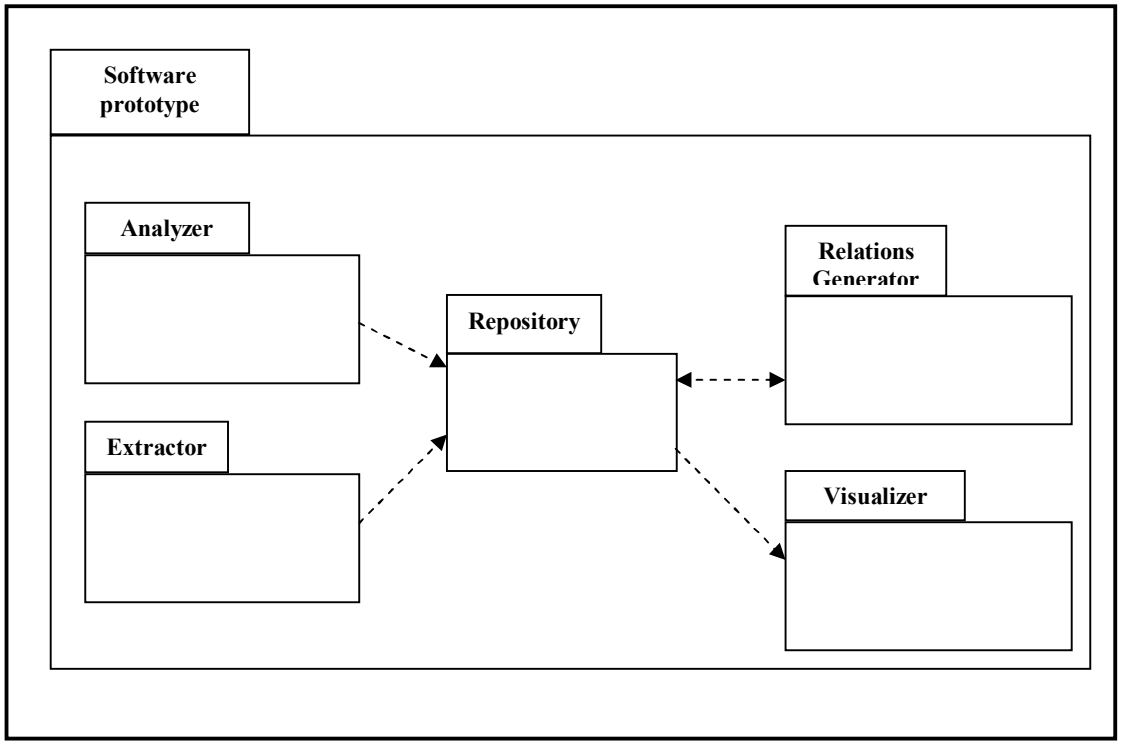

Figure 2: The interacted packages of the tool

6. The Proposed Prototype Software :

The implementation of the prototype system is done to test the visibility of the proposed approach to generate traceability relations, it can also be used to test the consistency between automatic and manual generation of the traceability relations.

7. Evaluation Method :

A case study of a Registration Student System that is available in the software literature is used for the evaluation.

The manual analysis of this case study documentations which will be used to measure the results of software prototype is obtained from Software Engineering text books [10].

8. Evaluation Results:

Table-1 illustrates the comparison between the results obtained from software prototype and the manual analysis of the case study.

Reqnt.ID / Used manner		Traced Use Cases	Traced Analysis Classes	Traced Attributes	Traced Operations
1	Manually	1	3	7	4
	By the Software	1	3	6	2
2	Manually	1	2	7	3
	By the Software	1	2	7	3
3	Manually	-	2	7	3
	By the Software	-	2	7	3

Reqnt.ID / Used manner		Traced Use Cases	Traced Analysis Classes	Traced Attributes	Traced Operations
4	Manually	1	2	7	3
	By the Software	1	2	7	3
5	Manually	2	5	11	8
	By the Software	2	5	11	8
6	Manually	1	4	9	3
	By the Software	1	4	9	3
7	Manually	-	-	-	-
	By the Software	-	-	-	-

Table 1: **Comparison between the results obtained from the and the manual analysis of the case study.**

9. Discussion:

As noticed from the table-1, the results are equivalent in most cases, with the exception of some cases such as:

- **Traced attributes :**

In Requirement-1, according to the manual analysis there are seven attributes traced to this requirement, our software discover six attributes traced to this requirement.

- **Traced operations :**

Based on the manual analysis Requirement-1, , there are four operations traced to this requirement, the software discovers only two traced operations.

10. Consistency:

Table-2 illustrates the consistency of the results that produced in case study.

Traced Artifacts of Case Study	Traced Use Cases	Traced Analysis Classes	Traced Attributes	Traced Operations
Manual vs. By the Software	100%	100%	97.5%	91.66%

Table 2: Consistency of the results of a case study

11. Conclusion :

In this work, an approach has been presented and implemented as a prototype case tool, which automatically generates relations between software artifacts produced during the system development process, using object-oriented approach. This software tool visually represents these generated relations to the developer.

12. Future Work:

Our discussion shows there is a place for more future improvements; these improvements can be summarized as follows:

- Working on Links requirements with its artifacts, in the other phases of software development process.
- Address the problem of the verbs inflection, which will improve the tool to discover additional operations from requirement statements.

- This tool can be used to trace the artifacts of Arabic software requirements, if it used with Arabic Part-of-Speech Tagger.

References :

[1] Brill tagger, http://www.ling.gu.se/~lager/mogul/brill-tagger/ , The Brill tagger is a method for doing part-of-speech tagging, date accessed November 2015.

[2] Bruegge, B., and Dutoit, A.,"Object-Oriented software engineering Using UML, Patterns, and Java", Pearson Edducation, Inc 2004.

[3] Jacobson, I., et al., "The Unified Software Development Process", Addison Wesley Longman, Inc, 1999.

[4] Leffingwell, D., and widring, D., "Managing Software Requirements" Pearson Edducation, Inc, 2003.

[5] Ramesh, B., and Jarke, M., "Toward Reference Models for Requirements Traceability", Georgia State University, 2001.

[6] Sherba, S., and Anderson, K., "A Framework for Mapping Traceability Relationships ", University of Colorado, 2003

[7] Stepanian, L., "Solving the Requirements Traceability Problem", University of Toronto, 2004.

[8] Weiss, D., and Kowalczykiewicz, K., "Traceability: Taming uncontrolled change in software development", Poznań University of Technology, 2002.

[9] Project Management Software, www.projectperfect.com.au, Version 5.1 released in December 2007, date accessed March 2009

[10] *Jacobson, I., et al., "The Unified Software Development Process", Addison Wesley Longman, Inc, 1999.*

[11] *JUDE Development Group, http://jude.change-vision.com/jude-web/product/index.html, the JUDE UML modeling Tool, date accessed November 2015.*