MULTIFACETED APPROACH IN GENERATING AND RECOVERING REQUIREMENT TRACEABILITY

Siti Rochimah[1], Wan M. N. Wan Kadir[2], Abdul H. Abdullah[2]

[1]Faculty of Information Technology, Sepuluh Nopember Institute of Technology, Surabaya, Indonesia 60111

[2]Faculty of Computer Science and Information System, University Technology Malaysia, Malaysia 81310

E-mail: [1]siti@its-sby.edu, [2]{wnasir,hanan}@utm.my

Abstract: Software evolution is inevitable. When a system evolves, there are certain relationships among software artifacts that must be maintained. Software must be continually changed to remain satisfactory in use. Requirement traceability is one of importance factor in facilitating software evolution since it maintains the artifacts relationship before and after a change is performed. This position paper aims at hypothesizing that utilizing a multifaceted approach to traceability generation and recovery can provide significant support for facilitating software evolution process. The multifaceted traceability approach contains three main facets that compose the whole traceability approach. Facet-1 is a syntactical similarity matching process, Facet-2 is a link prioritization process, and Facet-3 is a heuristic-based process. Those three facets works in an integrated mode to construct traceability links among artifacts and are run on top of a particular ready-to-use integrated environment development (IDE) in order to facilitate the traceability generation and recovery as well as to ease the artifacts maintenance.

Keyword: software evolution, requirement traceability, traceability generation, traceability recovery, multifaceted traceability approach.

1. INTRODUCTION

Software traceability is becoming increasingly significant element in software development life cycle. It provides critical function in the development and maintenance of software systems during its life cycle. There are many traceability approaches that have been proposed from a very simple way using manual and spreadsheet tools to the latest models that apply some formal or complex techniques. Most of them apply a particular technique specific for the related approach.

Besides, there is a fact that software evolution is inevitable since software must be continually changed to remain satisfactory in use. It is recognized that software evolution is about changes to software. From the software evolution point of view, software traceability is one of important factors in facilitating software evolution as it maintains the artifacts relationship before and after a change is performed.

However, research has shown that evolution (and maintenance) are the most expensive activities in the software process, consuming 60% to 80% of the total time spent on a software system [1-3]. Besides, based on the literature, it is found that software traceability is not easy in practice. Nevertheless, most of traceability approaches focus only on limited aspects in generating traceability. Usually, the aspects lead to creating an automated model of traceability generation and maintenance. Most of approaches aim to automate requirements tracing, but tracing automation is still complex and error prone [4], in term of the obtained results. Furthermore, automation alone cannot really reduce efforts of software evolution.

This position paper aims at hypothesizing that utilizing a multifaceted approach to traceability generation and recovery can provide significant support for facilitating software evolution process. Facilitating software evolution process can provide minimal software evolution effort. The paper is organized as follows: Section 2 describes the literature reviews, i.e. the software evolution definition, taxonomy and the state-of-the-art traceability approaches in a brief way, as well as an evaluation of the approaches. Section 3 presents the conceptual framework of the proposed approach, followed by descriptions of the facets in Section 4. Section 5 describes future works of the research and finally, Section 6 discusses the conclusion.

2. LITERATURE REVIEW

2.1 Definition and Taxonomy of Software Evolution

The standard definition of software evolution has not been established. Different researchers use the term in different ways. Belady and Lehman in [5, 6] define software evolution as "... *the dynamic behavior of programming systems as they are maintained and enhanced over their life times*." Somerville in [7] views software evolution as a spiral model. Some researchers use the term *software evolution* as a synonym or preferable substitution for *software maintenance*, due to their similarity in the basic operation, i.e. changing the software artifacts, especially in the source code [8, 9].

Chapin et al. in [10] refines the typology into 12 types of software *evolution* and maintenance based on the purpose of change. Recently, Buckley et al. [11] defines software evolution taxonomy based on the characterizing mechanisms of change and the factors that

influence these mechanisms. The taxonomy is organized into the following logical groupings: *temporal properties* that is a dimension of software evolution that captures timing aspect of change; *objects of change* that captures location aspect of change, i.e. which part of software can be changed; *system properties* that captures the characteristic of software while it is changed; and *change support* that captures support mechanism while software is being changed.

2.2 Overview of Traceability Approaches

Gotel and Finkelstein in [12] define requirement traceability as "*the ability to describe and follow the life of a requirement, in both a forwards and backwards direction*". IEEE Standard 830-1998 [13] states that "*An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the* referencing *of each requirement in future development or enhancement documentation*". From those definitions, it is showed that traceability involves various relationships among artifacts that are produced during a software development projects.

After reviewing about hundred of recent papers relating traceability topics, the authors interested in seven approaches that include specific subject, i.e. they put the requirement as one of the main artifacts to perform traceability.

It is resumed here the traceability approaches, which will be further evaluated. Each approach is cited in term of the mechanisms and algorithms that are utilized, metrics that are used to measure the results, and scope of tracing that is covered. Those characteristics will be used to evaluate the approach in the next sections.

2.2.1 Information Retrieval Based Approach (IR)

Recently, there are many researchers attempting to establish traceability link via information retrieval approach [14-26]. This approach focuses on automating the generation of traceability link by similarity comparison between two types of artifacts. The two basic IR models which commonly used in traceability generation are probabilistic and vector space models. Numerous variant models have also been applied including the popular model Latent Semantic Indexing (LSI) which is based on vector space model. In each model, one type of particular artifacts treats as a query and another type of artifacts treats as a document being searched in term of the query. For example, source code treats as a query against requirements specification as a document being searched based on the query.

The general steps include (i) preprocessing, i.e. stop-word removal and or stemming, (ii) analyzing and indexing of an incoming document collection, followed by constructing a

representation of each document and then archiving them, (iii) analyzing and representing an incoming queries, and using a matching or ranking algorithm to determine which document representations are similar to the query representation.

Two commonly measurements for evaluating candidate links (for the tools that implement IR) are recall and precision. Recall refers to the percentage of correct matches that are found. Precision refers to the percentage of found matches that are correct. The scope of tracing covers almost of artifacts including high-level and low-level requirements, manual documents, design elements, test cases, and source code.

2.2.2 Rule-Based Approach (RB)

Spanoudakis et al. in [27, 28] propose a method to automatically create traceability link using rules. They use two traceability rules, i.e. requirement-to-object-model traceability rule (RTOM rule) and inter-requirement traceability rule (IREQ rule). The rules are deployed into three specific documents types, i.e. (i) requirements statement documents (RSD), (ii) use case documents (UCD), and (iii) analysis object models (AOM). RTOM rules are used to trace the RSD and UCD to an AOM, while IREQ rules are used to trace between RSD and the UCD. The method assumes that all of document types are in XML-based format. The traceability rules are also represented in XML-based markup language.

The method consists of four stages, i.e. (i) grammatical tagging of the artifacts, (ii) converting the tagged artifacts into XML representations (iii) generating traceability relations between artifacts, and (iv) generating traceability relations between different parts of the artifacts.

RB uses recall and precision to measure the result. Whereas it covers requirement statement documents, use case documents, and analysis object models as the objects of tracing.

Also, Nentwich et al. in [29] build rule-based tool for consistency management for XML-based software artifacts, namely 'xlinkit'. First-order logics are utilized to describe consistency rules. The tool can be applied to all kind of textual software artifacts as long as these artifacts are in document-object-model (DOM) trees format.

2.2.3 Event-Based Approach (EB)

Cleland-Huang et al. in [30-32] propose event-based approach for updating and maintaining traceability relationships. Traceability relationships are defined as publisher-subscriber relationships in which dependent artifacts must subscribe to the requirements on which they depend. When a requirement change, the dependent artifacts are notified and subsequently proper action can be taken.

The method involves three main components, i.e. (i) the requirement manager which responsible for managing requirements and for publishing change event messages to the event server, (ii) the event server which responsible for establishing traceability by handling initial subscriptions placed by dependent entities, and also listening for event notifications from the requirement manager(s) and forwarding event messages to relevant subscribers, and (iii) the subscriber manager which responsible for listening on behalf of the subscribers that it manages for event notifications forwarded by the event server.

EB assumes that the traceability links among artifacts are already established before event-based algorithms are run. Consequently, the algorithms are focused only to manage up-to-date traceability links based on changes that may occur during system operational time. The algorithms have been implemented in a tool prototype to manage and maintain traceability between requirements and UML artifacts as well as test cases. EB uses event resolution, in term of path length and weighted measure as the metrics to evaluate the result obtained by the tool.

2.2.4 Hypertext-Based Approach (HB)

Maletic et al. in [33, 34] propose an approach uses hypertext model that allow complex linking as well as versioning of links. Also, Sherba in [35, 36] propose a hypertext-based traceability relationships generation using open hypermedia and information integration.

The approach utilizes XML as the main tool for representing models and created links. The models and their links are converted into XML-based representation. Models are categorized into anchor model and target model. The links are established between anchor and target model with particular link types, i.e. causal, non-causal, and or navigational links. Once the model-to-model traceability links have been established, meta-differencing mechanism is used to indicate if some changes have been occurred in the models. The evolution is supported by a fine-grained versioning technique.

A tool is built to implement the method, but the metrics is not discussed. The scope of tracing includes all types of artifacts.

2.2.5 Feature Model-Based Approach (FB)

Riebisch and Pashov in [37, 38] describe feature model-based method for requirement traceability. They utilize feature modeling which describes a requirements as an overview and models the variability of a product line. A feature model consists of a graph with features as nodes and feature relations as edges. If the number of features is very high, then the representation of features and their relations are displayed by tables. FB is applied for the

definition of a product by a customer. Every feature describes a property of a product from the customer's point of view. There are three categories of features, i.e. (i) *functional features* express the behavior or the way users may interact with a product, (ii) *interface features* express the product's conformance to a standard or a subsystem, and (iii) *parameter features* express enumerable, listable environmental or non-functional properties.

The features are structured by hierarchical relations. Classifications of feature relations are (i) *hierarchical relations* which describe the sequence of decisions of products. The most important features are placed higher in the hierarchy, (ii) *refinement relations* which describe relations of generalization and specialization as well as aggregation, and (iii) *requires* or *excludes* relations or *multiplicity-grouping relations* which describe constraints between variable features that have an influence on the sequence of decisions of products.

The scope of traceability includes requirements to features and elements of the solution, i.e. object model and source code. Metrics and tool support are not discussed.

2.2.6 Value-Based Approach (VB)

Zemont in [39] proposes a framework for assessing the value that traceability can provide to an organization. Furthermore, Heindl and Biffl in [4] propose value-based requirement tracing. This approach provides technical support to perform requirements tracing as well as take value and cost (economical aspect) considerations into account. Thus, it provides a technical model and an economic model for requirement tracing based on some criteria.

VB consists of five processes, i.e. (i) *requirements definition*, that is identifying atomic requirements and assigning an identifier to each of them, (ii) *requirements prioritization*, that is estimating the value, risk, and effort of each requirement, (iii) *requirements packaging*, that is identifying clusters of requirements, (iv) *requirements linking*, that is establishing traceability links between requirements and other artifacts, and (v) *evaluation*, that is utilizing generated traces for certain purposes, e.g. to estimate the impact of change for particular requirements.

In the case study performed by [4], they use a manual way in obtaining the traceability links and in performing a change in the software artifacts. Metrics and tool support are not discussed.

2.2.7 Scenario-Based Approach (SB)

Egyed et al. in [40-42] propose scenario-based approach. SB uses a hypothesized trace information that have to be manually entered. Then, it uses runtime information to creating trace links. Test case scenarios are executed on a running system and execution information is

obtained using a monitoring tool. The information is then combined with the hypothesized trace information to form a footprint graph. This graph shows the relationship among artifacts in the system.

Ibrahim et al. propose document-based traceability to support software change impact analysis, which basically adopt SB approach to obtain the relationship among artifacts [43, 44]. They extend the functionality by utilizing those relationships to provide software traceability as well as for software change impact analysis visibility of different artifacts.

The traceability links are created automatically, but the hypothesized trace information must be manually entered. In SB, traceability links can only be created once a running system is available.

2.3 Evaluation of Traceability Approaches

In 1998 Von Knethen et al. [45] performed a comparative case study with industrial requirement engineering approaches. More recently, Ingeniera TCP team [46] performed a comparative study between requirement management and engineering tools.

Here, the authors specifically evaluate requirements traceability approaches in term of their support for software evolution using the software evolution taxonomy proposed by [11]. The evaluation results are shown on Table 1 through Table 4 below. The justification of each factor can be found at our previous paper [47].

Table 1. Evaluation Result on 'Temporal Properties' Dimension

Aspects	IR	RB	EB	HB	FB	VB	SB
Time of Change							
• Compile	√	√	√	√	√	√	√
• Load							
• Runtime							
Change History							
• Versioning	√	√	√	√	√		
• Sequential							
• Parallel			√				
Change Frequency							
• Continuous							
• Periodic							
• Arbitrary	√	√	√	√	√	√	√

Table 2. Evaluation Result on 'Object of Change' Dimension

Aspects	IR	RB	EB	HB	FB	VB	SB
Artifact							
• High-level	√	√	√	√	√	√	√
• Low-level	√	√	√	√	√	√	√
Granularity							
• Coarse							
• Medium			√		√		
• Fine	√	√		√		√	√
Impact							
• Local	√	√		√	√		√
• System-wide	√	√	√	√	√	√	√
Ch Propagation*							
• CIA			√			√	√
• TA	√	√	√	√	√	√	√
• EE							

* CIA: change impact analysis; TA: traceability analysis; EE: effort estimation

Table 3. Evaluation Result on 'System Properties' Dimension

Aspects	IR	RB	EB	HB	FB	VB	SB
Availability							
• Partially avail.	√	√	√	√	√	√	√
• Permanently av.							
Activeness							
• Reactive	√	√	√	√	√	√	√
• Proactive							
Openness							
• Open	√	√	√	√	√	√	√
• Closed							
Safety							
• Static	√	√	√	√	√	√	√
• Dynamic							

Table 4. Evaluation Result on 'Change Support' Dimension

Aspects	IR	RB	EB	HB	FB	VB	SB
Deg. of Automation							
• Automated	√	√	√	√			
• Semi-automated					√	√	√
• Manual							
Degree of Formality							
• Ad-hoc							
• Semi-formal	√	√	√	√	√	√	√
• Formal							
Change Type							
• Structural	√	√	√	√	√	√	√
• Semantic	√	√	√	√	√	√	√

2.4 Critical Discussion

In the context of software evolution support, the discussion can be divided into two disjoint groups. The first relates to the similarity characteristics belong to each approach, and the second is the opposite.

The similarity values which arise among them are caused by the natural or inherent characteristics of requirement traceability process. These values have to be taken for granted. It means that **no lacks has to be improved.** It is mainly due to the natural characteristics such as the need for user justification in the final results, the nature of change that may occur anytime, the origin of change that usually comes from the requirement as well as from other artifacts, but rarely (or never) from the executable artifact, etc. Table 5 below lists the similarity aspects among the approaches.

Table 5. Similarity Aspects among the Approaches

No.	Aspect	Rationale
1	Time of change: compile	The origin of change usually comes from the requirement or other artifacts, but rarely (or never) from the executable artifact
2	Change frequency: arbitrary	The nature of change that can be occurred anytime
3	Artifacts: high and low level	Usually the traceability links covers all artifacts in the software systems
4	Impact: system-wide	It is closely related to point (3) in which the link coverage are all artifacts, thus the impact is also at a system-wide.
5	Change propagation: traceability analysis	Traceability process is the main focus of the approaches
6	Availability: partial	The software systems can be stopped from running mode while it is being modified
7	Activeness: reactive	The change performed on the artifacts must be driven by an external agent, i.e. user or stakeholder
8	Openness: open	Traceability process facilitates the software to be extended or modified
9	Safety: static	The traceability process guarantees a certain degree of behavioral safety, in which the change is behavior-preserving with respect to the original behavior
10	Degree of formality: semi-formal	Approaches are implemented on some underlying mathematical model especially for the core tracing algorithms while some other parts are based on informal model (user's justification)
11	Change type: structural and semantic	The change is performed on any kind of artifacts and it can be any kind of change type

The second group relate to the characteristics in which each approach have different value. Actually, the aspects which are included in this group express their support for software evolution. Based on the described evaluation, it is obviously cleared that some of the approaches already have their support but some others do not have. For example, IR based

approach already have an automated facility to generate traceability but VB approach does not. The way to improve the value to achieve a better support for software evolution is by **modifying the approach itself** (if the algorithms enable) or **by combining a particular approach to other approach that have a better value** (if the technology is enable).

Furthermore, those which are in the second group can be classified into two categories, i.e. strong effect and optional. 'Strong effect' means that the aspect has a very strong influence in addressing current traceability problems. An approach that already has strong effect aspect is ready to solve more problems other than those which do not have this aspect. 'Optional' means that the aspect can be included in the approach to broaden its capability, or it can be excluded because it is not the main facility that has to be covered by the approach. Table 6 below illustrates these aspects sorted by a priority in which strong effect aspect has a higher priority other than optional aspect.

Table 6. Aspect Priority to Support Evolution Process

No.	Aspect	Priority	Rationale
1	Degree of automation: automated versus semi-automated	Strong effect	Automation may minimize human interaction and simplify traceability process. It may address problems such as: - huge amount of artifacts - human labor and error-prone - complicated tracing procedure - up to date link maintenance
2	Change history: versioning	Strong effect	Closely related to the automation aspect. Versioning facility is important to maintain the version of artifacts configuration, especially after a change is performed. Versioning mechanism can be easily provided if the approach has an automated traceability process.
3	Granularity: medium versus fine-grained	Strong effect	Fine granularity is important since it indicates that the traceability information covers the source-code. This aspect may address problem such as inaccurate and incomplete traceability results
4	Impact: local	Optional	Although it is closely related to the granularity aspect, this local impact is optional, since an approach may select the level of source code to be covered in the process. This aspect indicates that a change can be performed in the source code, especially in the statement level of granularity, so that it is possible to change a local variable that may affect only at the related class.
5	Change propagation: CIA	Optional	Change Impact Analysis facility can be provided separately from the traceability approach.
6	Change history: parallel	Optional	A traceability approach can be implemented on a stand alone as well as on a distributed environment.

3. CONCEPTUAL FRAMEWORK

From the evaluation it is cleared that the current approaches have variety of capability to support software evolution, but none of them support the whole aspects. Most approaches typically provide only limited supports to software evolution to perform requirements tracing and maintaining the established links.

In contrast, software maintenance and evolution are the **most expensive activities** in the software process. If an approach is designed to support software evolution activities, then to be effectively utilized, it should take as much as possible important aspects for software evolution into account.

Regarding traceability generation and recovery processes, there is one inherent characteristic of requirement tracing process in that the final result must come from a user, i.e. the analyst, or maintainer. The inherent characteristic of traceability is that the user participation in the traceability process can not be excluded. So, the automation is just to minimize the effort required by the user. The lesser user effort required the better the approach to support the software evolution, since it will minimize the effort.

This research proposed a multifaceted traceability generation and recovery to facilitate software evolution. The conceptual framework of the research is shown on Figure 1 below.

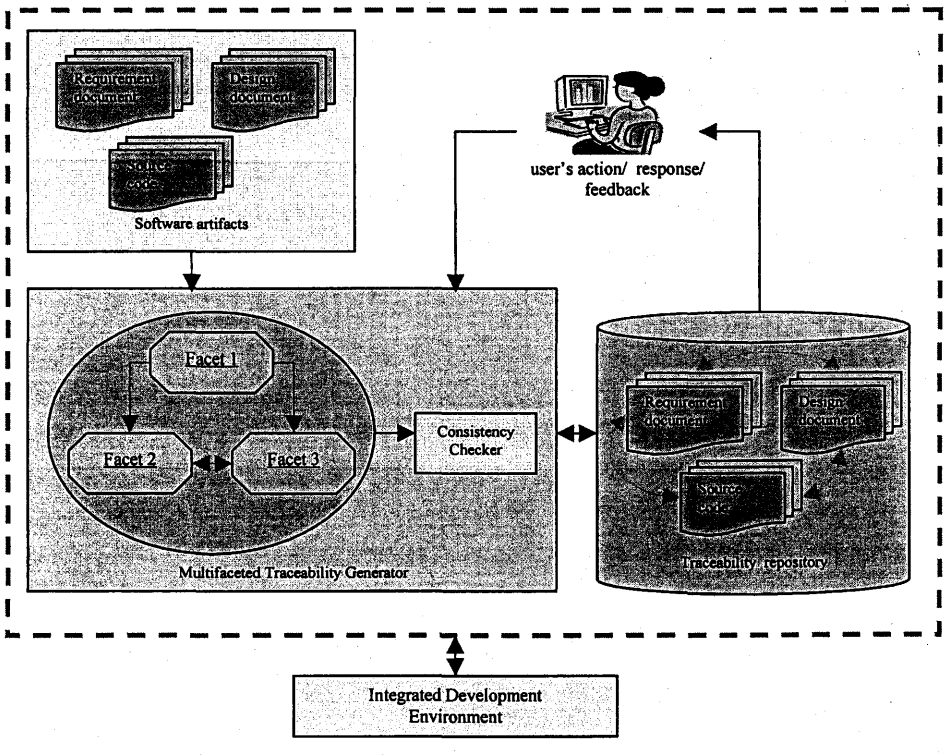

Figure 1. Conceptual Framework of the Research

As depicted in the figure, there are three main elements involved in the approach, i.e. the software artifacts in which their traceability will be created and managed, the main traceability generator which creates, recovers, and manages the traceability among artifacts, and the traceability repository which records the linked artifacts. All of them are integrated with (run on top of) a typical ready-to-use Integrated Development Environment (IDE).

The software artifacts that will be processed in this approach are the requirements documents in the form of use case specifications, the design documents in the form of class diagrams, and the source code. Those three types of artifacts represent the smaller scope of a large complete system.

The multifaceted traceability generator contains two sub-elements for the process, i.e. the traceability-techniques element (Facet-1, Facet-2, and Facet-3) and the consistency checker element. The traceability-techniques elements are composed from three different facets that are integrated altogether. Facet-1 is a syntactical similarity matching process. Facet-2 is a link prioritization process, and Facet-3 is a heuristic-based process. Those three facets work in an integrated mode to construct traceability links among artifacts. The traceability generation and recovery process must be started from Facet-1, and the subsequent process can be either from Facet-2 to Facet-3 or from Facet-3 to Facet-2. The second element of the traceability generator is the consistency checker which ensures that all of the artifacts are in the proper configurations time to time, especially after a change has been made.

The traceability repository records all of the relationship information including up-to-date links among artifacts, historical links, and other attributes related to the links. This element is also responsible for providing the artifacts traceability information that is queried by the user.

Those three main elements that compose the whole traceability approach are integrated with a particular ready-to-use integrated development environment (IDE) in order to facilitate the traceability generation and recovery as well as to ease the artifacts maintenance. This is due to the facts that the artifacts are usually be created and maintained in a particular IDE or CASE tool.

The role of the user's response in this approach is related to the inherent characteristic of traceability problem. As previously mentioned, the inherent characteristic of traceability problem is that the user participation in the traceability process can not be excluded. (S)he has to finally justify whether a created link is a true link or not. The knowledge about that is laid on the human side. So, the automation is merely to minimize the effort required by the user.

4. DESCRIPTION OF THE FACETS

This approach integrates three different processes. This integration is utilized to take multiple aspects into account while generating the traceability links. The description of those facets is described below.

Facet-1 is a syntactical similarity matching process. It allows dynamically generating traceability links in a fully automated process. This technique is used to produce links with highest recall, in order to ensure that all of candidate links have been generated. Thus, candidate links with high and low confidence will be shown as well.

Facet-2 is a link prioritization process. This technique is used to refine the results obtained by Facet-1 or Facet-3, depending on the process sequence that is chosen. This facet will prioritize the links according to the set of defined value. The defined value is the value that is assigned to each of the requirement based on its contribution or criticality in the whole project. For example, the requirement that is closely related to the application core is assigned a higher value than the others, or, the requirement that is related to government regulation (must be obeyed) is assigned a high value as well. The user has to assign a value to each requirement or group of requirements prior to the traceability generation. This is a semi-automated process, since the user intervention can not be avoided to justify the value. Every generated link is then examined and assigned a value according to its relation to the requirement (its parent). Facet-2, thus, play an important role in defining which links have to be firstly prioritized to be ordered in evaluation, based on its value, upon a change has been made on part of an artifact. This prioritization is crucial in order to minimize the effort of link evaluation conducted by the user, since only the links that have the high value is mandatory to be firstly evaluated, while the links with low value are optional.

Facet-3 is a heuristic-rule based process. In order to maximize the precision of the generated links, each of the candidate links has to be evaluated using heuristic list to filter which is the 'possibly' true link and which is not. Those that are indicated as false links will be sent to the 'recycle bin' to be evaluated later. The heuristic list works as a simple semantic traceability relationship rule, i.e. a simple rule to relate one artifact to another. For example, a heuristic rule between use case specification and class diagram can be '*a class name in the class diagram is related to an actor in the use case specification*', or '*a method name in the class diagram is related to a description in the use case specification*'. Thus, if a link satisfies the condition in the heuristic list, then the link confidence will be increased and will be indicated as true link.

5. FUTURE WORK

This research is expected to produce a new traceability approach especially to support software evolution. The future works include:

(i) Developing a software traceability model and approach;

(ii) Developing a supporting prototype tools in traceability generation and recovery; and

(iii) Analyzing results of conducted experiment to determine the effectiveness of the approach.

6. CONCLUSION

This paper has presented the background and rationale behind a proposed multifaceted traceability approach to support software evolution. It has described the evaluation of seven recent requirements traceability approaches, especially in their support for software evolution. It also described the conceptual framework of the multifaceted traceability approach. Finally, it described some future works of this research.

REFERENCES

1. Carrol, P.B., *Computer Glitch: Patching up Software Occupies Programmers and Disables Systems.* Wall Street Journal, 1988.
2. Hanna, M., *Maintenance Burden Begging for a Remedy*, in *Datamation.* 1993. p. 53-63.
3. Pfleeger, S.L., *Software Engineering: Theory and Practice.* 1998: Prentice-Hall.
4. Heindl, M. and S. Biffl. *A Case Study on Value-Based Requirement Tracing.* in *International Conference on Empirical Software Engineering (ESEC-FSE'05).* 2005. Lisbon, Portugal: ACM Press.
5. Belady, L.A. and M.M. Lehman, *A Model of Large Program Development.* IBM System Journal, 1976. **15**(1): p. 225-252.
6. Lehman, M.M., et al. *Metrics and Laws of Software Evolution - the Nineties View.* in *4th International Symposium on Software Metrics (Metrics 97).* 1997: IEEE Computer Society.
7. Sommerville, I., *Software Engineering.* 7th ed. 2004: Addison-Wesley.

8. *IEEE Std 1219-1998: IEEE Standard for Software Maintenance.* 1999.

9. Bennett, K. and V. Rajlich. *Software Maintenance and Evolution: A Roadmap.* in *Proceedings of the Conference on the Future of Software Engineering.* 2000. USA: ACM Press.

10. Chapin N., H.J., Khan K., Ramil J., Than W.G., *Types of Software Evolution and Software Maintenance.* Journal of Software Maintenance and Evolution, 2001: p. 3-20.

11. Buckley, J., et al., *Towards a Taxonomy of Software Change.* Journal of Software Maintenance and Evolution: Research and Practice, 2003. **17**(5): p. 309 - 332.

12. Gotel, O. and A. Finkelstein. *An Analysis of the Requirements Traceability Problem.* in *Proceeding of 1st International Conf. on Requirement Engineering.* 1994.

13. *IEEE Std 830-1998: IEEE Recommended Practice for Software Requirement Specification.* 1998.

14. Antoniol, G., et al., *Recovering traceability links between code and documentation.* IEEE Transactions on Software Engineering, 2002. **28**(10): p. 970-983.

15. Antoniol, G., et al. *Information Retrieval Models for Recovering Traceability Links between Code and Documentation.* in *IEEE International Conference on Software Maintenance.* 2000.

16. Cleland-Huang, J., et al. *Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability.* in *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05).* 2005.

17. De Lucia, A., et al. *ADAMS Re-Trace: a Traceability Recovery Tool.* in *9th European Conference on Software Maintenance and Reengineering (CSMR'05).* 2005: IEEE Computer Society.

18. Hayes, J.H., A. Dekhtyar, and J.M. Carigan. *Recommending a Framework for Comparison of Requirements Tracing Experiments.* in *Workshop on Empirical Studies of Software Maintenance (WESS 2004).* 2004. Chicago, IL.

19. Hayes, J.H., A. Dekhtyar, and J. Osborne. *Improving Requirements Tracing via Information Retrieval.* in *Proceedings of the 11th IEEE International Requirements Engineering Conference.* 2003.

20. Hayes, J.H., A. Dekhtyar, and S.K. Sundaram, *Improving After-the-fact Tracing and Mapping: Supporting Software Quality Predictions,* in *IEEE Software.* 2005. p. 30-37.

21. Hayes, J.H., A. Dekhtyar, and S.K. Sundaram, *Advancing candidate link generation for requirements tracing: The study of methods.* IEEE Transactions on Software Engineering, 2006. **32**(1).

22. Hayes, J.H., et al. *Helping analysts trace requirements: An objective look.* in *Proceeding of 12th IEEE International Requirements Engineering Conference (RE 2004).* 2004.

23. Lin, J., et al. *Poirot: A Distributed Tool Supporting Enterprise-Wide Automated Traceability.* in *14th IEEE International Requirement Engineering Conference (RE'06).* 2006: IEEE Computer Society.

24. Marcus, A. and I.J. Maletic. *Recovering Documentation-to-Source Code Traceability Link using Latent Semantic Indexing.* in *Proceedings of the 25th IEEE International Conference on Software Engineering.* 2003.

25. Settimi, R., et al. *Supporting software evolution through dynamically retrieving traces to UML artifacts.* in *Proceedings - 7th International Workshop on Principles of Software Evolution, IWPSE 2004 (In Conjunction with RE 2004).* 2004.

26. Zou, X., R. Settimi, and J. Cleland-Huang. *Phrasing in Dynamic Requirements Trace Retrieval.* in *Proceeding of 30th Annual International Computer Software and Applications Conference (COMPSAC'06).* 2006.

27. Spanoudakis, G. *Plausible and Adaptive Requirement Traceability Structures.* in *Proc. 14th Int'l Conf. Software Eng. and Knowledge Eng.* 2002.

28. Spanoudakis, G., et al., *Rule-Based Generation of Requirements Traceability Relations.* Journal of Systems and Software, 2004: p. 105-127.

29. Nentwich, C., et al., *xlinkit: A Consistency Checking and Smart Link Generation Services.* ACM Transactions on Internet Technology, 2002. **2**(2): p. 151-185.

30. Cleland-Huang, J., et al. *Automating Speculative Queries through Event-based Requirements Traceability.* in *Proceedings of the IEEE Joint International Conference on Requirements Engineering.* 2002.

31. Cleland-Huang, J., C.K. Chang, and G. Y. *Supporting Event Based Traceability through High-Level Recognition of Change Events.* in *IEEE Proc. Int'l Computer Software and Applications Conf. (COMPSAC).* 2002.

32. Cleland-Huang, J., C.K. Chang, and M. Christensen, *Event-based traceability for managing evolutionary change.* IEEE Trans. on Software Engineering, 2003. **29**(9): p. 796-810.

33. Maletic, J.I., et al. *Using a Hypertext Model for Traceability Link Conformance Analysis.* in *Proceedings of 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'03).* 2003.

34. Maletic, J.I., M.L. Collard, and B. Simoes. *An XML Based Approach to Support the Evolution of Model-to-Model Traceability Links*. in *Proceedings of 4th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'05)*. 2005.

35. Sherba, S.A., *Towards Automating Traceability: An Incremental and Scalable Approach*, in *Department of Computer Science*. 2005, University of Colorado: Colorado.

36. Sherba, S.A., K.M. Anderson, and M. Faisal. *A Framework for Mapping Traceability Relationships*. in *2nd International Workshop on Traceability in Emerging Forms of Software Engineering. (TEFSE '2003)*. 2003. Montreal, Canada

37. Riebisch, M. *Supporting evolutionary development by feature models and traceability links*. in *Proceedings - 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*. 2004.

38. Pashov, I. and M. Riebisch. *Using feature modeling for program comprehension and software architecture recovery*. in *Proc. of 11th IEEE Int'l Conf. and Workshop on the Engineering of Computer-Based Systems*. 2004.

39. Zemont, G., *Towards Value-Based Requirements Traceability*, in *Department of Computer Science*. 2005, De Paul University: Chicago, Illinois. p. 80.

40. Egyed, A. *A Scenario-Driven Approach to Traceability*. in *23rd International Conference on Software Engineering*. 2001. Toronto, Ontario, Canada: IEEE Computer Society.

41. Egyed, A. and P. Grunbacher. *Automating Requirements Traceability: Beyond the Record & Replay Paradigm*. in *17th IEEE International Conference on Automated Software Engineering (ASE'02)*. 2002: IEEE Computer Society.

42. Egyed, A. and P. Grunbacher, *Supporting Software Understanding with Automated Requirements Traceability*. International Journal of Software Engineering and Knowledge Engineering, 2005. **15**.

43. Ibrahim, S., M. Munro, and A. Deraman. *Implementing a Document-based Requirements Traceability: A Case Study*. in *IASTED International Conference on Software Engineering*. 2005.

44. Ibrahim, S., *A Document-Based Software Traceability to Support Change Impact Analysis of Object-Oriented Software*. 2006, Universiti Teknologi Malaysia: Kuala Lumpur.

45. Von Knethen, A., et al. *A Comparative Case Study with Industrial Requirements Engineering Methods*. 1998 [cited 2006 October]; Available from: se.informatik.uni-oldenburg.de/pubdb_files/pdf/ vknethen98a.pdf.

46. Ingenieria, T.S.e. *Comparative Study between Requirements Management and Engineering Tools*. 2004 [cited 2006 October]; Available from: www.qasystems.de/downloads/deutsch/downloads/downloads_irqa/ComparativeStudyRMEtools.pdf.

47. Rochimah, S., W.M.N. Wan Kadir, and A.H. Abdullah. *An Evaluation of Traceability Approaches to Support Software Evolution*. in *2nd International Conference on Software Engineering Advances*. 2007. France: IEEE Computer Society.