

A Case Study on Multifaceted Requirement Traceability

Siti Rochimah

Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya Indonesia
+62 31 5939214
siti@its-sby.edu

Wan M.N. Wan Kadir

Software Engineering Department
Faculty of Comp. Sci. and Inf. System
Universiti Teknologi Malaysia
+60 7 5532348
wnasir@utm.my

Abdul H. Abdullah

Software Engineering Department
Faculty of Comp. Sci. and Inf. System
Universiti Teknologi Malaysia
+60 7 5532348
hanan@utm.my

ABSTRACT

Requirements traceability can be expensive activities. Many researchers have addressed the problem of requirements traceability, especially to support software evolution activities. Yet, the evaluation results of those approaches show that most of them typically provide only limited supports to software evolution to perform requirements tracing and maintaining the established links. Based on the problems of requirement traceability, we have identified three directions that are important for traceability to support software evolution, i.e. process automation, procedure simplicity, and best results achievement. Those three directions are addressed in our multifaceted approach of requirement traceability. This approach utilizes three facets to generate links between artifacts, i.e. syntactical similarity matching, link prioritization, and heuristic-list based processes. This paper presents the result of preliminary experiment that has been applied in case study to illustrate the ability of the approach in facilitating software evolution, especially for best result achievement in term of highest accuracy. The case study results show that utilization of the three facets in generating the traceability among artifacts is promising in term of its accuracy. The initial results show that accuracy can be achieved up to 93% recall and 61% precision.

Categories and Subject Descriptors

D.2.1 [Requirements/Specifications]: Methodologies, Tools.

General Terms

Algorithms, Design, Experimentation.

Keywords

Requirement traceability, metamodel.

1. INTRODUCTION

Requirements management and especially requirements

traceability can be expensive activities. Conversely, from software evolution viewpoint, requirement traceability is one of importance factor in facilitating software evolution since it maintains the artifacts relationship before and after a change is performed.

Many researchers have addressed the problem of requirements traceability, especially to support software evolution activities. The evaluation results of those approaches show that most of them typically provide only limited supports to software evolution to perform requirements tracing and maintaining the established links [1].

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 [2-4]. Besides, it is found that software traceability is not easy in practice. It is shown that many organizations do not seriously implement traceability practices [5-7]. They feel that traceability process seem too costly for routine use in practice, even when it is supported with tools [6]. In addition, as time-pressured practitioners often fail to consistently maintain links and update impacted artifacts each time a change occurs [8], creating out-of-date links and improper artifacts relationships. It is due to the aspects such as huge amount of software artifacts to be tracked and managed, improper change mechanisms that is used, imprecise implemented traceability approach, or complicated tracing procedures faced by users or maintainers [9].

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 [6], in term of the obtained results. Furthermore, automation alone cannot really reduce efforts of software evolution.

Based on the problems, we have identified three directions that are important for traceability to support software evolution. First, the traceability practices must utilize an appropriate traceability approach to facilitate software evolution process in a significant degree. Second, the traceability approach that is utilized must be able to simplify the traceability practices. Third, the traceability approach that is utilized must consider more than one aspect that has an influence in minimizing software evolution effort (multifaceted approach). The notion of 'multifaceted' that is used in the approach is refer to something that has many facets or aspects [10, 11], a many-sided subject [12], or something that has numerous aspects or attributes [13].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PARS'08, Jul 2-3, 2008, UTM, Johor, Malaysia.
Copyright 2008 ACM 1-58113-000-0/00/0004...\$5.00.

Furthermore, those important aspects can be listed as follows: (i) **process automation** for generating and recovering traceability information; (ii) **procedure simplicity** for a whole tracing process; and (iii) **best results achievement** in term of highest accuracy, completeness, and up-to-date traceability information. Rochimah et al. in [14] have hypothesized the utilization of those three aspects in generating and recovering traceability links to support software evolution.

The aim of this paper is to present the result of preliminary experiment that has been applied in case study to illustrate the ability of the approach in facilitating software evolution, especially for best result achievement in term of highest accuracy. The case study results show that utilization of the three facets in generating the traceability among artifacts is promising in term of its accuracy. The other important aspects for traceability to support software evolution, as previously mentioned, are remain as our future works.

This paper is organized as follows. Section 1 is introduction. Section 2 to 5 describes the multifaceted approach in term of its conceptual framework, facets description, algorithms, and measurements. Section 6 describes the case study application. Section 7 shows the results and its discussion. Finally, Section 8 describes related work followed by conclusion in Section 9.

2. CONCEPTUAL FRAMEWORK

The conceptual framework of multifaceted requirement traceability approach can be found in Figure 1. 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 and other information. All of them are integrated and run on top of a typical ready-to-use Integrated Development Environment (IDE).

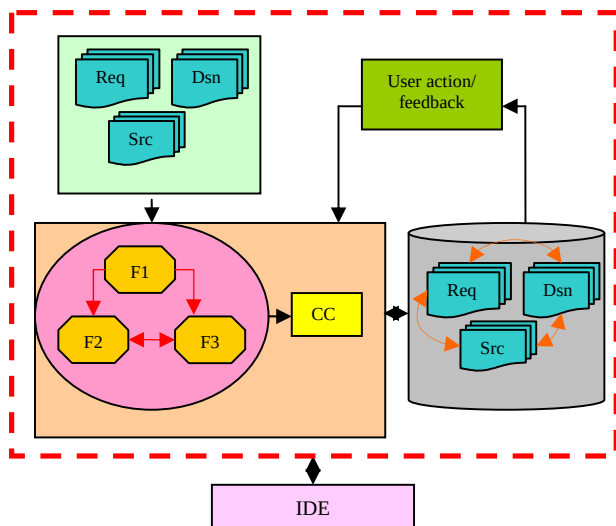


Figure 1. Conceptual Framework

The software artifacts that will be processed in this approach are requirements documents in the form of use case specifications,

design documents in the form of class diagrams, test cases, and source codes. Those four 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 (F-1, F-2, and F-3) and the consistency checker element (CC). The traceability-techniques elements are composed from three different facets that are integrated altogether. F-1 is a syntactical similarity matching process. F-2 is a heuristic-based process, and F-3 is a link prioritization process. Those three facets work in an integrated mode to construct traceability links among artifacts. 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 environment development (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 action / feedback in this approach is related to the inherent characteristic of traceability problem, i.e. 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.

3. DESCRIPTION OF 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-I 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-II is a link prioritization process. This technique is used to refine the results obtained by Facet-I or Facet-III, 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-II, 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-III is a heuristic-lists 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.

4. ALGORITHMS

Traceability generation process can be done in the following algorithms. In general, the algorithms contain three main processes, i.e. pre-processing, generating links, and displaying the results. The algorithms are shown in Figure 2 below.

```

Input artifacts
Pre-processing
  Convert artifacts into indexed format      // to be used for F1
  Extract artifacts information into database // to be used for F2
                                              and F3
  Save pre-processed artifacts into repository
Traceability generation
  // To obtain high recall using F1
  Repeat
    Generate traceability link between artifacts using F1
  Until all artifacts are linked
  // recall and precision are obtained
  // low confidence links are excluded from 'true link set' but
  // are saved in the database
  // To prioritize requirement based on its contribution using F2
  For each listed query:
    Descending sort document based on its weight
  // invariant: for each query, document links are sorted
  // based on pre-defined document category
  // To improve precision (as well as recall) using F3
  Utilize proper heuristic list to separate the 'true links'
  from 'false positive links'
  // heuristic list are specific for each type of artifacts
  // Traceability links have been generated
Display the candidate links to the user
// User evaluation are then needed to justify if the link is 'real' true
link or false positive
Evaluate the generated result to obtain the final result

```

Figure 2. Algorithms of Traceability Generation

The main objective of pre-processing is to prepare artifacts to be ready for use in the subsequent processes. There are three tasks in the artifact pre-processing. The purpose of converting artifacts into indexed format is to make an indexed document of an artifact that is ready for the syntactical similarity matching process. This means preparing the raw document collection into an easily accessible representation of documents. The conversion includes stop-word removal, tokenization, stemming, and weighting. The artifact information extraction is useful to build an easily accessible artifact database that will be used as a lookup information as well as query data, especially for heuristic-list references. All of the pre-processed artifacts have to be saved into the repository.

The traceability generation includes three subsequent tasks representing three facets. The traceability generation process must be started from F1, and the subsequent process can be either from F2 to F3 or from F3 to F2. Link generation using F1 is syntactical similarity matching process, also known as information retrieval technique. We use vector space model [15, 16] to compute a similarity between artifact documents. Document weight is computed using the inverse document frequency (idf) corresponding to a given term:

$$d_{ij} = t_{ij} \times idf_j$$

where d_{ij} is the weight of the j th entry in the vector corresponding to document i ;

t_{ij} is the number of occurrences of term t_j in document D_i , also referred to as *term frequency*; and

idf_j is $\log(d/df_j)$ where d is the total number of documents, and df_j is number of documents which contain t_j , also referred to as *document frequency*. This is the *inverse document frequency*.

The similarity between two documents is then measured using cosine similarity. Given a document vector $d=(w_1, \dots, w_N)$ and a similarly computed query vector $q=(q_1, \dots, q_N)$, the similarity between d and q is defined as the cosine of the angle between the vectors:

$$Sim(d,q) = \cos(d,q) = \frac{\sum_{i=1}^N w_i \cdot q_i}{\sqrt{\sum_{i=1}^N w_i^2 \cdot \sum_{i=1}^N q_i^2}}$$

In order to obtain highest recall, threshold must be set as low as possible, although it will reduce its precision at first stage. Yet, later this precision will be improved by F3 process. Artifact that contains more term is treated as a document and artifact that has less term is treated as query. This is based on the assumption that queries typically are short. The following table is the list of artifact 'role' in F1 process.

It is shown from Table 1 that class diagram is always treated as a query against other artifacts since it usually contains less term than the others. On the contrary, use case is always treated as document since it usually contains huge of terms.

Table 1. Artifact ‘Role’ in F1 Process

Query versus Document	
Class Diagram	Use Case
Class Diagram	Source Code
Class Diagram	Test Case
Source Code	Use Case
Source Code	Test Case
Test Case	Use Case

Link prioritization using F2 is performed by sorting all of the generated links in descending order. For each query corresponding to documents, the following condition must be satisfied: a document that has higher priority must be appeared higher up in the list. Given a generated link from query q to collection of documents $q = (d1, \dots, dN)$, the following invariant must be satisfied in F2:

For each d_i and $d_j, j > i$ if and only if $\text{priority}(d_i) > \text{priority}(d_j)$.

Document priority can be looked up in the artifact database and must be previously determined by user. The generated links are not changed in term of their quantity by this facet.

The objective of F3 is to improve recall and precision of the generated links by applying proper heuristic lists that are specific for each type of artifacts. By this facet, the generated links might be changed in term of their quantity as well as their content. In general, heuristics lists include two main categories, i.e. heuristics list to improve recall and heuristics lists to improve precision. Each of them is specific for each type of artifacts. The complete heuristics lists of all artifacts are saved in the repository.

The following example is heuristics list of class diagram as query to improve recall.

For each query q that has no link to any document:

- check member association in the association table
- add all of the links of its association-member to its links.

If a class c (in this case as query q) has an association with another class e , then this heuristic list will add some candidate links which are belonged to e , into class c . Thus, the candidate links will be increased.

Another example of a heuristic list of class diagram as query to improve precision is shown in the Table 2 below. This table can be read as follows. Take an example of heuristics A of the table. For each link l from class c to use case u , the link must satisfy the condition that:

- a class name is related to a use case name OR
- a class name is related to an actor OR
- a class name is related to steps OR
- a class attribute is related to pre-condition OR
- a class attribute is related to steps OR
- a class attribute is related to post-condition OR
- a class method is related to description OR
- a class method is related to steps.

If a link fails to satisfy one of those conditions then the link can be considered as false link.

Table 2. Example of Heuristic Lists of Class Diagram

A	Class Diagram	Use Case
	Class name	Use case name
		Actor
		Steps
	Class attribute	Pre-condition
		Steps
		Post-condition
	Class method	Description
		Steps
B	Class Diagram	Source Code
	Class name	Class Id
	Class attribute	Attribute Id
	Class method	Method Id
C	Class Diagram	Test Case
	Class name	Test case name
		Test description
		Test procedure
	Class attribute	Input
		Test procedure
		Expected output
	Class method	Test case name
		Test description
		Test procedure

After all traceability links have been automatically generated, the results are then displayed to the user. At this stage, the user must finally evaluate the results based on her/his knowledge about the systems.

5. ACCURACY MEASUREMENT

The standard measures of accuracy are recall and precision. Recall is a coverage measure. Given the theoretical ‘true trace’ or ‘answer set’, it measures the percentage of true links retrieved.

$$\text{Recall} = \frac{C}{C + M} \times 100\%$$

where C is the number of correct retrieved links
 M is the number of correct missed links.

$$\text{Precision} = \frac{C}{C + F} \times 100\%$$

where C is the number of correct retrieved links
 F is the number of retrieved false positives.

Accuracy can be measured objectively only when there is already a theoretical ‘true trace’ provided. In case of the absence of ‘true trace’, the user can build such a ‘true trace’ at the end of the evaluation process. The accuracy is then measured by comparing the candidate link retrieved to the final ‘true trace’ obtained from the evaluation process. Thus, s(he) can finally get the recall and precision of the results.

6. CASE STUDY APPLICATION

The case study that we apply to experiment the approach is the Asset and Facility Management System (AaFMS) owned by Universiti Teknologi Malaysia (UTM). It is an application for recording, tracking, maintaining, and disposing assets that are owned by UTM. Assets are divided into two categories, i.e. physical and intangible assets.

- ◆ Physical Assets - Those are required to maintain regularly
- ◆ Intangible Asset - Those are required to maintain when a problem is occurred

AaFMS assists users to make complaint on their asset. All the complaints will be processed online, so that the complaint will be settled in short time. The maintenance worker will record the work progress on asset and users can view the work progress on their asset. Besides, AaFMS assists user to make decision in disposing their assets. Base on maintenance history, AaFMS will calculate the asset FCI so that user can compare the cost of buying and the cost of maintenance.

In general, Figure 3 shows the basic conceptual business flow of existing asset management system.

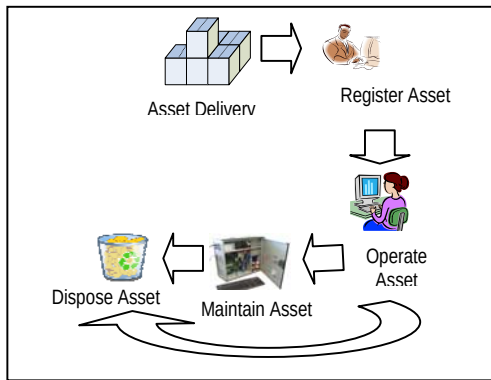


Figure 3. Asset Life Cycle

AaFMS is relatively small systems. It contains 11 use cases and 19 classes. Table 3 shows the example of use case.

Table 3. Example of AaFMS Use Case

1. Use case Register Asset	
Description	The asset needs to be registered by User after delivered to her/him
Category	Core Business
Actor	UTM, Supervisor
Pre-condition	Asset has not been registered form
Steps	1. user login 2. user verification 3. display asset form 4. If asset = Building then Insert building property and identification Else If asset = Space then Insert Space property and identification Else Insert equipment property and id 5. save the record into database
Post-condition	Asset has been registered

We have applied the approach to the case study to experiment the ability of the approach to support software evolution. We applied the approach semi manually using a popular spreadsheet and word processor to the case study. This is possible since we use a relatively small case study. In this experiment, we applied the algorithms to create traceability link between use cases and class diagram which contains 19 classes.

We made 19 queries from 19 classes, and 11 documents from 11 use cases. The queries are then matched to the documents using three facets according to the algorithms that have been defined. In order to easily obtain the accuracy results, we determine theoretical true trace (answer set) between both of artifacts. We use threshold of 0.1, 0.05, 0.03, and 0.02 in F1 to filter the results. However, F2 and F3 are only applied in the lowest threshold since the results is the best one in term of highest recall and precision.

7. RESULTS AND DISCUSSION

Table 4 shows the results of applying case study to the approach.

Table 4. Results of Case Study Application

Threshold	Recall (%)	Precision (%)
0.1	40	72
0.05	61	65
0.03	73	66
0.02	79	63
0.02 (plus F2 and F3)	93	61

It is shown from the table that recall increases when threshold is turn down. This is matched with previously concept that threshold must be set as low as possible to get the highest recall. However, precision is reduced while recall is increased. Yet, precision reduction still gives considerable results. As a simple example, precision for at least 50% means that for every 1 retrieved link, there are 1 false positive. Thus, if the precision is higher than 50%, it means that there are several, around 11%, true links without noise are retrieved.

It is also shown that, with the same threshold, recall significantly increases from 79 to 93, when F2 and F3 is applied, although precision reduced in a small value. It means that increasing 14% recall will only reduce 2% precision.

It is noted from the experiment that 100% recall can not be achieved even when the threshold is set to 0. Our investigation into the artifacts indicates that there are number of classes that have no term related to any use cases although those classes have a semantic relationship to the use cases. It means that the semantic relationship is not supported by its syntactical term.

8. RELATED WORKS

Many researchers have addressed the problem of requirements traceability. They have conducted experiments to test their approach.

Antoniol et al. apply two basic IR models mentioned above for creating traceability link between source code and free text

documentation [17]. Hayes et al. perform a thorough work on implementing various IR techniques for requirement tracing to support independent verification and validation process [18-22]. Marcus and Maletic describe method for creating traceability link using LSI [23]. They apply the method to generate traceability between source code and free text documentations. Settimi et al. compare several IR models in generating traceability from requirements to UML artifacts, source code, and test cases [24]. Cleland-Huang et al. in [25] followed by Zou et al. in [26, 27] propose various enhancement strategies to IR models based on probabilistic network model that previously worked by Wong and Yao [28, 29]. Spanoudakis et al. propose a method to automatically create traceability link using rules [30, 31]. Nentwich et al. build rule-based tool for consistency management for XML-based software artifacts, namely 'xlinkit' [32]. Heindl and Biffl propose value-based requirement tracing [6]. They focus on applying case study in real-life project setting to measure the effectiveness of the approach in term of reduced cost/effort comparing with full tracing. Egyed et al. propose scenario-based approach [33-35]. They use hypothesized trace information that have to be manually entered. Ibrahim et al. propose document-based traceability to support software change impact analysis, which basically adopt scenario-based approach to obtain the relationship among artifacts [36, 37]. They extend the functionality by utilizing those relationships to provide software traceability as well as for software change impact analysis visibility of different artifacts.

9. CONCLUSION AND FURTHER WORK

In this paper, we have reported the application of our proposed traceability approach to a case study. We have shown the results that are produced from the case study application. The initial results indicate that our proposed approach is promising in term of its accuracy. Our future work will be to conduct further experiment to test other aspects that we have defined, and also to build a supporting tool as well.

10. ACKNOWLEDGMENTS

Our thanks to Islamic Development Bank, Universiti Teknologi Malaysia, and Institut Teknologi Sepuluh Nopember Surabaya for their supports.

11. REFERENCES

- [1] 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.
- [2] Carrol, P.B., *Computer Glitch: Patching up Software Occupies Programmers and Disables Systems*. Wall Street Journal, 1988.
- [3] Hanna, M., *Maintenance Burden Begging for a Remedy*, in *Datamation*. 1993. p. 53-63.
- [4] Pfleeger, S.L., *Software Engineering: Theory and Practice*. 1998: Prentice-Hall.
- [5] Lormans, M., et al. *Managing Evolving Requirements in an Outsourcing Context: An Industrial Experience Report*. in *International Workshop on Principles of Software Evolution (IWPE04)*. 2004.
- [6] 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.
- [7] Gotel, O. and A. Finkelstein. *Extended Requirement Traceability: Results of an Industrial Case Study*. in *Third IEEE International Symposium on Requirements Engineering*. 1997: IEEE Computer Society.
- [8] 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.
- [9] Sherba, S.A., *Towards Automating Traceability: An Incremental and Scalable Approach*, in *Department of Computer Science*. 2005, University of Colorado: Colorado.
- [10] *Miriam Webster Online Dictionary*. [cited 2007; Available from: <http://mw1.merriam-webster.com/dictionary/>].
- [11] *Oxford Dictionary Online*. [cited 2007].
- [12] *Online Free Dictionary*. [cited 2007].
- [13] www.hudsonvalley.org. [cited 2007; Available from: www.hudsonvalley.org/education/Background/Vocabulary/vocabulary.html].
- [14] Rochimah, S., W.M.N. Wan Kadir, and A.H. Abdullah. *Introducing a Multifaceted Traceability Approach to Support Software Evolution*. in *Postgraduate Annual Research Seminar 2008 (PARS'08)*. 2007. Universiti Teknologi Malaysia: FSKSM UTM.
- [15] Salton, G., C.S. Yang, and A. Wong, *A Vector Space Model for Automatic Indexing*. *Communication of the ACM*, 1975. 18(11): p. 613-620.
- [16] Grossman, D.A. and O. Frieder, *Information Retrieval Algorithms and Heuristics*. 2nd ed, ed. Springer. 2004: Springer Verlag.
- [17] Antoniol, G., et al., *Recovering traceability links between code and documentation*. *IEEE Transactions on Software Engineering*, 2002. 28(10): p. 970-983.
- [18] 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).
- [19] 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.
- [20] 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.
- [21] Hayes, J.H., et al. *Helping analysts trace requirements: An objective look*. in *Proceeding of 12th IEEE International Requirements Engineering Conference (RE 2004)*. 2004.
- [22] 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.
- [23] 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.
- [24] 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.
- [25] 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.
- [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] Zou, X., et al. *Poirot:TraceMaker: A Tool for Dynamically Retrieving Traceability Links*. 2005 [cited 2006 September]; Available from: <http://re.cti.depaul.edu/Poirot.pdf>.
- [28] Wong, S.K.M. and Y.Y. Yao, *On Modeling Information Retrieval with Probabilistic Inference*. *ACM Transactions on Information Systems*, 1995. 13(1): p. 38-68.
- [29] Wong, S.K.M. and Y.Y. Yao, *A probabilistic inference model for information retrieval*, in *Information Systems*. 1991. p. 301-321.
- [30] Spanoudakis, G. *Plausible and Adaptive Requirement Traceability Structures*. in *Proc. 14th Int'l Conf. Software Eng. and Knowledge Eng.* 2002.
- [31] Spanoudakis, G., et al., *Rule-Based Generation of Requirements Traceability Relations*. *Journal of Systems and Software*, 2004: p. 105-127.
- [32] Nentwich, C., et al., *xlinkit: A Consistency Checking and Smart Link Generation Services*. *ACM Transactions on Internet Technology*, 2002. 2(2): p. 151-185.
- [33] Egyed, A. *A Scenario-Driven Approach to Traceability*. in *23rd International Conference on Software Engineering*. 2001. Toronto, Ontario, Canada: IEEE Computer Society.
- [34] 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.
- [35] Egyed, A. and P. Grunbacher, *Supporting Software Understanding with Automated Requirements Traceability*. *International Journal of Software Engineering and Knowledge Engineering*, 2005. 15.
- [36] Ibrahim, S., M. Munro, and A. Deraman. *Implementing a Document-based Requirements Traceability: A Case Study*. in *IASTED International Conference on Software Engineering*. 2005.
- [37] Ibrahim, S., *A Document-Based Software Traceability to Support Change Impact Analysis of Object-Oriented Software*. 2006, Universiti Teknologi Malaysia: Kuala Lumpur.
- [38]