Multifaceted Requirement Traceability Approach to Support Software Evolution

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
Software evolution is inevitable. When a system evolves, there are certain relationships among software artifacts that must be maintained. Requirement traceability is one of importance factor in facilitating software evolution since it maintains the artifacts relationship before and after a change is performed. 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. 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 utilization of multifaceted approach to traceability generation and recovery in facilitating software evolution process. We describe our new approach in traceability called Multifaceted Requirement Traceability Approach (MRTA) to support software evolution. We present the conceptual framework as well as the design of this new approach. We have applied the tool in real case study to measure accuracy of this approach. Also, a controlled experiment using MRTA Tool has been conducted to measure its usability.

Categories and Subject Descriptors
D.2.1 [Requirements/Specifications]: Methodologies, Tools.

General Terms

Keywords
Requirement traceability, metamodel.

1. INTRODUCTION
Traceability 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 important factor in facilitating software evolution since it maintains artifacts relationship before and after a change is performed. Many researchers have addressed problem of 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, 2].

Research has shown that evolution and maintenance are the most expensive activities in software process, consuming 60% to 80% of total time spent on a software system [3-5]. Yet, it is found that traceability is not easy in practice. It is shown that many organizations do not seriously implement traceability practices [6-8]. They feel that traceability process seem too costly for routine use in practice, even when it is supported with tools [7]. In addition, as time-pressured practitioners often fail to consistently maintain links and update impacted artifacts each time a change occurs [9], creating out-of-date links and improper artifacts relationships. It is due to 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 [10]. For this reason, often traceability is completely absent during software development, even though it is a very important element in development process.

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 the approaches aim to automate requirements tracing, but tracing automation is still complex and error prone [7]. Furthermore, automation alone cannot really reduce efforts of software evolution. Hence, currently, traceability problems are still become a challenging issue for many researchers and practitioners [11].

However, based on those problems, we have identified three directions that are important for traceability to support software evolution. First, traceability practices must utilize an appropriate traceability approach to facilitate software evolution process in a significant degree. Second, traceability approach that is utilized must be able to simplify traceability practices. Third, 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’ is refer to something that has many facets or aspects [12, 13], a many-sided subject [14], or something that has numerous aspects or attributes [15]. Thus, multifaceted traceability means traceability that utilizes more than one aspect to generate links. Furthermore, those important aspects can be listed as follows:
• process automation for generating and recovering traceability information,
• procedure simplicity for a whole tracing process,
• best results achievement in term of highest accuracy, completeness, and up-to-date traceability information.

This paper discusses utilization of our traceability approach called Multifaceted Requirement Traceability Approach (MRTA) to traceability generation and recovery in facilitating software evolution process. We use ‘requirement traceability’ instead of ‘traceability’ since we consider requirement as an origin of change that may be occurred in the systems. This paper presents the design and algorithm of traceability generation and recovery performed by MRTA.

The remains of this chapter are organized as follows. Section 2 describes the objectives of this research. Section 3 describes the methodology. Section 4 explains the result and its discussions. Section 5 describes the contribution of this research and section 6 describes conclusion and further works.

2. OBJECTIVES
This research encompasses a set of objectives that is associated with milestones of the research process. The research objectives are mentioned below.

(i) To investigate current traceability approaches to support software evolution.
(ii) To develop new traceability approach that integrates multiple important aspects into consideration (MRTA).
(iii) To develop supporting prototype tool that supports MRTA.
(iv) To demonstrate capability of MRTA to support software evolution process.
(v) To evaluate effectiveness of MRTA based on traceability results obtained by the tool.

3. METHODOLOGY

3.1 General Research Procedure
This research develops a new traceability approach to support software evolution (called MRTA). Producing an approach is an engineering problem. Thus, this research is performed in an engineering design, i.e. modeling, constructing, and evaluating new object. The complete research procedure is then accomplished in the steps as illustrated in Figure 1. The operational framework that is used to control the milestone during the research process is shown in Table 1.

![Figure 1. Research procedure.](image-url)
Table 1. Operational framework.

<table>
<thead>
<tr>
<th>No.</th>
<th>Research Question</th>
<th>Objective</th>
<th>Activity</th>
<th>Deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>Why the current traceability approaches are still not able to provide significant support for software evolution?</td>
<td>To investigate current software traceability approaches to support software evolution</td>
<td>Building an operational framework - Literature study - Comparative evaluation on existing approaches</td>
<td>Operational framework - Results of comparative evaluation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ii)</td>
<td>What is the best way to generate and recover the traceability of the software artifacts to support software evolution?</td>
<td>To develop new software traceability approach that integrates multiple important aspects into consideration</td>
<td>Building a conceptual framework - Building a meta model</td>
<td>Conceptual framework - Meta model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iii)</td>
<td>How to measure the generated (or recovered) traceability of the software artifacts?</td>
<td>To demonstrate the capability of the proposed software traceability approach to support software evolution process</td>
<td>Designing the prototype tool - Coding - Packaging the tool</td>
<td>Design documentation - Source code - Executable tool</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iv)</td>
<td>How to validate the usefulness of the proposed approach to support software evolution process in a significant degree?</td>
<td>To evaluate the effectiveness of software traceability approach based on the traceability results obtained by the tool</td>
<td>Conducting a controlled experiment - Measuring the result obtained by the tool using the traceability metrics - Analyzing the results</td>
<td>Usability values - Analysis results</td>
</tr>
</tbody>
</table>

Validation process of recall and precision metrics are performed through a series of experimentation of MRTA by applying in two real case studies upon the tool. Validation process of usability metrics is performed through a controlled experiment using 20 users to operate MRTA Tool.

3.2 Conceptual Framework

The conceptual framework of MRTA can be found in Figure 2. There are three main involved elements, i.e. software artifacts in which their traceability will be created and managed, main traceability generator which creates, recovers, and manages traceability among artifacts, and traceability repository which records linked artifacts and other information. All of them are run on top of a typical ready-to-use CASE Tool.

Software artifacts that are processed in this approach are requirement document in the form of use case specification, design document in the form of class diagram, test case, and source code. Those four types of artifacts may represent the smaller scope of a large complete system.

MRTA traceability generator contains two sub-elements for the process, i.e. traceability-techniques element (F1, F2, and F3) and consistency checker element (CC). Traceability-techniques element is composed from three different facets that are integrated altogether. F1 is a syntactical similarity matching process. F2 is a link prioritization process, and F3 is a heuristic-list based process. Those three facets work in an integrated mode to construct traceability links among artifacts. The second element of traceability generator is consistency checker which ensures that all artifacts are in proper configurations time to time, especially after a change has been made.

![Figure 2. MRTA conceptual framework.](image-url)

Traceability repository records all 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 artifacts traceability information that is queried by user.
Those three main elements that compose the whole traceability approach are run on top of a particular ready-to-use CASE Tool in order to facilitate traceability generation and recovery as well as to ease artifacts maintenance. This is due to the facts that artifacts are usually created and maintained in a particular CASE tool.

The role of user’s action/feedback in this approach is related to inherent characteristic of traceability problem, i.e. user participation in 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, automation is merely to minimize the effort required by user.

3.3 Description of Facets
MRTA integrates three different processes. This integration is utilized to take multiple aspects into account while generating traceability links. The traceability generation process must be started from F1, followed by F2 and F3. Figure 3 shows traceability generator in a more detail view and description of those facets is discussed below.

![Figure 3. MRTA traceability generator.](image)

3.3.1 F1: Syntactical Similarity Matching Process
F1 is a syntactical similarity matching process, also known as information retrieval technique. 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.

We use vector space model [16, 17] to compute similarity between artifact documents. A document whose vector is closer to the query vector in this model is scored higher. The similarity between query \( q \) and document \( d \) is computed as follows:

\[
Sim(q,d) = \sum_{t \in q} \left( \sqrt{tf(d)} \cdot idf(t) \cdot \frac{1}{\sqrt{nt}} \right)
\]

Where: \( t \) is term contained in query \( q \). 
\( tf(d) \) is term frequency, that is the number of times term \( t \) appears in the document \( d \).
\( idf(t) \) is the inverse of document frequency. Document frequency is defined as the number of documents in which the term \( t \) appears.
\( nt \) is the total number of term contained in document \( d \). This is used as document length normalization.

We use:

\[
idf(t) = 1 + \log \left( \frac{Nd}{df + 1} \right)
\]

Where: \( Nd \) is total number of document. 
\( df \) is document frequency.

In order to obtain highest recall, threshold must be set as low as possible. 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 [17]. Table 8 contains the list of artifact ’role’ in F1 process. It is shown that class diagram is always treated as a query against other artifacts since it usually contains less term than the others. On the contrary, source code is always treated as document since it usually contains huge of terms.

3.3.2 F2: Link Prioritization Process
F2 is a link prioritization process. This technique is used to refine the results obtained by F1 or F3, depending on process sequence that is chosen. This facet will prioritize links according to the set of defined priority value. Defined priority value is the priority value that is assigned to each of requirement based on its contribution or criticality in the whole project. For example, requirement that is closely related to application core should be assigned a higher priority value than others, or, requirement that is related to government regulation (must be obeyed) should be assigned a high value as well. Defined priority value of each requirement or group of requirements is included in use case specific document and has to be recognized prior to traceability generation. Every generated link is then examined according to its relation to requirement (its parent). F2, thus, play an important role in defining which links have to be firstly prioritized to be ordered in evaluation, based on its priority value. This prioritization is crucial in order to minimize effort of link evaluation conducted by user, since only links that have high value is mandatory to be firstly evaluated, while links with low value are optional.

Link prioritization is performed by sorting all 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, \( q = (d_1, ..., d_n) \), the following invariant must be satisfied:

\[
\forall \ (d_i, d_j), j > i \ if \ and \ only \ if \ priority(d_j) > priority(d_i)
\]

Document priority can be looked up in artifact database and must be previously determined by user. For example, priority of
requirement can be identified from Category field that exists in the requirement specification, as shown on Figure 4 below. Use case Register Asset is indicated to be in high priority requirement since it is a core business transaction in this project. Other priority types are Supplementary, or Optional. User may define his/her own set of priority of requirement prior to traceability generation. The generated links are not changed in term of their quantity by this facet.

<table>
<thead>
<tr>
<th>1. Use case Register Asset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description The asset needs to be registered by User after delivered to her/him</td>
</tr>
<tr>
<td>Category Core Business</td>
</tr>
<tr>
<td>Actor UTM, Supervisor</td>
</tr>
<tr>
<td>Pre-condition Asset has not been registered form</td>
</tr>
<tr>
<td>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 the Insert Space property and identification Else Insert equipment property and identification 5. save the record into database</td>
</tr>
<tr>
<td>Post-condition Asset has been registered</td>
</tr>
</tbody>
</table>

Figure 4. Example of high priority use case specification.

3.3.3 F3: Heuristic-list Based Process

F3 is a heuristic-list based process. In order to maximize precision of generated links, each of candidate links has to be evaluated using heuristic-list in the form of look-up table and simple statement to filter which is '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. By this facet, generated links may be changed in term of their quantity as well as their content.

Actually, there are two types of heuristic-list. The first type is the main heuristic-list look-up table with two-column in size, and is used in all types of artifact. This look-up table is utilized to filter result from noise, thus it acts as a heuristic to improve precision. It acts like a simple relationship rule to relate one artifact to another. For example, a relationship 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 condition in heuristic-list look-up table, then the link can be indicated as true link.

Table 2 below shows part of look-up table for all types of artifact. There are six groups of heuristic-list of relationship between two artifacts, named A, B, C, D, E, and F. This table can be read as follows. Taken heuristic A as an example, we can read the relationship rule between class-diagram and use case specification in this way:

for each link / from class c to use case u, the link must satisfy the condition that:
- a class name in c is related to a use case name in u OR
- a class name in c is related to an actor in u OR
- a class name in c is related to steps in u OR
- a class attribute in c is related to pre-condition in u OR
- a class attribute in c is related to steps in u OR
- a class attribute in c is related to post-condition in u OR
- a class method in c is related to description in u OR
- a class method in c is related to steps in u.

If a link fails to satisfy one of those conditions then the link can be considered as false link. The complete heuristics lists of all artifacts are saved in the repository. User may modify this default table in order to maximize the results according to his/her needs. If user feels that there are lots of similarities word in all artifacts then (s)he can reduce the number of relationship in each heuristic-list of the table. For example, taken heuristic-list A, (s)he may discard 'Steps' from the relationship to increase precision of the results.

Table 2. Example of heuristic-list look-up table.

<table>
<thead>
<tr>
<th>A</th>
<th>Class diagram</th>
<th>Use case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class name</td>
<td>Use case name</td>
<td></td>
</tr>
<tr>
<td>Class attribute</td>
<td>Pre-condition</td>
<td></td>
</tr>
<tr>
<td>Class method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Class diagram</td>
<td>Code</td>
</tr>
<tr>
<td>---</td>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td>Class name</td>
<td>Class Id</td>
<td></td>
</tr>
<tr>
<td>Class attribute</td>
<td>Attribute Id</td>
<td></td>
</tr>
<tr>
<td>Class method</td>
<td>Method Id</td>
<td></td>
</tr>
</tbody>
</table>

The second type of heuristic-list is in the form of a simple statement. This statement is used specifically for class diagram, which always be treated as query, to improve recall. The following figure is a simple statement that has to be satisfied when class diagram acts as a query to other artifacts.

For each query q that has no link to any document: check member association of q in the association table, add all of the links of its association-member to q’s links.

Figure 5. Heuristic-list as a simple statement specific for class diagram.

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

3.4 MRTA Elements and Process Flow

At design level, MRTA uses seven elements to perform traceability tasks, i.e. Artifact Editor, Converter, Pre-processor,
Traceability Generator, Analyzer, Consistency Checker, and Versioning Checker or Change Log. Each of them is described below.

Artifact Editor is utilized to create artifacts in graphical as well as textual format. This editor is embedded in a particular ready-to-use CASE Tool, so that any creation or modification on the artifacts can be done in an easy way. Converter is an element to convert artifact format from original CASE Tool format into textual format which will be needed for further process. Pre-Processor performs indexing textual artifact format into indexed format. This index format is needed in F1 process. Traceability Generator is a main element in MRTA to generate and recover traceability. Analyzer is an element that is used by user to analyze traceability result from generator into final true traceability. Consistency Checker is used to check consistency among artifacts especially upon a change has been made on artifact. Last, Versioning Checker is used together with Analyzer to assist user in determining the final true link especially after a change has been made on the artifact. The process flow of MRTA is shown in Figure 6 below.

Besides, during traceability generation, artifacts are converted into formats which are suitable for the process. There are five types of artifacts that are involved in the whole process. We named them as X1, X2, X3, X4, and X5 format. Each type of artifact is described as follows:

- format X1: artifact in original format, can be textual or graphical format (from CASE Tool),
- format X2: temporary artifact file in text format (*.txt and *.java),
- format X3: artifact indexed file (from textual indexing Tool),
• format X4: link candidates (indexed file format), and
• format X5: final true links (indexed file format)

3.5 MRTA Architecture
MRTA uses two main packages for deployment of the tool. The first package is MRTA Graphical Editor. This package is responsible for providing artifact in original CASE Tool format.

The second package is the main MRTA Tool. Six elements of MRTA are deployed in this package. Both of these packages are run on top of (embedded in) particular ready-to-use CASE Tool. Repository is build separated from these two packages for independency reason. Figure 7 below shows the MRTA Architecture.

3.6 Accuracy Measurement
The standard measures of accuracy are recall and precision [16, 17]. Recall is a coverage measure. Given the theoretical ‘true trace’ or ‘answer set’, it measures the percentage of true links retrieved.

Recall = \( \frac{C}{C+M} \times 100\% \) \hspace{1cm} (4)

where \( C \) is the number of correct retrieved links, \( M \) is the number of correct missed links.

Precision = \( \frac{C}{C+F} \times 100\% \) \hspace{1cm} (5)

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’, user can build such a ‘true trace’ at the end of the evaluation process. The accuracy is then measured by comparing 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.

3.7 MRTA Implementation
MRTA Tool is implemented using Java programming language. An open source GME is selected as a CASE tool to create artifacts in graphical format. This GME uses meta-modeling as a basic process to build any model upon it. In this research, we build meta-model named ArtifactEditor.mga to create use case diagram, class diagram, and test case in GME. Once artifacts have been created, a tailor-made Java component named GraphicToText extracts their textual parts into text files.

Thus, there are two types of tools that are used to create software artifacts along with their requirement traceability using Multifaceted Requirement Traceability Approach (MRTA), i.e.:

(a) Generic Modeling Environment (GME)
GME is an open source CASE Tool built by Vanderbilt University. It is the tool to make any model through creating its meta-model first. In MRTA, the meta-model for creating software artifacts model is named ArtifactEditor.mga. Using this meta-model, software artifacts can be created in the form of use case diagram, class diagram, and test case. After all artifacts have been created, a tailor-made Java component named GraphicToText, which is embedded in this GME,
extracts (convert) their textual parts into text files to be further processed.

(b) MRTA Tool
MRTA Tool is used to create traceability between artifacts. It reads text files that are produced by GraphicToText artifact converter, makes the indexes, and generates traceability among them. Finally, the generated traceability links have to be analyzed by user in order to justify whether the link is true link or not.

The diagram below show how this process works:

![Diagram of MRTA implementation](image)

**Figure 7. MRTA implementation.**

### 3.8 MRTA User Interface
Artifact that is created in GME Editor using ArtifactEditor.mga meta-model can be easily created and modified. Below is an example of user interface to create the artifact.

![Example of GME user interface using ArtifactEditor.mga meta-model](image)

**Figure 8. Example of GME user interface using ArtifactEditor.mga meta-model.**

Once the artifacts have been created, user can extract the textual part of the diagram using GraphicToText that acts as a converter. This component is embedded in GME, and is represented as a Java Bean icon on top of GME screen.

![Java Bean icon](image)

**Figure 9. Java Bean icon representing GraphicToText artifacts converter.**

The MRTA repository is represented as a predefined directory contains all types of files involved in the project.

![MRTA Repository](image)

**Figure 10. MRTA Repository.**

MRTA Tool which is the main traceability tool reads text files from repository and then generate or recover the artifacts traceability. Below is an example of MRTA Tool user interface.

![Example of MRTA Tool user interface](image)

**Figure 11. Example of MRTA Tool user interface.**

### 4. CASE STUDY AND CONTROLLED EXPERIMENT
A real case study has been applied in MRTA Tool to obtain recall and precision values. This case study, called Integrated Facility and Asset Maintenance and Management Systems (IFAMMS) owned by Universiti Teknologi Malaysia is chosen because of the nature of its business environment whereas rapid changing of user or customer requirements is one of the main factors. IFAMMS has
19 classes connected in one class diagram, 15 use cases, 13 test cases, and about 15 classes bundled into two packages source codes. We use 20 iterations using filters ranged from 0.0 as a minimum to 0.5 as a maximum. Those filters are used to obtain 20 pairs of recall and precision for F1 process and 20 pairs for combination F1, F2, and F3 processes. Some of the results are shown below (Figure 12 until Figure 15).

![Figure 12. Result of Use Case vs Class Diagram.](image)

![Figure 13. Result of Class Diagram vs Test Case.](image)

![Figure 14. Result of Class Diagram vs Source Code.](image)

![Figure 15. Result of Test Case vs Source Code.](image)

A controlled experiment involving 20 users has been conducted to obtain usability value of the metric. The users vary in their experience although currently they are all UTM students. Below is the illustration of user’s background.

### Table 3. Cross tabulation of user’s experience versus frequencies.

<table>
<thead>
<tr>
<th>Working Experience is Software Engineering Fields</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>No experience at all</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Less than 1 year</td>
<td>8</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>1 - 3 years</td>
<td>4</td>
<td>20</td>
<td>65</td>
</tr>
<tr>
<td>4 - 6 years</td>
<td>3</td>
<td>15</td>
<td>80</td>
</tr>
<tr>
<td>more than 6 years</td>
<td>4</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4. Cross tabulation of user’s previous jobs versus frequencies.

<table>
<thead>
<tr>
<th>Previous Job</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer</td>
<td>5</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Software Engineer/ System Analyst</td>
<td>5</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>Project Leader/ Project manager</td>
<td>1</td>
<td>5</td>
<td>55</td>
</tr>
<tr>
<td>Lecturer/ Academician</td>
<td>3</td>
<td>15</td>
<td>70</td>
</tr>
<tr>
<td>Student</td>
<td>6</td>
<td>30</td>
<td>100</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questioners are distributed over 20 users after they use MRTA Tool. There are two questioners in which user must fill in. The first is questioner to evaluate the usability of Artifact Editor and Converter which are embedded and run from GME, and the second is to evaluate MRTA Tool as an artifact traceability generator. They have to evaluate the tool in term of its usability especially to support software evolution tasks. Below are some of the results.
5. RESULT AND DISCUSSION
In general, from case study application, it is obviously shown that using multifaceted approach (F1, F2, and F3) always produce higher or at least the same recall and precision values than using F1 only. From controlled experiment result, it is shown that the usability of the approach reaches about 65% and over 50% useful for each feature of the approach.

6. CONTRIBUTION
This research has made a novelty approach in generating and recovering software artifacts traceability, especially to support software evolution tasks.

7. CONCLUSION AND FURTHER WORK
In this paper, we have presented one of new approach in software traceability. We have described our traceability approach named Multifaceted Requirement Traceability Approach (MRTA). Also, we have describe the results of case study application and also from controlled experiment. The conclusion is that MRTA is a new traceability approach that can be used to support software evolution tasks in a significant degree. Our future work will be to apply this approach in more real case studies to refine our results.

8. REFERENCES


