Tool Support for Traceability of Software Artefacts

K. Kamalabalan, T. Uruththirakodeeswaran, G. Thiyagalingam, D. B. Wijesinghe, I. Perera, D. Meedeniya
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka

D. Balasubramaniam
School of Computer Science, University of St Andrews, UK

***Abstract*—Artefact management in a software development process is a challenging problem. Often there is a wide variety of artefacts, which are maintained separately within a software development process, such as requirement specifications, architectural concerns, design specifications, source codes and test cases, which are essential to software engineering. Artefact inconsistency is a major problem since these artefacts evolve at different rates. Maintaining traceability links among these artefacts and updating those artefacts accordingly can be a solution to address artefact inconsistency. There is a need for establishing these artefact traceability links in semi-automatic way. Proper management and visualization tool is required for effective software artefact management in an incremental software development. We present a prototype tool to establish artefact traceability links and visualization of those. This paper describes the research methodology and relevant research carried out for semi-automatic traceability link establishment and visualization of software artefacts.**

Keywords—Artefacts; Traceability Links; Artefact Traceability Visualization; Artefacts Consistency Management

I. INTRODUCTION

Software development lifecycle consists of different process that result in different artefacts such as requirements specification, software architecture, design specification, source code, test cases for verification and validation of the system, and etc. They are usually maintained in isolation and evolve at different rates. Even though all of these artefacts are aimed at facilitating software products developed with the expected quality parameters and fulfilling functional and domain requirements, once produced they often are subjected to different priority levels for their maintenance; some artefacts are hardly maintained and updated as the project progresses whereas certain high priority artefacts, such as the source code, are regularly updated and maintained. This can lead to artefacts rapidly becoming inconsistent with one another and losing their value for development and documentation purposes [1]. Because of this artefact consistency management is a complex challenge in software engineering.

Research on artefact consistency management is an important area of study in software engineering. It is getting popular attracting many researchers who are interested in different aspects of the artefact consistency [2]. Research studies are currently being carried out to tackle software artefact inconsistency problem. A main area of study to solve this problem is the establishment of traceability links among software artefacts. For example, we may wish to indicate that a particular requirement is the reason for the existence of certain design elements in the design specification and such design element can result in a particular type of code snippet thereby creating a faint link between the three artefacts. This may be more straightforward in cases where a generative approach such as Model Driven Development is used to produce one artefact from another, since a mapping is likely to be created and can be used to establish and maintain links in between. However, this single process of generating an artefact from another, which allows clear and easily manageable traceability links, is not always feasible and links may have to be established between existing artefacts and between existing and new artefacts. The default way of doing this is to define the inter-artefact links manually, which is a labour-intensive and potentially error-prone process. Therefore, we establish our main objective of this research as to explore the semi-automatic identification and specification of traceability links among the various software artefacts.

Effective artefact relationship management is a key point for the success of the software process being used for the development. For modelling and storing these artefact relationship links we used graph database approach, which has already been identified as a very effective mechanism for managing unstructured, dynamic relationships and they are especially suited well to store data that already has a graph like structure. These databases natively store graphs with their nodes and edges and offer querying technologies [3]. We use neo4j [4] graph database for relationship modelling, which is one of the most widely used graph databases written in Java; it supports other languages as well. Information on different artefacts, such as requirements, design diagrams and source code, is extracted and stored in a graph database called Neo4j. Graph databases enable data to be stored in a graph structure as nodes and edges. Nodes in this database represent artefact elements such as requirement descriptions and methods in a class. Edges denote relationships among those nodes. This data set is used to analyse the impact of changes to artefacts and to identify the elements that are affected so that changes can be propagated where necessary.

Having a well-engineered visualization system is necessary for a complete toolset of artefact management. We designed a visualizer and a graphical editor over the graph database to support the activities involved in artefact consistency management, such as viewing artefacts in different levels of abstractions, exploring intra and inter artefact relationships, allowing users to traverse the graph for impact analysis and editing the graph to propagate necessary changes to elements and relationships [5], [6].

This paper is arranged into following sections: Section II describes the related work containing the details about

relevant researches previously, problems encountered and solutions; it further elaborates the background information on the research area. Abstract solution details of the complete research, architecture of the proposed solution and how the development tool is considered are included in the methodology section (Section III). In Section IV an introduction to the expected analysis tasks with the tool evaluation and experiment details are given. Finally, Section V concludes the paper with possible future extensions to the research.

II. Related Work

In this section, we explore existing research studies related to artefact traceability extraction, modelling, processing, management and visualization. Our research has been categorized into three areas such as extract information using Natural Language Processing (NLP), establishing traceability links and visualizing the traceability links.

Our first research area is information extraction using NLP. For analysing the given text, the most natural language processing systems are based on the following levels, i.e., morphological level, lexical level, syntactic level, semantic level, discourse level and pragmatic level. Popular NLP frameworks such as GATE, OpenNLP, WordNET and Stanford NLP are widely used in the literature.

GATE has been used in an approach [7] where following rules were used for extraction: nouns were converted to entities, gerunds were identified as relationships and specialization's relationships were found from structure like "is a", etc. NLPC(natural language processing for classes) tool [8] was created to analyze a given input. It supports many processing stages such as Pre-processing, Part of Speech (POS) Tagging, Class Identification, Attribute and Function identification and then plotting the classes. NLPC used WordNet 2.1 for their processing.

In another approach a well-defined framework [9] is used a framework to extract information from requirement specification. It first analyses the requirement specification. Then parse by using Natural Language Processing (NLP) parsing techniques to get the nouns & verbs. With the help of domain expert knowledge results are then improved. Then it uses WordNet as the ontology to get the semantic relation between the classes & attributes. POS (Part of Speech) Tagger is used to get the stem (root) of the word. This analysis helps in finding and identifying classes, attributes methods, actors and messages between them.

Our next research area is establishing traceability links. CLIME tool [10] is designed to tell the developer when artefacts become unsynchronized and to indicate what needs to be changed or updated to achieve system wide consistency. This tool first extracts relevant information from each of the software artefacts and stores it in a database. Next, it uses this stored information along with a description of the constraints among the artefacts to build the complete set of current constraints for the software system. Third, it uses the information in the database to test the validity of each of these constraints. Finally, it presents the results of these tests to the developers so that they can resolve any inconsistencies [10]. In another research, two different Information Retrieval (IR) models, Probabilistic Model (PM) and Vector Space Model (VSM), are applied to extract links between code and documentation [11]. The results show that IR provides a practical solution for automated traceability recovery.

An IR based traceability link recovery technique is used with two basic steps to build traceability links [12]. First, a corpus for the target artefacts is constructed and after construction, pre-processing is applied. In pre-processing removal of non-literal characters, splitting of identifier names by using either the camel case or underscore conventions, word stemming and removal of stop words have been performed. Each element of the source artefacts is converted into a query and then compared to each document in the corpus. After the comparison it provides similar results list to the user.

In another approach researchers have implemented a method to creating traceability links between UML design specifications and its implementation, i.e., Java source code [6]. First they converted UML design specification to XMI (XML metadata interchange) and then converted a java source code file into JavaML file. After the conversion they used Xlink methodologies to design linking information between a UML design specification and Java source code which is named as link-base.

Visualizing traceability links enables users to recover, browse, and maintain inter-relationships between artefacts in a natural and intuitive way. However, it is a big challenge to visualize an overwhelmingly large number of traceability links effectively and efficiently because of scalability and visual clutter issues. Graph based visualization technique is an easiest and efficient approach to represent artefacts as nodes and traceability links between artefacts as edges to form a graph.

A hierarchical graphical structure to visualize links was proposed [13] where requirements are represented by leaf nodes and titles and other hierarchical information are represented in internal nodes. But this representation is not scalable well with large data set and not efficient. Traceability visualization approach using sunburst and netmap techniques to display traceability links between requirements knowledge elements [14] also has the above same problem. Although the two techniques can visualize the overall hierarchical structure and can easily browse links, the graph can become very large leading to visual clutter when dealing with a large number of traceability links.

Treemap and hierarchical tree visualization techniques [15] which integrates enclosure and node-link visualization representations to support the overall overview of traceability in the system and the detailed overview of each link while still being highly scalable and interactive. There are some limitations in this approach such as the hierarchical structure of the system is not well communicated in the treemap and not scalable well with nodes which have large number of links.

III. Methodology

A. High Level Architecture

Software artefacts related to development process are stored in a file server. The files are tracked to detect changes of the incremental development. After that artefacts are translated into structured XML files. (Fig. 1 shows high level architecture of our tool). In each iteration, modified files proceed for the artefact XML translation process.

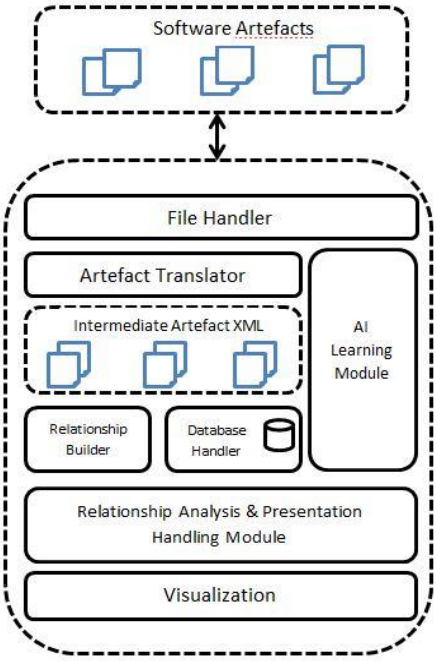

Fig. 1. High Level Architecture of the tool

Our tool is named as SATAnalyzer (Software Artefact Traceability Analyzer) and it has access to the translated structured XML files. At the artefact XML pool data level, version control module tracks the changes of incremental iterations and redirect modified artefact XML models for relationship extraction process. These version control processes in each data levels assure the performance of the system by avoiding redundant processes.

After the artefact extraction process, relationships among artefacts are built using semi-automated process. Learning algorithms are able to extract specific patterns involved in a specific environment such as naming differences between design and source artefacts due to coding conventions. These extracted patterns are used to extract relationships in later iterations of artefact development.

After building the relationship XML files, relationships are stored in a graph database. While the incremental development happens, main server analyses the artefacts and provides the information about a relationship violation and a possible correction. Since relationship violations are relative with artefacts, various techniques used to triangulate best base points to provide accurate suggestions for relationship violation corrections.

Presentation handing module is used to handle various analysis and presentation result restructuring process for better user experience.

B. Artefact Management

Artefact selection process for a given software development process is done manually. User has freedom of selecting which artefacts are maintained by the traceability system. In the incremental development snapshots of artefacts are stored and modified artefacts are selected for further processing.

C. Artefact Extraction

For further processing, various different artefacts such as requirements, design and the source-code are extracted to structured format of information as XML data. Natural Language Processing (NLP) and Machine Learning (ML) techniques are used to extract information from the requirement documents with user feedback [6], [8], [16], [17].

```
<Artefacts>
    <Artefact type="UMLDiagram">
        <FileSystemLocation></FileSystemLocation>
            <ArtefactElement name="..." type="..." id="...">
                <ArtefactSubElement name="..." type="..." visibility="..."
                                    id="..." variableType="...">
                </ArtefactSubElement>
                <ArtefactSubElement name="..." parameters="..."
                    visibility="..." returnType="..." type="..." id="...">
                </ArtefactSubElement>
            </ArtefactElement>
    </Artefact>
</Artefacts>
```

Fig. 2. Artefact Extraction XML Model for UML Artefact

```
<Artefacts>
    <Artefact type="Requirement">
        <FileSystemLocation></FileSystemLocation>
            <ArtefactElement name="R1" id="RQ1">
                <Title>...</Title>
                <Content>...</Content>
                <Priority>...</Priority>
                <Type>...</Type>
            </ArtefactElement>
            <ArtefactElement ... >
                ...
            </ArtefactElement>
            ...
```

Fig. 3. Requirement Artefact's Extraction XML Model

For artefact standardization, artefacts' xml file structures are defined for each type of artefacts (Fig. 2 and Fig. 3 show the pre-defined artefacts XML model used for requirements and UML artefacts, respectively). These extracted artefact information is used to build relation among the artefacts. In an incremental development process, only the modified artefacts are used for artefact extraction process.

D. Relationship Extraction

For a better impact analysis of artefacts in an incremental development process, information about how these artefacts

are related is very important. By extracting the artefacts relationships and modelling those as a graph representation produces a very effective relationship model which can be efficient in processing those relationships.

Some artefact definitions are very strong such as design of a class and its implementation while some artefacts are loosely defined such as a relationship from the requirements. Patterns learned regarding the system by NLP and ML techniques are used to extract these relationships while learning the patterns of relationships and provide suggestions is the procedure of the current relationship extraction (Fig. 4 shows the pre-defined relationship XML model used for relationship extraction) [10].

```
<Relations>
    <Relation id="1">
        <SourceNode>...</SourceNode>
        <TargetNode>...</TargetNode>
    </Relation>
    <Relation ...>
        ...
    </Relation>
</Relations>
```

Fig. 4. Relationship XML Model.

E. Relationship Modelling

Graph representation is used as the relationship model. In the relationship model artefact elements are represented as nodes and relationships among them as edges of the graph. Traversing the graph will allow the analysis of the relationship violation. Directional relationships are used for the relationship model.

Existing relational graph databases lack support for sparse connected relation data storing and process. Database performance become worse as the outlier data multiplies and structure of the database become less uniform and more complex [3].

F. Recommendation System

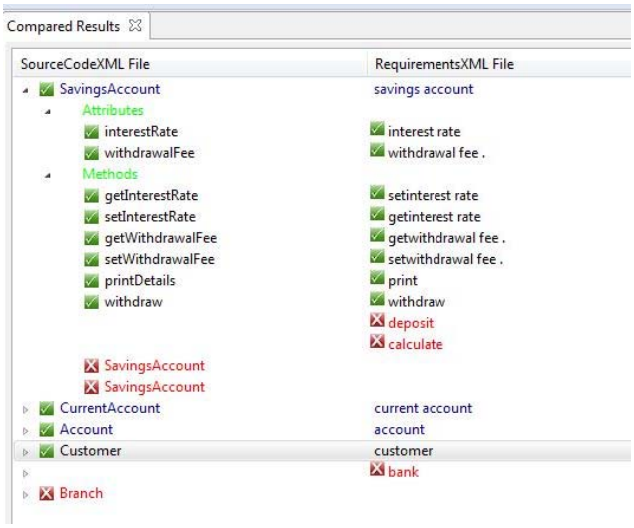

Fig. 5. Comparison between Requirement and UML Artefacts

In the analysis process, impact analysis of the changes is important. When changes are detected while browsing new versions of XML files, affected artefacts are recognized by analysis of the relationship graph. This output is used to correct the relationship violation in artefacts in a development environment for a user. It provides suggestions to correct relationship violations and if possible correct those violations are the main functionalities of the recommendation system. Fig. 5 shows the relationship violation between these requirements and the corresponding UML artefact.

G. Visualization

Visualization of relationships as a graph with nodes and links is a very understandable view for a typical user as shown in Fig. 6. In a generic software development process, there can be millions of artefacts. When the numbers of artefacts are growing the complexity of the graph view of the relationships is also increased. Using pre-defined clustered view of relationships for reduced complexity in the graph view we can reduce the complexity of the graph and provide effective analysis view.

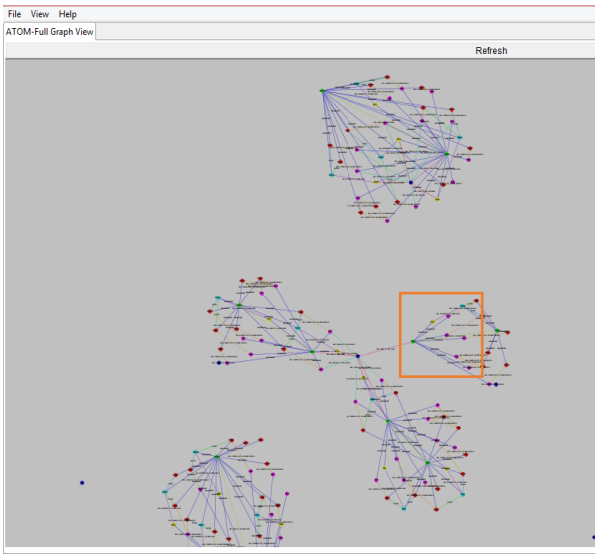

Fig. 6. Graph Representation of Relationships.

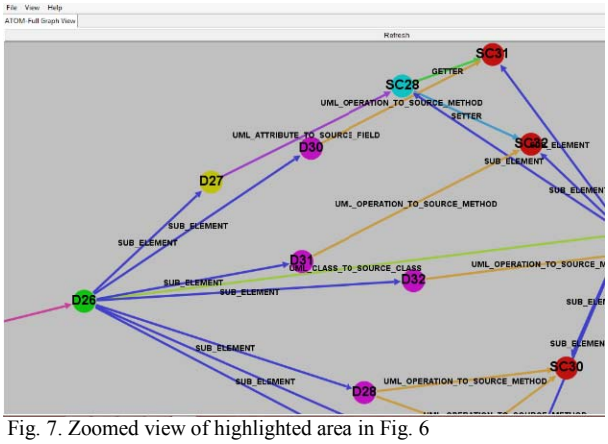

Fig. 7. Zoomed view of highlighted area in Fig. 6

With compact files view with indicated impacts in a representation with low graphical view can help to analyse more artefacts in parallel. Also to maximize the user experience compact to file view analysis is strongly recommended.

Also the visualization system supports editing functionality which let users to edit nodes, relationships and their properties which includes modification or deletion. Also changes made in visualization is propagated backward and entire system is made consistent after the changes.

IV. Implementation and Analysis

A. Extract information from Requirement

Requirements are structured and stored in XML data. Each requirement explores specific information in natural language about what system should do. First information has been extracted from the '<Title>' and '<Content>' nodes within the '<ArtefactElement>' nodes as shown in Fig. 8. Title gives short description. It gives class or behaviour of the system. Content gives long description. It gives class, attribute and behaviour.

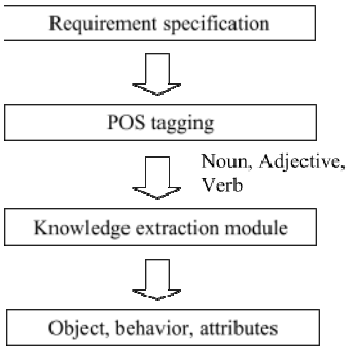

Fig. 8. Information extraction from Requirements Artefact

We have come up with a solution to extract information (Identifiers) from each requirement specification. We used Standford NLP tool [18] to extract relevant information from each requirement and designed an algorithm to identify classes, attributes and behaviours. Pre-processing to find identifiers such as stemming, stop words removal, remove person name, location name using NLP tool, was implemented. For identifying class, we extract noun from <Title> node, and if the <content> node has "such as" sub string, we split the sentence and get the nouns from the first part of sentence. Default words such as detail, database, and system are removed. If the noun matches in both nodes, then it is identified as a class.

To find out the attributes of a class from the second part of the sentence which is got by splitting the <content> node by "such as" word as mentioned above, "," is used as separator. Behaviours of the class are identified by extracting verbs from the <title> node. The content gives the additional information of the particular behaviour. By following the Object Oriented Programming (OOP) concept, behaviours for the identified attributes are created using getter and setter methods.

B. Establish traceability links among software artefacts

Initially we read each XML file and created a list of artefacts which are classes. Each artefacts has list of sub elements/ sub artefacts such as attribute and behaviour. The artefact list is iterated to find relationship among classes and sub artefact list is iterated to find relationship among attributes and behaviours. Traceability links are created using the following algorithms while iterate through the list. We have used Levenshtein distance algorithm for finding similarity between two words, and used WordNet for semantic analysis, and additional algorithms for establishing traceability links among artefacts were used.

As shown in Fig 4, when we trace between artefacts, first we check whether the class is matched with given two artefacts. If that succeed, then we compare the attributes and behaviour. By following this same procedure for all artefacts, we can establish traceability links among software artefacts of a system.

C. Visualize artefact elements and relationship among artefacts

We have choosen Neo4j Graph Database for modelling Artefact Relationships from various graph database solutions. By using Neo4j we can maintain our database small, simple and efficient. Also Neo4j can handle billions of nodes and relationships with properties. The structure of a graph in Neo4j instance does not follow any defined schema which is semi-structured, following only the rules of a simple property graph which is key/value-based, directed and multi relational graph [3].

We used Apache Lucene Indexing API [19] for searching nodes and edges of Neo4j graph because Neo4j does not provide an implementation for an indexing mechanism. To visualize the relationship among artefacts we designed a User Interface (UI) with the aid of Gephi toolkit API [20] functionalities. Graph visualized in the tool can be editable providing modification and deletion facilities.

After extracting the artefacts data as XML model data and relationships among artefacts, traceability links are stored in the Neo4j graph database. User can see the visualization of artefacts traceability links in tool as an overview of entire system as in Fig. 6 to get overall structure or as cluster views of filtered artefacts or relationships to reduce visual clutter and get more in-depth details.

When a particular element (node or edge) is selected in visualization, user can see the properties of it, edit the properties, see impact of the element in the system (which means highlighting elements connected to it) and delete the selected element.

V. TOOL Evaluation

As the next step of this research we have conducted an extensive evaluation process of the tool that we have developed against its expected functionality of semi-automatic artefacts consistency management along with therich visualisation support. Tool has been evaluated using two different data sets which are bank system data and bus booking system data.

According to the obtained results, the tool performs well in terms of extracting classes, attributes and behaviours from requirements artefact, establishing traceability links among three artefacts, visualize system artefacts with their relationships, show the impact analysis, change management and editing features such as artefacts selection, artefacts properties visualization and artefacts deletion. The result which we gained for our tool is used to calculate the precision and recall as in (1) and (2). When we analyse the tool, results are recorded as in TABLE I. which describes the accuracy of our tool.

$$Precision = \frac{\#\ of\ correct\ relationship\ traced}{\#\ of\ relationship\ traced\ in\ tool} \quad (1)$$

$$Recall = \frac{\#\ of\ correct\ relationship\ traced}{\#\ of\ relationship\ in\ system} \quad (2)$$

TABLE I. TOOL ACCURACY EVALUATION.

Bank system	**Precision**	**Recall**
Finding requirement elements	100%	100%
Establishing traceability links	100%	100%
Bus Booking system	**Precision**	**Recall**
Finding requirement elements	100%	100%
Establishing traceability links	100%	76.08%

As we said earlier this is semi-automatic tool which mean developer can also add or remove relation through the tool manually. The result shown in TABLE I is got directly from the tool without users' modification to the system. When a user manually update relationship which are presented because of the faults in system's original artefacts, then all relationship in the system are traced more accurately than before.

VI. CONCLUSIONS

A. Future Work

There is a number of possible further extensions can be identified with respect to the research and the tool that we have developed as part of this research. A few of those possible future research extensions can be listed as follows:

Developing a more generic platform for artefact management without enforcing special development methodologies and development tools is required to deploy in an existing development environment. In the current tool, support for distribution development environment is minimal as a design decision that we have taken considering the time and resource availability in line with the scope of the work. This can be improved as a future development. Improve learning and Artificial Intelligence on understanding artefacts and model relationships is positive future development. It is the solution to ease the use of artefacts management.

B. Concluding Remarks

Dealing with variety of artefacts to trace each other is a complex problem in software engineering due to artefacts having different formats. Artefact extraction and relationship modelling heavily depend on user feedback. Automating the artefact extraction and relationship mining is restricted due to existence of various types of artefacts, schematic interpretation restrictions, unstructured artefact documents, various file formats of design tools. In this research we have successfully established the required functionality for semi-automatic artefact consistency management with visualisation tool support. Moreover, the natural language processing as part of the AI module for artefact extraction can be seen as a unique approach that we have introduced into the existing artefact consistency management techniques. With the completion of the tool development and evaluation, we believe that the solution we have introduced can support the software professionals in doing their software engineering tasks efficiently and effectively.

ACKNOWLEDGMENT

This work is done with collaborative support of artefact consistency research group from University of St. Andrews. Ildiko Pete provided feedback and initial support.

REFERENCES

[1] I. Pete and D. Balasubramaniam, "A Framework for Maintaining Artefact Consistency during Software Development," p. 24, 2013.

[2] S. Winkler and J. Pilgrim, "A survey of traceability in requirements engineering and model-driven development," Softw. Syst. Model., vol. 9, no. 4, pp. 529–565, Dec. 2009.

[3] H. Wendel, "Using Provenance to Trace Software Development Processes," 2010.

[4] "Neo4j - The World's Leading Graph Database." [Online]. Available: http://www.neo4j.org/.

[5] S. C. Tan, "Supporting Visualization and Analysis of Requirements Evolution," 2009.

[6] J. Alves-foss, D. C. De Leon, and P. Oman, "Experiments in the Use of XML to Enhance Traceability Between Object-Oriented Design Specifications and Source Code," vol. 00, no. c, pp. 1–8, 2002.

[7] H. Herchi and W. Ben Abdessalem, "From user requirements to UML class diagram," 1996.

[8] P. R. Kothari, "Processing Natural Language Requirement to Extract Basic Elements of a Class," vol. 3, no. 7, pp. 39–42, 2012.

[9] S. K. Shinde, "NLP based Object Oriented Analysis and Design from Requirement Specification," vol. 47, no. 21, pp. 30–34, 2012.

[10] S. P. Reiss, "Incremental Maintenance of Software Artifects - CLIME," Softw. Eng. IEEE Trans., vol. 32, no. 9, pp. 682 – 697, 2006.

[11] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," vol. 28, no. 10, pp. 970–983, 2002.

[12] S. Klock, M. Gethers, B. Dit, and D. Poshyvanyk, "Traceclipse□: An Eclipse Plug-in for Traceability Link Recovery and Management Categories and Subject Descriptors," pp. 24–30, 2011.

[13] J. Cleland-Huang and R. Habrat, "Visual support in automated tracing," IEEE Comput. Soc., no. IEEE Computer Society, pp. 4–8, 2007.

[14] T. Merten, J. Daniela, S. Augustin, and A. Delater, "Improved Representation of Traceability Links in Requirements Engineering Knowledge using Sunburst and Netmap Visualizations," pp. 17–21, 2011.

[15] X. Chen, J. Hosking, and J. Grundy, "Visualizing traceability links between source code and documentation," 2012 IEEE Symp. Vis. Lang. Human-Centric Comput., pp. 119–126, Sep. 2012.

[16] J. I. Maletic, M. L. Collard, and B. Simoes, "An XML based approach to support the evolution of model-to-model traceability links," Proc. 3rd Int. Work. Traceability Emerg. forms Softw. Eng. - TEFSE '05, p. 67, 2005.

[17] S. D. Joshi, "Textual Requirement Analysis for UML Diagram Extraction by using NLP," vol. 50, no. 8, pp. 42–46, 2012.

[18] "The Standford NLP (Natural Language Processing) Group." [Online]. Available: http://nlp.stanford.edu/software/.

[19] "Apache Lucene - Apache Lucene Core." [Online]. Available: http://lucene.apache.org/core/.

[20] "Gephi Toolkit." [Online]. Available: http://gephi.github.io/toolkit/.