2014 International Conference on Advances in ICT for Emerging Regions (ICTer): 055 - 062

Establishing Traceability Links among Software Artefacts

Diunuge B. Wijesinghe, Karthigesu Kamalabalan, Thanuja Uruththirakodeeswaran, Gitanjali Thiyagalingam, Indika Perera, Dulani Meedeniya

Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka

diunuge.10@cse.mrt.ac.lk
kamalan.10@cse.mrt.ac.lk
thanuja.10@cse.mrt.ac.lk
anjali.10@cse.mrt.ac.lk
indika@cse.mrt.ac.lk
dulanim@cse.mrt.ac.lk

***Abstract*— Artefact management in a software development process is a difficult problem in software engineering. Usually 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 to name a few. 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 provide a prototype tool to establish artefact traceability links and visualization. This paper describes the research methodology and relevant research carried out for semi-automatic traceability link establishment and visualization of software artefacts.**

***Keywords*— Software Artefacts, Traceability Links, Visualisation, Traceability Management**

I. INTRODUCTION

Artefact consistency management is a complex challenge in software engineering. Different process activities in a software lifecycle that result in artefacts such as requirements specification, software architecture, design specification, source code, test cases for verification and validation of the system etc. are usually maintained in isolation and evolve at different rates. To elaborate further, although, all of these artefacts are aimed at facilitating software product 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].

Research on artefact consistency management is an important area of study in software engineering; it is still getting popular attracting many researchers who are interested in different aspects of the artefact consistency, however [2]. Some research studies are currently being carried out to tackle software artefact inconsistency problem. A main area of study to attack 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 (Fig. 1 shows an overview of having interconnected artefacts of a software process). 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 one 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 applied process.

978-1-4799-7732-1/14/$31.00 ©2014 IEEE

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. At present we use neo4j graph database for relationship modelling [3]. 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 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.

Also having a well-engineered visualization system is necessary for a complete toolset of artefact management. We designed a visualiser 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 [4], [5].

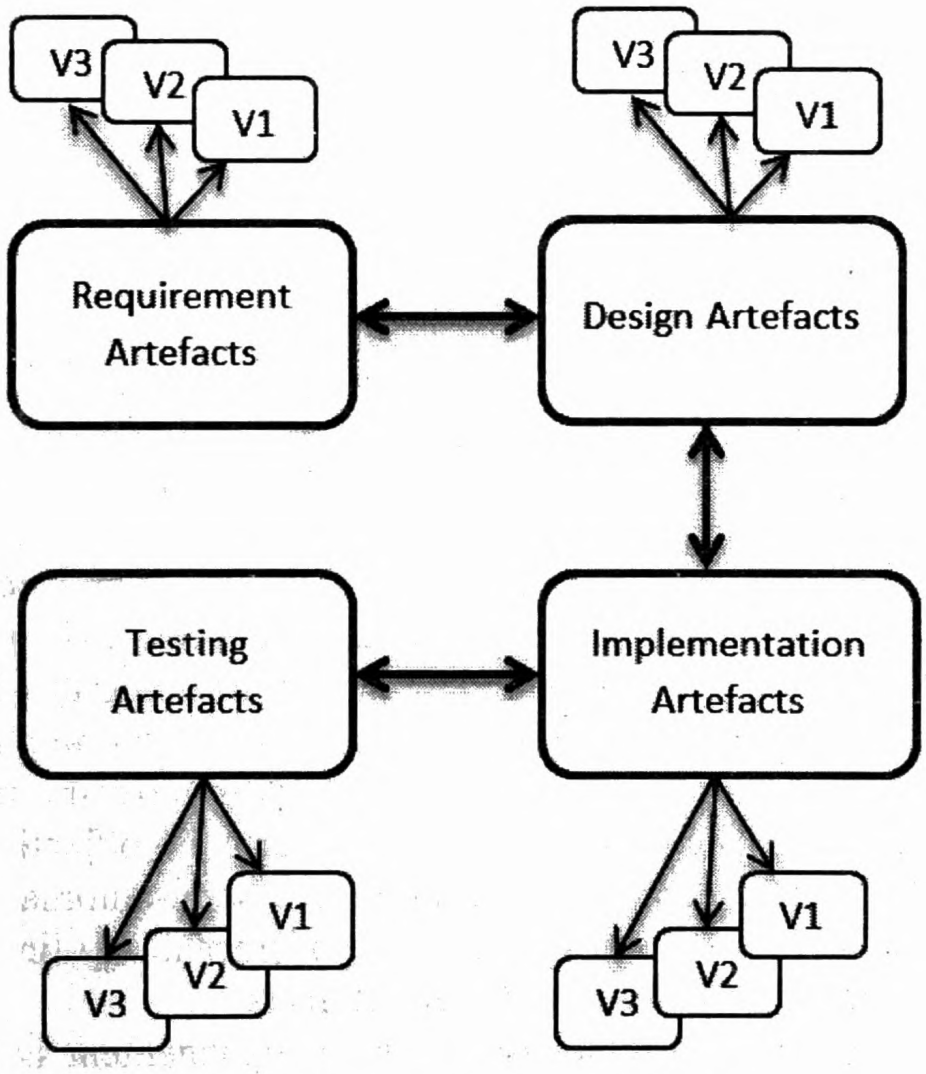

Fig. 1 Different Artefact Abstraction Level.

This paper is arranged into following sections: Section II describes the related work containing the details about relevant research done for the projects, 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). 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, research details discussion provided for artefact traceability extraction, modelling, processing, management and visualization.

From the review, it was found that Steven P. Reiss [6] has created a tool named as CLIME, which 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 [6].

G. Antoniol et al. [7] apply two different Information Retrieval (IR) models, Probabilistic Model (PM) and Vector Space Model (VSM), to extract links between code and documentation. The results show that IR provides a practical solution for automated traceability recovery.

Shinde, Bhojane and Mahajan [8] used a framework for 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 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 & identifying classes, attributes methods, actors and messages between them.

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. But existing researches regarding visualization has several problems such as cannot represent hierarchical structure, not scalable with higher dataset, not space efficient and causes visual clutter while displaying large set of traceability links on top of a graph [4].

A hierarchical graphical structure to visualize links is proposed by Cleland-Huang and Habrat [9] where requirements are represented by leaf nodes and titles and other hierarchical information are represented in internal nodes. But this representation is not scalable with large data set and efficient.

Zhou et al [10] adopt a hyperbolic tree view with the enhancement of a focus plus context approach to facilitate software traceability understanding. This view allows users to maintain a global view of links as well as being able to dive deep into an interesting traceability path but it is also not space efficient.

TraceVis [11], visualizes a dynamic list of hierarchies and adjacency relations using icicle plots and hierarchical edge bundling [12] techniques to support the hierarchical structure and to reduce visual clutter. But it is not space efficient and causes visual clutter when the dataset is large or lateral relations visualized [11].

Traceability visualization approach proposed by Merten et al [13] also has the above same problem which used sunburst and netmap techniques to display traceability links between requirements knowledge elements with overall hierarchical structure. We tried to minimize the above mentioned problem in our graph based visualization technique.

III. Methodology

Major target deploy environment of a software artefact management tool is an existing software development process which supports current methodologies.

Usually, there are more than one people involved with an enterprise software development process. To support distributed development process tool is developed using Client-Server Architecture.

A. High Level Architecture

The main server part of the tool has access to the file server of the development process where the software artefacts are stored (Fig. 5 shows the client server architecture of the system which supported distributed artefact management). These files are tracked to detect changes the incremental development. A client with required permission level such as project manager can selects the artefacts relevant to a specific project. After that artefacts are translated into structured xml files using client feedbacks (Fig. 6 shows the layered architecture of the system which shows the data transfer). In the each iteration, modified files are proceeds for the artefact xml translation process.

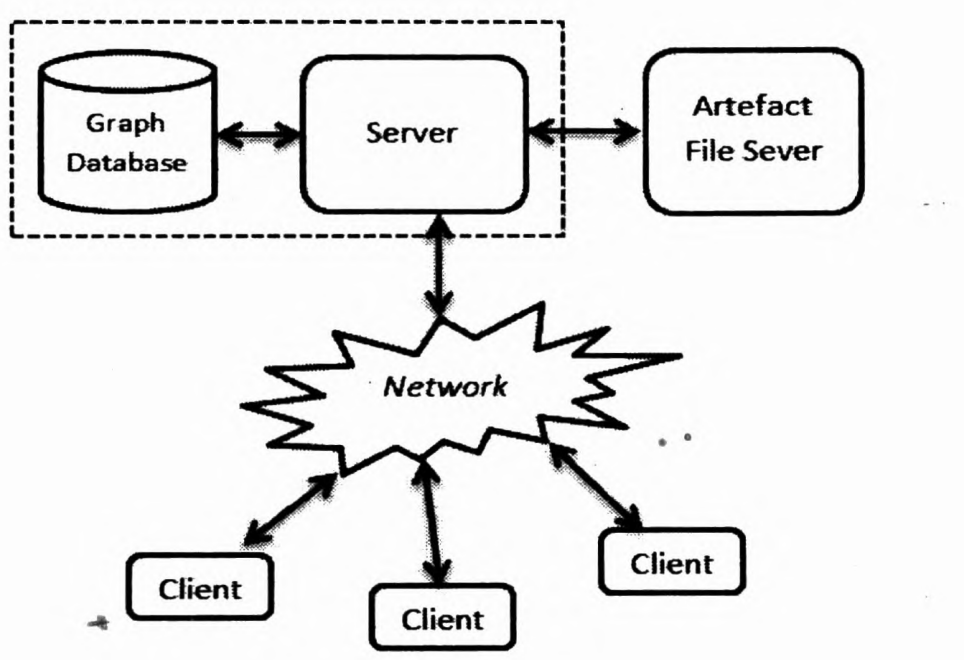

Fig. 5 Client-Server Architecture of the Artefact Management Tool.

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 building using semi-automated process. Users can provide feedbacks for relationships while learning algorithms are enable to extract specific patters involved with specific environment such as naming differences between design & 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 analyse the artefacts and provide the information about relationship violation and 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.

Security is handled within each layer and user management is done to support collaborative distributed development.

User can view and correct the artefact violation via client application which uses textual based RESTful communication protocol with the server.

B. Artefact Management

Artefact selection process for a given software development process is done manually via graphical interface. User has freedom of selecting which artefacts are maintained by the traceability system.

In the incremental development we store snapshots of artefacts and select modified artefacts for further processing.

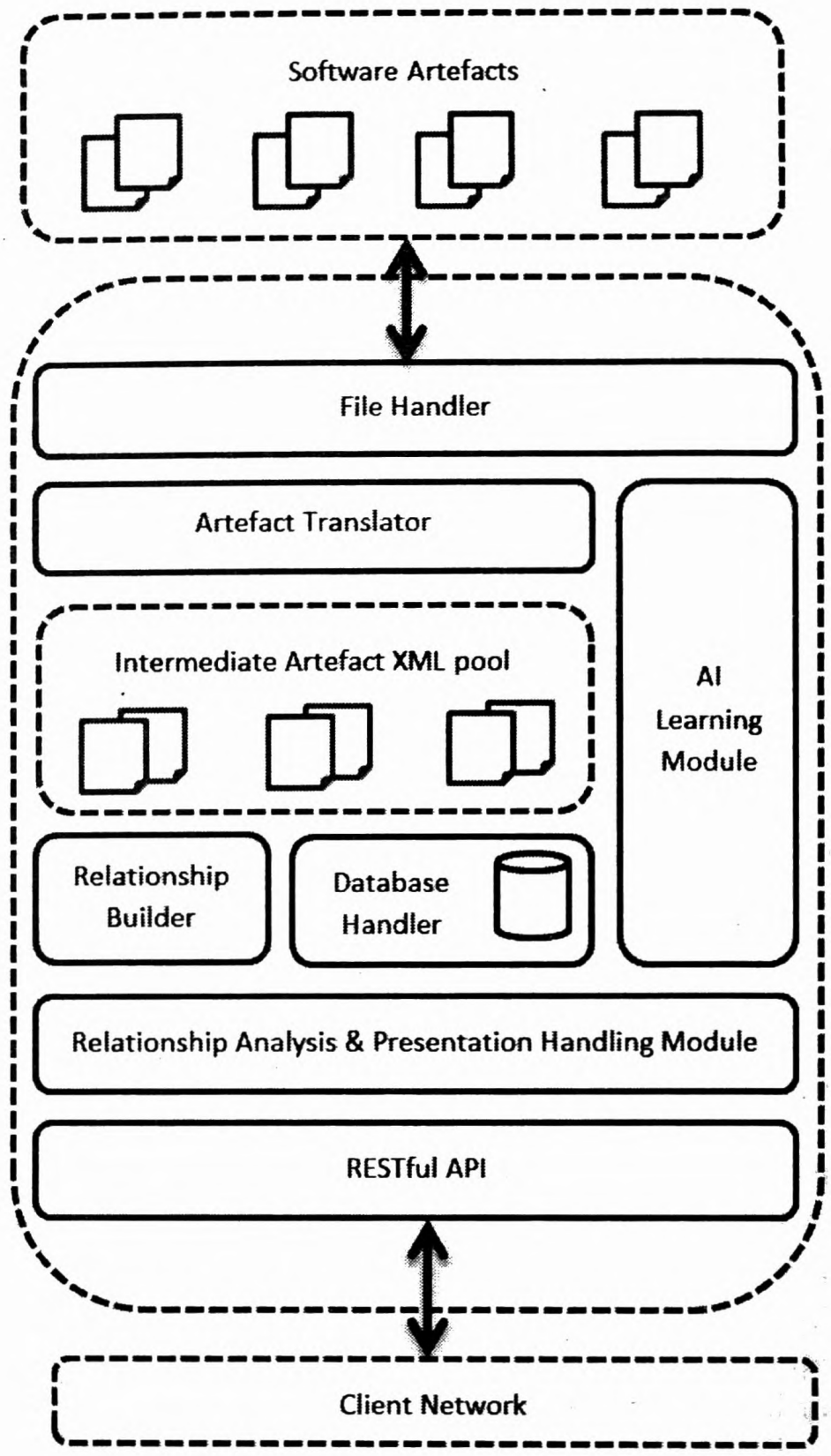

Fig. 6 Server Layered Architecture.

C. Artefact Extraction

For further processing, various different artefacts such as requirements, design & the source are extracted to structured format of information as XML data. Natural language & machine learning techniques are used to extract information from the requirement documents with user feedbacks [14]–[17]. For artefact standardization, artefacts xml are defined for each type of artefacts (Fig. 2 shows the pre-defined artefacts xml model used for requirements artefacts).

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 it as a graph representation is very effective relationship model which can be efficient in processing those relationships.

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

            <IntraConnections>
                <Connection>
                    <Type>...</Type>
                    <StartPoint>...</StartPoint>
                    <EndPoint>...</EndPoint>
                    <Annotation>...</Annotation>
                </Connection>
                ...
            </IntraConnections>
    </Artefact>
</Artefacts>
```

Fig. 2 Artefact Extraction XML Model for Requirement Artefacts.

Some artefact definitions are very strong such as design of a class and its implementation while some artefacts are loosely defined such as relationship from the requirements. Using user feedbacks to extract these relationships while learning the patterns of relationships and provide suggestions is the procedure of the current relationship extraction (Fig. 3 shows the pre-defined relationship xml model used for relationship extraction) [6].

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

Fig. 3 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 (Fig. 4 shows the model used to represent relationships). Traversing the graph will allow the analysis of the relationship violation. Directional relationships are used for the relationship model.

Existing relational graph databases lacks 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 [18].

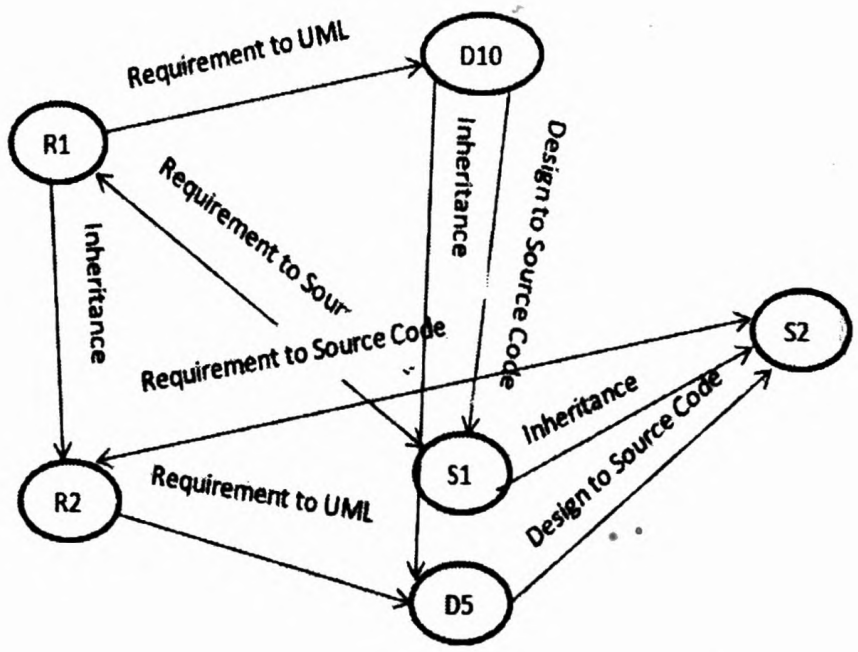

Fig. 4 Graph Representation of Relationships.

F. Recommendation System

In the analysis process, impact analysis of the changes is important. After an iteration of change of artefacts, effected 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.

Provide a suggestions to correct relationship violation & if correct possible violations is the main functionality of the recommendation system. Using AI technique and capturing user feedbacks to exact patterns is used in the recommendation.

G. Visualization

Visualization of relationships as a graph with nodes and links is a very understandable view for typical user. In typical software development process there can be thousands and millions of artefacts. When the numbers of artifacts are growing 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 provided effective analysis view.

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

IV. IMPLEMENTATION

The Server part of the tool is developed using Java EE. We have developed a java based client for artefact management and viewing process. Communication between client and server is handled by Jersey java library which is an implementation of JAX-RS Java API for RESTful web service.

We used Neo4j Graph Database for modelling Artefact Relationships from various graph database solutions. It's a more generic and has balanced performances records for both graph storage and graph processing [18].

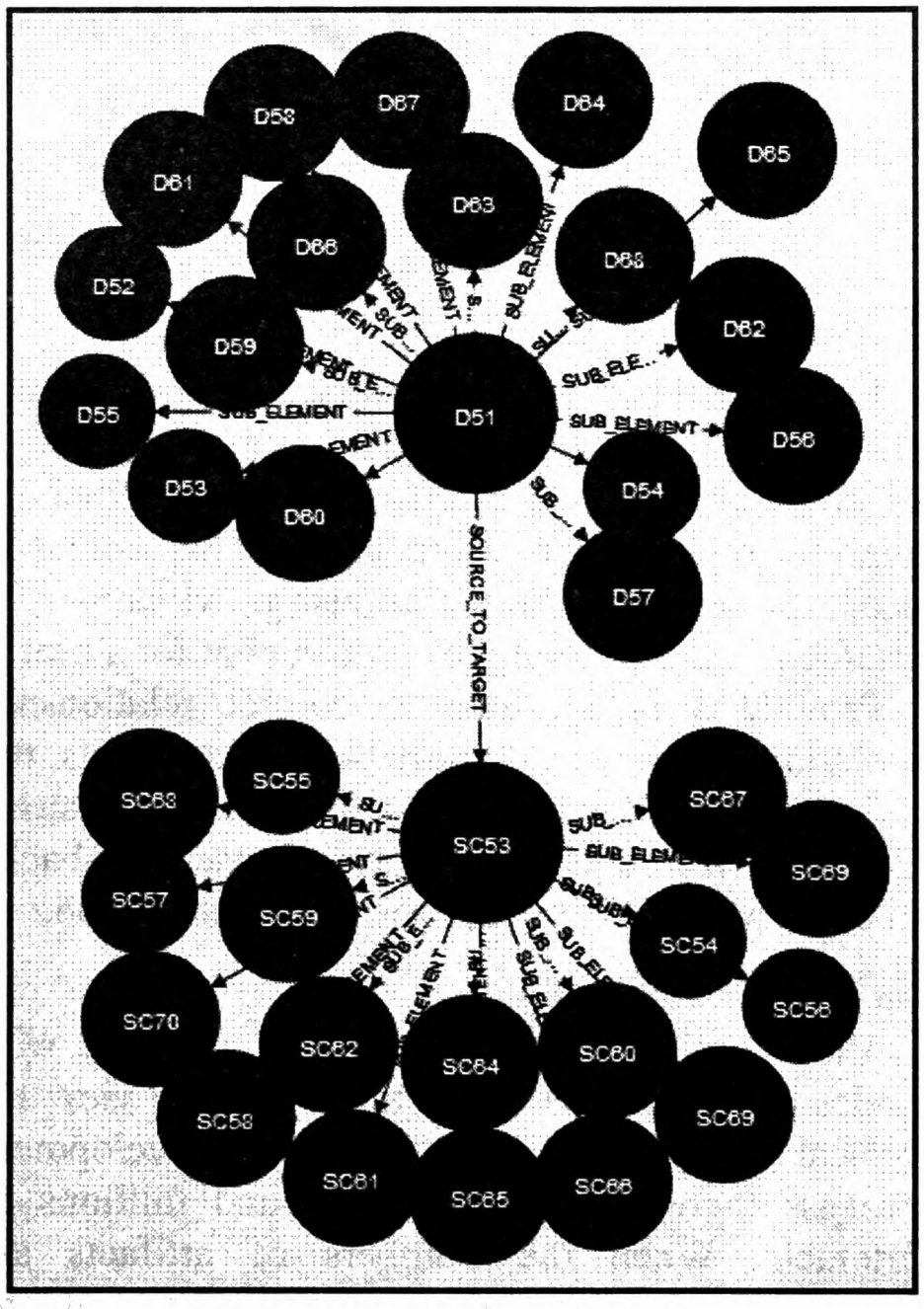

Fig. 7 Artefact relationship analysis view.

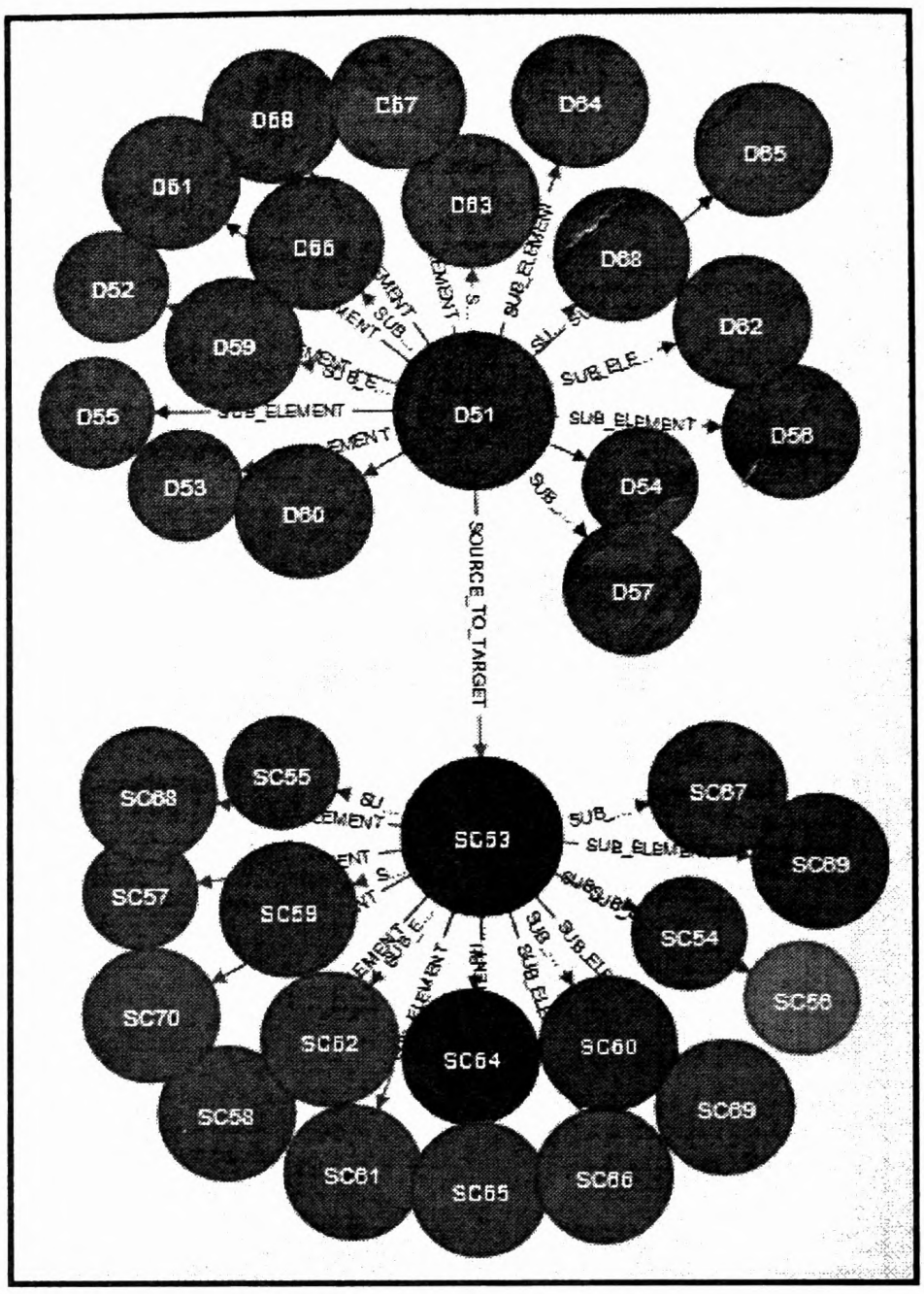

Fig. 8 Artefact impact analysis view.

After extracting the artefacts data as XML model data and relationships among artefacts, traceability links are stored in graph database (Fig. 7 shows a relationship analysis overview of artefact cluster). When an incremental artefacts change is submitted to the system, system provide an impact analysis and possible suggestions for correct relationship violation (Fig. 8 shows an impact analysis on change of SC53-source artefact).

Since, relationship violations are relevant providing exact suggestions was problematic. We considered modification behaviours, requirement artefact priorities, degree of artefact to identify the base points to provide accurate suggestions. Most of the time relationship violation corrections are heavily depends on the user feedbacks. Extracting artefacts and learning relationships required higher degree of user guidance when new artefacts are added or relationships are added.

Currently, there is no any semi-automatic artefact extraction method to support an existing software development process. So, performance or quality comparison capability is very limited.

V. EXPERIMENT AND ANALYSIS

As the next step of this research we are conducting an extensive evaluation process of the tool that we have developed using regression testing and user studies. By applying two unique evaluation techniques we expect to validate the system against its expected functionality of semi-automatic artefacts consistency management along with the rich visualisation support provided to the software engineers and architects as the end users. A sample of participants from these end users, who are using different software development methodologies, will be selected and asked to evaluate the system as the user study of the evaluation.

During the user studies, end users are asked to use our tool from beginning of their software development. This user study aims to evaluate tool properties such as reliability, correctness, robustness and flexibility by following repetitive experiment to analyse major functionalities of the tool which includes importing artefacts into tool, establishing traceability links among them, visualize the traceability links and tracking tool backward traceability maintenance by editing artefacts from visualization interface against previously mentioned tool properties.

By comparing results of the repetitive experiment done for a same project we can identify whether our tool is able to produce similar result always and time taken to complete the same task in tool for a developer decreases with repetition which will conclude that our tools is robust enough and highly flexible by increasing users' learnability and supporting to handle different sizes of artefacts. Comparing traceability results of different project and user satisfaction, we can say whether our tool is reliable and performs its functionalities correctly.

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. A few of those possible future research ideas 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 being a complex problem in software engineering due to artefacts having different formats. Artefact extraction and relationship modelling heavily depends 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 help support the software professional in doing their software engineering tasks efficiently and effectively.

ACKNOWLEDGMENT

This work is done with collaborative support of artefact consistency research group from University St. Andrews. Dharini Balasubramaniam & 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] "Neo4j - The World's Leading Graph Database." [Online]. Available: http://www.neo4j.org.

[4] 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.

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

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

[7] G. Antoniol, G. Canfora, G. Casazza, a. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," IEEE Trans. Softw. Eng., vol. 28, no. 10, pp. 970–983, Oct. 2002.

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

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

[10] J. X. Xin Zhou, Zhenzhong Huo, Yaowen Huang, "Facilitating software traceability understanding with ENVISION," no. COMPSAC '08. 32nd Annual IEEE International, pp. 295 – 302, 2008.

[11] W. J. P. Van Ravensteijn, "Visual traceability across dynamic ordered hierarchies," Master's thesis, Eindhoven Univ. Technol., 2011.

[12] D. Holten, "Hierarchical edge bundles: visualization of adjacency relations in hierarchical data," IEEE Trans. Vis. Comput. Graph. vol. 05, pp. 741–748, 2006.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] E. E. Ian Robinson, Jim Webber, Graph Databases. O'Reilly Media, 2013.