Extraction and Visualization of Traceability Relationships between Documents and Source Code

Xiaofan Chen
Department of Computer Science
The University of Auckland
Auckland, New Zealand
(0064 9) 373-7999
xche044@aucklanduni.ac.nz

ABSTRACT
Traceability links between artifacts in a software system aid developers in comprehension, development, and effective management of the system. Traceability systems to date have been confronting the difficulties in retrieving relationships between artifacts with high quality and accuracy, and in visualizing extracted relationships in a natural and intuitive way. This research aims to combine several traceability recovery techniques to make up for each other's weaknesses to extract relationships between artifacts at a high-level accuracy and quality. Moreover, the recovered relationships are visualized in a hierarchical rich graphical tree that can be expanded and contracted to help users easily interact with these links and move easily between artifacts and their related artifacts and vice versa. Our preliminary evaluation demonstrated that integration of several traceability recovery techniques can improve the quality and accuracy of retrieved links.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *corrections, documentation, enhancement, extensibility, portability, restructuring, reverse engineering, and reengineering, version control.* H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *clustering, information filtering, query formulation, relevance feedback, retrieval models, search process, selection process.*

General Terms
Management; Documentation.

Keywords
Traceability; Information retrieval; Text Mining; Links Visualization.

ASE'10, September 20–24, 2010, Antwerp, Belgium.

1. INTRODUCTION
Traceability has been recognized as a critical success factor for effective development and management of a complex software system throughout the software development life cycle (SDLC) [4, 7, 19]. Tracing and maintaining interrelationships between artifacts within a software system enables developers to better understand the system, undertake improved maintenance of the system, and ultimately to produce a higher quality system [2, 4, 20, 22]. Source code and documents (e.g. tutorial, handbook, developer or user's guide, API documentation, architecture, design rationale etc.) usually written in natural languages are two of the major artifacts produced during the SDLC. Retrieving traceability relationships between this variety of artifacts is very challenging [2, 12, 16, 26].

Our particular focus in this research is on traceability between documents and source code. The objective of our research is to provide users with an effective environment enabling them to retrieve, create, browse, edit, and maintain traceability links between artifacts in a natural and intuitive way. With this environment users can trace relationships between various documents and source code, automatically recover traceability links at low cost and high accuracy, easily create and change links as well as conveniently browse and maintain links.

2. BACKGROUND AND MOTIVATION
In practice software artifacts created during the software development process end up being disconnected from each other. This is because they are often separated into different documents and repositories, created and maintained by different individuals and evolve at different rates [1, 27]. Implementing effective traceability during development can ameliorate this issue. However, there are two main issues in current traceability systems hindering organizations to incorporate traceability into their practice: the low quality and accuracy of extracted traceability links, and the undesirable visualization of retrieved links.

Many traceability recovery techniques have been developed to improve the quality and accuracy of recovered traceability links. These recovery approaches can be classified into two main groups: semi-automatic recovery and automatic recovery. Nevertheless, each approach has its own limitations. Semi-automatic recovery techniques are those that need human intervention during the traceability link extraction process, such as rule-based, scenario-driven, and value-based. The rule-based approach [13] depends on grammatical structures present in the

natural language sentences, traceability rules have to be expanded to allow generation of relations that consider all possible grammatical structures. Moreover, building rules is time-consuming. The scenario-driven technique [9] does not support trace relationships to program variables and other types – other than classes. In addition, the correctness and completeness of the hypothesized traces largely affects the quality of recovered relationships. The value-based approach [10] produces high quality trace relationships among high-value artifacts, but the quality of relationships is undesirable among low-value artifacts. Although this approach can save cost due to the focus on artifacts with high values, the determination of the value of every artifact is complex and time-consuming.

Automatic recovery techniques include Information Retrieval (IR) and Text Mining (TM). The IR techniques [2, 5, 15, 16, 26] only recover information that is deemed relevant to a given query. The accuracy rate of link recovery deeply relies on a threshold and the same threshold may or may not be best suited for different software systems. Using a low threshold retrieves a larger number of accurate relationships than using a high threshold, but more fault relationships are captured at the same time. Compared with IR, the TM technique [27] extract from documents salient facts about pre-specified types of events, entities, or relationships. Although it generates relationships with high accuracy, types of entities have to be pre-defined, and grammar rules have to be built for detecting complex named entities.

To varying degrees, current approaches fail to fully and accurately capture relations between artifacts. In order to recover relationships between artifacts at a high-level accuracy and quality, we are attempting to combine several extraction techniques to counteract each other's weaknesses plus support developers creating and maintaining non-discoverable relationships themselves.

Visualizing recovered links enables users to browse and maintain these links in a natural and intuitive way. Traditionally, these links are stored in a document (e.g. XML format document) [13] or are represented in Matrix [9]. These approaches cannot support users to maintain or interpret links easily and conveniently. Using graph to display links attempts to overcome the shortages of XML documents and Matrix. Unfortunately, most traceability systems to date [15, 17] can only display trace links between selected source and target artifacts, and fail to show links for multiple artifacts at the same time. Although Poirot [5] uses a hierarchical graphical structure to represent links, it is hard to understand the graph as the less of description to nodes and displaying unnecessary parts of tree without candidate links. We use a hierarchical, graphical traceability relationship visualization that can be expanded and contracted to enable users to interact with large numbers of extracted relationships.

In the development process change is inevitable. Many of the original requirements, design or model details usually cannot make it to the final release of a software system [24]. Larman [14] states that only tested code demonstrates the true system design. Source code represents the low-level structure, realised design and fulfilled requirements of the system. Trace links anchored on source code are more stable and less likely to deteriorate than trace links centered on other artifacts. In addition, studies have shown that having a clear map of code can help users understand a system and establish a clear mental model of the system [6, 21]. Therefore, our research utilizes source code dependency analysis to extract and visualize relations between documents and source code.

3. METHODOLODY AND APPROACH

Our traceability system is seamlessly integrated with the Eclipse integrated development environment (IDE) to provide users with both IDE and traceability support.

Most traceability systems to date have used one recovery relationship technique (e.g. IR, TM, Scenario-based etc.) to retrieve links between artifacts. These systems are unable to accurately capture traceability links due to limitations of their employed recovery technique. To improve this issue, an approach of combining IR, TM and digital library [23] techniques is utilized to recover traceability links in this research. By using this mult-technique approach we aim to compensate for different technique's weaknesses and play to each's strengths.

Although several traceability tools [5, 15, 17] provide traceability link visualization support, we have found none that display links in an integrated, hierarchical rich graphical tree that is able to expand and contract. Our research takes advantage of the graphical drawing ability of ZEST [28] to establish such a tree to represent traceability links between documents and source code. A tree structure produces not only the main backbone of a system but the detailed leaves as well. This allows users to easily grasp the main structure of the system, effectively and efficiently browse links among artifacts as well as maintain them.

Comparison and usability studies will be conducted to evaluate the strengths and weaknesses of our traceability system. We will use several open source projects which have both code and rich documentation artifacts to do this comparison. In comparison evaluation, we will compare our traceability system with other traceability systems to find out whether our system provides better quality and higher accuracy than others. Conducting usability studies aims to show whether the system is easy to use, whether the system is able to help developers in comprehension, development, and management of the software system, and whether the links visualization can help developers browse and edit the recovered relationships easily.

4. IMPLEMENTATION

A prototype of our novel traceability system has been developed. This prototype is embedded within Eclipse, and automatically extracts relationships between documents and source code, and visualizes these links. Documents include any kinds of documents generated during and after the development of a project, such as user guides, developer manuals, tutorials, test cases and so on. Figure 1 shows the user interface of the prototype. The traceability perspective includes three parts. The left part is the navigation view, which displays details of projects, e.g. headings inside PDF documents. The top right area is the edit area which shows java files and PDF files. The bottom right area is the traceability view that visualizes extracted relationships.

The Navigator in the traceability view includes two sections. The Source Code section includes all java files in the project. The Documents section contains all PDF files in the project. When a class or a heading is selected, the right graph area shows artifacts

only related to the selected item. For example, when PrintJob.java is selected, all direct or indirect related nodes are displayed.

4.1 Features Completed

4.1.1 Extracting Traceability Links

We combined IR and TM techniques to extract relationships between documents and code. The IR engine is Apache Lucene, which is a full-featured text search engine written in Java [3]. The TM engine we used is the Information Extraction system of GATE [11], called ANNIE – a nearly-new information extraction system. Source code is analyzed by the JDT Java Parser [8] provided by Eclipse to extract source code identifiers (every class, method, package name) and relations between classes and methods. IR Query phrases are class names split by uppercase. We define every class identifier as an entity that belongs to the "Class" type, and then create grammar rules that help TM to only match and annotate information about these entities. Currently, the prototype only extracts links between classes and headings of PDF documents. Each link has a score representing the similarity rate. The rules for merging relationships recovered by IR and TM are as follows:

- Relationships that hit the same section but different sentences are merged to one link.
- If a link is retrieved by both IR and TM, then its similarity score is assigned to 1.0.
- If a link is only recovered by TM, then its similarity score is set to 1.0.
- If a link is only recovered by IR, then its similarity score is that provided by the IR analysis.
- Relationships with a score under 0.1 are cut off.

A set of candidate traceability links between headings and classes is produced after going through the five merging rules. The similarity score for every link in the set is above or equal to 0.1.

4.1.2 Visualizing Extracted Links

We used Zest to visualize extracted links. Zest, the Eclipse Visualization toolkit, is built for Eclipse to represent graphs [28]. As shown in Figure 1, the first column is methods in a class, the second column is classes in the project, the third is headings in a PDF file, the fourth is PDF files in the project. When a node is selected, all adjecent nodes are highlighted, and the file related to the selected node is opened in the top right edit area.

The retrieved relationships between classes and documents are also represented in a Matrix table. The columns of the table are classes in the project, the rows are headings of PDF files. Cells are scores generated by IR engine. When a cell is selected, all related files (java file and PDF file) are opened in the top edit area.

4.2 Features to Be Completed

4.2.1 Extracting Traceability Links

Key features that we are planning to implement are as follows:

- Extend the prototype to deal with other format documents, e.g. xml, html, txt, and Wiki documents. Currently it uses Java code and PDF documents.
- Develop a common representational technique for rich inter-artifact relationship representation and management.
- Attempt to add a digital library approach [23] to recover relationships using a very different set of discovery techniques to IR and TM, combining these using our common representational technique.

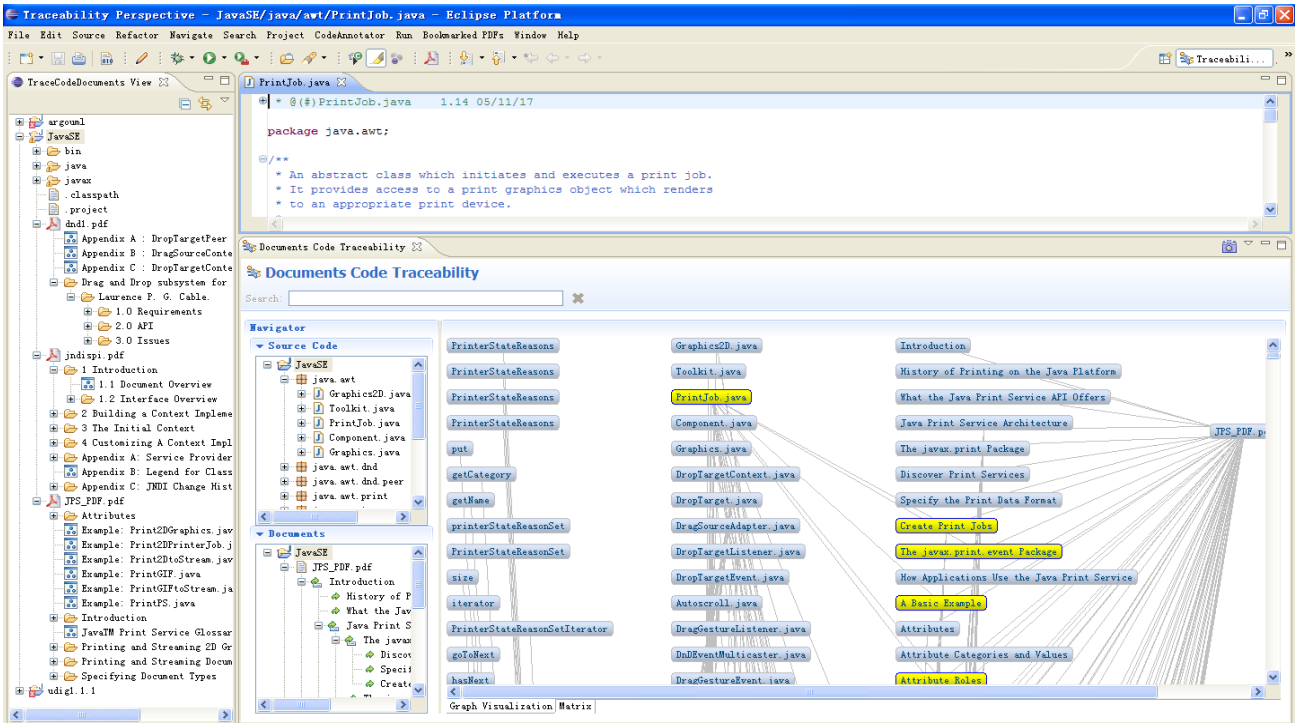

Figure 1. User interface of traceability prototype in Eclipse

- Combine results generated by IR, TM, and digital library to refine and enhance the extracted links using our common representational technique to model the composite, extracted relationships.
- Change the query phrases to include comments, properties and other meta-data for classes, interfaces, fields and methods to improve the accuracy of extracted links.
- Test the prototype on some different systems (e.g. JDK, uDig etc.) using comparison and usability studies to find out its strengths and weaknesses as compared to existing traceability support platforms and tools.

4.2.2 Visualizing Extracted Links

Key visualization features that we are planning to implement are as follows:

- Allow user editing of relationships via drag & drop between artifact elements of interest.
- When users select a node in the graph, all related nodes will be shown at the top to allow users easily to browse, and then users don't need to scroll up or down to find related items.
- Transform the graph to a hierarchical graph that can be expanded or contracted. For example, users can choose to expand or contract a node if it has children.
- When users select a node in the graph, related java file and documents will be opened, and related contents in the opened files will be highlighted, and the cursor will go to the first highlighted line.
- Integrate extracted relationships into the Eclipse IDE providing developers non-intrusive but effective access to related artifact elements in-situ when editing code or documentation elements.
- Use extracted and user-defined relationships for code and documentation quality analysis, e.g. to identify undocumented parts of code, potentially "stale" documentation with no code links, inconsistent documentation of code etc.

5. EXPERIMENTAL RESULTS

We have set up two case studies to evaluate the integration of IR and TM traceability recovery approach and compare the results of our technique with other recovery techniques. The first case study employs the Java SE Development Kit Version 5 (JDK 1.5) and a set of related PDF documents [18]. The second example is the User-friendly Desktop Internet Graphic Information System Version 1.1.1 (uDig 1.1.1) and a set of related PDF documents [25]. The experimental results show if we use IR or TM alone to recover links, some correct links are missed. If we only use the IR approach to extract relationships, some correct relationships cannot be captured: 30% for JDK1.5, 10% for uDig. Furthermore, for JDK, the Precision is only 19.49%, 35.47% for uDig. Both Precisions are quite low. If we only use the TM approach to extract relationships, 27% for JDK1.5 and 50% for uDig of total correct relationships cannot be recovered. However, it achieves a high Precision: 92.07% for JDK, and 80.47% for uDig. Combining IR and TM techniques can improve the Precision value and capture relationships that are missed by one or other technique.

We compared our new traceability recovery approach to the TM system developed by Witte et al [27]. Table 1 shows a comparison of results. We compared our result at each cut point to the evaluation result for the named entity recognition performance with the source code ontology in [27]. At the 0.1 cut point, our approach has the worst Precision 31%, though the Recall (98%) is higher than that of the TM system (87%) in [27]. Our Precision at the 0.3 cut point is only 2% lower than that of the TM system, but we have a low Recall 71%. Nevertheless, when the cut point is not less than 0.5, our approach achieves a 15%-19% increase in the Precision, but a 24%-27% decrease in the Recall on average. The main reason for causing this outcome is that our approach can deal with any kinds of document, including documents that contain source code identifiers and documents that have no or few code identifiers. Our approach can still reach a high Precision for documents that contain few code identifiers. The TM system of Witte et al is unable to retrieve relationships between source code and documents with few or no code identifiers because TM requires a direct match of tokens to code identifiers. Our integrated traceability recovery approach has better overall performance than the TM system of Witte et al.

Table 1. Comparison of results

		JDK1.5	uDig1.1.1	Average
Our proposed traceability recovery approach:				
>=0.1	Precision	24%	38%	31%
	Recall	99%	98%	98%
>=0.3	Precision	58%	72%	65%
	Recall	87%	55%	71%
>=0.5	Precision	84%	80%	82%
	Recall	80%	47%	63%
>=0.7	Precision	91%	80%	86%
	Recall	75%	46%	61%
>=0.9	Precision	92%	80%	86%
	Recall	74%	46%	60%
TM System by Witte et al [27]:				
	Precision	76%	58%	67%
	Recall	87%	87%	87%

6. CONCLUSION

Traceability systems face two key challenges: retrieving relationships between artifacts of a system at a high-level of quality and accuracy, and visualizing retrieved relationships in a natural and intuitive way. Many systems exist but none so far produces sufficiently consistent and high quality results as developers require. Our traceability system combines several traceability recovery techniques to extract relationships between artifacts to counteract each technique's weaknesses. These recovered relationships are visualized in a hierarchical rich graphical tree to help users easily browse and edit these links. We expect that this system will provide the following key novel contributions: recover relationships with higher quality and accuracy than existing tools, efficient and effective browsing and

maintainance of retrieved relationships; ease of use and understanding for developers during their everyday coding and documentation; and utilization of recovered, combined traceability relationships for software quality analysis. The preliminary experimental results demonstrated that combining IR and TM techniques can eliminate some limitations of IR and TM alone, and improve precision, recover relationships that are missed by IR or TM alone, and process a wide range of documents with no or few source code identifiers.

7. ACKNOWLEDGMENTS

The author wishes to gratefully acknowledge Prof. John Grundy from Swinburne University of Technology and Prof. John Hosking from the University of Auckland for their supervisions and contributions to this work; the support of the Foundation for Research, Science and Technology and the University of Auckland for their financial support of this research.

8. REFERENCES

[1] Antoniol, G., Canfora, G., Casazza, G., and Lucia, A. D. 2000. Information retrieval models for recovering traceability links between code and documentation. In Proc. Of IEEE Intl. Conf. On Software Maintenance 2000, San Jose, CA, USA

[2] Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D., and Merlo, E. 2002. Recovering traceability links between code and documentations. IEEE Transactions on Software Engineering, Vol. 28, No. 10, Oct. 2002, pp. 970-983

[3] Apache Lucene – Overview, 2009, Extracted on 4 December 2009 from http://lucene.apache.org/java/docs/

[4] Asuncion, H. U., Francois, F., and Taylor, R. N. 2007. An end-to-end industrial software traceability tool. ESEC-FSE'07, Sep. 3-7, 2007, Cavtat near Dubrovnik, Croatia, pp. 115-124

[5] Cleland-Huang, J., Settimi, R., Romanova, E., Berenbach, B., and Clark, S. 2007. Best practices for automated traceability. Computer, Vol. 40, Issue 6, pp. 27-35

[6] Desmond, M., Storey, M.-A., and Exton, C. 2006. Fluid source code views for just in-time comprehension. SPLAT'06, Bonn, Germany

[7] Domges, R. and Pohl, K. 1998. Adapting traceability environments to project specific needs. CACM, 41(12), pp. 54-62

[8] Eclipse Java Development Tools (JDT), 2010, Extracted on 7 April 2010 from http://www.eclipse.org/jdt

[9] Egyed, A. 2003. A scenario-driven approach to trace dependency analysis. IEEE Transactions on Software Engineering, 29(2), PP. 116-132

[10] Egyed, A., Biffl, S., Heindl, M., and Grunbacher, P. 2005. A value-based approach for understanding cost-benefit trade-offs during automated software traceability. TEFSE 2005, California, USA, pp. 2-7

[11] GATE – General Architecture for Text Engineering, 2009, Extracted on 4 Decmeber 2009 from http://gate.ac.uk/

[12] Gotel, O.C. and Finkelstein, A. C. W. 1994. An analysis of the requirements traceability problem. Proceedings 1st Int. Conf. in Requirement Engineering, Colorado Springs, Colo., pp. 94-101

[13] Jirapanthong, W. and Zisman, A. 2009. XTraQue: traceability for product line systems. Software and System Modeling Journal, Vol. 8, No. 1, pp. 1619-1366

[14] Larman, C. Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development (3rd edition). Published by Prentice Hall

[15] Lucia, A. D., Oliveto, R., and Tortora, G. 2008. ADAMS Re-Trace: traceability link recovery via latent semantic indexing. ICSE'08, Leipzig, Germany, pp. 839-842

[16] Marcus, A. and Maletic, J. I. 2003. Recovering documentation-to-source-code traceability links using latent semantic indexing. 25th ICSE'03, pp. 125-135

[17] Marcus, A., Xie, X., and Poshyvanyk, D. 2005. When and how to visualize traceability links? TEFSE'05, Long Beach, California, USA, pp. 56-61

[18] PDF Version of JDK Documentation, 2010, Extracted on 10 Feb. 2010 from http://java.sun.com/j2se/1.5.0/download-pdf.html

[19] Ramesh, B., Stubbs, C., Powers, T., and Edwards, M. 1997. Requirements traceability: theory and practice. Annals of Software Engineering 3 (1997), pp. 397-415

[20] Seacord, R., Plakosh, D., and Lewis, G. 2003. Modernizing legacy systems: software technologies, engineering processes, and business practices. SEI Series in SE. Addison-Wesley

[21] Storey, M.-A. 2006. Theories, tools and research methods in program comprehension: past, present and future. Software Quality Journal, Springer, 2006

[22] Tempero, E., Noble, J., and Biddle, R. 2002. Tool support for traceability between requirements and design. Technical report CS-TR-02/27, Computer Science, Victoria University of Wellington, New Zealand

[23] The New Zealand Digital Library Project, 2009. Extracted on 3 December 2009 from http://www.nzdl.org/cgi-bin/library.cgi

[24] The Standish Group Report: Chaos, 2007, Extracted on 1 May 2010 from http://www.projectsmart.co.uk/docs/chaos_report.pdf

[25] uDig GIS Documentation, 2010, Extracted on 10 Feb. 2010 from http://udig.refractions.net

[26] Wang, X., Lai, G., and Liu, C. 2009. Recovering relationships between documentation and source code based on the characteristics of software engineering. Electronic Notes in Theoretical Computer Science 243 (2009), Elsevier B. V., pp. 121-137

[27] Witte, R., Li, Q., Zhang, Y., and Rilling, J. 2007. Ontological text mining of software documents. NLDB 2007, pp. 168-180, Paris, Springer LNCS 4592

[28] Zest: the eclipse visualization toolkit, 2009, Extracted on 3 December 2009 from http://www.eclipse.org/gef/zes