

Automated Documentation to Code Traceability Link Recovery and Visualization

Xiaofan Chen

This thesis is submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, completed at the University of Auckland

August 2012

Abstract

Documentation written in natural language and source code are two of the major artifacts of a system. Tracking a variety of traceability links between documentation and code assists developers in comprehension, efficient development, and effective management of a system. However, automated traceability systems to date have faced with three major open research challenges.

The first challenge is how to extract links with both high precision and high recall. We introduce an approach that combines three supporting techniques, Regular Expression, Key Phrases, and Clustering, with Information Retrieval (IR) models to improve the performance of automated traceability recovery between documents and source code. This combination approach takes advantage of strengths of the three techniques to ameliorate limitations of IR models. Our experimental results show that our approach improves the performance of IR models, increases the precision of retrieved links, and recovers more true links than IR alone.

The second challenge is how to establish robust traceability benchmarks to evaluate traceability recovery techniques. We describe an approach and guidelines to enable researchers to establish affordable and robust traceability benchmarks. We have designed rigorous manual identification and verification strategies to determine whether or not a link is correct. We have developed a formula to calculate the probability of errors made in created benchmarks. The analysis of error probability results shows that our approach can build high quality benchmarks and our strategies significantly reduce the error probability in them.

The third challenge is how to efficiently visualize links for complex systems because of scalability and visual clutter issues. We present a new approach that combines treemap and hierarchical tree techniques to reduce visual clutter and to allow the visualization of the global structure of traces and a detailed overview of each trace, while still being highly scalable and interactive. The usability evaluation results show that our approach can effectively and efficiently help software developers comprehend, browse, and maintain large numbers of links.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude and thanks to my main supervisors, Prof. John Hosking and Prof. John Grundy, for their support, guidance, and mentorship over the past four years. I thank both of them for their patience, steadfast encouragement, invaluable advice, and for making this journey the most challenging and rewarding experience it has been. They have been my inspiration as I hurdle all the obstacles in the completion of this research work. I have always counted myself very blessed and lucky to have both of them as my supervisors.

I would like to extend my sincerest thanks and appreciation to my supervisor, Prof. Robert Amor, for his unselfish and unfailing support and guidance throughout my last year's PhD study. I thank him for being so willing to put me under his wings.

I would like to express my highest and warmest gratitude, appreciation, and thanks to my husband (Richard Lian Hsu), daughter (Doris Jia Hsin Hsu), parents, parents-in-law, and sisters for their endless love, encouragement, and support to me to go through the ups and downs of this journey.

I would also like to gratefully acknowledge Alberto Bacchelli at Università della Svizzera Italiana for agreeing to share his exemplar text sets and oracles.

I would like to acknowledge the financial support provided by the Foundation for Research, Science and Technology, the PReSS account of University of Auckland, the CS Graduate Student Travel fund, and the Software Process and Product Improvement Funding.

Last but not least, I would like to thank all the friends, colleagues, and staff at the Computer Science Department for all their help and support.

Table of Contents

| | |
|--|-----|
| Abstract | I |
| Acknowledgements | III |
| List of Tables | IX |
| List of Figures | XI |
| Chapter 1 -- Introduction | 1 |
| 1.1 Research Background and Motivation..... | 1 |
| 1.2 Research Questions..... | 5 |
| 1.3 Research Methodology | 6 |
| 1.4 Our Approach..... | 10 |
| 1.5 Research Contributions..... | 12 |
| 1.6 Thesis Organization | 14 |
| Chapter 2 -- Related Work | 17 |
| 2.1 Introduction..... | 17 |
| 2.2 Software Traceability | 18 |
| 2.3 Traceability Link Classifications | 19 |
| 2.4 Traceability Link Recovery | 22 |
| 2.4.1 Semi-automatic Traceability Recovery Techniques | 25 |
| 2.4.2 Automatic Traceability Recovery Techniques..... | 27 |
| 2.4.3 Enhancing Discovery Techniques | 37 |
| 2.4.4 Requirements for Successful Traceability Recovery Techniques | 39 |
| 2.5 Traceability Link Visualization..... | 41 |
| 2.5.1 Traditional Traceability Visualization Techniques | 45 |
| 2.5.2 Graph-based Traceability Visualization Techniques | 47 |
| 2.5.3 Other Traceability Visualization Techniques..... | 57 |
| 2.5.4 Hierarchical Structural Visualization Techniques | 61 |
| 2.5.5 Requirements for Traceability Visualization Systems..... | 62 |
| 2.6 Summary..... | 65 |
| Chapter 3 -- Traceability Link Recovery | 67 |
| 3.1 Introduction..... | 67 |

| | |
|--|------------|
| 3.2 Motivation | 69 |
| 3.3 IRETrace -- Information Retrieval Enhancement Traceability | 72 |
| 3.3.1 The Basic Retrieval Technique | 73 |
| 3.3.2 Regular Expression (RE) | 77 |
| 3.3.3 Key Phrases (KP)..... | 79 |
| 3.3.4 Clustering..... | 80 |
| 3.4 Implementation..... | 85 |
| 3.5 Summary | 90 |
| Chapter 4-- Traceability Benchmark | 91 |
| 4.1 Introduction | 91 |
| 4.2 Background and Motivation..... | 94 |
| 4.3 Traceability Benchmark Development..... | 96 |
| 4.3.1 Task Identification..... | 97 |
| 4.3.2 Artifact Selection | 98 |
| 4.3.3 Project Selection | 98 |
| 4.3.4 Oracle/True Traceability Link Set Development..... | 99 |
| 4.3.5 Evaluation Metrics..... | 105 |
| 4.4 Case Study | 107 |
| 4.5 Evaluation..... | 112 |
| 4.5.1 Requirements Review | 112 |
| 4.5.2 Probability of Errors in JDK1.5-SUBSET Benchmark | 114 |
| 4.5.3 Cost-quality Tradeoffs | 119 |
| 4.6 Discussion | 121 |
| 4.7 Summary | 123 |
| Chapter 5 -- Evaluation of Traceability Link Recovery | 125 |
| 5.1 Case Studies..... | 125 |
| 5.2 Evaluation Results..... | 126 |
| 5.2.1 Evaluation Results from Using the Apache Lucene Engine | 127 |
| 5.2.2 Evaluation Results from Using the Terrier IR Platform | 131 |
| 5.2.3 Comparison Results | 143 |
| 5.3 Performance of Our Approach | 145 |

| | |
|---|------------|
| 5.4 Discussion | 147 |
| 5.5 Summary | 151 |
| Chapter 6 -- Traceability Link Visualization | 153 |
| 6.1 Introduction | 153 |
| 6.2 Motivation | 155 |
| 6.3 DCTracVis -- Traceability Visualization between Documents and Source Code | 160 |
| 6.3.1 Treemap View | 161 |
| 6.3.2 Hierarchical Tree Views | 163 |
| 6.3.3 Editing Traceability Links | 165 |
| 6.3.4 Other Functionality | 166 |
| 6.4 Implementation | 166 |
| 6.4.1 User Interface of DCTracVis | 168 |
| 6.4.2 Treemap View | 170 |
| 6.4.3 Hierarchical Tree Views | 171 |
| 6.4.4 Editing Traceability Links | 173 |
| 6.4.5 Other Functionality | 179 |
| 6.5 Summary | 183 |
| Chapter 7 -- Evaluation of Traceability Link Visualization | 187 |
| 7.1 Introduction | 187 |
| 7.2 Evaluation Design | 188 |
| 7.2.1 Tasks | 188 |
| 7.2.2 Observation | 188 |
| 7.2.3 Questionnaire | 189 |
| 7.3 Evaluation Method | 190 |
| 7.4 Evaluation Results | 191 |
| 7.4.1 Background of Participants | 191 |
| 7.4.2 Tasks and Observation | 192 |
| 7.4.3 DCTracVis Usability Questions | 194 |
| 7.4.4 Comparative Questions of Visualization Views | 200 |
| 7.5 Discussion | 204 |
| 7.6 Summary | 205 |

| | |
|-------------------------------------|------------|
| Chapter 8 – Conclusions..... | 207 |
| 8.1 Thesis Summary | 207 |
| 8.2 Research Contributions | 210 |
| 8.3 Future Work | 211 |
| 8.4 Summary | 214 |
| Appendix A – Surveys..... | 215 |
| Bibliography | 245 |

List of Tables

| | |
|---|-----|
| Table 2.1 Definitions of Software Traceability | 18 |
| Table 2.2 Comparison among Traceability Recovery Techniques | 23 |
| Table 2.3 Comparison among Traceability Visualization Systems to Date..... | 43 |
| Table 3.1 A description of IR retrieval models in Terrier | 75 |
| Table 3.2 Sections Related to java.awt.dnd.DragSource..... | 83 |
| Table 4.1 JDK1.5-SUBSET Packages and Documents..... | 107 |
| Table 4.2 Comments of the First Six Participants at the First Stage | 110 |
| Table 4.3 No. of Conflict Links Produced at the end of Each Stage | 111 |
| Table 4.4 Comments of the Rest Five Participants | 111 |
| Table 4.5 Probability Distribution for the JDK1.5-SUBSET Oracle Link Set Based on the Example Error Rates of the Three Types of Participants | 115 |
| Table 4.6 Actual Error Rates for the Three Types of Participants during the Establishment of the JDK1.5-SUBSET Oracle Link Set | 117 |
| Table 4.7 Probability Distribution for the JDK1.5-SUBSET Oracle Link Set Based on the Estimated Error Rates for the Three Types of Participants | 118 |
| Table 5.1 Details of Each Case Study | 126 |
| Table 5.2 Precision and Recall Results at Cut Points 0.02 and 0.9 Using VSM..... | 127 |
| Table 5.3 Precision and Recall Results at Cut Points 0.02 and 0.9 Using TF_IDF | 132 |
| Table 5.4 Precision and Recall Results at Cut Points 0.02 and 0.9 Using PL2 | 134 |
| Table 5.5 Precision and Recall Results at Cut Points 0.02 and 0.9 Using BM25 | 136 |
| Table 5.6 Precision and Recall Results at Cut Points 0.02 and 0.9 Using DLH | 138 |
| Table 5.7 Precision and Recall Results at Cut Points 0.02 and 0.9 Using IFB2 | 141 |
| Table 5.8 Percentages of Sections/Emails No Class Names in the Four Cases | 147 |
| Table 5.9 Percentages of Classes without Comments in the Four Cases | 148 |
| Table 5.10 Details of Classes with Extracted Key Phrases in the Four Cases | 149 |
| Table 5.11 Percentages of Classes with Related Key Phrases in the Four Cases..... | 149 |
| Table 6.1 Three Colour Ranges Indicating the Number of Links Each Node has | 163 |
| Table 7.1 DCTracVis Functions Usage Statistic | 193 |

| | |
|--|-----|
| Table 7.2 Comments of Participants Made to DCTracVis | 199 |
| Table 7.3 Reasons Given by Participants in Deciding the Best Visualization View..... | 202 |
| Table 7.4 Comments of Participants Made to the Visualization Views | 203 |

List of Figures

| | |
|--|-----|
| Figure 1.1 Development Process of Our Traceability Recovery Technique | 7 |
| Figure 1.2 Development Process of Our Traceability Visualization System | 10 |
| Figure 2.1 Traditional Traceability Visualization Techniques (Winkler and Pilgrim, 2010) | 45 |
| Figure 2.2 Traceability Matrix in Trace/Analyzer (Egyed, 2006)..... | 46 |
| Figure 2.3 Traceability Link Definition Form in ADAMS (ADAMS 2009) | 48 |
| Figure 2.4 Traceability Graph Visualization in ADAMS (ADAMS 2009)..... | 48 |
| Figure 2.5 GUI Prototype Showing Links in a Hierarchical Graphical Structure (Cleland-Huang and Habrat, 2007)..... | 49 |
| Figure 2.6 ENVISION User Interface (Zhou et al., 2008)..... | 51 |
| Figure 2.7 TBreq Traceability Link Visualization (LDRA, 2012)..... | 52 |
| Figure 2.8 The Sunburst Visualization (Merten et al., 2011) | 53 |
| Figure 2.9 The Netmap Visualization (Merten et al., 2011)..... | 53 |
| Figure 2.10 The User Interface of EXTRAVIS (Cornelissen et al., 2007)..... | 55 |
| Figure 2.11 The Visualization of TraceVis (van Ravensteijn, 2011)..... | 56 |
| Figure 2.12 The Visualization User Interface of UniTI (Pilgrim et al., 2008) | 57 |
| Figure 2.13 Showing Links in Clusters in Poirot (Cleland-Huang and Habrat, 2007) | 58 |
| Figure 2.14 The TraceViz User Interface (Marcus et al., 2005)..... | 59 |
| Figure 2.15 The User Interface of LeanArt (Grechanik et al., 2007)..... | 60 |
| Figure 2.16 Hierarchical Tree Visualization (Holten, 2006) | 62 |
| Figure 2.17 Treemap Layout (Holten, 2006)..... | 62 |
| Figure 3.1 the Sequence Diagram of Clustering Retrieved Links of a Class | 80 |
| Figure 3.2 Precision, Recall, and F-measure Results when $s = 0.2, 0.3, 0.4,$ or 0.5 in JDK1.5-SUBSET | 81 |
| Figure 3.3 Clustering Links of <code>java.awt.dnd.DragSource</code> | 84 |
| Figure 3.4 Traceability Recovery Process of IRETrace | 86 |
| Figure 3.5 Architecture of IRETrace | 88 |
| Figure 3.6 The User Interface of IRETrace | 89 |
| Figure 4.1 The Strategy of an Oracle Traceability Link Set Development | 100 |

| | |
|--|-----|
| Figure 4.2 Time Taken by the First Six Participants in Capturing Links at the First Stage..... | 109 |
| Figure 4.3 Percentages of Links Captured by the First Six Participants at the First Stage (Determining Correct Links vs. Conflict Links)..... | 109 |
| Figure 4.4 Time Taken by Rest Participants in Verifying Retrieved Links | 111 |
| Figure 5.1 Precision Results for the Four Cases Using VSM..... | 128 |
| Figure 5.2 Recall Results for the Four Cases Using VSM | 128 |
| Figure 5.3 F-measure ($\beta=1$) Results for the Four Cases Using VSM..... | 130 |
| Figure 5.4 Precision/Recall Results for the Four Cases Using TF_IDF | 132 |
| Figure 5.5 F-measure ($\beta=1$) Results for the Four Cases Using TF_IDF | 133 |
| Figure 5.6 Precision/Recall Results for the Four Cases Using PL2 | 134 |
| Figure 5.7 F-measure ($\beta=1$) Results for the Four Cases Using PL2..... | 135 |
| Figure 5.8 Precision/Recall Results for the Four Cases Using BM25 | 137 |
| Figure 5.9 F-measure ($\beta=1$) Results for the Four Cases Using BM25 | 137 |
| Figure 5.10 Precision/Recall Results for the Four Cases Using DLH..... | 139 |
| Figure 5.11 F-measure ($\beta=1$) Results for the Four Cases Using DLH | 140 |
| Figure 5.12 Precision/Recall Results for the Four Cases Using IFB2 | 141 |
| Figure 5.13 F-measure ($\beta=1$) Results for the Four Cases Using IFB2 | 142 |
| Figure 5.14 Comparison Results between IR Only and IR+RE+KP+Clustering | 144 |
| Figure 5.15 Execution Times for Different Combinations with Different IR models | 146 |
| Figure 6.1 Displaying Traceability Links between Nodes Using (a) Straight/linear edges; (b) Curved link edges; (c) and (d) Edges grouped by Hierarchical Edge Bundling. (Holten, 2006) .. | 162 |
| Figure 6.2 Showing Traceability Links in the Hierarchical Tree Layout: (a) Links as edges between nodes (Holten, 2006), (b) Links as children of nodes..... | 164 |
| Figure 6.3 the Sequence Diagram for Visualizing Links in a Project..... | 165 |
| Figure 6.4 Architecture of DCTracVis | 167 |
| Figure 6.5 How to Start Visualizing Links in a Project under Trace | 168 |
| Figure 6.6 The User Interface of DCTracVis Using the Treemap | 169 |
| Figure 6.7 The User Interface of DCTracVis Using the Whole HT | 169 |
| Figure 6.8 The Treemap View..... | 170 |
| Figure 6.9 The Whole HT View When a Class is Selected..... | 171 |
| Figure 6.10 The Whole HT View When a Section is Selected | 172 |

| | |
|---|-----|
| Figure 6.11 The Detail HT Contracted | 172 |
| Figure 6.12 The Detail HT Expanded | 173 |
| Figure 6.13 Popup Menu in the Treemap | 173 |
| Figure 6.14 Popup Menu in the Whole HT | 174 |
| Figure 6.15 Popup Menu in the Detail HT | 175 |
| Figure 6.16 Add a Link in the Detail HT View | 176 |
| Figure 6.17 A New Link is Added to “Binding” Class Shown in the Treemap | 176 |
| Figure 6.18 A New Link is Added to “Binding” Class Shown in the Whole HT | 177 |
| Figure 6.19 Change the Similarity Value in the Detail HT | 177 |
| Figure 6.20 Editing an Existing Link in the Detail HT | 177 |
| Figure 6.21 Changing an Existing Link in the Detail HT is Reflected in the Treemap | 178 |
| Figure 6.22 Changing an Existing Link in the Detail HT is Reflected in the Whole HT | 178 |
| Figure 6.23 Showing Contents of a Node in the Treemap | 179 |
| Figure 6.24 Selecting a Node in the Navigator is Reflected in the Treemap | 180 |
| Figure 6.25 Selecting a Node in the Navigator is Reflected in the Whole HT | 180 |
| Figure 6.26 Use Search to Find a Node | 181 |
| Figure 6.27 Methods in the Filter | 181 |
| Figure 6.28 Links Visualization of Using Different Similarity Score Level | 182 |
| Figure 7.1 The Background Information of Participants | 191 |
| Figure 7.2 Times Taken by Participants in Completing Each of the Three Tasks | 192 |
| Figure 7.3 Results of the Evaluation in the Functionality of DCTracVis | 195 |
| Figure 7.4 Average Ratings of the Functionality of DCTracVis | 196 |
| Figure 7.5 Results of the Evaluation in the Overall Performance of DCTracVis | 197 |
| Figure 7.6 Average Ratings of the Overall Performance of DCTracVis | 198 |
| Figure 7.7 Results of Comparison among the Three Visualization Views | 201 |

Chapter 1 -- Introduction

This chapter provides an overview of this thesis. It starts with the research background and motivation from which our research derived its goals and objectives. The key research questions are then introduced. Subsequently, our research methodology is outlined, followed by a description of the approach employed to fulfill the objectives of this research. Finally, the key research contributions are presented and an outline of the thesis structure is provided.

1.1 Research Background and Motivation

Have you ever struggled to understand a new system and to connect code with corresponding documents? Have you ever faced the difficulty of identifying impacted artifacts while making changes? Studies of software maintainers have shown that more than 50% of their time is spent on the process of program comprehension (Fjeldstad and Hamlen, 1983; Standish, 1984). In practice, artifacts produced during the software development life cycle (SDLC), such as source code, designs, requirements and documentation, end up being disconnected from each other. They are often separated into different documents, file formats and repositories, created and maintained by different individuals, and evolve at different rates (Antoniol et al., 2000b; Jin and Cordy 2005; Settimi et al., 2004). These disconnected artifacts hinder engineers' comprehension and undertaking effective development, efficient management, and improved maintenance of a system.

Implementing effective traceability support during the SDLC can ameliorate this issue by allowing engineers to navigate between and browse more effectively related artifacts (Cleland-Huang et al., 2007; Gotel and Finkelstein 1994; Rilling et al., 2007; Watkins and Neal 1994). Gotel and Finkelstein (1994) define software traceability as the ability to follow the life of a requirement in a

forward and backward direction. In other words, traceability is the ability to relate artifacts produced during the SDLC. Nevertheless, traceability within the software development process has not been employed in many organizations due to high costs, complexity, limited tool support, and the need for human intervention in most traceability systems (Cleland-Huang et al., 2011; Oliveto et al., 2007; Ramesh and Jarke, 2001). This is even though standards like IEEE Std. 830-1998, DO-178B, ISO15504, CMMI mandate use of traceability techniques during software development (Cleland-Huang et al., 2007; Rilling et al., 2007).

Unfortunately, it is painstaking, error-prone, complex and time-consuming work to manually retrieve and maintain traceability relationships, or links, between software artifacts. These efforts can be significantly reduced by applying traceability recovery approaches to automatically obtain high quality relationships/links between elements in one artifact and elements in another (Penta et al., 2002; Settimi et al., 2004; Spanoudakis and Zisman, 2005), and adopting traceability visualization techniques to represent these retrieved relationships in a natural and intuitive way (Asuncion et al., 2007; Roman & Cox, 1992). High quality relationships represent a link set containing as many as possible correct relationships and as few as possible incorrect relationships. Moreover, high quality relationships connect elements of different artifacts on a fine-grained level of detail e.g. part of a design document description and its related source code elements.

Various traceability recovery techniques (discussed in detail in Chapter 2) have been developed to capture traceability links between artifacts. Some need human intervention while others can automatically capture links. Unfortunately, no recovery approaches to date have the capability of recovering all possible links between artifacts automatically and accurately. Most automatic traceability recovery approaches use Information Retrieval (IR) models to extract links between artifacts. IR is an area that studies the problem of finding relevant information in text collections based on user queries (Hayes et al., 2003; Spanoudakis and Zisman, 2005). In other words, IR models determine how relevant a piece of text is to a query that represents a user's interest by computing a similarity value according to the frequency and distribution of keywords or terms in textual format document collections (Antoniol et al., 2002; Marcus and Maletic, 2003). The accuracy rate of the extracted traceability links depends heavily on a cut point that decides which links can be recovered. Only links that have a similarity value greater or equal to the cut point are

finally retrieved. As a result, many incorrect links are obtained at low cut points and few correct links are returned at high cut points. This issue motivated us to develop a new traceability recovery approach that can automatically recover high quality links between artifacts in the traced system at all cut points. This was the first challenge of our research.

In order to evaluate the performance of a traceability recovery technique, validate the technique, and compare its performance with other traceability recovery techniques, traceability benchmarks are one of the essential elements in the evaluation process (Cleland-Huang et al., 2011; Dekhtyar et al., 2007; Sim et al., 2003). A traceability benchmark serves as a basis for assessing a technique and then compares it to others (Cleland-Huang et al., 2011). A benchmark can determine whether a retrieved link is correct or incorrect and whether any correct links fail to be retrieved. Without such data, the performance of a traceability recovery technique cannot be measured or compared with the performance of others. However, it is difficult to obtain or establish such benchmarks (Cleland-Huang et al., 2006). Moreover, there has been little research on building generally agreed or applied approaches or guidelines to assist researchers in manually creating traceability benchmarks. Therefore, we were motivated to propose a new approach and set of operational guidelines for the manual establishment of a robust benchmark. This was the second challenge of our research.

While traceability links between artifacts are captured by a traceability recovery technique, a remaining key issue is how to represent these retrieved links to assist software engineers to effectively and efficiently understand, browse, and maintain them. Adopting software visualization techniques (e.g. tree-based, graph-based, or 3D-based approaches) is a common way to display retrieved links (Asuncion et al., 2007; Roman & Cox, 1992). However, displaying an overwhelmingly large number of traceability links effectively and efficiently is a big challenge, because a software system with large numbers of artifacts, and thus very large numbers of traceability links between artifacts, quickly gives rise to scalability and visual clutter issues (Cornelissen et al., 2007; Holten, 2006; Merten et al., 2011). Moreover, the efficient visualization of both the structure of the traced system and the enormous number of links between artifacts is far from trivial (Cornelissen et al., 2007; Marcus et al., 2005). Many traceability visualization techniques (discussed in detail in Chapter 2) have been designed and developed to represent

traceability links. To date, however, no traceability visualization techniques can visualize an overwhelmingly large number of traceability links effectively and efficiently without scalability and visual clutter issues. In order to remedy these issues, we were motivated to design and develop a new traceability visualization technique to visualize traceability links in a natural and intuitive way. This was the third challenge of our research.

Source code and documents constitute a large number of artifacts produced during the SDLC. Typical documents are those written in natural language such as tutorials, handbooks, developer or user guides, API documentation, architecture descriptions, design rationale, emails and so on. Software developers develop code and documentation artifacts during the lifetime of a system. Sometimes this is done concurrently while at other times different developers develop different artifacts, and often these artifacts get out of synchronization. Tracing and maintaining interrelationships between code and documentation can assist developers to better understand systems, better maintenance of systems, and to produce higher quality systems. Our research focuses on recovering and displaying relationships between source code and documents generated during the SDLC.

Overall, the main motivation of our research lay in the potential to recover traceability links with high accuracy at all cut points, to make maintenance of links more efficient and effective, and to make manually establishing a robust traceability benchmark easier and more effective. Our research targets links between source code and documentation. The aim of our research was to provide users with an effective environment enabling them to automatically capture, browse, and maintain traceability links between artifacts effectively and efficiently. With this environment, users can trace links between various documents and source code, automatically recover links at low cost and high accuracy, easily create and modify links as well as conveniently browse and maintain links.

1.2 Research Questions

The main research question for our research can be posed as:

Can traceability between artifacts in a system enable software engineers to better understand, maintain, and manage the system?

In order to address this main question, we divided it into three smaller research questions.

- 1) *Can the performance of an IR-based traceability recovery technique be improved to retrieve high quality links at all cut points?* This question focuses on how to ameliorate the limitations of IR-based traceability recovery techniques to produce as many correct links and as few incorrect links as possible at any cut point. To answer this question, we proposed a combination approach, implemented a prototype of the approach in an automatic traceability tool, and conducted a performance evaluation of the technique. This is addressed in Chapter 3 (Traceability Link Recovery) and Chapter 5 (Evaluation of Traceability Link Recovery).
- 2) *How can we manually build a robust traceability benchmark to evaluate the performance of a traceability recovery technique?* This question focuses on the establishment of a robust traceability benchmark. To answer it, we proposed a new approach and set of operational guidelines to assist users to create such a benchmark manually and of high quality. This is addressed in Chapter 4 (Traceability Benchmark).
- 3) *Can traceability visualization enable users to better understand, browse, and maintain traceability links in a system?* This question focuses on how to visualize traceability links captured by a traceability recovery technique in a natural and intuitive way. To answer this research question we designed, prototyped and evaluated a combination visualization technique. This is addressed in Chapter 6 (Traceability Link Visualization) and Chapter 7 (Evaluation of Traceability Link Visualization).

1.3 Research Methodology

Our approach for addressing the research questions was based on the following methodology:

- 1) In order to address the first research question, we have adopted the iterative and incremental development method (Larman and Basili, 2006; Robey et al., 2001; Zhang et al., 2010) to develop our recovery system through repeated cycles and in smaller portions at a time. This method explores issues and problems in designing a system (Robey et al., 2001; Zhang et al., 2010). Moreover, it gives software engineers opportunities to improve the development process and the product quality (Cockburn, 2008). The key steps are as follows.
 - a. We conducted an extensive literature review of traceability recovery techniques. We studied many papers that described traceability recovery techniques to capture links between artifacts in a system. This step allowed us to compare and analyze their approaches for link recovery, identify common properties in traceability systems, and explore valuable ideas that can be incorporated into a new, improved recovery system. The aim of this task was to assist us in the development of our own traceability recovery system to create high quality links in the system.
 - b. From this, we identified a set of key requirements for an improved traceability recovery system. The findings from the literature helped us to obtain these requirements. This set of requirements helped us to explore what an improved traceability recovery system needs and then to design and develop our traceability recovery system.
 - c. In light of the set of requirements identified from the literature, we developed a combination link recovery approach through the iterative and incremental development method, which is a recommended practice widely used in the software development processes (Larman and Basili, 2006; Robey et al., 2001; Zhang et al., 2010). We chose this because iterative and incremental development is not just about revisiting work but also evolutionary advancement (Larman and Basili, 2006). It improves the weaknesses of the waterfall method that include no feedback between phases, hard dates on phases, and no completion criteria (Martin, 1999). We had developed several prototypes for our recovery system (see Figure 1.1). Because prototypes are generally produced quickly and offer valuable feedback on the feasibility and usefulness of a system's design and specifications

(Robey et al., 2001). Initially, we attempted to adopt an IR model, Vector Space Model (VSM), to retrieve links between artifacts (Prototype 1). The limitations of VSM motivated us to add supporting techniques into VSM to improve its performance. We then explored adding Text Mining (TM) into VSM (Prototype 2) to seek to recover more correct links than VSM alone. The issues we experienced in using TM inspired us to find an alternative. We employed Regular Expression (RE) as the alternative supporting technique to be integrated with VSM (Prototype 3) to increase the number of correct links. The limitations of RE motivated us to add other supporting techniques. In the fourth prototype, we used Key Phrases (KP) to capture links that are possibly missed by VSM+RE. We adopted Clustering technique to reduce the number of retrieved incorrect links in the fifth prototype. In the final prototype, we integrated the three supporting techniques with five additional IR models to explore whether they could improve other IR models' performances. Our final traceability recovery approach incorporated three enhancement strategies, RE, KP, and Clustering, into IR models to improve the performance of automated traceability link recovery to obtain high quality links.

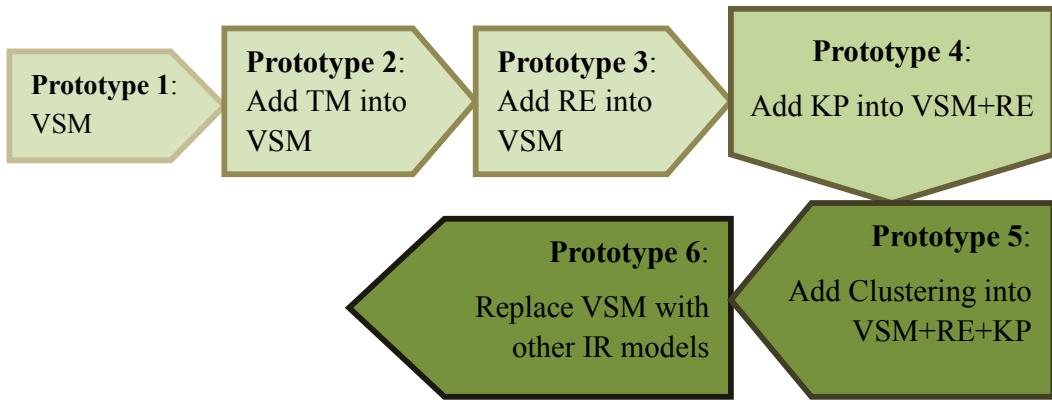


FIGURE 1.1 DEVELOPMENT PROCESS OF OUR TRACEABILITY RECOVERY TECHNIQUE

- d. After implementing the combination recovery approach, we evaluated it using a case studies approach, which is an empirical inquiry that investigates a phenomenon within its real-life context to help researchers gain a sharpened understanding of why the phenomenon happened as it did and encourages exploration into possible future research (Easterbrook et al., 2007; Yin, 2009). We chose this approach as it is feasible to use when investigating how or why certain phenomena occur (Easterbrook et al., 2007; Yin, 2009). Moreover, it is widely and commonly used for the evaluation of a traceability recovery

technique (Antoniol et al., 2002; Cleland-Huang et al., 2005; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009). We applied four different case studies with their own benchmarks. We used it with six IR models to investigate whether the three supporting strategies could improve the key limitations of IR models. The aim of this evaluation was to find out whether our recovery approach could create high quality links at all cut points. This evaluation helped us to achieve in-depth understanding of how and why our recovery approach could improve the performance of automated link recovery, and reveal limitations of our recovery approach. The experimental results for each case study were compared with its benchmark to calculate precision, recall, and F-measure. The values for the three metrics helped us to measure how the performance of our approach was.

- 2) In order to conduct the evaluation of the proposed recovery approach, traceability benchmarks need to be created. To address the second research question, we first conducted a literature review of traceability benchmarks. We studied papers that discussed traceability benchmarks. This helped us to identify the properties for benchmarks. We then proposed a new procedure and set of guidelines to help the manual establishment of a high quality, robust benchmark. The case study approach (Easterbrook et al., 2007; Yin, 2009) was also used to evaluate our approach because it is a feasible method to explore how and why our proposed guideline can produce a robust and effective benchmark. We created a traceability benchmark for a case project following our proposed guidelines. We used a probability formula to calculate the probability of errors made in the benchmark. The aim of this formula was to show the accuracy of the benchmark that was created using our approach and explore what factors could influence the accuracy of the benchmark. The case study approach helped us to gain the understanding of how our proposed guidelines would work and reveal the shortcomings of our approach.
- 3) In order to answer the third research question, we also adopted the iterative and incremental development approach (Larman and Basili, 2006; Robey et al., 2001; Zhang et al., 2010) to develop our traceability visualization system. The key steps are as follows.
 - a. We conducted an extensive literature review of traceability visualization techniques. We studied many papers that described traceability visualization techniques as a supporting

technique for software engineers in the system comprehension, management, and maintenance. We compared and analyzed these approaches for link visualization to identify common properties in traceability visualization systems and to explore valuable ideas for building a new, improved traceability visualization system. The aim of this task was to assist us in the development of our traceability visualization system to effectively and efficiently create and visualize links in the system.

- b. From this, we identified a set of key requirements for an improved traceability visualization system. The findings from the literature helped us to obtain these requirements and thus to understand what an improved traceability visualization system needs, and to design and develop our traceability visualization system.
- c. Based on the set of requirements identified from the literature, we designed and developed a combination visualization technique applying the iterative and incremental development method. We chose this method as it is a highly recommended practice used in the software development processes (Larman and Basili, 2006; Robey et al., 2001; Zhang et al., 2010). We had developed several prototypes for our visualization system (see Figure 1.2) to explore issues and problems occurred in developing our visualization system. Our initial attempt was to visualize retrieved links in a tree that could not be expanded or collapsed and to store them in a traceability matrix. The difficulties we experienced in browsing and maintaining links in this attempt motivated us to develop another prototype. The second prototype was to display links in a treemap to effectively use the display space, and apply colors to differentiate the link status of each node in the treemap instead of directly drawing edges between related nodes on top of the treemap to avoid visual clutter. The limitations of treemap in this prototype inspired us to add supporting techniques to improve the link visualization. In the third prototype, we additionally visualized links in a hierarchical tree that had the ability to expand and collapse to be space-effective and added links as children of nodes in the hierarchical tree to avoid visual clutter. Our final traceability visualization technique was to visualize retrieved links in the treemap and two hierarchical trees to support the efficient visualization of the structure and traceability links of the traced system.

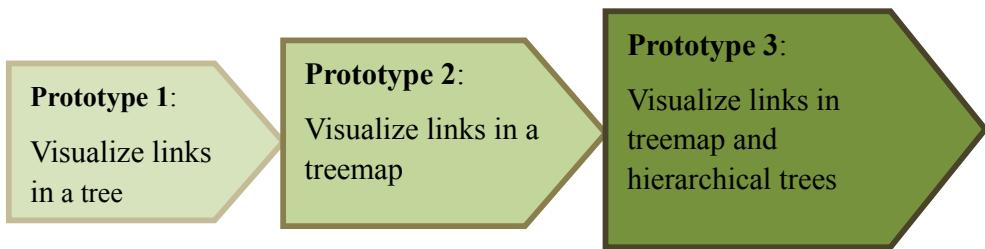


FIGURE 1.2 DEVELOPMENT PROCESS OF OUR TRACEABILITY VISUALIZATION SYSTEM

- d. After the implementation of the new visualization technique we conducted a formal usability evaluation, which is the only mechanism that allows researchers to obtain direct, detailed information on the user's experience with the system being tested (Nielsen, 1993). We chose this because it helps developers to understand barriers to user interface and design errors, and then to refine the system (Nielsen, 1993). This study helped us to understand whether and how our approach assisted users in comprehension, development, and management of the traced system. It also showed whether such link visualizations could help users understand, browse, and maintain the recovered links easily and efficiently. Moreover, this study revealed the strengths and weaknesses of our system. The study was carried out with participants who had computer science or software engineering background. There were no requirements for participants to have some knowledge about the traced system. The methods we employed in our study were: questionnaires, observation, and think aloud.

1.4 Our Approach

We briefly describe our approach that recovers and visualizes traceability links and our guidelines for the manual establishment of robust traceability benchmarks. The detailed descriptions are in Chapters 3, 4 and 6.

In order to improve the accuracy of retrieved traceability links to a reasonably high level at all levels of cut points, we have been exploring a new approach combining Regular Expressions (RE), Key Phrases (KP), and Clustering techniques with IR models to recover links between sections in

documents and class entities. Our approach is intended to overcome the limitations of IR by taking advantage of the strengths of RE, KP, and Clustering. Combining RE with IR models allows extraction of more correct links at high cut points. As long as class names are retrieved correctly and refined regular expressions are built, RE can retrieve all possible links that are related to these class names and return few incorrect links as well. Adding KP enables IR to generate all potential links by extending the IR queries to include key phrases from comments in the source code. If source code is well documented, KP can extract key phrases from comments closely related to classes. The majority of incorrect links at low cut points are discarded by adopting Clustering, which takes advantage of the inherent hierarchical structure of documents to cluster links retrieved by IR models, RE, and KP. Therefore, our combination approach increases the number of correct links at high cut points and reduces the number of incorrect links at any cut point.

In terms of the manual establishment of robust traceability benchmarks, we have defined a traceability benchmark to include tasks, dataset, an oracle (or true) traceability link set, and measures. We have proposed five steps to establish such a robust traceability benchmark: task identification, artifact selection, project selection, oracle/true traceability link set development, and evaluation metrics.

- (1) Task identification aims at capturing the set of traceability tasks.
- (2) Artifact selection is to choose which artifacts to collect based on the tasks.
- (3) Project selection is to find appropriate software projects containing artifacts that are chosen in the step two.
- (4) Oracle/true traceability link set development is to manually establish correct links between selected artifacts. We have designed rigorous manual identification and verification strategies to assist researchers to identify and verify links.
- (5) Evaluation metrics are to provide metrics that can be used to measure the performance of recovery techniques.

In order to provide effective and efficient traceability visualization, we explored an approach of combining enclosure and node-link representations to display the structure of the traced system and the overall overview of traceability links, and to provide a detailed overview of each link

while still being highly scalable and interactive. We utilize a treemap view to display the structure of the system under trace and the overall overview of links. In order to remedy the visual clutter issue, we apply colours to differentiate the relationship status of each node in the treemap instead of drawing edges directly over the treemap. We adopt two hierarchical trees that can be expanded and contracted to visualize links. A hierarchical tree (the whole HT) is to illustrate the whole system and links in it to convey the hierarchical structure of the system. When an item is selected in the treemap view or the whole HT, the detailed dependency information of the selected item is displayed in a new hierarchical tree (the detailed HT), which is treated as the supplement of the treemap and the Whole HT. Any changes to links made in the treemap are reflected in the two hierarchical trees, and vice versa.

1.5 Research Contributions

Our research presented and discussed in this thesis contributes to the field of software engineering particularly in the area of software traceability. The main contributions from our research are as follows.

- 1) We invented a new automatic traceability recovery technique, **IРЕTrace**, which incorporates three supporting techniques, Regular Expression (RE), Key Phrases (KP) and Clustering, into IR models to improve the performance of IR models. This technique can increase the number of retrieved correct links and decrease the number of retrieved incorrect links at all cut points. Papers describing this work include the following.
 - a. “Extracting and Visualizing Traceability Links between Documents and Source Code” which was published in Proceedings of the 21st Australian Software Engineering Conference (ASWEC 2010).
 - b. “Extraction and Visualization of Traceability Relationships between Documents and Source Code” which was published in Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE 2010).
 - c. “Enhancing Automated Traceability via Combination of Multiple Techniques” which was published in Proceedings of the 33rd International Conference on Software Engineering

(ICSE 2011).

- d. “Improving Automated Documentation to Code Traceability by Combining Retrieval Techniques” which was published in Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011).
- 2) We have proposed a new approach and operational guidelines to assist engineers to manually develop their own robust traceability benchmark easily and effectively. We used this approach and guidelines while creating a new traceability link benchmark for the JDK1.5-SUBSET source code and documentation artifacts. We have made our new JDK1.5-SUBSET benchmark public and allow other users to access or download it for free. Our benchmark is represented in a spreadsheet format. Anyone can review the data, apply it to evaluate their traceability approaches, and probably extend it to better meet their own needs. Users can download it from: <http://tinyurl.com/7l3ohe4>. The development of the JDK1.5-SUBSET benchmark is described in the following paper.
- a. “Improving Automated Documentation to Code Traceability by Combining Retrieval Techniques” which was published in Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011).
- 3) We have invented a combination traceability visualization technique that combines treemap and hierarchical tree techniques to display the structure and traceability links of a system effectively and efficiently without scalability and visual clutter issues. We then adopted this visualization approach to build a new traceability visualization system, **DCTracVis**. This system includes navigator, search, and filter functions to help engineers locate particular nodes and filter out uninteresting links. This research was reported in the following papers:
- a. “Extracting and Visualizing Traceability Links between Documents and Source Code” which was published in Proceedings of the 21st Australian Software Engineering Conference (ASWEC 2010).
 - b. “Extraction and Visualization of Traceability Relationships between Documents and Source Code” which was published in Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE 2010).
 - c. “Visualizing Traceability Links between Source Code and Documentation” which was

published in Proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2012).

1.6 Thesis Organization

The remaining chapters are organized as follows.

Chapter 2: Related Work

This chapter reviews existing traceability recovery techniques and traceability visualization systems. These techniques are compared and analyzed to investigate and explore important requirements that a successful traceability recovery and visualization system needs to meet.

Chapter 3: Traceability Link Recovery

This chapter describes the design and implementation of a combination traceability recovery technique that we have invented, called **IRETrace**. This more effectively addresses the first challenge we encounter in software traceability - how to go about automatically retrieving high quality relationships between software artifacts.

Chapter 4: Traceability Benchmark

This chapter tackles the second major challenge we face: establishing appropriate traceability benchmarks to evaluate a traceability recovery techniques against prior studies and techniques. We describe a new approach that we have developed and trialed for researchers to manually build robust traceability benchmarks more easily and more effectively. We propose a formula to calculate the probability of errors in the created traceability benchmarks. We used our new approach to create a benchmark for JDK1.5-SUBSET source code and documentation.

Chapter 5: Evaluation of Traceability Link Recovery

This chapter presents an evaluation of our combination traceability recovery approach, IRETrace. We applied six Information Retrieval (IR) models to four case studies varying in size and context. The evaluation results are then analyzed to explore the strengths and

weaknesses of IRETrace.

Chapter 6: Traceability Link Visualization

The third challenge we face in software traceability is to how to efficiently and effectively visualize traceability links retrieved by a traceability recovery technique to support the comprehension, browsing, and maintenance of links in a system. This chapter describes a combination visualization system that we have invented, called **DCTracVis**. It allows software engineers to recover, browse, understand, and maintain links in a natural and intuitive way.

Chapter 7: Evaluation of Traceability Link Visualization

This chapter presents a usability evaluation of our traceability link visualization tool, DCTracVis, and discusses the evaluation results to explore its strengths and weaknesses from the perspective of several software engineering participants.

Chapter 8: Conclusions

This final chapter summarizes and concludes the work presented in this thesis, reiterates our key research contributions, and suggests some possible future research directions.

Chapter 2 -- Related Work

This chapter presents a literature review. Related work on software traceability link recovery and visualization is reviewed to identify key strengths and weaknesses in current approaches. We then identify key requirements that any new system must meet in order to achieve successful traceability recovery and visualization techniques. Such a system must be able to retrieve links with both high precision and recall and must also be able to represent retrieved links in a natural and intuitive way.

2.1 Introduction

Due to the importance of tracing the relationships between artifacts in a system to assist software engineers in software development activities, extensive efforts have been put into improving the precision and recall of recovered links between artifacts through various traceability recovery techniques. Moreover, many researchers have designed and developed traceability visualization systems to provide engineers with an effective visualization environment enabling them to retrieve, browse, edit, and maintain retrieved links effectively and efficiently. In this chapter, we analyze many existing traceability recovery techniques and traceability visualization systems to determine a set of requirements for a successful traceability recovery and visualization system.

We firstly introduce what software traceability is, followed by a description of traceability link types. We then discuss representative traceability recovery techniques invented to date to explore requirements involved in accomplishing successful link recovery. Next, a variety of traceability visualization systems invented to date is discussed to seek feasible functionality in achieving effective and efficient link visualization. We also describe some other techniques that can be used

to recover links or display links in a system. Finally, a summary is presented.

2.2 Software Traceability

Traceability in the software engineering area is the ability to relate artifacts in a software system (Spanoudakis and Zisman, 2005). Table 2.1 lists some definitions of software traceability found in the literature. Definitions in Table 2.1 indicate that traceability generally means the ability to trace dependent artifacts produced during the software development life cycle and to capture interrelationships among artifacts within the software system.

TABLE 2.1 DEFINITIONS OF SOFTWARE TRACEABILITY

| Definition | Defined by (year) |
|--|------------------------------|
| “Traceability refers to the ability to allow changes to any artifacts -- requirements, specification, implementation -- to be traced throughout the system”. | Greenspan and McGowan (1978) |
| “Traceability refers to the ability to discover the history of every feature of a system”. | Hamilton and Beeby (1991) |
| “Traceability is a technique used to provide a relationship between the requirements, the design, and the final implementation of the system”. | Edwards and Howell (1992) |
| “Traceability is a link or definable relationship document between entities of a software system”. | Watkins and Neal (1994) |
| “Requirement Traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)”. | Gotel and Finkelstein (1994) |
| “Traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design, and implementation”. | Palmer (1997) |
| “Traceability refers to the ability to determine relationships between different development artifacts in the software development process”. | Tempero et al. (2002) |
| “Traceability is the ability to record dependencies between artifacts and to identify and control changes and assess their impact on the various phases of the development process”. | Pulham and Wills (2008) |

These traceability relationships enable software engineers to: detect whether a system meets all requirements; perform early recognition of those requirements not satisfied by the system; track and

update any artifacts reflecting changes while changes occur; and answer questions about completeness, conflict, coverage, or consistency (Antoniol et al., 2002; Egyed et al., 2007; Gotel and Finkelstein, 1994; Pfleeger & Bohner 1990; Ramesh et al., 1997; Seacord et al., 2003). They also can facilitate the reuse of system components where the components of existing systems are used to implement similar requirements of new systems (Conklin and Begeman, 1988). Furthermore, they can aid both top-down and bottom-up system comprehension i.e. they allow users to map high-level documents, and thus abstract concepts, to low-level artifacts (Oliveto et al., 2007). In addition, they can support system testing and devise complete and comprehensive test cases (Antoniol et al., 2002; Spanoudakis and Zisman, 2005).

Overall, traceability is used to improve the quality of software systems (Spanoudakis and Zisman, 2005), to make the documentation of the systems clear and consistent, and to make the process of maintaining the systems less dependent on individual experts (Lindval and Sandhal, 1996).

2.3 Traceability Link Classifications

In order to help software engineers comprehend traceability, many traceability link classifications have been defined based on different aspects of traceability. Some classifications are based on the artifact-centric aspect. Gotel and Finkelstein (1994) introduced two basic types of traceability: *pre-traceability* and *post-traceability*. *Pre-traceability* is used to capture relations between requirements and the sources that have helped the generation of these requirements, i.e. the views and needs of the stakeholders. *Post-traceability* includes relationships between requirements and artifacts produced during the software development life cycle. These two types of traceability concentrate on relationships among different artifacts of a system. These are what Pfleeger refers to as *horizontal traceability* (Pfleeger, 1998; Pfleeger & Bohner 1990). Pfleeger (1998) also defined *vertical traceability* as relationships among the different elements of an artifact.

A classification focusing on requirements was proposed by Pohl (1996), who categorized eighteen different traceability links into five groups based on the action-centric view: *condition links*,

content links, documents links, evolutionary links, and abstraction links. (1) *Condition links* refer to relations between requirements and restrictions associated with them. (2) *Content links* indicate comparisons, contradictions, and conflicts among requirements. (3) *Documents links* associate software documents to requirements. (4) *Evolutionary links* are replacement relations between requirements, i.e. a requirement A is replaced by a requirement B. (5) *Abstraction links* represent generalization and refinement between requirements.

Ramesh and Jarke (2001) defined four types of traceability links based on a meta-model for requirement traceability: *satisfaction links, dependency links, evolution links, and rationale links.* *Satisfaction links* and *dependency links* form a *product-related* group, which describe properties and relationships of design objects independent of how they were created. (1) *Satisfaction links* refer to relationships between lower-level objects and constraints/goals defined in high-level objects (requirements, standards, policies, complex design objects etc.). (2) *Dependency links* indicate dependency between lower-level objects. *Evolution links* and *rationale links* belong to another group called *process-related*. These can only be captured by looking at the history of actions taken during the process itself and cannot be recovered from the product relationships alone. (3) *Evolution links* identify the origins of design objects by documenting the relationships of actions leading from existing design objects to new or modified design objects. (4) *Rationale links* are captured in design objects based on the history of actions of how design objects are created.

Spanoudakis and Zisman (2005) organized various types of requirement traceability links proposed by previous researchers into eight main groups: *dependency, generalisation/refinement, evolution, satisfaction, overlap, conflicting, rationalisation, and contribution links.* (1) *Dependency links* represent the dependency between elements in a system; an element A depends on an element B, if the existence of A relies on the existence of B, or if changes in B have to be reflected in A. (2) *Generalisation/refinement links* identify how complex elements of a system can be broken down into components, how elements of a system can be combined to form other elements, and how an element can be refined by other elements. (3) *Evolution links* signify the evolution of elements of software artifacts e.g. element A evolves-to element B if A has been replaced by B during the software development life cycle. (4) *Satisfiability links* refer to the

satisfaction between elements in a system e.g. element A satisfies element B, if A meets the constraints, expectation, needs, desires, and goals of B. (5) *Overlap links*: In this type of links, element A overlaps element B, if A and B refer to common features/domain of a system. (6) *Conflict links* indicate conflicts between elements in a system, i.e. when two requirements conflict with each other. (7) *Rationalisation links* are used to represent and maintain the rationale behind the creation and evolution of elements, and decisions about the system at different levels of detail. (8) *Contribution links* represent associations between requirements and stakeholders that have contributed to the generation of the requirements.

Traceability links retrieved by a traceability recovery technique can be classified into four groups based on a recovery method-centric view: *lost links*, *warning links*, *false positive links*, and *normal links* (Lucia et al., 2005). (1) *Lost links* are those that are recovered by the recovery technique but not captured by the user. (2) *Warning links* are relations that are captured by the user but missed by the recovery technique. (3) *False positive links* refer to links that are retrieved by the recovery technique but considered as incorrect by the user. (4) *Normal links* are recovered by the recovery technique and confirmed as correct by the user. For this classification, the verification of the retrieved links is totally dependent on the knowledge of the user to determine whether a retrieved link is correct or not.

In our research, we focus on tracing relationships between source code and documentation that is generated during the software development life cycle and written in natural language, e.g. tutorials, handbooks, developer or user guides, API documentation, architecture documentation, design rationale, requirements, emails, and so on. These relationships belong to the horizontal traceability category. We also adopt a recovery method-centric view to classify traceability links. However, we utilize traceability benchmarks (discussed in Chapter 4) to verify links retrieved by a traceability recovery technique. The main component of a traceability benchmark of a traced system is the oracle traceability link set that consists of all the correct, or true, traceability links for the system. We compare the set of retrieved links with the oracle link set to determine whether a retrieved link is correct or not. Therefore, we define three types of traceability links: *correct/true links*, *incorrect/fault links*, and *missing links*.

- (1) *Correct/true links* are relations that are recovered by the recovery technique and also included in the benchmark. If a retrieved link L is in the benchmark B, then L is a correct/true link.
- (2) *Incorrect/fault links* are those retrieved links that are not in the benchmark. If retrieved link L is not in the benchmark B, then L is an incorrect/fault link.
- (3) *Missing links* are links that are missed by the recovery technique but included in the benchmark. If link L is in the benchmark B but fails to be retrieved by the recovery technique, then L is a missing link.

2.4 Traceability Link Recovery

Many contemporary requirements engineering systems (CORE, 2012; Kaindl, 1992; IBM Rational DOORS, 2012; Pinheiro and Goguen, 1996; RTM, 2012) offer limited support for traceability as they require users to create traceability links manually. For example, creating traceability with IBM Rational DOORS is as simple as drag and drop between items on the screen, i.e. link a requirement to a design item, to a test case or to another requirement with the click of a mouse (Teleologic, 2012). However, effective traceability is rarely established manually because the manual creation of traceability links is difficult, error-prone, time-consuming and complex (Spanoudakis and Zisman, 2005).

To alleviate this problem and improve the accuracy of recovered traceability links among different artifacts in a system, various traceability recovery techniques have been developed. These approaches can be classified into two main groups: semi-automatic recovery and automatic recovery. Table 2.2 compares representative traceability recovery techniques to date in detail: whether they can automatically retrieve traceability links between artifacts in a system; what techniques they utilize to recover links; whether or not they rely on cut points; whether they improve precision or recall; whether they support tracing of systems that are written in different programming languages (e.g. Java, C++, VB); and what artifacts they are targeted on. In the following sections, we discuss these representative traceability recovery techniques in detail.

TABLE 2.2 COMPARISON AMONG TRACEABILITY RECOVERY TECHNIQUES

| Authors | Automatic Link Recovery | Traceability Recovery Technique | Rely on Cut Points | Baseline [Precision, ↓:Decreased] | Precision (↑:Improved ↓:Decreased) | Recall (↑:Improved ↓:Decreased) | Traced System | Target |
|-----------------------------|-------------------------|------------------------------------|--------------------|-----------------------------------|------------------------------------|---------------------------------|---------------|---|
| Jirapanthong&Zisman (2005) | | Rule-based | | | | | Multi-format | Feature-based object-oriented documents |
| Egyed (2003) | | Scenario-driven | | | | | Multi-format | Models, scenarios, Code |
| Egyed et al. (2005) | | Value-based | | | | | Multi-format | Software artifacts, Code |
| Yoshikawa et al. (2009) | | Ontology-based | | [70%, 49%] | ↑ | ↑ | Java-based | Features sentences, Code |
| Grechanik et al. (2007) | | Machine learning | | | | 64% (average) | Java-based | Use case diagrams, Code |
| Cleland-Huang et al. (2007) | √ | Probabilistic Model (PM) | √ | | 20-35% | 90% | Multi-format | Requirements, UML class diagrams |
| Antoniol et al. (2002) | √ | Vector Space Model (VSM) | √ | | 7-20% | 90% | Multi-format | Source code, documentation |
| Marcus&Maletic (2003) | √ | Latent Semantic Indexing (LSI) | √ | | 20-50% | 90% | Multi-format | Source code, documentation |
| Abadi et al. (2008) | √ | Jensen-Shannon (JS) model | √ | | 8-15% | 90% | Multi-format | Source code, documentation |
| Capobianco et al. (2009) | √ | B-spline | √ | | 20-40% | 90% | Java-based | Source code, documentation |
| Gibiec et al. (2010) | √ | PM+Query expansion | √ | [15-17%, 90%] | ↑ | | Multi-format | Requirements, regulations |
| Zou et al. (2006) | √ | PM+Phrasing +Project glossary | √ | [20%, 90%] | ↑ | | Multi-format | Requirements, UML class diagrams |
| Cleland-Huang et al. (2005) | √ | PM+Hierarchy +Artifact clustering+ | √ | [18-38%, 90%] | ↑ | | Multi-format | Requirements, UML class diagrams |

Table 2.2 -- Continued

| Authors | Automatic Link Recovery | Traceability Recovery Technique | Rely on Cut Points | Baseline [Precision, Recall] | Precision (\uparrow :Improved \downarrow :Decreased) | Recall (\uparrow :Improved \downarrow :Decreased) | Traced System | Target |
|---------------------------|-------------------------|--|--------------------|------------------------------|---|--|------------------------|--|
| Settimi et al. (2004) | ✓ | VSM+General thesaurus | ✓ | [11-16%, 90%] | ↓ | ↓ | Java-based | Requirements, UML artifacts, code |
| Hayes et al. (2003) | ✓ | VSM+Context-specific thesaurus | ✓ | [17%, 23%] | ↑(sometimes) | ↑ | Multi-format | High-level and low-level requirements |
| Wang et al. (2009) | ✓ | LSI+Code clustering+Identifier classifying+Thesaurus+s+Hierarchy | ✓ | [21%, 90%] | ↑ | ↓ | Multi-format | Source code, documentation |
| Hayes et al. (2006) | ✓ | LSI+Thesaurus | ✓ | [13%, 70%] | ↑(sometimes) | ↑ | Multi-format | High-level and low-level requirements, design elements |
| Lucia et al. (2011) | ✓ | VSM/LSI+Smoothin g filters | ✓ | [20-35%, 90%] | ↑ | ↑ | Multi-format | Different types of artifacts |
| Witte et al. (2007) | ✓ | Text Mining (TM) | | 90% (average) | 62% (average) | Java-based | Source code, documents | |
| Bacchelli et al. (2010) | ✓ | Regular expressions (REs) | | 15-49% | 38-72% | Multi-format | Source code, emails | |
| Dagenais&Robillard (2012) | ✓ | REs+Partial Program Analysis (PPA)+Filtering heuristics | | 96% (average) | 96% (average) | Java-based | Documentation, API | |

2.4.1 Semi-automatic Traceability Recovery Techniques

Semi-automatic recovery techniques are those that need human intervention during the traceability link extraction process, such as rule-based, scenario-driven, value-based, ontology-based, and machine learning approaches. XTraQue developed by Jirapanthong and Zisman (2005; 2009) utilizes a rule-based approach to define traceability relations between feature-based object-oriented documents (feature model, use cases, class diagram etc.). The rule-based approach uses traceability rules to identify links. Traceability rules are created based on the following aspects: (1) the semantics of the documents being compared, (2) the various types of traceability relations in the product line domain, (3) the grammatical roles of the words in the textual parts of the documents, and (4) synonyms and distance of words being compared in a text (Jirapanthong and Zisman, 2009). The four aspects show that traceability rules are largely dependent on grammatical structures present in the natural language sentences. In order to recover all possible links in different systems, traceability rules have to be expanded to consider all possible grammatical structures. Moreover, building rules is time-consuming.

The scenario-driven technique (Egyed, 2003) combines hypothesized traces and test scenarios to generate traceability relations between models, scenarios and code. The hypothesized traces link models with test scenarios describing test cases or usage scenarios for these models and are executed on a running software system. Traceability relations are then created through analysis of the runtime behavior of these test scenarios and the hypothesized traces. The rationale for trace generation is simple: the code accessed during the testing of a model must implement that model, and multiple model elements sharing some lines of code have a trace dependency among them. STRADA (Egyed et al., 2007) is a tool for scenario-based trace detection between features and code. However, the correctness and completeness of the hypothesized traces can affect the quality of recovered links.

Most current traceability research considers that all artifacts in a system have equal value (Boehm, 2003). However, the value-based approach (Egyed et al., 2005) does not treat every artifact as equally important, so that not all trace relationships between software artifacts and code are

equally important in the context of traceability. The value-based approach produces high quality trace relationships among high-value artifacts on a finer-grained level of detail: high-value artifacts are mapped to the method level. However, the quality of relationships among low-value artifacts is undesirable because they are based on a coarser-grained level of detail: low-value artifacts are mapped to the class level. Although this approach can save cost due to its focus on artifacts with high values, the determination of the value of every artifact is complex and time-consuming.

Yoshikawa et al. (2009) introduced domain ontologies for recovering traceability links between sentences and source code. The sentences are those that describe features of a software product. Domain ontologies can structurally represent knowledge about concepts and their relationships for a specific problem domain of the software. The semantic relationships of the ontologies can detect which methods fulfill features described in sentences. Their evaluation results show that both precision and recall are improved. The limitation is that domain ontologies are manually created.

Machine learning employs learning algorithms to search for similarity between terms in artifacts after being trained (Grechanik et al., 2007). LeanArt (Grechanik et al., 2007) combines program analysis, run-time monitoring, and machine learning to recover links between use-case-diagrams (UCDs) and source code. LeanArt uses the values of program variables, initial traces, and the names of program entities and elements of UCDs to train the learning algorithm to identify entities with similar values and names. The values of program variables are collected when the software is executed and run-time monitoring of program variables is performed. Initial traces are created by programmers to link a small percentage of program entities to elements of UCDs. Then the learning algorithm classifies the rest of program entities by matching them with the names of the elements of UCDs. Experimental results show that LeanArt recovers 87% links in the best case, 64% on average, and 34% in the worst case if 6% initial traces are created. Unfortunately, learning algorithms that can deliver consistently good results for different types of input data have proved difficult to find. Moreover, users are required to manually prepare the initial traces between artifacts. In addition, the run time on a program with over 20,000 lines of code is around thirty minutes.

To sum up, semi-automatic recovery techniques are unable to generate traceability links automatically without human intervention. To successfully implement these techniques, some initial data needs to be manually prepared, such as rules, domain ontology, initial traces etc. They require users to have some knowledge of the traced system. Therefore, it is difficult to adopt these techniques to capture links among artifacts in a system for users who are unfamiliar with the system. Automatic recovery techniques can alleviate this issue. Most of these techniques can be applied to trace links in multi-format software projects.

2.4.2 Automatic Traceability Recovery Techniques

Automatic traceability recovery techniques include lightweight and heavyweight techniques. Lightweight techniques do not require pre-computation of the input and can be directly executed at run-time, such as Regular Expressions (Bacchelli et al., 2009). Heavyweight techniques, in contrast, require pre-processing of their input. These techniques include Information Retrieval (IR) and Text Mining (TM). In the following, we discuss a range of these automatic recovery techniques developed to date in detail.

2.4.2.1 *IR-based traceability recovery techniques*

The most-studied and often-used techniques in automated traceability link recovery to date are Information Retrieval (IR) models. Many traceability recovery techniques (Antoniol et al., 2002; Cleland-Huang et al., 2005; Hayes et al., 2003; Lucia et al., 2007; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009) use a variety of Information Retrieval (IR) approaches to automatically recover traceability links between artifacts in a system.

IR models

Early IR systems were Boolean models which used a complex combination of Boolean ANDs, ORs, and NOTs to specify users' needs (Singhal, 2001). However, Boolean models are not very effective as they do not support ranked retrieval. Therefore, current IR models rank documents by their estimation of the relevance of a document for a query and assign a similarity value to every

document and rank documents by this value (Singhal, 2001). The three most-used IR models to date are Probabilistic Model (PM), Vector Space Model (VSM), and Latent Semantic Indexing (LSI).

The Probabilistic Model (PM) is based on the general principle that documents in a collection are ranked by decreasing probability of their relevance to a query (Singhal, 2001). The probability of a link between a query q and a traceable document d_j is defined as (Cleland-Huang et al., 2005):

$$pr(d_j | q) = \left[\sum_{i=1}^k pr(d_j | t_i) pr(q, t_i) \right] / pr(q).$$

$pr(d_j | t_i)$ and $pr(q, t_i)$ represent the dispersion of the term t_i within the document d_j and query q respectively by computing the normalized frequency of terms. $pr(d_j | t_i)$ is computed by considering the frequency with which t_i occurs in d_j , normalized over the total number of words in d_j . This is

defined as: $pr(d_j | t_i) = \frac{freq(d_j, t_i)}{\sum_k freq(d_j, t_k)}$. Similarly, $pr(q, t_i) = \frac{freq(q, t_i)}{n_i}$ where n_i is the number

of documents containing t_i . $pr(q) = \sum_i pr(q, t_i)$ computes the probability of the query q . Then

the similarity value ($pr(d_j | q)$) between the query q and each document d_j is computed. All documents are ranked based on their similarity values with q and are filtered by a predetermined cut point. This is used to retrieve only documents whose similarity value is higher than this specified cut point (Cleland-Huang et al., 2005; Wang et al., 2009). However, PM cannot deal with synonym and abbreviation problems (Wang et al., 2009).

Vector Space Model (VSM) is a widely used IR approach for constructing vector representations for documents (Marcus and Maletic, 2003). It treats documents and queries as vectors in an N -dimensional space, where N is the number of terms/words in the document collection's vocabulary (Antoniol et al., 2002; Hayes et al., 2003). Each document D_i is represented as a vector $[d_{i,1}, d_{i,2}, \dots, d_{i,N}]$ where the j^{th} element $d_{i,j}$ is a measure of the weight of the j^{th} term of the vocabulary in the document D_i . The vector element $d_{i,j}$ is defined as: $d_{i,j} = tf_{i,j} * \log(idf_j)$ where $tf_{i,j}$ is the term frequency of the j^{th} term in the document D_i , and $\log(idf_j)$ is the weight for the frequency of the j^{th} term in the document collection. idf_j is computed as

$idf_j = \frac{\text{Total number of documents}}{\text{Number of documents containing the } j^{\text{th}} \text{ term}}$. The more frequent the term is in the document collection, the more its presence is important to the document, i.e. the higher the weight (Antoniol et al., 2002; Hayes et al., 2003). Similarly, a user query Q is also converted into a similar vector $[q_1, q_2, \dots, q_N]$. The similarity value between a document D_i and a query Q is computed as the cosine of the angle between the document vector and the query vector in the N -dimensional space (Antoniol et al., 2002; Hayes et al., 2003):

$$\text{Similarity}(D_i, Q) = \cos(D_i, Q) = \frac{\sum_{j=1}^N d_{i,j} q_j}{\sqrt{\sum_{j=1}^N (d_{i,j})^2 * \sum_{j=1}^N (q_j)^2}}$$

All documents are ranked based on their similarity values and filtered by a predetermined cut point. However, it is difficult for VSM to cope with abbreviations, synonyms, and polysemy problems as it does not take into account relations between terms or words, i.e. all terms and words are independent (Marcus and Maletic, 2003; Wang et al., 2009).

Latent Semantic Indexing (LSI) is a VSM based method for inducing and representing aspects of the meaning of words reflective in their usage (Marcus and Maletic, 2003). The basic concept of LSI is that the information about word contexts, in which a particular word appears or not, provides a set of mutual constraints that decides the similarity of meaning of sets of words to each other (Marcus and Maletic, 2003). Therefore, LSI improves the synonym issue by taking the relations among terms into consideration. LSI firstly represents documents and queries in a VSM N -dimensional space. Then the VSM N -dimensional space is truncated and transformed to LSI subspace by applying Singular Value Decomposition (SVD) (Salton and McGill, 1983). The document and query vectors in the VSM N -dimensional space are orthogonally projected onto new corresponding vectors in the LSI subspace. Terms that occur less frequently in the document collection tend to be precluded from the LSI subspace (Marcus and Maletic, 2003). Finally, LSI uses the VSM similarity formula to compute similarity values between documents and queries but based on the LSI subspace. All similarity values are sorted decreasingly and filtered by a predetermined cut point.

Cut points

The accuracy rate of link recovery by using IR models heavily relies on a cut point; only links that have a similarity value greater than or equal to the cut point are shown to users (Antoniol et al., 2002; Cleland-Huang et al., 2005; Lucia et al., 2007; Marcus and Maletic, 2003; Wang et al., 2009). There are two ways to determine the cut point (Lucia et al., 2007; Lucia et al., 2006; Marcus and Maletic, 2003).

- One way is similarity-based, which uses a cut point on the similarity value that identifies which documents are linked. Only links that have a similarity value greater than the cut point will be captured.
- The second way is cut-point-based, which cuts the ranked links by imposing a cut point on the number of retrieved links, regardless of the actual similarity value. This means that the user can decide to retrieve the top ranked links among those that have a similarity value greater than the cut point.

However, both approaches lead to scenarios where: (i) the lower the cut point the much greater the number of incorrect links that are retrieved, and (ii) conversely, the higher the cut point the smaller the number of correct links retrieved. In other words, using a low cut point retrieves a larger number of accurate links than using a high cut point, but more incorrect links are also captured at the same time. This means that many potentially useful and important links are missed at high cut points. Similarly, many incorrect or unuseful links are extracted at low cut points and may confuse developers. Furthermore, the same cut point may or may not be best suited for different systems (Marcus and Maletic, 2003).

IR-based traceability recovery techniques

Cleland-Huang et al. (2007) applied PM to retrieve links between requirements and UML classes in the Poirot tool. Their experimental results suggested that 20%-35% precision should be achievable at a recall level of 90%.

Antoniol et al. (2002) applied two different IR models, PM and VSM, to extract links between

code and documentation. Their experimental results showed that IR provides a practical solution for automated traceability recovery. PM achieves 90% of recall with around 20% of precision. VSM has 7-20% of precision when reaching 90% of recall. The two IR models have similar performance when terms in artifacts perform a preliminary morphological stemming.

Marcus and Maletic (2003) introduced LSI, an extension of the VSM, to recover links between documentation and source code. Their experimental results showed that LSI achieves very good performance without the need for stemming, as required for PM and VSM. It reaches 20-50% of precision while recall is 90%. LSI needs less computation as it requires less processing of code and documentation.

Abadi et al. (2008) compared several IR models for retrieving links between source code and documentation. These IR models included VSM, LSI, Probabilistic Latent Semantic Indexing (PLSI), and Jensen-Shannon (JS) similarity model. PLSI uses the PM for document indexing (Abadi et al., 2008; Hofmann, 1999). The JS model is a new IR model and is driven by the PM and hypothesis testing techniques. The JS model represents each artifact through a probability distribution. In other words, an artifact is represented by a random variable where the probability of its states is given by the empirical distribution of terms occurring in the artifact. The empirical distribution of a term is based on the weight assigned to the term for the specific artifact. The similarity between two artifacts is computed as a distance of their probability distributions measured using the Jensen-Shannon Divergence (Cover and Thomas, 1991). As the JS model does not take into account relations between terms, it suffers from synonymy and polysemy problems. Their experimental comparison results show that VSM and JS have almost equal performance, LSI and PLSI have the worst performance, and PLSI is significantly worse than the others.

Capobianco et al. (2009) proposed a novel IR technique, called B-spline, based on numerical analysis for recovering links between code and documentation. This approach models the information contained in an artifact by particular interpolation curves of plots mapping terms and their frequency on the artifact. This mitigates synonymy and polysemy problems because the interpolation curves for artifacts can exploit information about co-occurrences of terms. The

similarity between artifacts is then computed by calculating the distance of the corresponding interpolation curves. Their experimental results demonstrate that this approach significantly outperforms both VSM and LSI, and is comparable and sometimes better than the JS model. It achieves 20-45% of precision while reaching 90% of recall. Their experimental results also revealed that both the artifact types and the language used to write artifacts significantly influence the retrieval accuracy of the four IR models.

Enhancement strategies

In order to improve the performance of IR-based traceability recovery techniques, various enhancement strategies have been developed. Gibiec et al. (2010) developed a web-mining approach to discover a new set of query terms that can be used to replace or augment the original PM query to mitigate the problem of stubborn traces. These new query terms are learned through seeding a web-based search with the original query and then processing the results to identify a set of domain-specific terms. Their experimental results showed that the stubborn trace queries are significantly improved.

Zou et al. (2006, 2008) incorporated the use of phrases detected and constructed from requirements using a part-of-speech tagger with PM to improve the precision of traces between requirements and UML classes. A project glossary is also used to find additional phrases and weight the contributions of all phrases. Phrases that are mentioned in the project glossary are weighted more heavily than those that are not in the project glossary. Their experimental results showed that combining phrasing and project glossary with PM has a significant improvement in precision especially within the top ranked links.

Cleland-Huang et al. (2005) proposed an approach to improve the performance of dynamic requirements traceability by incorporating three different strategies into PM, namely hierarchical modeling, logical clustering of artifacts, and semi-automated pruning of the probabilistic network. (1) Hierarchical enhancement is used to adopt the artifact hierarchical format in which the words used to name and describe the higher-level artifacts capture the general meaning of their

lower-level components. The ancestral information in artifacts strengthens the probability of true links according to their ancestral links. (2) Clustering enhancement is based on sibling artifacts and the premise that links tend to occur in clusters. If a link exists between artifact A and artifact B that is part of a cluster of artifacts, then there would be a higher probability that additional links should exist between the artifact A and other artifacts in the cluster. (3) Pruning enhancement is not intended to improve precision across all artifacts but focuses on particular areas in which the precision is particularly poor. It creates constraints between groups of artifacts to improve the precision of problematic area. Their experimental results indicated that the three strategies effectively improve the precision of retrieval traces.

Settimi et al. (2004) investigated the effectiveness of VSM and VSM with a general thesaurus for generating links between requirements, code, and UML design models. The comparison results showed that precision and recall are not improved by the use of a general thesaurus. Hayes et al. (2003) used VSM but with a context-specific thesaurus that is established based on technical terms in requirement documents to recover links between requirements. Their experimental results showed that improvements in recall and sometimes in precision are achieved.

Wang et al. (2009) presented four enhanced strategies to improve LSI, namely: source code clustering, identifier classifying, similarity thesaurus, and hierarchical structure enhancement. (1) Source code clustering is implemented based on the inheritance relationships among classes; if one class is the ancestor of another class then they are assigned to the same cluster. If there exists a link between document d and a class in cluster C , then the similarity between d and all classes in C is set to a higher value. (2) Identifier classifying is to classify identifiers in source code into class names, various comments, and general identifiers (all identifiers except of class names and comments), and to impose different weights on different identifiers. For example, if there exists a link between a document and a class such that the document contains the name of the class, then the similarity of the link is increased by 20%. (3) A similarity thesaurus is employed to cope with synonym and abbreviation problems. This thesaurus is constructed through a data dictionary that might exist in documentation and abbreviations in source code. (4) The hierarchical structure of documentation is used to divide all documents in a system into high-level conceptual documents

(e.g. requirements, summary designs, and manuals) and low-level implementation documents (e.g. detailed designs, API, data format specifications). Low-level documents are regarded as a bridge between source code and high-level documents. Links between low-level documents and source code are used as feedback to retrieve links from high-level documents. Their experimental comparison with PM and LSI indicated that this approach has higher precision than LSI and PM, but has lower recall.

Hayes et al. (2006) investigated the performance of LSI with the context-specific thesaurus from Hayes et al. (2003). Their experimental results showed that this approach improves recall and sometimes precision. Lucia et al. (2011) adopted a smoothing filter to improve the performance of VSM and LSI. The smoothing filter aims at removing the common information among artifacts of the same type (e.g. between use cases, or between source code) that does not help to characterize the artifact semantics. Use of the smoothing filter was experimented on different types of artifacts, use cases, requirements, UML diagrams, code, and test cases. Their experimental results indicated that the usage of the smoothing filter significantly improves the precisions of both VSM and LSI.

To sum up, IR-based traceability recovery techniques rely on a predetermined cut point to filter the retrieved links. Using different cut points generates different values of precision and recall. IR-based recovery techniques generally achieve low precision but high recall at low cut points and generally get high precision with low recall at high cut points (Abadi et al., 2008; Antoniol et al., 2002; Capobianco et al., 2009; Cleland-Huang et al., 2007; Marcus and Maletic, 2003). In other words, a larger number of incorrect links are captured at lower cut points and fewer correct links are returned at higher cut points. Although various strategies have been applied to enhance the performance of IR-based recovery techniques, no approaches to date can substantially decrease incorrect links at low cut points and significantly increase correct links at high cut points (Antoniol et al., 2002; Cleland-Huang et al., 2005; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009). Most of these techniques can retrieve links in multi-format systems.

2.4.2.2 Other traceability recovery techniques

In addition to IR-based recovery techniques, some other techniques have been invented to capture links between artifacts. The Text Mining (TM) technique organizes related texts in documents to extract domain-specific information from texts (Stranieri and Zeleznikow, 2005; Weiss et al., 2005). Witte et al. (2008) employed Information Extraction, a subfield of TM, to capture traceability links through extracting entities (e.g. methods, classes, packages, etc.) from software documents. It achieved a very high precision (90%) for the entity extraction with a recall of 62% on average. Its limitation is that it can only extract from documents salient facts about pre-specified types of events, entities, or relationships, although it generates these relationships with high accuracy (GATE Information Extraction, 2010; Weiss et al., 2005). Types of entities have to be pre-defined, and grammar rules have to be built for detecting complex named entities.

Bacchelli et al. (2010) built regular expressions to match class names to words in development email archives. The first regular expression is to simply use case sensitive matches between class names and words in emails. It achieves low precision (1-40%) but high recall (40-75%). The low precision is caused by the high number of class names that belong to common words as it is hard to tell whether or not the matched words in emails indicate class names. The second regular expression is to solve the issue of class names that are dictionary words. It requires the presence of the last part of the package of a class before the class name itself, which must be then followed by a source code extension or any kind of separator. This regular expression is defined as:

```
(.*) (\s|\.\|\|/) <packageTail> (\.\|\|/) <EntityName> ((\.(java|class|as|php|h|c)) | (\s)) (*)
```

When used alone, this regular expression achieves good precision (59-64%) and recall (59-65%) for Java systems. The third regular expression works for non-Java systems:

```
(.*) (:punct:\s) + <EntityName> (:punct:\s) + (*)
```

It requires that a class name is separated from other words by empty space or connected to it through source code tokens (i.e. punctuation). This expression gives high outcomes for non-Java systems; around 50% of precision and 72% of recall. These results show that using regular expressions achieves good accuracy. However, this approach fails to retrieve links between classes

and emails where class names do not explicitly appear but are mentioned implicitly, such as an email that describes tasks that a class should fulfill but does not directly mention its name. Moreover, as this approach performs strict matching between class entities and words in documents, it is unusable on cases that class entities are rarely mentioned in the documents. In addition, this approach returns documents without any ranking. A document either matches or does not match the regular expressions.

Dagenais and Robillard (2012) designed a technique that automatically analyzes the documentation (e.g. tutorials, reference manuals, mailing lists, or forums) of a system and that precisely links code-like terms (e.g. `year()`) in documents to specific code elements (e.g. `DateTime.year()`) in the API of the documented framework or library. A code-like term is a series of characters that matches a pattern associated with a code element kind (e.g., parentheses for functions, camel cases for types, anchors for XML elements). This technique first adopts regular expressions, as used in Bacchelli et al. (2010), and Partial Program Analysis (PPA), as used in Dagenais and Hendren (2008), to identify the code-like terms, the code snippets, and their probable kind (e.g. class, method, XML element, Java code snippet, XML code snippet) in the documentation of the system. PPA builds a typed intermediate representation (e.g. an abstract syntax tree) of incomplete Java program source code, recovers the declared types by performing partial type inference, and resolves syntactic ambiguities inherent to incomplete programs using heuristics (Dagenais and Hendren, 2008). Next, this technique considers the context in which a term is mentioned and applies a set of filtering heuristics to resolve four sources of ambiguity (declaration, overload, external reference, and language ambiguities) inherent to linking code-like terms in unstructured natural language documents, to ensure that terms referring to external code elements are not spuriously linked. Finally, links between code-like terms and code elements are created, based on the assumption that code elements mentioned in close vicinity are more likely to be related than code elements mentioned further apart. In other words, a code-like term A is closer to term B than to term C if B is in a more specific context than C. This technique is implemented in a tool called RecoDoc. The experimental results showed that this technique has an average precision and recall of 96%. However, this approach suffers from the same drawbacks as the approach using regular expressions in Bacchelli et al. (2009; 2010).

To sum up, these non-IR-based traceability recovery techniques are able to achieve good precision and recall. However, these techniques cannot rank retrieved links to allow users to select the best set of links. Moreover, as they all perform strict matching, they are unusable where documents use different words to describe the matching entities, or where documents rarely mention these entities. Only the approach designed by Bacchelli et al. (2009; 2010) accepts multi-format software projects.

2.4.3 Enhancing Discovery Techniques

Some techniques exist that may enhance the performance of IR-based traceability recovery techniques: keyphrase extraction techniques, that can automatically extract key phrases from the text; and clustering algorithms, that can group together similar items in a data set.

Key phrases, which can be single words or phrases of two or more words, represent the key ideas of the document (Turney, 1999; Witten et al., 1999). Key phrases can serve as a representative summary of the document, high quality index terms, and suggestions for refining search queries (Kim and Kan, 2009; Turney, 1999). In IR-based traceability recovery techniques, key phrases extracted from text can augment the original IR-queries or establish a thesaurus to improve their performance. Keyphrase extraction techniques employ lexical and IR techniques to extract phrases from the document text that are likely to characterize it (Turney, 1999; Witten et al., 1999). The simplest system is KEA (Frank et al., 1999; Witten et al., 1999) that uses the Naïve Bayes machine learning algorithm (Domingos and Pazzani, 1997) for training and keyphrase extraction. KEA first splits the input text into tokens and removes apostrophes and non-token characters. It then determines candidate phrases based on three rules: the length of each candidate phrase is limited to three words, candidate phrases cannot be proper names, and they cannot begin or end with a stop-word. Next, case-folding and stemming these determined candidate phrases discard any suffix and treat different variations on a phrase as the same thing. Two features are calculated for each candidate phrase: TF*IDF (i.e. term frequency * inverse document frequency) and first occurrence in the document. TF*IDF measures the frequency of a phrase in a document compared to its rarity in general use. The first occurrence is the distance into the document of the first appearance of the

phrase. Finally, the overall probability of each candidate phrase is calculated based on the two feature values. Candidate phrases are ranked according to their probabilities. Their experimental results show that KEA can on average match between one and two of the five keyphrases chosen by the author of the experimented documents.

Conceptually meaningful groups of artifacts or elements in an artifact that share common characteristics (e.g. inheritance relationships among classes in source code, hierarchical relationships among sections in documents) play an important role in analyzing and describing a system (Tan et al., 2005). In IR-based traceability recovery techniques, clustering can organize similar artifacts in a system into groups, and then re-weight the similarity of links that belong to the same group, such as the logical clustering of artifact strategy in Cleland-Huang et al. (2005) and the code clustering enhancement in Wang et al. (2009). The most common approach to clustering is to minimize the sums of the squares of the distances between points in a given data set and the centers, or means, of clusters (Gupta and Grossman, 2004). The K-means algorithm is a simple and popular heuristic solution to the clustering problem (MacQueen, 1967; Tan et al., 2005). This algorithm is composed of four steps: initialization, assignment, update, and re-computation (MacQueen, 1967; Tan et al., 2005). (1) Initialization is to select K initial centroids, one for each cluster. These centroids should be placed in a cunning way because different locations cause different results; place them as far away as possible from each other. (2) The next step, assignment, is to assign each point belonging to a given data set to the closest centroid. Each collection of points assigned to a centroid is a cluster. (3) The update step is then implemented. The centroid of each cluster is updated based on the points assigned to the cluster, as a result of which K new centroids are calculated. (4) The final step, re-computation, repeats the assignment and update steps until the K centroids remain the same. After K new centroids are re-calculated, a new binding has to be done between points in the same data set and the nearest new centroid. A loop of steps 2 and 3 has been generated. The loop terminates when there are no more changes made to the K centroids.

2.4.4 Requirements for Successful Traceability Recovery Techniques

From the above detailed review of traceability recovery techniques to date, we have identified six key challenges in developing a successful traceability recovery technique:

- 1) **Aim for target end users who want to get to know the target system.** Do not impose requirements that users must have some understanding of the target system.

Semi-automatic traceability recovery techniques (discussed above) require users to have some knowledge of the traced system. It is difficult for users who are unfamiliar with the traced system to use semi-automatic recovery techniques to retrieve links in the traced system. The Grand Challenges in Traceability (Cleland-Huang et al., 2006) defines that a successful traceability recovery technique needs near zero user effort when creating links.

- 2) **Support the ability to automatically capture traceability links between artifacts.** Do not require human intervention during the traceability link recovery process.

Manual traceability link establishment puts a huge burden on users. Traceability recovery techniques that need human intervention require users to have some understandings of the traced system in order to implement them successfully, such as semi-automatic recovery techniques. This can cause difficulties for users who are new to the traced system to retrieve links using such techniques. The Grand Challenges in Traceability (Cleland-Huang et al., 2006) specifies the dream process for traceability creation that is completely automated.

- 3) **Support the ability to rank the retrieved links to allow end users to select the best set of links.**

Retrieved links that are ranked in descending order based on their similarity scores can assist users in effectively and efficiently finding the most relevant links (Singhal, 2001).

- 4) **Retrieve as many correct links as possible and as few incorrect links as possible.** Aim to support both high precision and high recall at all cut points.

The dream process defined in the Grand Challenges in Traceability (Cleland-Huang et al., 2006) automatically retrieves links with 100% precision and recall. However, it is impossible for automatic traceability recovery techniques (that have evolved to date) to recover all

correct links without capturing incorrect links in practice. Moreover, the quality of links retrieved by IR-based recovery techniques depends heavily on a cut point. In general, the lower the cut point, the lower precision and the higher recall. Therefore, it is still a major challenge for researchers in the software traceability community to develop a traceability recovery technique that can achieve reasonably high precision and recall (Oliveto et al., 2007).

- 5) **Minimize or mitigate the issue of dependency on cut points.** This narrows the gap among the precision and recall results generated at different cut points.

Most automated recovery techniques to date have employed IR models to create links. The main limitation for IR models is the reliance on a cut point (Antoniol et al., 2002; Cleland-Huang et al., 2005; Lucia et al., 2007; Marcus and Maletic, 2003; Wang et al., 2009). At low cut points, low precision and high recall are obtained. At high cut points, high precision but low recall are achieved. Many supporting strategies (discussed above) have been developed to ameliorate the limitation of IR models. However, no approaches to date can largely increase precision at low cut points and significantly increase recall at high cut points to reduce the gap among them at different cut points (Antoniol et al., 2002; Cleland-Huang et al., 2005; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009).

- 6) **Provide support for multi-format systems and any kinds of artifacts produced during the software development life cycle.**

Most recovery techniques to date have supported retrieval of links in multi-format systems. However, no recovery approaches have yet been able to create links between any two artifacts produced during the software development life cycle. The Grand Challenges in Traceability (Cleland-Huang et al., 2006) specifies the dream process for traceability links anything within the traced system without a performance hit.

2.5 Traceability Link Visualization

There are no standard approaches to store or represent traceability links in a system. Software engineers traditionally represent traceability links in tabular formats using a spreadsheet, matrix, cross-references, or database. Although these traditional approaches are simple and easy to understand, they are unsuitable for the representation of links in big systems because they become unreadable when the number of artifacts in a system becomes large (Voytek and Nunez, 2011). More recently, research has focused on displaying links in a graph or tree due to the convenience and the ease of browsing and maintaining links. These methods are discussed in the following three sections.

Table 2.3 compares in detail some representative traceability visualization systems. They are compared in terms of eleven aspects: IDE support, semi-automatic or automatic link recovery, visualization technique, hierarchical structure of the system, overall overview of traces, details-on-demand, link edition, navigation, search, filter, and other support functions.

1. **IDE support:** Has the traceability visualization system been integrated within an Integrated Development Environment (IDE) (e.g. Eclipse or Visual Studio) or other software management tools? For example, if a traceability visualization system is embedded within Eclipse, then users can use the functionality provided not only within Eclipse but also the visualization system as a stand-alone tool.
2. **Semi-/automatic link recovery:** Can links between artifacts be generated by the traceability visualization system itself i.e. whether the visualization system is integrated with a traceability recovery technique?
3. **Visualization technique:** the technique used to display the links.
4. **Hierarchical structure of the system:** Can the visualization technique illustrate the hierarchical structure of the traced system?
5. **Overall overview of traces:** Can the visualization technique provide a global overview of links in the traced system? The overview of traces provides users with information about

the distribution of traces in the system and whether or not an artifact has links. Taking traceability links between classes and requirements as our example, the overview of traces provides information about how many classes in a package have related requirements, how many classes in the package have no links, which class in the package has no links, and which requirements have not been implemented (i.e. have no related classes) etc. Users don't need to check one by one to see which artifacts have no links.

6. **Details-on-demand:** Can detailed information be shown to users when needed? When an artifact is selected, its detailed link information is displayed, or its content is opened.
7. **Link edition:** Does the tool allow users to create new links and modify or delete existing links, or to allow users to modify existing links' similarity values?
8. **Navigation:** Does the tool allow users to navigate among predefined system artifacts to locate a specific item in the visualization view?
9. **Search:** Does the tool allow users to search a specific item by using key words or other searching methods?
10. **Filter:** Does the tool allow users to filter out uninteresting items, prune traces, or display links based on specific categories?
11. **Other support functions:** other functions the visualization system provides to help users to browse and maintain traceability links.

TABLE 2.3 COMPARISON AMONG TRACEABILITY VISUALIZATION SYSTEMS TO DATE

| Systems/Authors | IDE support | Semi-/automatic link recovery | Visualization technique | Hierarchical structure of the system | Overall structure of overview of traces | Details-on-demand | Link edition | Navigation | Search | Filter | Other support functions |
|-------------------------------|-------------|-------------------------------|---|--------------------------------------|---|-------------------|--------------|------------|--------|--------|-------------------------|
| Trace/Analyzer (Egyed, 2006) | | ✓ | Traceability matrix | | ✓ | ✓ | ✓ | | | | ✓ |
| ADAMS (Lucia et al., 2004) | Eclipse | ✓ | A graph: nodes as artifacts, edges as links | | | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Cleland-Huang & Habrat (2007) | | ✓ | Hierarchical graphical representation | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| ENVISION (Zhou et al., 2008) | Eclipse | ✓ | Hyperbolic tree view with focus+context | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | History view |
| TBreq (TBreq, 2012) | LDRA | ✓ | Artifacts listed horizontally, edges drawn between elements | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Merten et al. (2011) | Redmine | | Sunburst + Netmap techniques | ✓ | ✓ | ✓ | ✓ | | | | ✓ |

Table 2.3 -- Continued

| Systems | IDE support | Semi-/automatic link recovery | Visualization technique | Hierarchical structure of the system | Overall overview of traces | Details-on-demand | Link edition | Navigation | Search | Filter | Other support functions |
|--|-------------|-------------------------------|--|--------------------------------------|----------------------------|-------------------|--------------|------------|--------|--------|-------------------------|
| EXTRAVIS (Cornelissen et al., 2007) | | | Circular bundle view + Massive sequence view | ✓ | ✓ | ✓ | | | | | |
| TraceVis (van Ravenstein, 2011) | | | Icicle plots + Hierarchical edge bundling | ✓ | ✓ | ✓ | | | | | ✓ |
| UNITI (Pilgrim et al., 2008) | Eclipse | ✓ | A 3D approach: artifacts on a plane, links as edges between planes | | ✓ | | | | | | ✓ |
| Poirot (Cleland-Huang & Habrat, 2007) | | ✓ | Confidence levels + checkboxes + tabs | | | ✓ | | | | | |
| TraceViz (Marcus et al., 2005) | Eclipse | ✓ | A map with colored and labeled squares | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | History |
| LeanArt (Grechanik et al., 2007) | Eclipse | ✓ | An point-and-click graphical interface | | ✓ | | | | | | |

2.5.1 Traditional Traceability Visualization Techniques

Matrix and cross-reference techniques are very common, traditional methods of representing traceability links. A traceability matrix (see Figure 2.1a) is a two-dimensional grid that displays artifacts in rows and columns and represents traceability links as marks between row artifacts and column artifacts. Such a visualization is easy to understand and provides a quick overview of relations between two artifacts if the set of artifacts is small (van Ravenstein, 2011). However, such a matrix misses the inherent hierarchical structure in artifacts and becomes unreadable when the set of artifacts becomes large (Voytek and Nunez, 2011). The cross-reference pattern (see Figure 2.1b) lists each artifact using natural language and gives a list of related links for each artifact (van Ravenstein, 2011). It is easy to understand but cannot provide the overall structure of traces. It is difficult to identify individual traceability links as they are lost in this table structure. The approach, therefore, does not scale to large numbers of classes and documents.

| | Req 1 | Req 2 | Req 3 | Class 1 |
|---------|-------|-------|-------|---------|
| Req 1 | | ■ | ■ | ■ |
| Req 2 | | | ■ | |
| Req 3 | | ■ | | |
| Class 1 | | | | |

(a) Traceability matrix

| | | |
|-------|---------------------|---------------------------------|
| Req 1 | The system shall... | ► Req 2 ► Req 3 ► Class 1 |
| Req 2 | The system shall... | ► Req 3 ◀ Req 1 ◀ Req 3 |
| Req 3 | The system shall... | ► Req 2 ◀ Req 1 ◀ Req 2 |

(b) Cross-reference

FIGURE 2.1 TRADITIONAL TRACEABILITY VISUALIZATION TECHNIQUES (WINKLER AND PILGRIM, 2010)

The Trace/Analyzer tool developed by Egyed (2006) uses a traceability matrix to visualize the trace links among models, code, and test scenarios. This tool reasons about the model elements' ownership of the source code and infers trace dependencies among the model elements based on the ownership information (Egyed, 2006). The matrix depicts artifacts in rows and columns and uses colours or symbols to indicate whether two artifacts are related or not. Links between artifacts are generated based on the models' ownership of source code, which is refined from the hypotheses on how the various models map to source code that are provided by users.

Figure 2.2a shows a matrix between models and code. It shows the models in rows and the code elements in columns. It uses “I” to represent that a model owns a code element, “X” to represent that a model does not own a code element, or “.” to represent a unique code and “?” for shared code/utility code if a model is not known to own anything but might potentially own a code element as unique code or a code element as shared code/utility code (Egyed, 2006). Within the matrix, users can select any cell by clicking on it, and details of the corresponding model and code element are then displayed underneath. Users also can use a popup menu to add or remove the selected model to or from the excluded list of the selected code element.

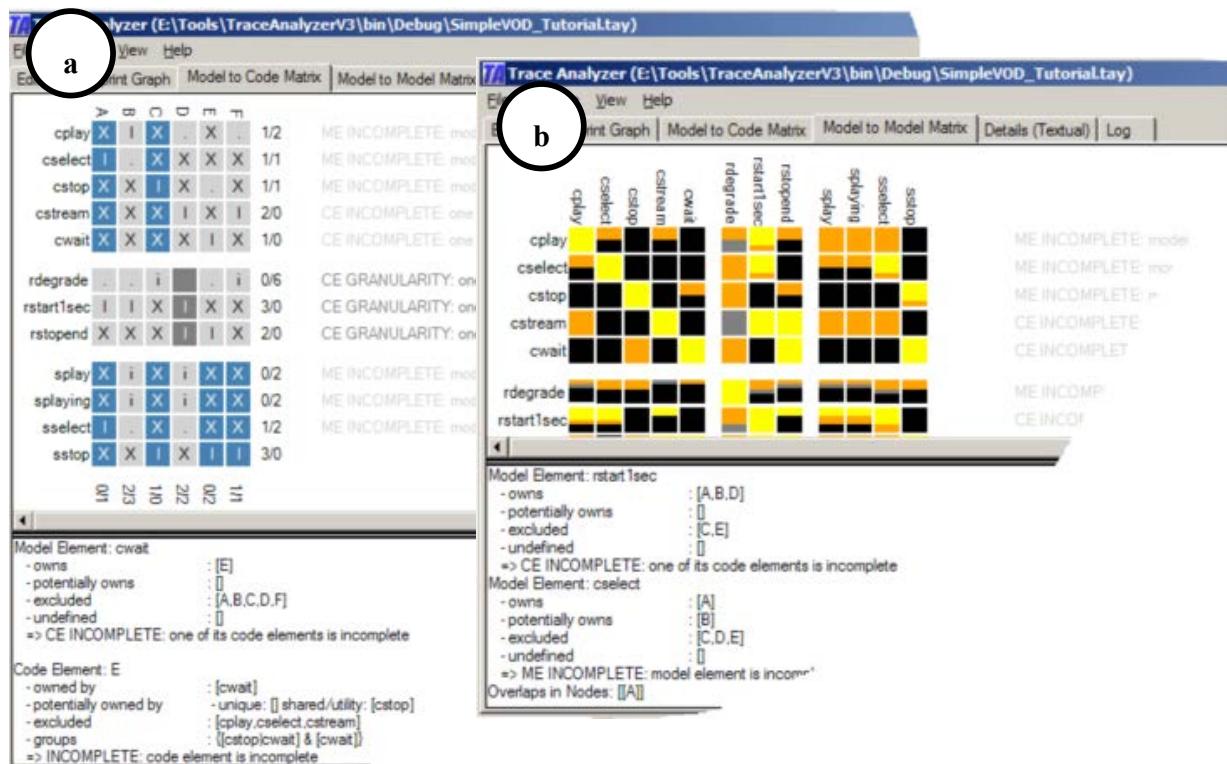


FIGURE 2.2 TRACEABILITY MATRIX IN TRACE/ANALYZER (EGYED, 2006)

Figure 2.2b shows a model-to-model matrix. This matrix depicts the same artifacts in rows and columns and uses colours to indicate whether two artifacts have a trace link (yellow), do not have a trace link (black), potentially have a trace link (orange), or are unknown to have a trace link (gray) (Egyed, 2006). Users can select any cell to show details about the selected model elements and to change the degree of trustworthiness. Users also can choose only to visualize overlaps that are known to exist (Display Trace Yes/No), to display potential overlaps (Display Trace

Yes/Maybe/No), or to display undefined overlaps (Display Trace Yes/Maybe/Undefined/No) (Egyed, 2006). Although the matrix is an effective and efficient way to represent links, interpreting links is difficult and time-consuming. Moreover, it is not suitable for visualizing links in a large set of artifacts.

To sum up, traditional traceability visualization techniques are simple and easy to understand. However, they do not scale to a large set of artifacts in a big system. They cannot provide an overview of the inherent hierarchical structure in the traced system. Although they can provide an overview of trace links, the set of artifacts must be small. Furthermore, they cannot support users to maintain or interpret links easily and conveniently. The actual task of creating, maintaining, and utilizing a trace matrix is arduous and error-prone, and many organizations fail to implement consistent and adequate traceability processes (Cleland-Huang and Habrat, 2007).

2.5.2 Graph-based Traceability Visualization Techniques

Graph-based visualization techniques represent artifacts as nodes and traceability links between artifacts as edges to form a graph or tree. Graph-based visualizations can show the overall overview of relationships between artifacts and can be used to easily browse links.

Tree visualizations

ADAMS (Lucia et al., 2004; Lucia et al., 2010) is an Eclipse plug-in and integrates the ADAMS Re-Trace recovery tool (Lucia et al., 2008) to retrieve links between artifacts using the LSI IR model. It supports specifying links between pairs of artifacts. Users first select the source and the target artifacts among pre-defined project artifacts (see Figure 2.3). Filters on the artifact type and/or on the name of the artifact can be specified to filter out uninteresting links. Then a list of candidate links calculating all the combinations of source and target artifacts is shown. Thus, users can select the links to trace specifying for each link and identify the dependence type of the link. Traceability links are organized in a traceability graph where nodes are represented by the artifacts and edges are the links. After users select a source artifact from the artifact list and define the dependence types they want to visualise in the graph, the traceability graph is built starting from the source artifact and finding all the dependencies of a specific type that involve the source

artifact either as source or target artifact (see Figure 2.4).

| | SOURCE | TARGET | | | | | | | | | | | | | | | | | | | |
|--|---|---|--|-------------------------------------|--|--|----|-----------------|----|-----------------|------------|-----|-----------------------|-----|-----------------------|-------------------------------------|-----|-----------------------|-----|-----------------------|--------------------------|
| Artefact Types | Statechart Diagram System Decomposition Test Case Test Execution Document Test Incident Test Plan UC Diagram UI Navigational Path UI Screen Mockup Use Case | Statechart Diagram System Decomposition Test Case Test Execution Document Test Incident Test Plan UC Diagram UI Navigational Path UI Screen Mockup Use Case | | | | | | | | | | | | | | | | | | | |
| Artefact Filter | Contains <input type="text"/> <input type="button" value="Next >"/> | Contains <input type="text"/> <input type="button" value="Next >"/> | | | | | | | | | | | | | | | | | | | |
| <table border="1"> <thead> <tr> <th colspan="4">Select links to trace (2 links proposed)</th> </tr> <tr> <th>ID</th> <th>Source Artefact</th> <th>ID</th> <th>Target Artefact</th> <th>Trace link</th> </tr> </thead> <tbody> <tr> <td>548</td> <td>Use Case 1 (Use Case)</td> <td>549</td> <td>Use Case 2 (Use Case)</td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td>549</td> <td>Use Case 2 (Use Case)</td> <td>548</td> <td>Use Case 1 (Use Case)</td> <td><input type="checkbox"/></td> </tr> </tbody> </table> <input type="button" value="Select All"/> <input type="button" value="Deselect All"/> <input type="button" value="Invert Selection"/> <input type="button" value="Insert Links"/> | | | Select links to trace (2 links proposed) | | | | ID | Source Artefact | ID | Target Artefact | Trace link | 548 | Use Case 1 (Use Case) | 549 | Use Case 2 (Use Case) | <input checked="" type="checkbox"/> | 549 | Use Case 2 (Use Case) | 548 | Use Case 1 (Use Case) | <input type="checkbox"/> |
| Select links to trace (2 links proposed) | | | | | | | | | | | | | | | | | | | | | |
| ID | Source Artefact | ID | Target Artefact | Trace link | | | | | | | | | | | | | | | | | |
| 548 | Use Case 1 (Use Case) | 549 | Use Case 2 (Use Case) | <input checked="" type="checkbox"/> | | | | | | | | | | | | | | | | | |
| 549 | Use Case 2 (Use Case) | 548 | Use Case 1 (Use Case) | <input type="checkbox"/> | | | | | | | | | | | | | | | | | |

FIGURE 2.3 TRACEABILITY LINK DEFINITION FORM IN ADAMS (ADAMS 2009)

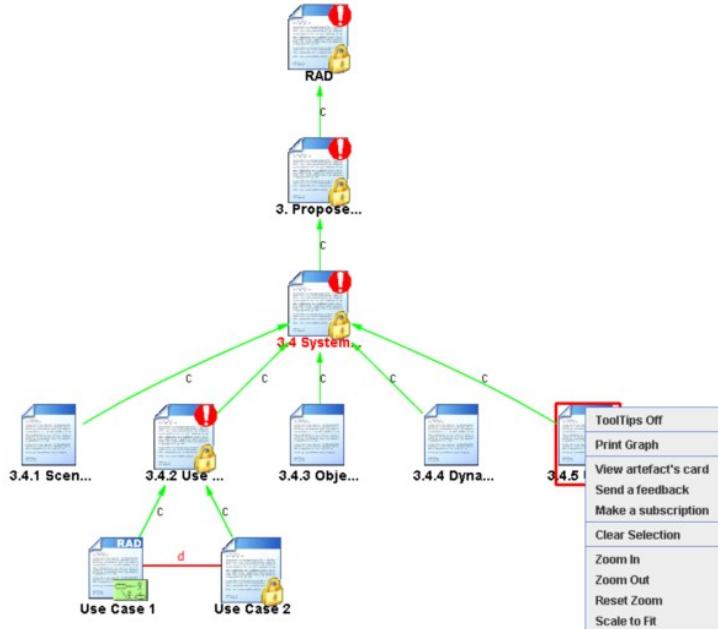


FIGURE 2.4 TRACEABILITY GRAPH VISUALIZATION IN ADAMS (ADAMS 2009)

Within the graph, users can identify traceability paths, i.e. sets of artefacts connected by traceability links. A path indicates the dependency of artifacts in the path; an artifact along a traceability path could be affected by each artifact appearing in the part of the path preceding it as well as it could affect each artifact appearing in the part of the path following it (Lucia et al., 2010).

In order to provide enhanced visual information about the artifacts, tool tips are used to show immediate artifact information while the user positions the mouse over a node. Moreover, the artifact used as query has been shown in red to point out its position within the traceability graph. Each different type of link for compositions and dependencies has been also associated with a specific notation. An exclamation mark and/or a lock mark an artifact node if this artifact is out of synchronization and/or its content is locked. In addition, the contextual menu shown on a right click of the mouse over an artifact of the graph has been customized to directly access to some artifact information (see Figure 2.4). This traceability graph performs very well in displaying all links of a selected source artifact. However, it fails to support the display of multiple artifacts' links.

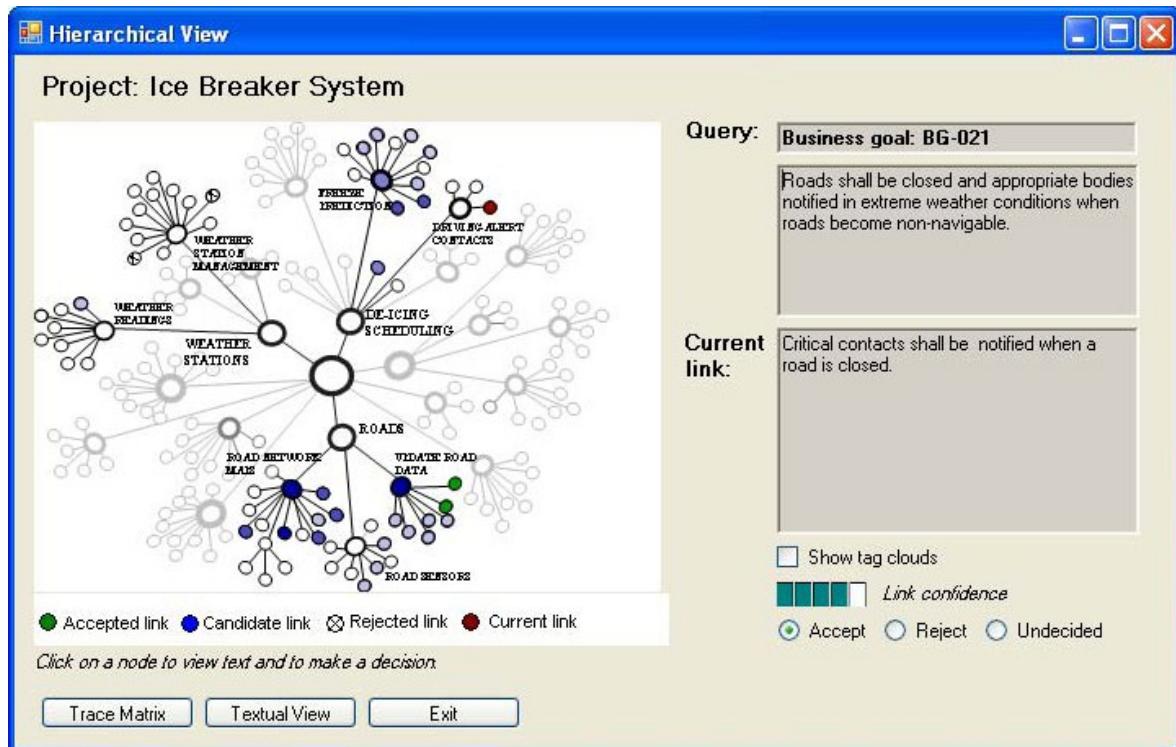


FIGURE 2.5 GUI PROTOTYPE SHOWING LINKS IN A HIERARCHICAL GRAPHICAL STRUCTURE
(CLELAND-HUANG AND HABRAT, 2007)

Cleland-Huang and Habrat (2007) proposed a visual prototype that graphically depicts the dispersion and probabilities of the links generated by the PM IR model in a hierarchical graphical structure (see Figure 2.5). In this graph, leaf nodes are represented by requirements while titles and other hierarchical information are represented as internal nodes. Nodes can be contracted and

expanded according to their depth in the tree. Node size is used to depict the distance from the root, meaning that requirements headings are portrayed by larger nodes than lower level requirements. Degrees of shading represent the likelihood of each node being linked to the query, i.e. darker nodes are more likely to be linked than paler ones, and unshaded nodes are considered very unlikely to represent true links. Parts of graph with no candidate links are faded out. When users click on a node that is shaded red, they can view related textual information and the level of its confidence score. Users can then use this view to accept or reject links. Rejected links are coloured white and marked with “x”, and accepted links are shaded green. This visualization graph provides a birds-eye-view of candidate links and their distribution across the set of traceable artifacts, and allows the user to explore groups of candidate links that naturally occur together in the document’s hierarchy (Cleland-Huang and Habrat, 2007). Unfortunately, this visualization is hard to understand due to the lack of description of the nodes. Moreover, it becomes very large as the data set gets bigger. In addition, it uses the display space inefficiently.

ENVISION, developed by Zhou et al. (2008), adopts a hyperbolic tree view with the enhancement of a “focus+context” approach to facilitate software traceability understanding (see Figure 2.6). It is an Eclipse-based prototype and automatically extracts traceability links between source code and UML design or JUnit tests. These extracted links are represented in the hyperbolic tree, in which nodes are used to represent artifacts and edges are links. Nodes are coloured based on their status, and edges are labeled with the information contained by them. Once a node is selected in the tree view, its properties are shown in the property view. Links in the tree view can be manipulated, such as adding a new link, deleting out-of-date links, and modifying the properties of a link. There are two ways to create a new link. The first is to allow users to directly build a link between the source and target artifacts if the two artifacts are shown on the screen. The newly-created link is highlighted and users can fill in a more detailed description for it. The second way is to locate the source artifact and make it appear on the screen, then use the search function to find the target artifact, and finally to drag and drop the target artifact in the search result view to the source artifact in the tree view. The search function can help users search and locate a specific node or edge by their name and type. The filter function can filter out uninterested nodes or edges from the tree view by their name and type. This prototype also provides a historical navigation view to capture and record the user’s navigation path in the tree view. The history view assists the user to

memorize the navigation process and quickly go back or forward by clicking on nodes in the path. The results of their empirical study show that the hyperbolic tree view allows users to have a global view of links as well as being able to dive deep into an interesting traceability path. The historical navigation makes users not becoming prone to getting lost. Moreover, the seamless integration of ENVISION with Eclipse IDE allows users to better understand the details of traceability links. However, the hyperbolic tree view is not space-efficient. It becomes too big to be shown in the screen as a whole if the number of artifacts becomes large.

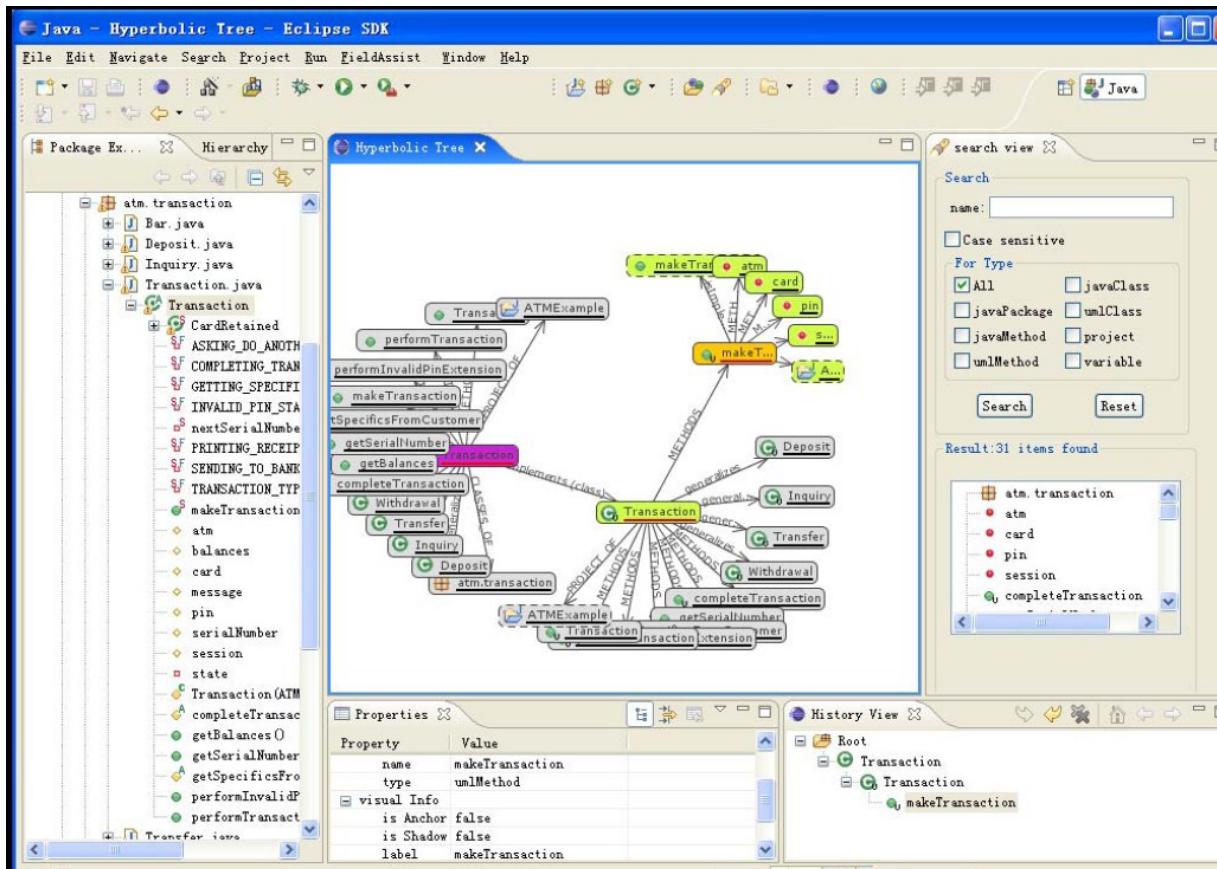


FIGURE 2.6 ENVISION USER INTERFACE (ZHOU ET AL., 2008)

Graph visualizations

TBreq (TBreq, 2012), integrated with the LDRA tool suite (LDRA, 2012), supports the tracing of requirements through the entire software development life cycle. The LDRA tool suite provides a comprehensive range of both static and dynamic software analysis, in addition to unit testing and requirements traceability (LDRA, 2012). TBreq provides end-to-end graphical traceability from requirements to design, code, and test in a single view. Users can see a complete picture of

traceability from requirements to other artifacts. TBreq captures requirements from any management tool or source, creates test specifications and executable test cases directly from requirements, and then link requirements to design, code, and test cases. It lists artifacts horizontally and draws linear edges between related items of artifacts (see Figure 2.7). It can show links between two non-consecutive artifacts by drawing edges under the artifacts. Once an item of an artifact is selected, all its related items are highlighted, and its attributes and contents are shown underneath. Links between artifacts can be modified. Although this graphical traceability view works well for small systems, systems with medium to large numbers of artifacts quickly suffer from severe visual clutter (van Ravensteijn, 2011). This view cannot provide the hierarchical structure of the traced system, although it shows the inherent hierarchy of each artifact.

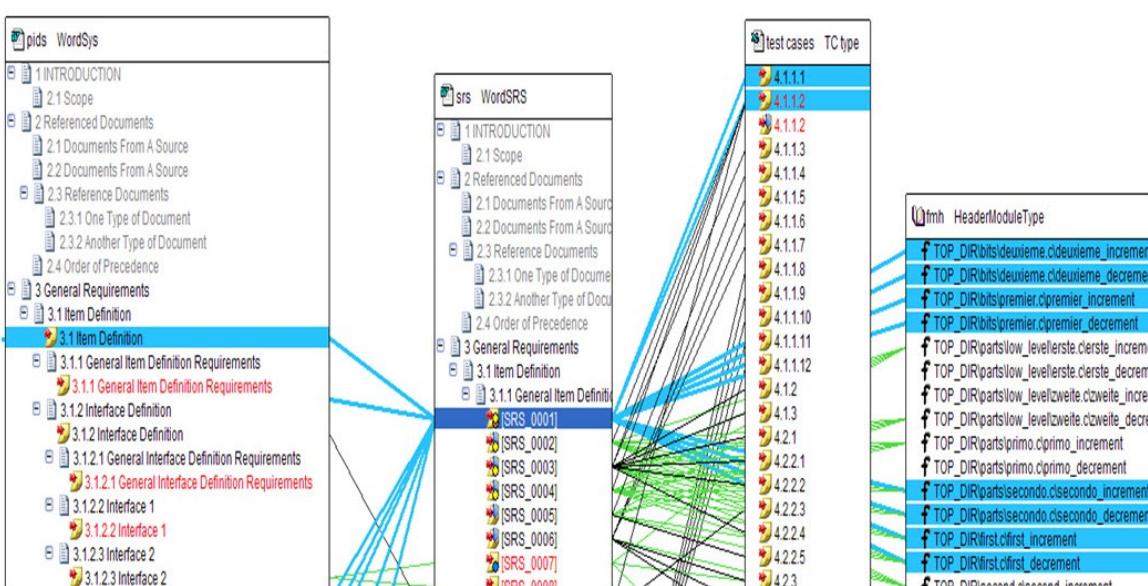
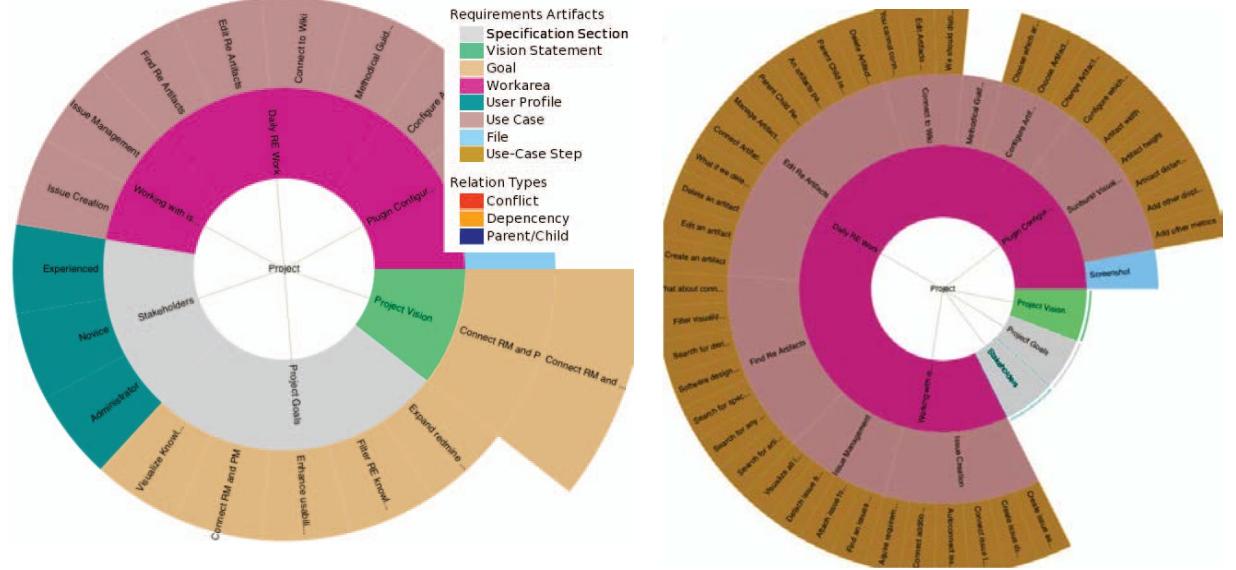


FIGURE 2.7 TBREQ TRACEABILITY LINK VISUALIZATION (LDRA, 2012)

Merten et al. (2011) developed the sunburst and netmap visualizations to display traceability links between requirements knowledge elements. The implementation of the visualizations is developed in conjunction with a requirement plugin for the Redmine project management tool (Lang, 2011). Traceability links between requirements are generated by the requirement plugin.



(a) Tasks (unfolded)

(b) Subtasks (folded)

FIGURE 2.8 THE SUNBURST VISUALIZATION (MERTEN ET AL., 2011)

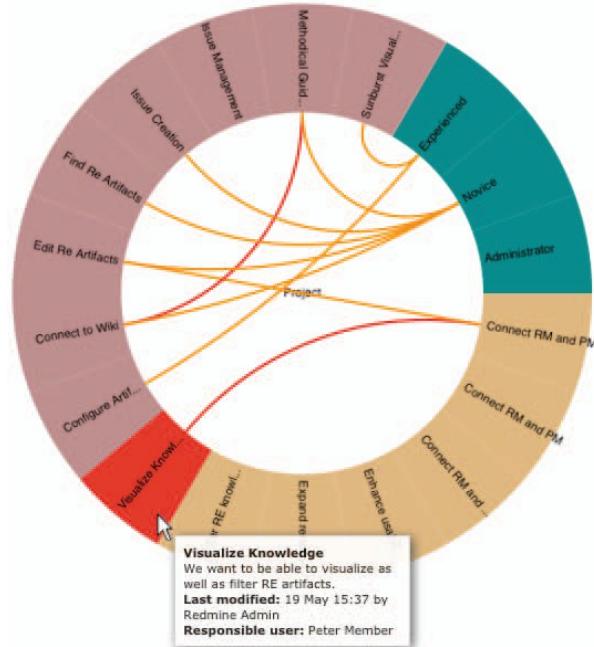


FIGURE 2.9 THE NETMAP VISUALIZATION (MERTEN ET AL., 2011)

The sunburst visualizes the hierarchical structure of the project under trace (see Figure 2.8). Nodes as requirement elements are arranged in a radial layout and are displayed on adjacent rings representing the tree structure. Requirement elements are coloured differently (see Figure 2.8a). In order to mitigate readability issue of nodes on the outer circles, the sunburst visualization supports folding (see Figure 2.8b) and unfolding (see Figure 2.8a) of nodes. Folded branches that are not of

current interest free space for other nodes, which are enlarged. If a certain branch in the sunburst needs to be further researched for links between requirement elements, a netmap helps the user to gain more insights. The netmap aims to represent links between requirements. The nodes in a netmap are in a circle and are segments of exactly one ring in the sunburst. Traceability links are drawn by using linear edges in the inner circle, lines with arrows for directed links and hyper/curved lines without arrows as undirected links. Links are coloured differently based on their types; conflict links are coloured red, dependency links orange, and parent/child links blue. When the mouse is over a node, its full name or description is shown. The filter function is provided to filter for requirement element types as well as traceability link types to help users focus on certain specification parts. Although the two visualization 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.

Edge aggregation

The issue of visual clutter can be mitigated by applying edge aggregation, which is to group edges together. The hierarchical edge bundling technique (Holten, 2006) can group edges based on visually bundling adjacent edges (i.e. non-hierarchical edges) together. EXTRAVIS (Cornelissen et al., 2007) employs the hierarchical edge bundling technique (Holten, 2006) to group edges based on the structure of a hierarchy to reduce visual clutter. This tool includes two views: a circular bundle view and a massive sequence view (see Figure 2.10). Using the circular bundle view shows a detailed visualization of the structural entities of the system under trace and their interrelationships. The hierarchies are shown by using an icicle plot based on the mirrored layout. Edges that are represented relations between nodes are bundled together. The hierarchical entities can be collapsed to enable users to focus on specific parts of the system. When a node is selected, its related nodes and their edges are highlighted. The massive sequence view provides an overview of (part of) the full trace links set in order to support users in identifying parts of the interested traces. Overall, the circular bundle view is aesthetically pleasing and helps users quickly gain insight in the adjacency relations in hierarchically organized systems (Holten, 2006). However, when considering a large number of traces, it becomes difficult to discern the various colours and to prevent bundles overlapping (Holten, 2006).

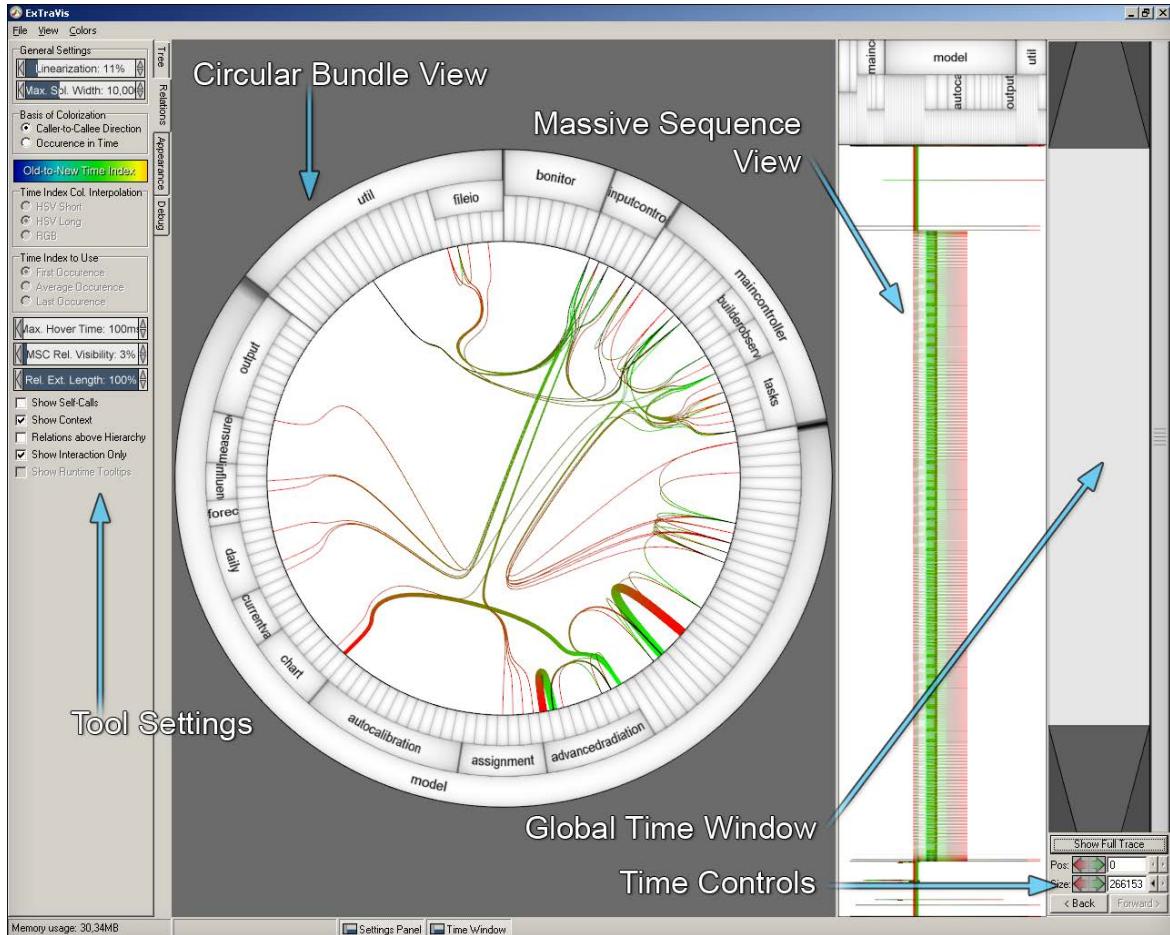


FIGURE 2.10 THE USER INTERFACE OF EXTRAVIS (CORNELISSEN ET AL., 2007)

TraceVis, developed by van Ravensteijn (2011), visualizes a dynamic list of hierarchies and adjacency relations that are provided manually (see Figure 2.11). It uses icicle plots and hierarchical edge bundling (Holten, 2006) techniques to support the hierarchical structure and to reduce visual clutter. Icicle plots are used to represent hierarchies vertically to provide a global overview of the connectivity between items and convey the structure of the hierarchies. An icicle plot is a space-filling solution and provides a fairly compact visualization of hierarchies (van Ravensteijn, 2011). TraceVis allows the connection of adjacency relations to a previous and a next hierarchy by horizontally mirroring the icicle plot. Adjacency relations are represented by drawing edges between related items. Edges are displayed using splines and are grouped using hierarchical edge bundling. When an item is selected, its related items and their edges are highlighted and detailed information about itself and/or its links are shown in the detailed information view. In order to reduce visual clutter and make the text in the nodes more readable, items of less interest and their connected relations can be hidden. Overall, TraceVis supports an overview as well as a

detailed insight into inter-related, hierarchically organized data. However, it uses space inefficiently and can result in visual clutter if the dataset is large or lateral relations visualized (van Ravensteijn, 2011).

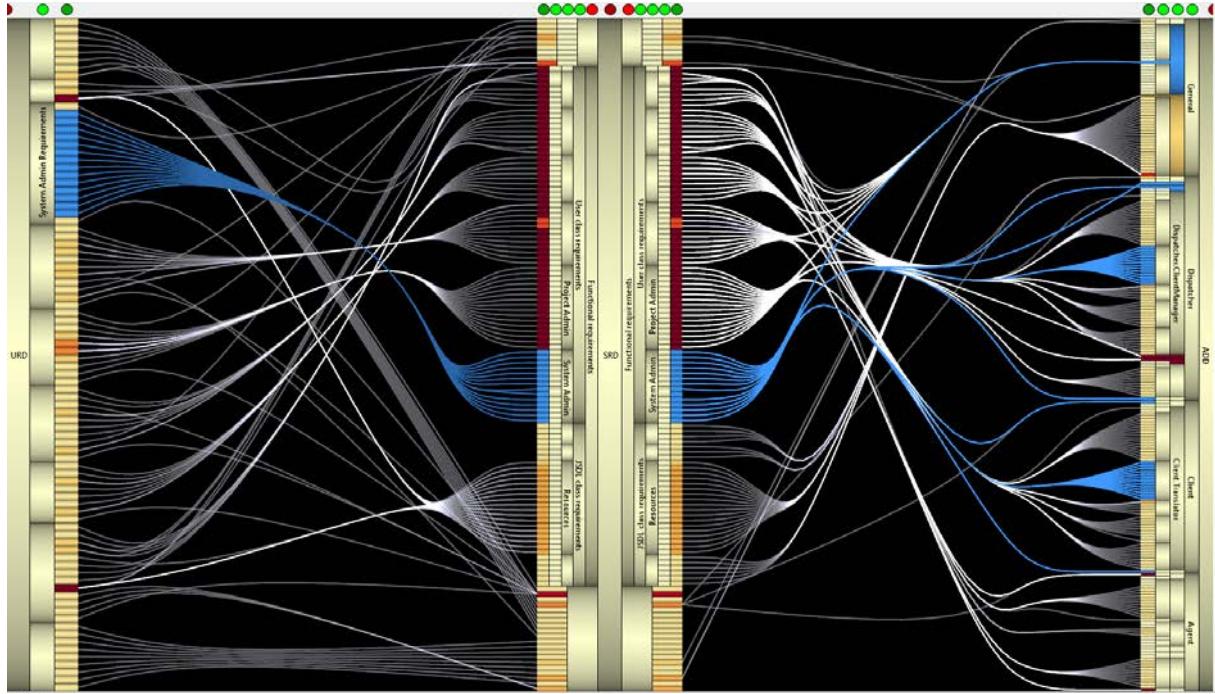


FIGURE 2.11 THE VISUALIZATION OF TRACEVIS (VAN RAVENSTEIJN, 2011)

3D approach

UNITI (Pilgrim et al., 2008) is an Eclipse plug-in for transformation chain modeling and execution with the support of traceability between UML model elements. Traces are generated by automatic model transformations and are represented by a 3D approach (see Figure 2.12). In the 3D approach, UML models are projected on layered planes. Traces between different levels of abstraction are visualized by using edges between planes. Traces are displayed in different colors depending on the level of the connecting elements; links connecting top-level elements (e.g. classes or associations) are coloured with red while green is for links connecting nested elements (e.g. operations). Uninteresting traces can be filtered out to make the traceability visualization clearer. Although presenting more content at once, giving an overview of traces, and grouping related information together, the 3D approach adds more complexity to the graph, and still leads to visual clutter when the data set becomes large.

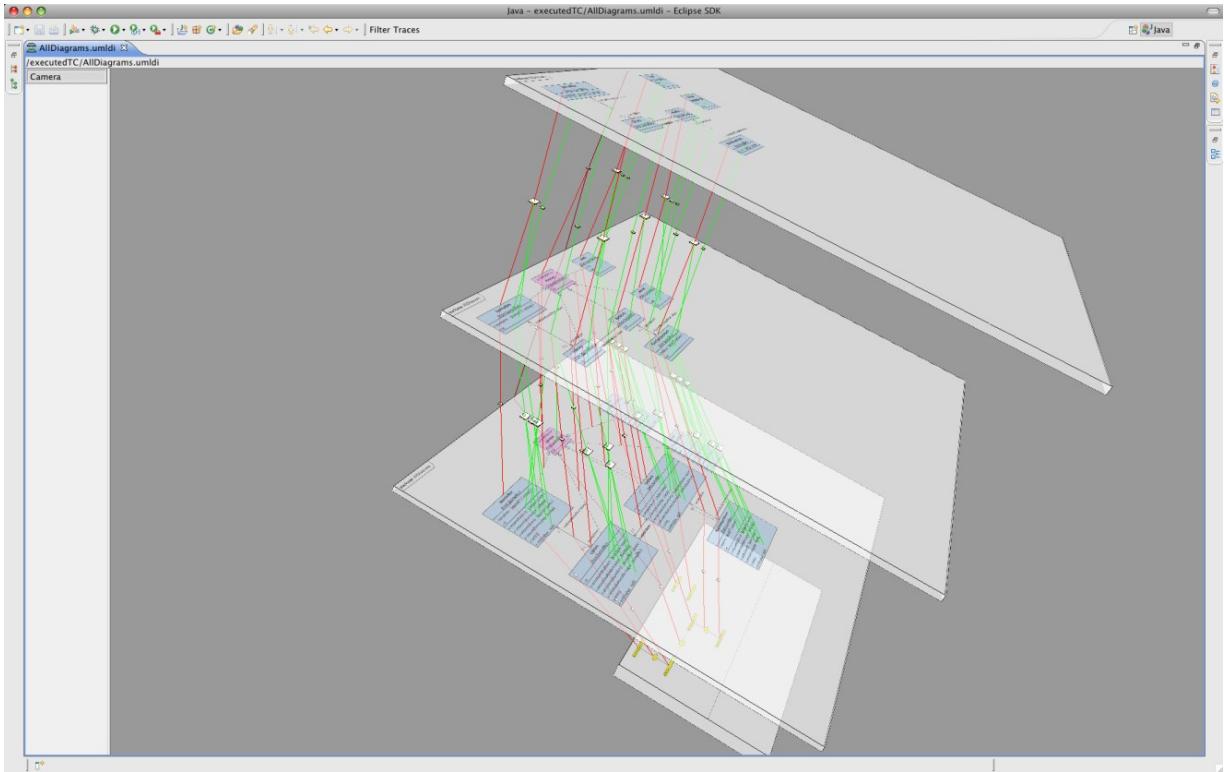


FIGURE 2.12 THE VISUALIZATION USER INTERFACE OF UNITI (PILGRIM ET AL., 2008)

To sum up, tree visualizations use display space inefficiently. They can grow quickly to an enormous size when dealing with large numbers of artifacts and traceability links. Graph visualizations can quickly lead to visual clutter when the traced system becomes large. Grouping links together using hierarchical edge bundling can ameliorate the visual clutter issue, but large systems may still have issues with bundle overlapping. Although showing an overview of traces, the 3D approach suffers from visual clutter when dealing with large numbers of artifacts.

2.5.3 Other Traceability Visualization Techniques

In addition to traditional approaches and the various graph representations similar to those reviewed above, there are several other approaches that have been used to visualize traceability links. Poirot (Cleland-Huang and Habrat, 2007) displays trace results in a textual format (see Figure 2.13). Poirot is a web-based traceability tool and adopts the PM IR model to retrieve links between artifacts. Retrieved links are clustered by feature to group links together in a meaningful way. Poirot uses confidence levels, user feedback checkboxes, and tabs separating likely and

unlikely links to assist the analyst in evaluating candidate links. The confidence score that indicates the likelihood of a link is illustrated under the column labelled confidence level. Link scores are categorized into 6 different levels by displaying from 0 to 5 bars representing the likelihood that the link is a correct one. Users can use the checkboxes to verify the correctness of a link and can also search a specific artifact using the ID of the artifact. However, it cannot visualize the overall structure of the traced system nor provide an overall view of traces.

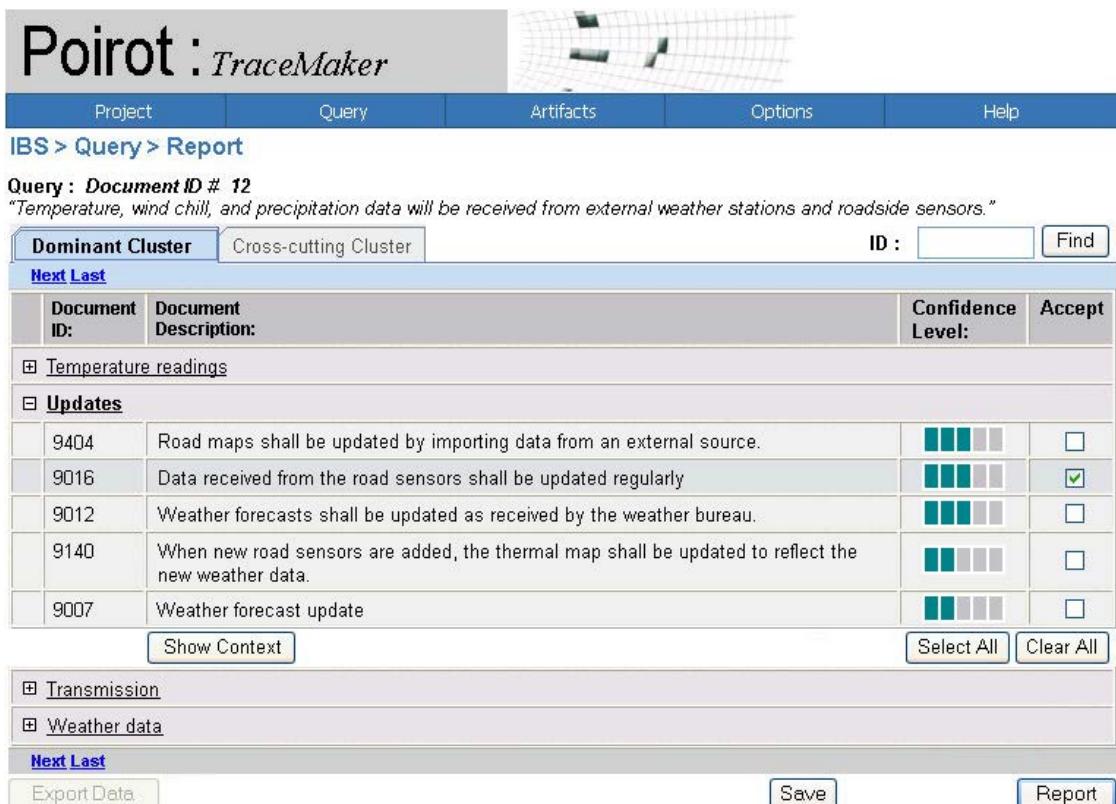


FIGURE 2.13 SHOWING LINKS IN CLUSTERS IN POIROT (CLELAND-HUANG AND HABRAT, 2007)

TraceViz (Marcus et al., 2005) employs a map consisting of coloured and labeled squares to display traceability links for a specific source or target artifact (see Figure 2.14). TraceViz is seamlessly integrated into the Eclipse IDE and uses a standard Eclipse view to host its user interface. It uses the LSI IR technique to recover traceability links between source code and documents. TraceViz is composed of three parts. First, the elements area on the left contains the source and target browsers. Second, the link area in the middle displays the links for a specific source or target. Links in TraceViz are grouped into categories and are represented by coloured squares. Third, the information area on the right shows the link properties and browsing history.

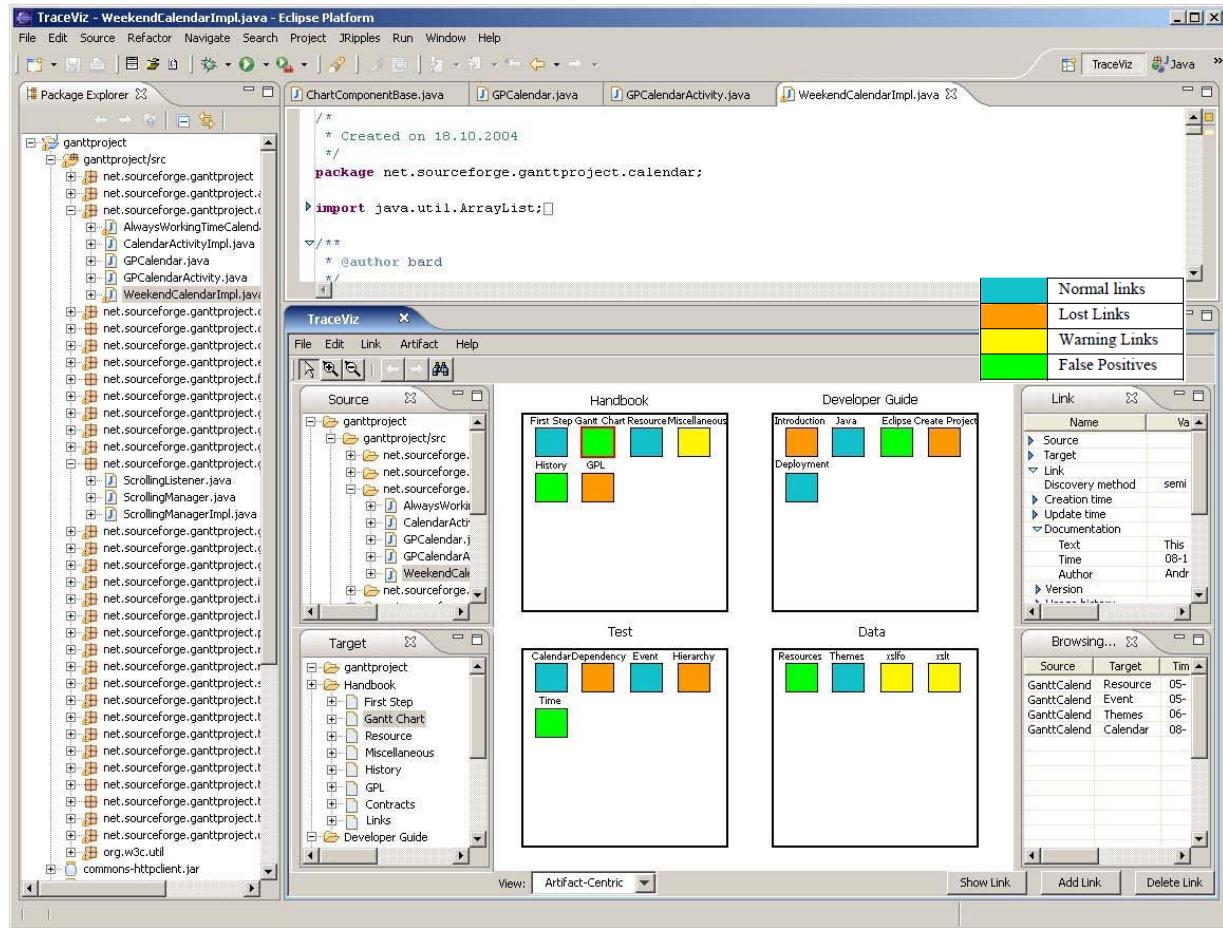


FIGURE 2.14 THE TRACEVIZ USER INTERFACE (MARCUS ET AL., 2005)

Users can launch TraceViz while editing any file in the Eclipse project and this edited document automatically becomes the source of the traceability links. Users select a file as the source or target of traceability links in the elements area. Then all links that have this file as their source or target artifact are represented as small squares in the link area. These links are grouped in larger squares and coloured to denote whether the link is a normal, lost, warning, or false positive link. Once a link in the link area is selected, all attributes and browsing history of the selected link and the time will be displayed in the information area. Users can add a new link and delete or modify existing links. Once a new link is created, the elements and attributes of the link need to be inserted. Users also can search a specific item using query-based artifact/link search or keyword search. Moreover, users can filter displayed links based on information about link category or link update time. Overall, TraceViz allows users to clearly visualize all links of a selected source artifact or a chosen target artifact. Unfortunately, it is unable to display links for multiple artifacts at the same time.

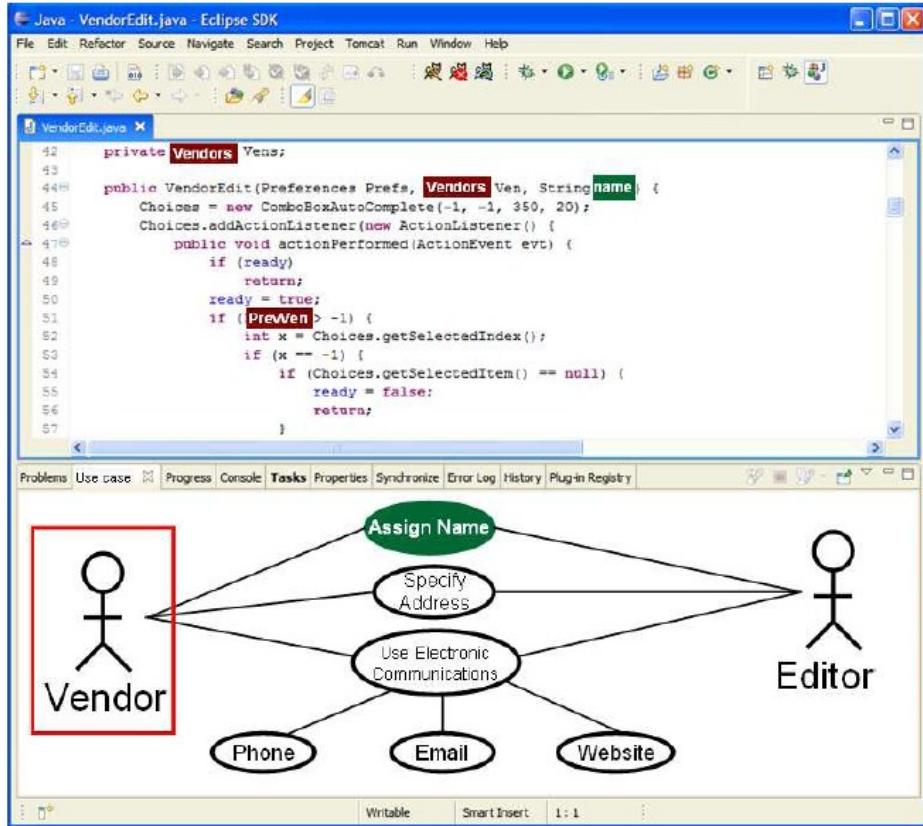


FIGURE 2.15 THE USER INTERFACE OF LEANART (GRECHANIK ET AL., 2007)

LeanArt (Grechanik et al., 2007) utilizes an intuitive point-and-click graphical interface to enable users to navigate to program entities linked to elements of use case diagrams (UCDs) by selecting these elements, and to navigate to elements of UCDs by selecting program entities to which these elements are linked (see Figure 2.15). It is an Eclipse plug-in and recovers links between UCDs and source code using a machine learning technique.

UCDs are shown in a tab called Use Cases in Eclipse. Users can navigate from elements of some UCD to program entities by selecting an element of the UCD and clicking on it. The selected element changes its colour and a frame is drawn around it. Then to what program entities this element is linked are determined based on the information of recovered links. Finally, LeanArt loads all source code that contains program entities linked to selected elements of the UCD, and highlights program entities with the selected colour. Conversely, users can navigate from program entities to elements of UCDs by right-clicking on the program entity that is highlighted. A context menu is presented with a selection of menu items, one of which shows the traceability links. Once users select this menu item, a list of relevant UCDs is shown, from which users select a subset of

the UCDs. Each selected UCD is loaded into a separate use case tab, and elements of these UCDs linked to highlighted program entity are coloured. The characteristic visualization of links in LeanArt is to select a source and then display targets linked to this source. It also fails to represent all links at the same time.

To sum up, non-graphic link visualizations fail to provide a suitable hierarchical structure of the traced system and also fail to provide a suitable overview of trace links.

2.5.4 Hierarchical Structural Visualization Techniques

The most widespread technique to show a system's structure is the node-link based layout, which represents parent-child relations as lines drawn between items that are nodes in the tree (Graham and Kennedy, 2010). The most common layout is the hierarchical tree (see Figure 2.16), which generally is drawn from top to bottom or from left to right. A hierarchical tree can convey the hierarchical nature of a structure in a graphical form by positioning children nodes below or after their common ancestor (Herman et al., 2000). This layout is very effective for small trees, but falls short when a large number of items are to be displayed simultaneously. It also is not space-efficient. This issue can be remedied to enable the hierarchical tree to collapse branches that are not interested and expand them when needed. The ability to expand and collapse can make the use of the display space more efficient.

Space filling approaches try to overcome space-inefficiency issues by using enclosure to represent structured data. The treemap layout (see Figure 2.17) is one of the most common space filling approaches. It indicates parent-child relations by placing child node representations within the boundaries of their parent node (Graham and Kennedy, 2010), which make it an ideal approach for displaying a large number of items (Shneiderman, 1992). Each branch of the hierarchical tree is given a rectangle, which is partitioned into smaller rectangles representing sub-branches. The size of each rectangle may represent the proportional significance of a given item (Herman et al., 2000). Each rectangle may be coloured to show a separate dimension of the data. The treemap layout can reflect the structure of the hierarchical tree as a result of its construction. However, it is difficult to perceive hierarchical structure using this layout (Herman et al., 2000).

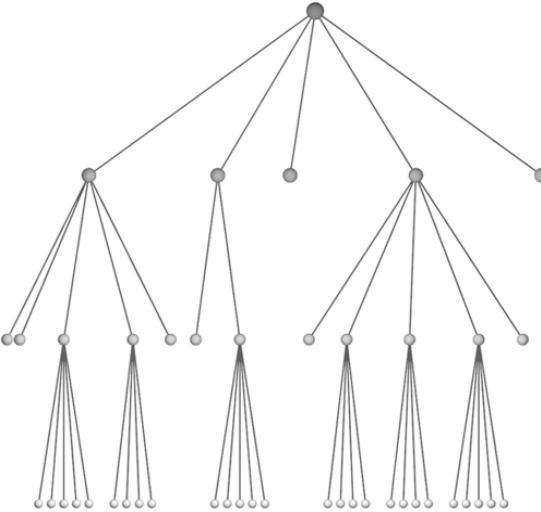


FIGURE 2.16 HIERARCHICAL TREE VISUALIZATION (HOLTEN, 2006)

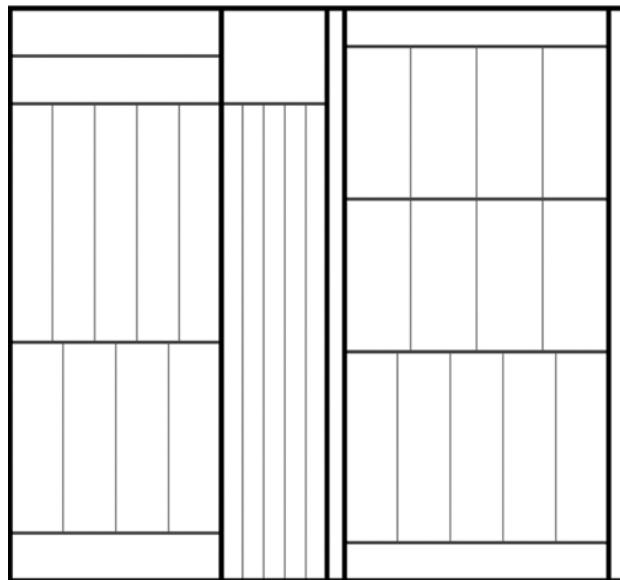


FIGURE 2.17 TREEMAP LAYOUT (HOLTEN, 2006)

2.5.5 Requirements for Traceability Visualization Systems

From the above discussion on existing traceability visualization systems, nine main technical challenges in developing a successful traceability visualization system have been identified:

- 1) **Need to seamlessly integrate the link visualization support within an IDE (e.g. Eclipse) or other software management tools** (Marcus et al., 2005; Zhou et al., 2008).

This kind of integration allows users to utilize functionality not only in the IDE or the management tool but also the link visualization system. For example, if the link visualization

system is integrated with Eclipse, users can change a piece of code in a class and simultaneously check what artifacts are impacted by this change through browsing artifacts related to this class. Integrating the link visualization system with an IDE or other systems can support a common look and feel for the artifacts and links between them (Marcus et al., 2005).

- 2) **Ability to integrate with traceability recovery tools to automatically capture traceability links between artifacts in the traced system** (Marcus et al., 2005; Zhou et al., 2008).

If links between artifacts in a traced system are created automatically, these retrieved links are visualized.

- 3) **Provide efficient and effective visualization techniques to represent retrieved links.**

Technical challenges for successful visualization techniques are as follows.

- ◆ **Use the display space efficiently when representing a large number of artifacts in the traced system.**

The link visualization techniques (discussed above) are unable to use display space efficiently; they can grow quickly to an enormous size when dealing with large numbers of artifacts and traceability links. This issue can reduce the readability of the graph that use to display artifacts in the traced system (Kienle and Muller, 2007).

- ◆ **Remedy the visual clutter issue when visualizing links in the system.**

Current graph link visualization techniques suffer from the visual clutter issue when displaying a large number of artifacts hence a large number of links between them. This issue can impede the ability to efficiently browse, analyze, and maintain traceability links between artifacts (Holten, 2006; van Ravensteijn, 2011).

- ◆ **Show the whole hierarchical structure of the system.**

Displaying the hierarchical structure of a system can assist users in understanding, managing, and maintaining the system (Holten, 2006; van Ravensteijn, 2011).

- ◆ **Show the overall overview of traces in the system** (van Ravensteijn, 2011; Zhou et al., 2008).

An overview of traces provides users with information about the distribution of traces in the system and whether or not an artifact has links. It provides users with a fast and efficient way to know which artifacts have links.

◆ **Illustrate detailed dependency information for a specific artifact.**

When an artifact is selected, its detailed dependency information is displayed. This information can assist users in comprehending and maintaining each artifact (van Ravensteijn, 2011; Zhou et al., 2008).

- 4) **Provide detailed information about each link or artifact to assist users to understand them** (Marcus et al., 2005; Merten et al., 2011; van Ravensteijn, 2011; Zhou et al., 2008).

Additional information concerning a selected artifact (e.g. the name of the artifact, its content, or its related artifacts) can assist users in comprehending the artifact and its related artifacts.

- 5) **Allow end users to add new links, delete or modify existing links and their similarity values, and edit the properties of links and their connected artifacts** (Marcus et al., 2005; Zhou et al., 2008).

A traceability recovery technique can return incorrect links to users and fail to capture some correct links. Support for link editing can allow users to delete incorrect links and add a correct link missed by the recovery technique.

- 6) **Allow end users to browse through artifacts to locate a specific item in the traceability visualization view** (Marcus et al., 2005; Zhou et al., 2008).

Navigation support can assist users in locating a particular item easily and efficiently (Marcus et al., 2005; Zhou et al., 2008).

- 7) **Allow end users to search a specific link or artifact through keywords or query** (Marcus et al., 2005; Zhou et al., 2008).

Kienle and Muller (2007) indicate that the lack of the searching support in software visualization definitely hinders users from finding specific items.

- 8) **Filter out some uninterested links or artifacts** (Marcus et al., 2005; Zhou et al., 2008).

Filtering information can allow users to reduce the amount of visualized data and to limit the

analysis (Kienle and Muller, 2007).

9) **Capture and record browsing history** (Marcus et al., 2005; Zhou et al., 2008).

A browsing history that contains all previously performed steps of graph operations can allow users to revert to previous states (Kienle and Muller, 2007).

2.6 Summary

Traceability helps software engineers to effectively and efficiently develop, manage, update, and reuse a software system throughout the software development life cycle. According to the literature discussed above, there are two major challenges that need to be addressed in designing a successful traceability system: (1) correctly recovering traceability links between software artifacts and (2) visualizing these links effectively and efficiently. None of the traceability recovery techniques developed so far is able to produce sufficiently consistent and high enough quality results to meet developers' needs. Semi-automatic techniques are unable to generate traceability links automatically without human intervention. Although automatic techniques improve this issue, their limitations impede them from capturing all potential correct links and having few incorrect links. None of the traceability visualization techniques developed so far can visualize an overwhelmingly large number of traceability links effectively and efficiently.

The research presented here concentrates on addressing the two major challenges and in doing so aims to design a system that can produce high quality and detailed trace links between artifacts (discussed in Chapter 3), and can visualize these links in a natural and intuitive way (discussed in Chapter 6) to meet the requirements we have explored. In Chapter 4, we also discuss the challenge of creating a robust and effective traceability benchmark to facilitate the evaluation and comparison of traceability recovery techniques.

Chapter 3 -- Traceability Link Recovery

The first challenge we encounter in software traceability is to how to go about automatically retrieving relationships between artifacts inside a system with both high precision and high recall. This chapter describes a new combination traceability link recovery technique that we have invented, called **IRETrace**, to more effectively address this challenge.

3.1 Introduction

Source code alone is not sufficient to capture all information about a software system. Software requirements, architectural decisions, detailed design, tutorials and user documentation and various types of technical system documentation (e.g. deployment configuration) are important artifacts produced while engineering software systems. Tracing and maintaining interrelationships between these various forms of software documentation and source code enables software engineers to understand systems better, to undertake improved maintenance of systems, and ultimately to produce higher quality systems (Antoniol et al., 2002; Antoniol, Casazza et al., 2000; Seacord et al., 2003). However, this relies on retrieving high quality candidate links between elements in one artifact (e.g. code constructs) and elements in another (e.g. requirements and detailed design documentation). A set of high quality candidate links represents a link set between these artifacts that contains as many true/correct links as possible and as few fault/incorrect links as possible. Moreover, a high quality candidate link set should connect elements of different artifacts at a fine-grained level of detail e.g. part of a design document description and its related source code elements. However, it is very challenging to automatically extract high quality candidate links among the wide variety of artifacts created during the software development life cycle (Antoniol et al., 2002; Gotel et al., 1994; MacQueen, 1967; Wang et al., 2009).

Many traceability recovery techniques have been invented to retrieve traceability links among artifacts. Some need human intervention while others can automatically generate traceability links. Unfortunately, no recovery approaches have the capability of recovering all possible links among artifacts automatically and accurately. This is due both to the inherent imprecision when expressing things in natural language and inherent information loss or addition when moving between software artifacts at different levels of abstraction. Some potentially useful and important links are missed by existing techniques. Similarly, some incorrect or unhelpful links are extracted and these may confuse developers. However, different link retrieval approaches have different strengths and weaknesses. To try to improve the performance of automated traceability link retrieval, we have developed an approach called Information Retrieval Enhancement Traceability (**IРЕTrace**) that combines Information Retrieval (IR) models with three supporting techniques: Regular Expression (RE), Key Phrases (KP), and Clustering. These particular techniques have quite different strengths and weaknesses and recover different sets of links due to their vastly different retrieval approaches. Our approach attempts to take advantage of the strengths of these techniques to automatically recover links between artifacts at both high precision and high recall.

Our particular focus is on retrieving links between class entities and sections in documents written in natural language, e.g. tutorials, handbooks, developer or user's guides, API documentation, architecture documentation, design rationale and emails. The objective of this research is to demonstrate that our new composite traceability link recovery approach can improve the automatic recovery of traceability links between these two sources of information with high precision and recall. We have conducted a detailed experiment (described in Chapter 5) with four case studies to evaluate the strengths and weaknesses of our approach. Analysis of these experimental results demonstrates that our approach improves the performance of IR, increases the precision of retrieved links, and recovers more true links than IR alone.

In this chapter, we firstly provide the motivation for this approach. Next, we introduce our traceability link recovery approach -- **IРЕTrace**. We then discuss its overall design and describe each technique combined in the approach. Following that, the implementation of our approach is described. The evaluation results of our approach are presented in Chapter 5.

3.2 Motivation

Consider manually retrieving relationships between artifacts of a software system. It is not hard but still time-consuming for engineers manually to find relationships between artifacts in a small system with up to 100 classes and several documents, especially for those who are expert in this system. However, it is an extremely arduous, time-consuming, tedious task to recover traceability links in a system with hundreds of classes and hundreds of documents. With the purpose of evaluating our traceability recovery approach, we manually created a traceability benchmark for a system called JDK1.5-SUBSET, which is discussed in Chapter 4. This comprised 294 classes and 3 PDF documents with 182 sections. The most important part of the traceability benchmark is to establish an oracle/true traceability link set, which consists of true/correct links between classes and sections. We recruited 11 analysts who have at least six years of Java programming experience. They manually identified and verified true links between classes and sections. When building the oracle link set, each participant spent one hour to identify related sections of 50 classes on average. It would be an enormous workload to capture traceability links if the number of classes and documents was significantly larger and to discover relationships not only between classes and documents but also between documents and documents. In order to ease the manual workload in the recovery of traceability links in a system, various traceability link recovery techniques have been invented.

In Chapter 2, we discussed techniques that have so far been developed to retrieve traceability links between artifacts of a software system. These techniques fall into two main groups: semi-automatic recovery and automatic recovery. Semi-automatic techniques are unable to generate traceability links automatically without human intervention. These include rule-based, scenario-driven, and value-based approaches. Rule-based approaches build traceability rules using the grammatical structures present in natural language sentences (Jiraphanthong and Zisman, 2005; Jiraphanthong and Zisman, 2009). A scenario-driven approach needs to create the hypothesized traces first (Egyed, 2003). A value-based approach requires the determination of the value of every artifact before starting the traceability link recovery process (Egyed et al., 2005). Semi-automatic techniques require engineers to have some knowledge of the system being traced. It is difficult to employ these techniques to retrieve traceability links between artifacts in a system for people who

are unfamiliar with the system. Automatic recovery techniques address this issue.

Automatic recovery techniques can be further categorized as lightweight and heavyweight techniques. Lightweight techniques do not require pre-computation of the input and can be directly executed at run-time, such as Regular Expressions. Regular Expressions (Bacchelli et al., 2009; Bacchelli et al., 2010) miss links where class names are mentioned only implicitly in documents. Heavyweight techniques, by contrast, require pre-processing of their input. These techniques include Text Mining (TM) and Information Retrieval (IR) models. The TM technique only extracts from documents salient facts about pre-specified types of events, entities, or relationship (GATE Information Extraction, 2010; Weiss et al., 2005). Using IR models discussed in Chapter 2 (e.g. Probabilistic Model (PM), Vector Space Model (VSM), and Latent Semantic Indexing (LSI)) to retrieve traceability links has a significant limitation: the accuracy rate of link recovery heavily relies on a *cut point*; only links that have a similarity value greater than or equal to the *cut point* are recovered (Cleland-Huang et al., 2005; Lucia et al., 2007). In other words, retrieved links above the cut point are retained and links below are discarded. Using a low cut point retrieves a larger number of true/correct links than using a high cut point. However, more fault/incorrect links are captured at the same time. In other words, IR models gain low precision but high recall at low cut points, and high precision and low recall at high cut points. Moreover, the same cut point may or may not be best suited for different systems.

In order to improve the performance of IR models, many strategies have been invented to ameliorate the limitations of IR models. For example, Cleland-Huang et al. (2005) proposed an approach to improve the performance of dynamic requirements traceability by incorporating three different strategies into PM, namely hierarchical modeling, logical clustering of artifacts, and semi-automated pruning of the probabilistic network. Their results indicate that the three strategies effectively improve trace retrieval performance. Hayes et al., (2003; 2006) used VSM but with a context-specific thesaurus that is established based on technical terms in requirements documents to recover links between requirements. This combination approach improves recall and sometimes precision. Wang et al., (2009) presented four enhanced strategies to improve LSI, namely, source code clustering, identifier classifying, similarity thesaurus, and hierarchical structure enhancement. This approach achieves higher precision than LSI and PM, but lower recall. Although various

strategies have been applied to enhance the performance of IR techniques, no approaches to date can substantially decrease incorrect links at low cut points and significantly increase true links at high cut points (Antoniol et al., 2002; Cleland-Huang et al., 2005; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009).

Semi-automatic recovery techniques need human intervention during the traceability link extraction process. Although automatic techniques improve this issue, their limitations impede the capture of all potential true links and few incorrect links. To varying degrees, none of the traceability recovery techniques developed so far can produce sufficiently consistent and high enough quality results to meet developers' needs (Antoniol et al., 2002; Bacchelli et al., 2010; Cleland-Huang et al., 2005; Egyed et al., 2005; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009). This leads us to the following three main requirements for a more comprehensive traceability link recovery technique aimed at target end users who want to get to know the traced system (The more detailed requirements are described in Chapter 2):

- Ability to automatically capture traceability links between artifacts.
- Ability to retrieve as many true links as possible and as few fault/incorrect links as possible; high precision and high recall at all cut points.
- Ability to minimize/mitigate the issue of dependency on cut points; to narrow the gap between the precision and recall results generated at different cut points.

These issues motivated us to invent a new traceability recovery approach that is effortless to employ yet retrieves links at a high-level of quality and precision. The discussion in Chapter 2 on recovery techniques shows that combining recovery techniques can improve performance. For example, adding IR into RE augments the retrieved link set produced by RE alone, as IR can capture links that cannot be recovered by RE. Our approach is to combine several existing techniques to counteract each other's weaknesses by taking advantage of their strengths. In addition, as change is inevitable during the software development and evolution process, many of the original requirements, designs and models usually do not make it to the final release of a software system (The Standish Group Report: Chaos, 2007). Larman (2005) stated that only tested code demonstrates the true system design. Source code represents the low-level structure, realised

design and fulfilled requirements of the system. Relationships anchored on source code are thus more stable and less likely to deteriorate than relationships centered on other artifacts. Therefore, our research adopts a code-centric approach and employs source code dependency analysis to extract relations between documents and source code.

3.3 IRETrace -- Information Retrieval Enhancement Traceability

In order to recover traceability links at a reasonable high-level of precision and recall, we have explored an approach incorporating three supporting techniques, Regular Expression (RE), Key Phrases (KP), and Clustering, into IR models to recover links between sections in documents and class entities. Our approach is intended to overcome the limitations of IR models by taking advantage of the strengths of RE, KP, and Clustering.

At first, we use an IR model, a Vector Space Model (VSM), as the fundamental basis of our approach. This is because VSM can retrieve all potential links with appropriate queries. However, VSM has three main limitations (Antoniol, Canfora et al., 2000; Antoniol, Canfora et al., 2002; Cleland-Huang et al., 2005; Hayes et al., 2006; Lucia et al., 2007; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009).

- First, very few true links are retrieved at high cut points, i.e. there is low recall at high cut points.
- Second, many fault/incorrect links are captured at low cut points, i.e. there is low precision at low cut points.
- The third limitation is that VSM misses links in two situations: class names that do not follow a common naming convention strategy; and documents that use different words to describe related classes. Our experience shows these can have significant occurrence in some systems.

Combining the first supporting technique, Regular Expression (RE), with VSM allows extraction

of more true links at high cut points. As long as class names are retrieved correctly and refined regular expressions are built, RE can retrieve all possible links that are related to these class names and return few incorrect links as well. The second supporting technique we add to our approach, Key Phrases (KP), recovers links missed by VSM. We extend the VSM queries to include key phrases of comments in the source code. If code is well commented, KP can extract key phrases from code comments closely related to classes. Clustering, the third supporting technique we incorporated, aims to eliminate incorrect links at low cut points by refining existing retrieved traceability links. As the aim of our approach is to trace useful links between class entities and sections in documents, we take advantage of the inherent hierarchical structure of documents to cluster links retrieved by VSM, RE, and KP. Therefore, our combination approach increases precision at any cut point and retrieves links with a reasonably high recall.

We then extend our approach to employ other IR models to explore whether augmenting IR with the three supporting techniques can improve its performance. A well-known problem with IR models is their dependency on cut points. This causes all IR models to suffer from the same limitations that VSM has. Our hypothesis is that the strengths of the three complementary techniques can make up for the inherent weaknesses of IR models to significantly enhance both precision and recall. The following four sections describe the four techniques we used in detail.

3.3.1 The Basic Retrieval Technique

Information Retrieval (IR) is widely used in searching fields such as web search engines and library document search. We decided to employ an IR technique as the foundation of our traceability links retrieval approach as its query-based approach has potential to recover all types of links, if appropriate queries are constructed.

The IR engine we initially employed was Apache Lucene, a full-featured text search engine written in Java (Apache Lucene – Overview, 2009). We chose this as it is widely used for IR experimentation and practice. Lucene uses VSM to index text and determine how relevant a section is to a query (Apache Lucene – Overview, 2009; Konchady, 2008). As many papers have extensively discussed VSM (Antoniol, Canfora et al., 2000; Antoniol, Canfora et al., 2002;

Antoniol, Casazza et al., 2000; Cleland-Huang et al., 2005; Hayes et al., 2003; Lucia et al., 2007; Marcus and Maletic, 2003; Wang et al., 2009), we only briefly describe how queries are built and similarity scores of links are calculated.

A class name (or identifier) composed of two or more words is split into separate words. A query string for VSM is established by using the OR operator to combine the name and the separated words. For example, “DragSource” is split into the words drag and source, then the query string is “DragSource OR drag source OR drag OR source”. The query is case-insensitive.

The output of the indexing process is a *term-by-document* matrix, where *term* represents all words that occur in documents, and *document* indicates all documents in the VSM corpus. Each entry $a_{i,j}$ of this matrix denotes a weight for the frequency of the i^{th} term in the j^{th} document. Each matrix column is considered as a vector that describes a document. Queries are represented in a similar way by a matrix, where each vector indicates a query. The similarity between a document and a query is measured by the cosine of the angle between the corresponding vectors. In other words, a matching document may have one or more query terms and is ranked according to the frequency of term occurrence and number of query terms present in the document (Antoniol et al., 2002; Konchady, 2008; Lucia et al., 2007). In the end, traceability links between documents and classes are retrieved. Each link has a similarity score ($0 \leq \text{similarity score} \leq 1$) that represents how much each document and class match.

There are three main drawbacks to using VSM. The method calculating link similarity values results in some true links with a very low similarity score and most retrieved links have low similarity values. Therefore, the lower the cut point that is used, the more possible links are retrieved but also the more incorrect links are captured as well. This leads to the first limitation that very few true links are captured at high cut points. VSM possesses high precision but low recall at high cut points. The second limitation is that many incorrect links are extracted at low cut points. This leads to low precision but high recall at low cut points. The third limitation is that links are missed in two situations: class names not following a naming convention strategy and documents using different words to describe related classes. We have found that these are both common occurrences in many software documentation artifacts.

The second IR engine that we utilized in this research is the Terrier IR platform (Terrier IR platform, 2010). We chose to experiment with this engine as it is a comprehensive, flexible, and transparent platform for research and experimentation in text retrieval. Terrier is written in Java and is a highly flexible, efficient open source search engine for the rapid development and evaluation of large-scale retrieval applications. It provides multiple indexing strategies and various retrieval approaches to meet different requirements in text retrieval. Table 3.1 shows the five additional IR models used in our research that are supported by Terrier: TF-IDF, BM25, DLH, PL2, and IFB2 (See Configuring retrieval in Terrier, 2010).

TABLE 3.1 A DESCRIPTION OF IR RETRIEVAL MODELS IN TERRIER

| IR modles | Description |
|-----------|--|
| TF-IDF | The $tf * idf$ weighting function, where tf is given by Robertson's tf (Robertson, 2004) and idf is given by the standard Sparck Jones' idf (Sparck Jones, 1972). |
| BM25 | The BM25 probabilistic model. (Robertson and Walker, 1994) |
| PL2 | Poisson estimation for randomness, Laplace succession for first normalization, and Normalization 2 for term frequency normalization. (Amati, 2003; DFR framwork, 2011) |
| IFB2 | Inverse Term Frequency model for randomness, the ratio of two Bernoulli's processes for first normalization, and Normalization 2 for term frequency normalization. (Amati, 2003; DFR framwork, 2011) |
| DLH | The DLH hyper-geometric DFR model. (Amati, 2006) |

TF-IDF. Essentially, the TF-IDF weighting approach is to determine the relative frequency of words in a specific document compared to the inverse proportion of that word over the entire document corpus (Ramos, 2003). In other words, it decides how relevant a given word is in a particular document. Given a document collection D and an individual document $d \in D$, and let $V=\{k_1, \dots, k_N\}$ be a list of keywords of a given document d . Then the keyword weights (w_1, \dots, w_N) of a document d is calculated: $w_i = tf_i(d) * idf_i$. Where $tf_i(d)$, called *term frequency*, equals the frequency of keyword k_i in the document d , and idf_i , called *inverse document frequency*, is computed as $idf_i = \log_2(|D|/df_i)$, where $|D|$ is the size of documents in the collection, and df_i equals the number of documents in which keywords k_i appears in D . (Hayes et al., 2006; Ramos, 2003; Salton & Buckley, 1988)

BM25. The BM25 is a probabilistic model that incorporates attributes of documents, such as term frequencies, document frequencies, and document length (Robertson and Walker, 1994; Sparck

Jones et al., 2000). It ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document (e.g., their relative proximity) (Amati, 2003; Robertson and Walker, 1994). Moreover, it includes the document length as an explicit random variable of the probability space in normalizing the term-frequency (Amati, 2003).

PL2 and IFB2. PL2 and IFB2 are two of Divergence From Randomness (DFR) models. In the DFR models, the term-weight is inversely related to the probability of term-frequency within the document obtained by a model of randomness. PL2 uses the Poisson estimation for randomness in combination with the Laplace succession for the computation of the information-gain with a term within a document and normalizing the term-frequency by taking the document length into consideration. IFB2 uses the Inverse Term Frequency model for randomness in combination with the ratio of two Bernoulli's processes for computing the information-gain and normalizing the term-frequency by considering the document length. (Amati, 2003; DFR framwork, 2011; He and Ounis, 2005)

DLH. The DLH model is a generalization of the parameter-free hyper-geometric DFR model in a binomial case (Amati, 2006; He and Ounis, 2007). It considers that the occurrences of a query term in a document are samples from the whole collection instead of from the document alone. Furthermore, it does not contain a term frequency normalization. Neither does it have any parameters that require relevance tuning because all the variables of the document weighting formula are automatically set from the collection statistics (He and Ounis, 2007).

By adopting different weighting methods, these IR techniques fundamentally extract from a collection a subset of sections that are deemed relevant to a given query. They then assign a similarity score ($0 \leq$ similarity score ≤ 1) to each retrieved section based on frequency and distribution of key words in the query (Amati, 2003; Antoniol, Canfora et al., 2002; Antoniol, Casazza et al., 2000; Cleland-Huang et al., 2005; Hayes et al., 2003; Lucia et al., 2007; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009). This can result in some accurate links having a very low similarity score. The lower the cut point that is used, the more possible links are retrieved. However, more incorrect links are captured as well. In other words, at a high cut point,

IR captures few links with few positive links. These IR models possess the same limitation as VSM in low precision at low cut points and low recall at high cut points. Certain IR models such as LSI may solve the synonym issue in text retrieval by producing positive similarity between related artifacts sharing no terms (Marcus and Maletic, 2003). Nevertheless, IR models still fail to retrieve some links when class names do not follow the naming convention strategy or documents use different words to describe related classes.

3.3.2 Regular Expression (RE)

In order for us to augment the number of retrieved links at high cut points, a RE technique is used. A regular expression, which is a pattern of characters that describes a set of strings, is constructed and used to find all of the occurrences of this pattern in an input sequence. Here, we use REs to find class names in documents. This technique is case sensitive because the first letter of every class name is generally capitalized.

Class names can be placed into two groups. One group is class names containing only one word, such as Control, Main, Graphics etc. Another is class names formed by compound words, such as NamingExceptionEvent, DragSource etc. For the second group, the class names are most likely not to be part of common words that can be found in a dictionary. Therefore, once they appear in documents, most likely they represent class names. For the first group, class names probably belong to common words. Then we need to make sure the same words found in documents indicate class names and not other names.

For the second group, simply matching class names against their occurrence in documents suffices. From inspection of typical documents, we observe that class names can be surrounded by a wide variety of non-word characters but must exclude the hyphen “-”. A hyphen attached before or after a class name can be part of another class name. For example, the string “DragSource” matches a class named “DragSource”, but also a class name is written as “DragSource-Listener” in documents when a class name is separated over two lines and is connected by a hyphen: “DragSource-“ is at the end of a line, “Listener” is at the beginning of the following line. It raises another issue that hyphens may exist inside class names, e.g. “DragSource-Listener”. Therefore,

we extend the regular expressions developed by Bacchelli et al. (2009; 2010) to the following regular expression code (take the class named “Control” for the example):

$$(.*)(^a-zA-Z0-9\>)-<C-?o-?n-?t-?r-?o-?l>(^a-zA-Z0-9\>)(.*)$$

In order to identify class names in the first group, we can additionally match different parts of the package name of a class in documents. For example, a package named “javax.naming.event” has three parts: javax, naming, event. It is not feasible to require the package name to be presented before the class name, because it is very rare that a package name is cited before the class name in documents. If the class name, the last part of the package name, and at least one of other parts of the package name are found, then the single word in documents denote a class name. This method also can apply to identify classes sharing the same name but belonging to two different packages. The regular expression code for matching each part of package names is (take the package named “javax.naming.event” for the example):

$$\begin{aligned} & (.*)(^a-zA-Z0-9\>)-<\text{javax}>(^a-zA-Z0-9\>)(.*) \\ & (.*)(^a-zA-Z0-9\>)-<\text{naming}>(^a-zA-Z0-9\>)(.*) \\ & (.*)(^a-zA-Z0-9\>)-<\text{event}>(^a-zA-Z0-9\>)(*) \end{aligned}$$

These two regular expressions can correctly capture all documents directly containing class names and return few unrelated documents. Therefore, links recovered by RE are considered as links that are most likely to be true or correct. They are assigned with the highest similarity value (=1). This largely expands the retrieved link sets at high cut points but does not change the incorrect links recovered by an IR model. This approach still fails to retrieve links that are missed by an IR model.

Both TM, discussed in Chapter 2, and RE can fulfill class name entity recognition. We found through experimentation that the results obtained from both approaches were the same in the case study, JDK1.5-SUBSET (discussed in Chapter 4). Both TM and RE retrieved 665 links for JDK1.5-SUBSET. However, TM spent double the time to capture links. Moreover, combining TM into our traceability recovery system made the whole system much slower than RE. Therefore, we

chose to use RE rather than more sophisticated TM techniques in our current tool.

3.3.3 Key Phrases (KP)

Key Phrases provide a brief summary of a document's content (Witten et al., 1999). We use the KP technique to extract key words (or key phrases) from comments of code to provide a brief summary of each class's description comment and use these to augment our IR technique's link recovery.

In Section 3.3.1, we discussed that an IR query is built by using the OR operator to combine the class name and its separated words if it is formed by compound words. This means that queries probably contain unhelpful terms if programmers use unrelated words to name classes, or probably miss useful terms if the class name fails to reflect the purpose of the class. We hypothesize that extending the IR queries with some helpful terms can assist IR techniques in extracting more useful links. Then the question is: where and how to find helpful terms that are related to classes?

There are two situations where IR is unable to retrieve correct links. Firstly, when class names do not follow a naming convention strategy, IR struggles to retrieve documents that do not explicitly mention the class name. For example, for a class named "RefAddr", its IR query is "RefAddr OR ref addr OR ref OR addr", IR is unable to retrieve documents not containing "RefAddr" as "ref" and "addr" are not common words. Secondly, documents implicitly mentioning a class but not explicitly using the same word as the class name or separated words of the compounded class name are also problematic. For example, a class is named "Media", but documents may use "medium" to indicate this class. We have found that these two issues can be addressed by taking the comments in source code into consideration.

Generally, software developers provide comments to describe the purpose of the class or what tasks the class fulfills. Extracting key phrases from comments can help find alternative words to the class name or words indicating what tasks the class fulfills. For example, "medium" indicates the class "Media", "reference address" refers to the purpose of the class "RefAddr". As long as comments in each class are well written, KP can extract all possible key phrases that summarize

the purpose of each class. We found that adding these extracted key phrases to the IR queries enables our approach to work in the above two contexts. However, many incorrect links at low cut points are also recovered.

3.3.4 Clustering

In general, every document has an inherent hierarchical structure. Documents are usually divided into sections with headings. Each section has a direct parent or some direct children or some siblings. There exist tangled relationships between these sections. For example, in this Chapter, “Section 3.3.1 The Basic Retrieval Approach” has a direct parent, “Section 3.3 IRETrace -- Information Retrieval enhancement traceability”, and three siblings, “Sections 3.3.2, 3.3.3, and 3.3.4”. It has no children. Section 3.3.1, 3.3.2, 3.3.3, and 3.3.4 cross-reference each other to some extent. We take advantage of these tangled relationships to reduce the number of incorrect links retrieved by using Clustering.

Clustering is a division of a set of objects into groups of similar objects: clusters (MacQueen, 1967). We modified the K-mean clustering algorithm (MacQueen, 1967) to meet our needs. There are three main steps in this: initialization, assignment, and removal. Before implementing the Clustering technique, all retrieved links are grouped based on classes; that is, links related to the same class are grouped together. Clustering is performed on each group that represents sections related to the same class. Figure 3.1 illustrates the sequence diagram of clustering links in each group.

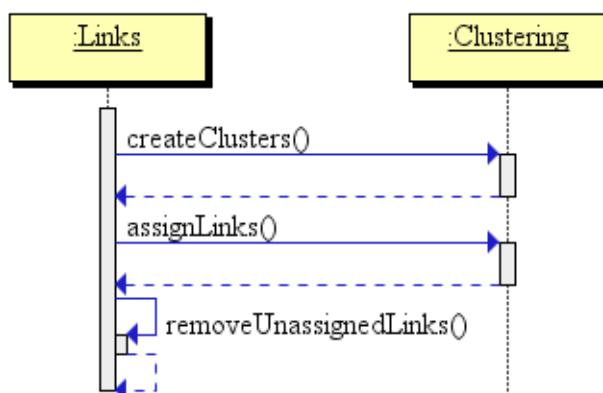


FIGURE 3.1 THE SEQUENCE DIAGRAM OF CLUSTERING RETRIEVED LINKS OF A CLASS

The first step is to create clusters in each group; the algorithm selects k clusters according to the number of links with similarity values $\geq s$. Each cluster contains one of these related sections. When the group contains links with a similarity value that equals 1, the algorithm uses $s = 1$. Otherwise, the algorithm uses $s = 0.3$ to create clusters. From empirical observation we found three reasons to use this latter value when none of the links' similarity value in the group is equal to 1. Firstly, a majority of incorrect links have a similarity score ≤ 0.3 . Links with similarity > 0.3 are more likely to be true. Secondly, if we use $s \leq 0.3$, our approach retrieves many incorrect links and only slightly more true links. Thirdly, if $s \geq 0.3$, our approach slightly decreases the number of incorrect links but does not obtain more true links.

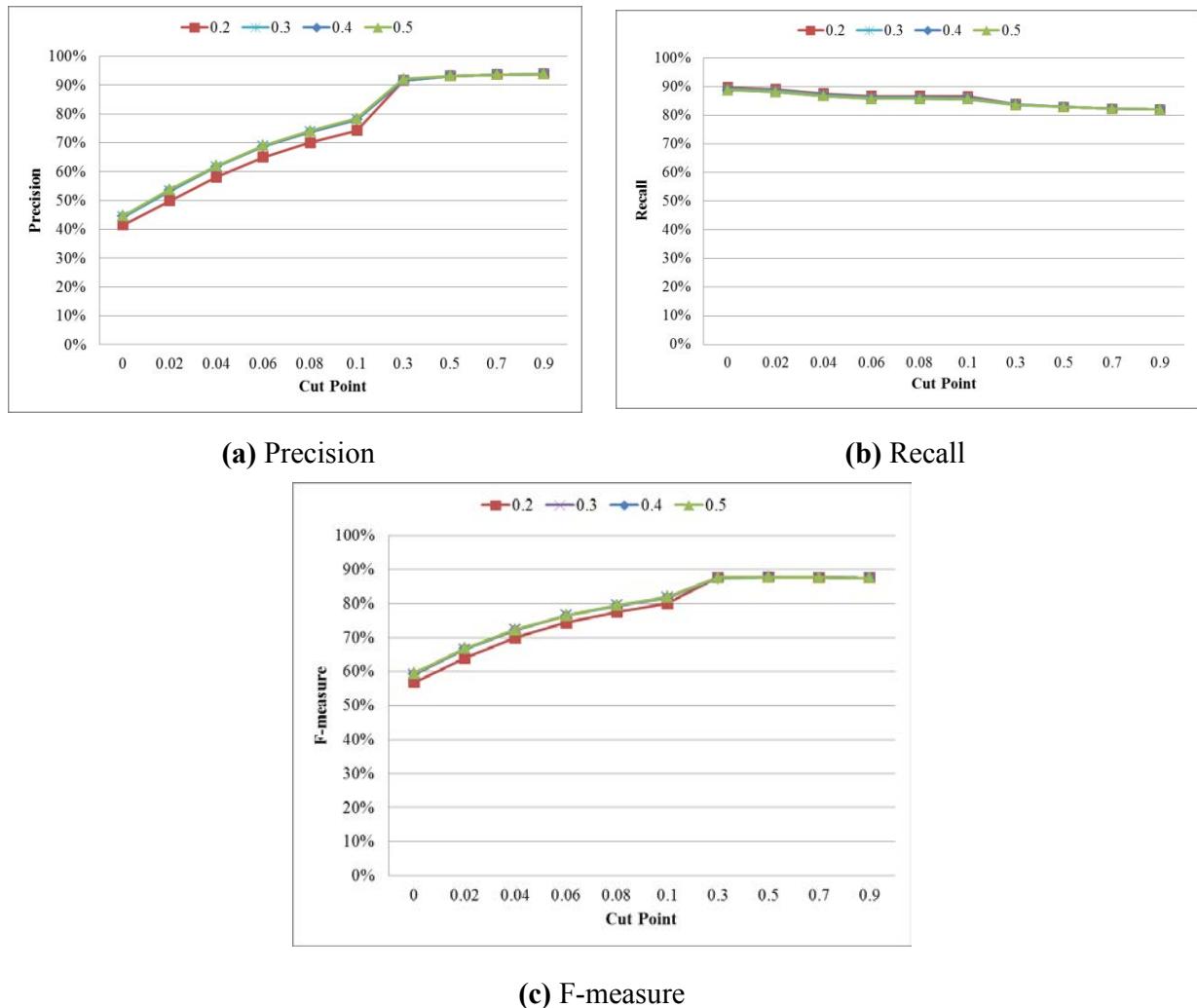


FIGURE 3.2 PRECISION, RECALL, AND F-MEASURE RESULTS WHEN $s = 0.2, 0.3, 0.4$, OR 0.5 IN JDK1.5-SUBSET

Figure 3.2 shows the precision, recall and F-measure results when using different s values ($s=0.2$,

0.3, 0.4 or 0.5) to create clusters for the JDK1.5-SUBSET case (discussed in Chapter 4). When using $s = 0.3, 0.4$ or 0.5 , there is no obvious difference for the precision, recall, and F-measure values. However, precision and F-measure are decreased and recall has no obvious improvement if using $s = 0.2$. Empirically, therefore, we found $s = 0.3$ to be the best choice for groups that do not contain links with a similarity score = 1 in our experiment. We need to conduct more experiments, however, to validate its suitability for other systems.

Next, in the assignment step, the algorithm assigns the rest of links in each group to the created clusters. Before the assignment, the inherent hierarchical information about the direct parent, all direct children and all siblings of the initial section in each cluster are required. This information can be acquired by looking through the document near where the initial section is. Based on the inherent hierarchical information, a remaining link is assigned to a cluster subject to the following: (1) only if its related section is a direct parent or a direct child or a sibling of the initial section to the cluster, (2) it is not already included in other clusters, and (3) it is in the retrieved link set. Finally, the removal step can be implemented by discarding any link that is not assigned to any cluster.

We take a group containing links between the class “java.awt.dnd.DragSource” and sections in documents in the JDK1.5-SUBSET case (discussed in Chapter 4) as an example to illustrate our clustering algorithm. Table 3.2 shows 34 sections in the “dnd1.pdf” document (PDF Version of JDK Documentation, 2010) are related to “DragSource”. Each line represents a link. The first 15 links have similarity value =1. Lines coloured blue and italicized refer to true links, i.e. links of No. 1-15, 17, and 19 are true links. Other links are not correct, i.e. they are fault links.

In the first step, initialization, as this group has links with similarity value = 1, we use $s = 1$ to create clusters. 15 clusters are created because 15 links (from No. 1 to 15) have a similarity score = 1. Each cluster contains a related section. For example, for link No. 1, as its similarity value is 1, it is considered to be a cluster, which contains the related section 2.1.

TABLE 3.2 SECTIONS RELATED TO JAVA.AWT.DND.DRAGSOURCE

| No. | Similarity Score | File Name | Section | Status |
|-----|------------------|-----------|---|--------|
| 1 | 1.0 | dnd1.pdf | 2.1 Overview | True |
| 2 | 1.0 | dnd1.pdf | 2.2.1 DragGestureRecognizer | True |
| 3 | 1.0 | dnd1.pdf | 2.3 Drag Source | True |
| 4 | 1.0 | dnd1.pdf | 2.3.1 The DragSource definition | True |
| 5 | 1.0 | dnd1.pdf | 2.3.2 The DragSourceContext Definition | True |
| 6 | 1.0 | dnd1.pdf | 2.3.5 The DragSourceDragEvent Definition | True |
| 7 | 1.0 | dnd1.pdf | 2.3.6 The DragSourceDropEvent Definition | True |
| 8 | 1.0 | dnd1.pdf | 2.4.3 The DropTargetContext Definition | True |
| 9 | 1.0 | dnd1.pdf | 2.4.4 The DropTargetListener Definition | True |
| 10 | 1.0 | dnd1.pdf | 2.4.5 The DropTargetDragEvent and ... | True |
| 11 | 1.0 | dnd1.pdf | 2.5 Data Transfer Phase | True |
| 12 | 1.0 | dnd1.pdf | 2.5.1 FlavorMap and SystemFlavorMap | True |
| 13 | 1.0 | dnd1.pdf | 2.5.2 Transferring Data across the JVM ... | True |
| 14 | 1.0 | dnd1.pdf | 3.0.1 What are the implications of the ... | True |
| 15 | 1.0 | dnd1.pdf | 3.0.3 Lifetime of the Transferable(s)? | True |
| 16 | 0.0623 | dnd1.pdf | 3.0.4 Implications of ACTION_... | Fault |
| 17 | 0.0590 | dnd1.pdf | 2.3.3 The DragSourceListener Definition | True |
| 18 | 0.0548 | dnd1.pdf | 2.5.3 Transferring lists of files across... | Fault |
| 19 | 0.0506 | dnd1.pdf | 2.3.4 The DragSourceEvent Definition | True |
| 20 | 0.0443 | dnd1.pdf | 3.0.2 Inter/Intra VM transfers? | Fault |
| 21 | 0.0418 | dnd1.pdf | 3.0.5 Semantics of ACTION_... | Fault |
| 22 | 0.0308 | dnd1.pdf | 2.4.1 java.awt.Component additions... | Fault |
| 23 | 0.0299 | dnd1.pdf | 2.5.4 Transferring java.rmi.Remote ... | Fault |
| 24 | 0.0279 | dnd1.pdf | 3.0 Issues | Fault |
| 25 | 0.0279 | dnd1.pdf | 2.0 API | Fault |
| 26 | 0.0266 | dnd1.pdf | 2.4.2 The DropTarget Definition | Fault |
| 27 | 0.0216 | dnd1.pdf | 1.1 Provision of a platform independent | Fault |
| 28 | 0.0193 | dnd1.pdf | 2.2 Drag Gesture Recognition | Fault |
| 29 | 0.0133 | dnd1.pdf | 1.2 Integration with platform ... | Fault |
| 30 | 0.0133 | dnd1.pdf | Appendix A : DropTargetPeer definition | Fault |
| 31 | 0.0132 | dnd1.pdf | 1.0 Requirements | Fault |
| 32 | 0.0097 | dnd1.pdf | 2.4.6 Autoscrolling support | Fault |
| 33 | 0.0006 | dnd1.pdf | Appendix B : DragSourceContextPeer definition | Fault |
| 34 | 0.0006 | dnd1.pdf | Appendix C : DropTargetContextPeer definition | Fault |

In the assignment step, we assign the remaining links to the 15 clusters based on the inherent hierarchical information about the initial section in each cluster. This information can be obtained by examining the “dnd1.pdf” document. Take the cluster for section 2.1 in Table 3.2 for example (i.e. the first line, link No. 1). Section 2.1 in the “dnd1.pdf” document has a parent, section 2.0, no direct children, and four siblings, sections 2.2, 2.3, 2.4, and 2.5. As sections 2.3 and 2.5 (link Numbers 3 and 11) are initially considered to be clusters, these two sections cannot be assigned to this cluster. Sections 2.0 and 2.2 can be assigned to this cluster only if they do not belong to other clusters. Section 2.4 is not assigned as it is not in the retrieved link set. After the assignment, 13 remaining links (No. 16-26, 28, and 32) are assigned to the 15 clusters. These contain 2 true links and 11 fault links. Lines coloured red (bold) and blue (italics) indicate links included in the 15 clusters.

In the final removal step, 6 links out of 34 are discarded in the group for “DragSource”, i.e. link Numbers 27, 29-31, and 33-34 are removed as they are not in any cluster, and so are regarded as fault/incorrect links (which, in this case, they are). Thus, after the clustering, 6 fault links are discarded. We have found that our clustering approach eliminates many fault links at low cut points.

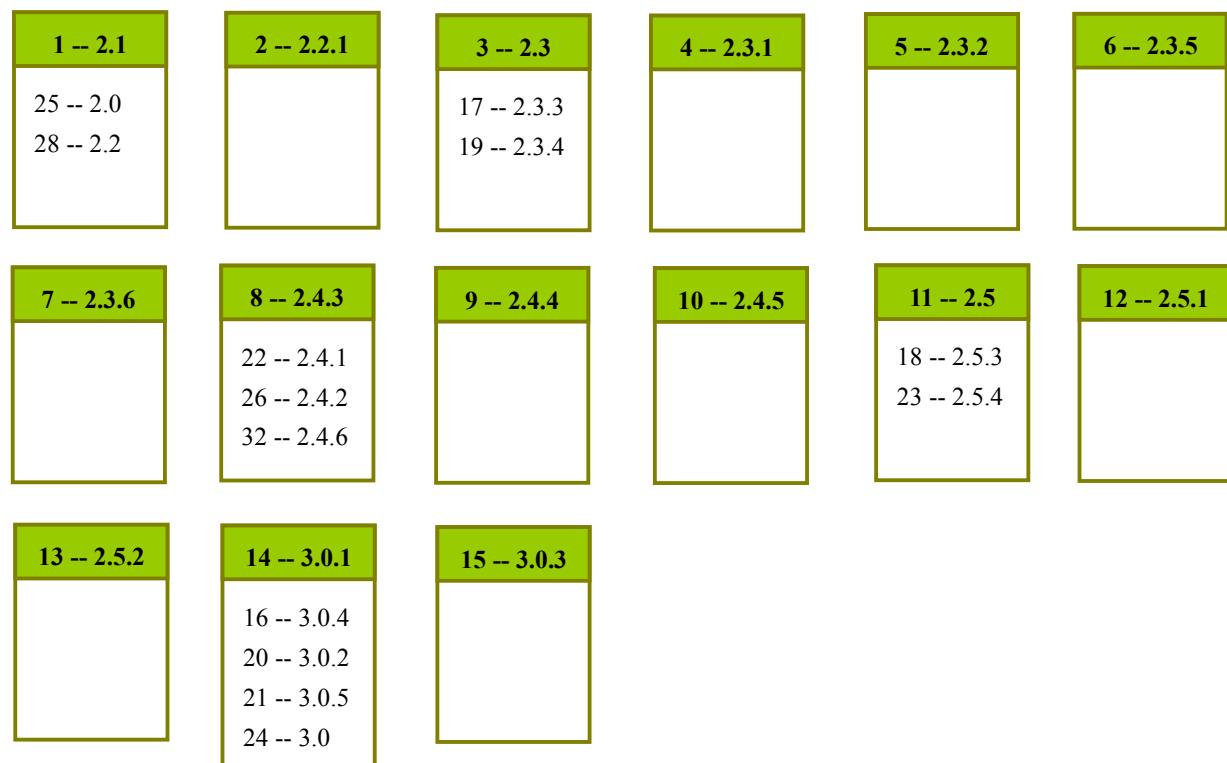


FIGURE 3.3 CLUSTERING LINKS OF JAVA.AWT.DND.DRAGSOURCE

Figure 3.3 illustrates how to cluster links of “DragSource”. Here, each rectangle box represents a named cluster and a list of assigned links. Take the first cluster as an example. Its name is “1 -- 2.1”, “1” referring to the line number in Table 3.2, and “2.1” indicates the related section number. Two sections are assigned to this cluster, “25 -- 2.0” and “28 -- 2.2”, “25” and “28” are the line numbers in Table 3.2, and “2.0” and “2.2” are the assigned section numbers. It is clear in Figure 3.2 that 15 clusters are created. The rest of links are assigned to five clusters, “1 -- 2.1”, “3 -- 2.3”, “8 -- 2.4.3”, “11 -- 2.5”, and “14 -- 3.0.1”. Those links that are not assigned to one of the 15 clusters are removed.

3.4 Implementation

Figure 3.4 illustrates the complete traceability recovery process of our approach. First, if a document contains sections, it is partitioned into small sub-documents according to sections or headings (1). For example, if a PDF document contains 10 headings, it is split into 10 sub-documents; the contents of each are the text between its heading and the following one. These sub-documents are then preprocessed.

Next, source code is analyzed by a code dependency analysis system in order to extract source code identifiers (every class, method, package name), and comments inside code (2). Code dependency analysis is based on Eclipse’s JDT Java parser (Eclipse Java Development Tools (JDT), 2010). These extracted class names are passed to the Regular Expression (RE) processor. Based on the two regular expressions described in Section 3.3.2, the RE processor finds sections that directly mention class names (3). Links retrieved by the RE processor are assigned the highest similarity score (= 1), and form the RE link set. Hence, the number of retrieved links at high cut points is largely increased.

At the same time, extracted comments inside code are passed to the Key phrases extraction system (4). This is based on KEA (KEA: keyphrase extraction algorithm, 2010), an automatic keyphrase extraction algorithm developed by Witten et al. (1999). KEA extracts candidate key phrases using lexical methods, computes feature values for each candidate, and adopts a machine-learning

algorithm to predict which candidates are good key phrases (Witten et al., 1999). We employ KEA to extract key phrases from comments that are likely to characterize them. IR techniques may extract more useful links by including these extracted key phrases because they may contain meaningful terms. These extracted key phrases are combined with extracted class names to form IR queries (5). A query string for IR is established by using OR operators to combine the class name, the separated words if the class name is formed by compound words, and key phrases extracted from comments in the class code.

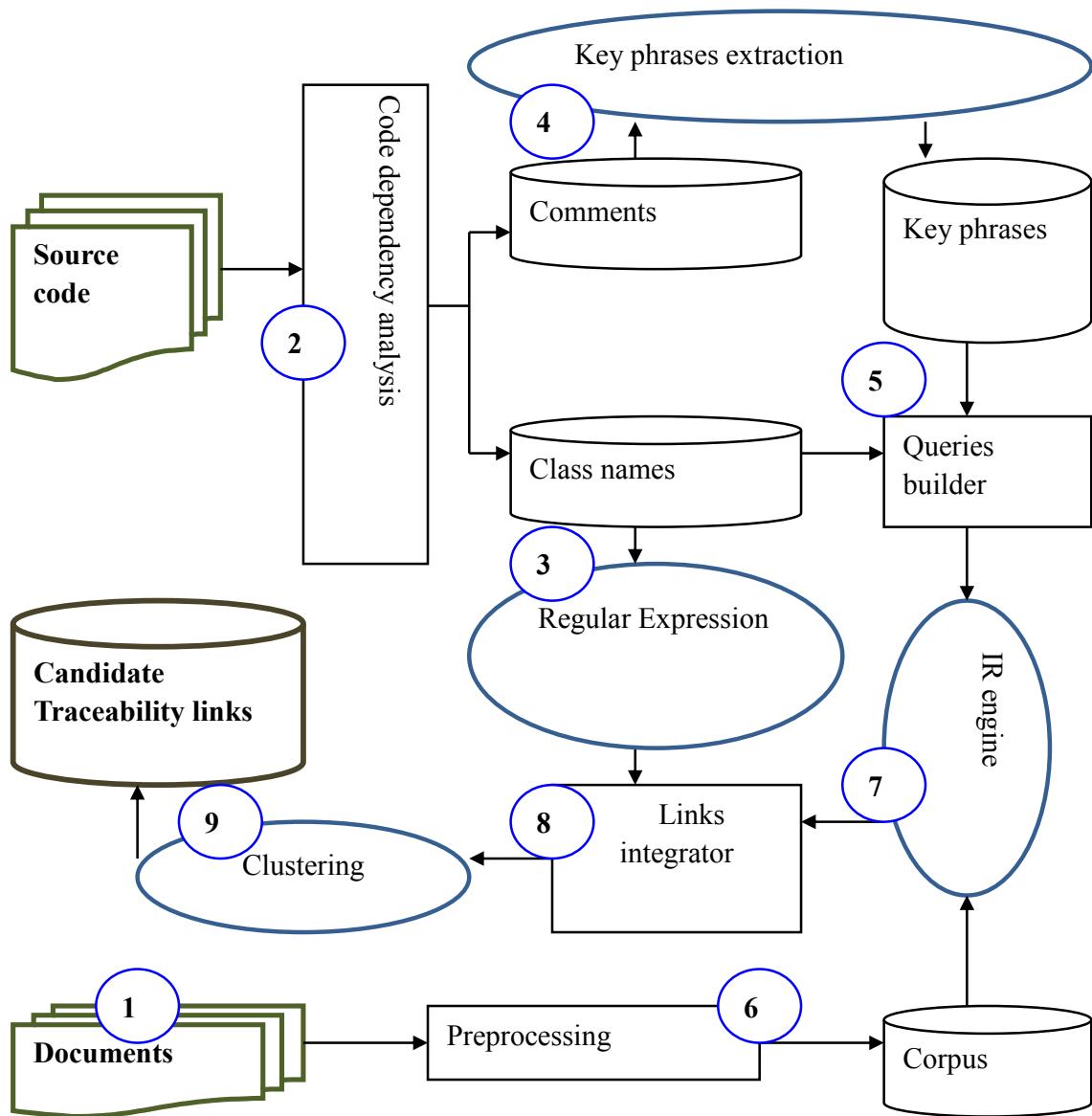


FIGURE 3.4 TRACEABILITY RECOVERY PROCESS OF IRETRACE

Before using an IR engine to capture links between sections and class entities, sections in the documents are preprocessed (6). Their preprocessing starts by generating tokens from consecutive

letters in the text stream according to token boundaries that are defined at non-letter characters. Next, non-textual tokens (i.e. special symbols, numbers etc.) are dropped. A lower case filter transforms all capital letters into lower case letters, and a stop-words filter removes common words (i.e. articles, adverbs, etc.). Finally, an IR corpus is generated containing all documents and words (or tokens) in the documents. The IR engine retrieves traceability links according to queries, and computes similarity scores ($0 \leq \text{similarity score} \leq 1$) based on the frequency and distribution of the key words or phrases (7). We use two IR engines: the Apache Lucene IR engine (Apache Lucene – Overview, 2009) and Terrier (Terrier IR platform, 2010). Lucene employs VSM capturing traceability links. Terrier provides a number of IR models to satisfy different needs in text retrieval: TF-IDF, BM25, PL2, IFB2, DLH and so on. We discuss the experimental results of using different IR models in Chapter 5. Recovered links forms the IR link set. The RE link set and the IR link set are then merged (8). If a link can be found in both sets, then the one in the IR set is removed and we leave the link in the RE set (i.e. with higher rank). Finally, the merged link set passes through the Clustering system to refine the link set to produce the final candidate traceability links (9). The Clustering system clusters retrieved links based on the inherent hierarchical information contained in the traced documents. Many unrelated links can be discarded through the Clustering system.

We built a system to implement and evaluate our traceability recovery technique, IRETrace, which automatically extracts relationships between classes and sections in documents. This system is seamlessly integrated with the Eclipse Integrated Development Environment (IDE) to provide users with both IDE and traceability support. The current version of this system only traces relationships in software projects that are written in Java. The overall architecture of IRETrace is shown in Figure 3.5.

Before starting to capture traceability links in a project, the project including source code and documents needs to be imported into Eclipse. Source code and documents of the project are preprocessed to obtain class identifiers and sections in documents. Candidate traceability links between classes and sections are then retrieved through using our recovery approach (the traceability recovery process described in Figure 3.4). In order to conduct a comparison of our recovery approach with other approaches, we used a filter to support different selections of

applying different combination approaches to recover links, e.g. use IR only, the combination of IR and RE (IR+RE), the combination of IR, RE, and KP (IR+RE+KP), or the combination of IR, RE, KP, and Clustering (IR+RE+KP+Clustering). In other words, candidate traceability links can be captured using different combinations. Finally, these candidate traceability links are stored in a table-format file and are displayed using a graph, which is implemented using Zest (Zest, 2009), and a matrix.

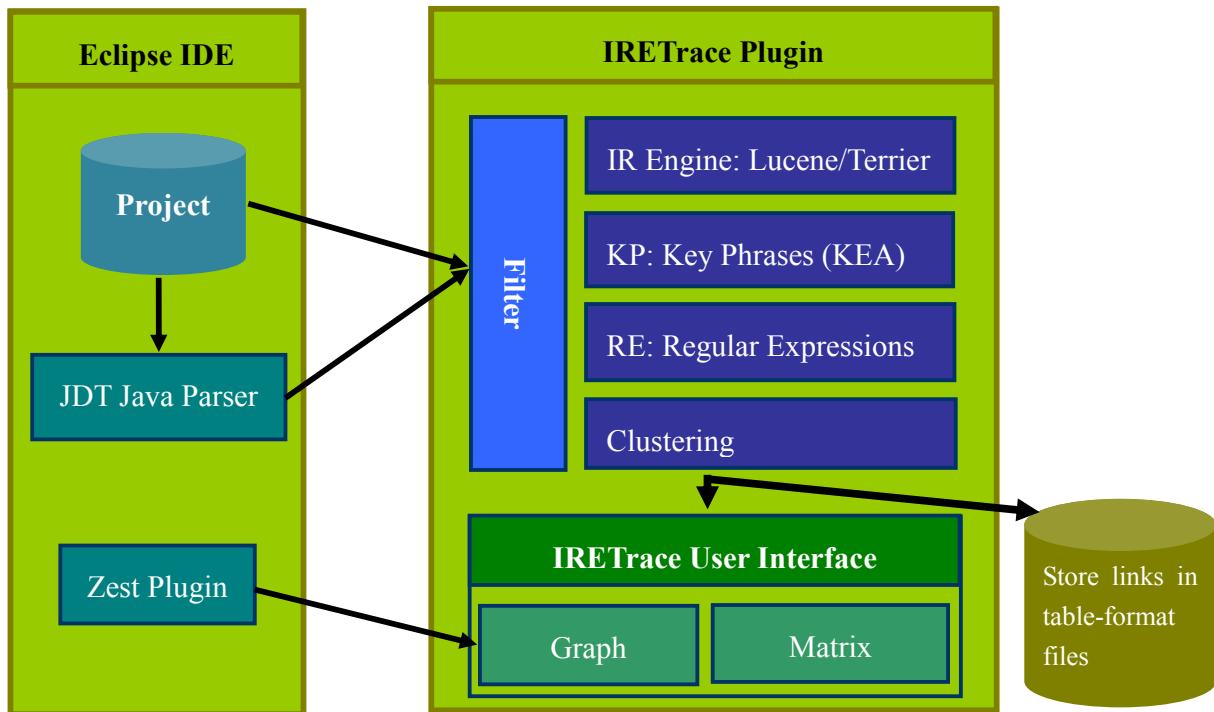


FIGURE 3.5 ARCHITECTURE OF IRETRACE

Figure 3.6 shows the user interface of the IRETrace 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 in a tree. We used Zest to visualize extracted links. Zest, the Eclipse Visualization toolkit, is built for Eclipse to represent graphs (Zest, 2009). As shown in Figure 3.6, 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 (for example, “PrintJob.java”), all adjacent 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 stored in an internal matrix table.

The columns of the table are classes in the project, the rows are headings of PDF files. Cells are similarity scores generated by IR engine.

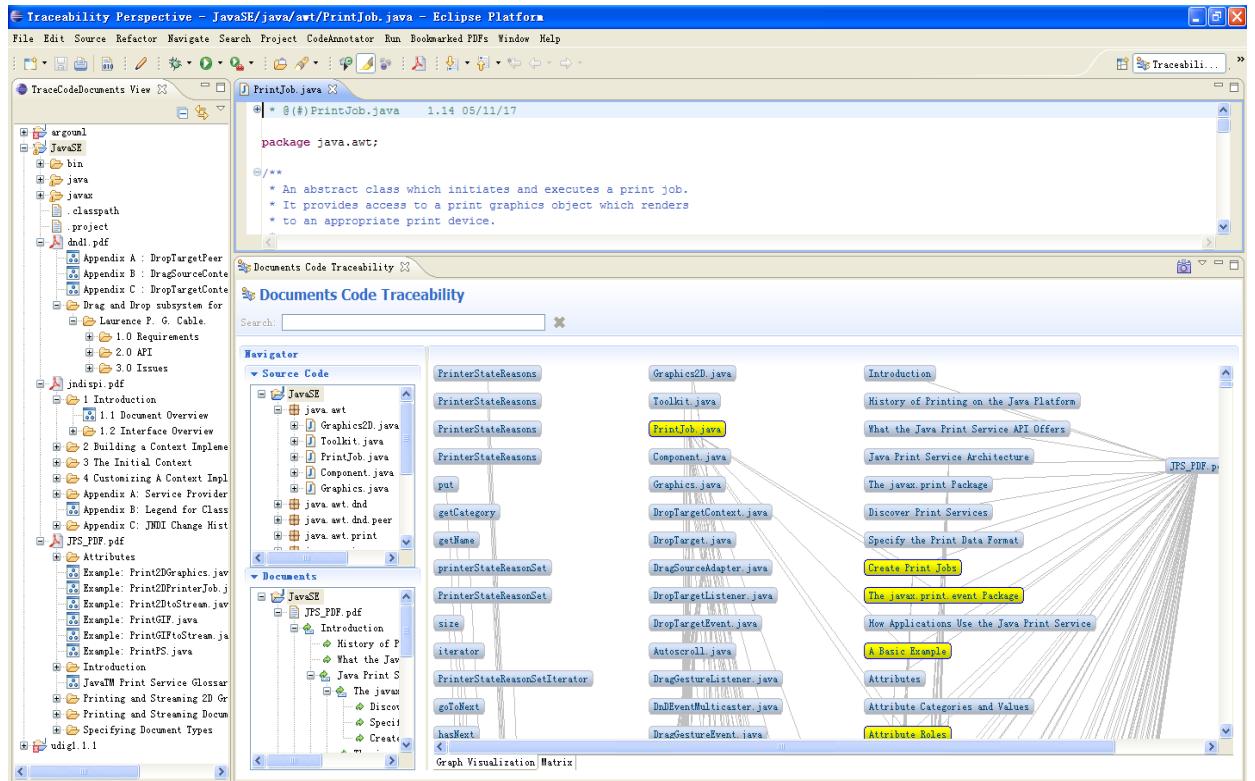


FIGURE 3.6 THE USER INTERFACE OF IRETRACE

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 related only to the selected item. For example, when “PrintJob.java” is selected, all related nodes are displayed. Although the “Navigator” assists users in locating a specific node in the graph, the graph view cannot fully support users to maintain or interpret links easily and conveniently. Moreover, if the number of artifacts in the system becomes large, the graph view easily becomes overcrowded, which impedes a user’s ability to browse and maintain links. In addition, links automatically retrieved by IRETrace may contain incorrect links or miss some correct links. So these retrieved links need to be analyzed to delete incorrect links or to add missing links. A challenge then arises: how to visualize retrieved links efficiently and effectively to help users browse, maintain, and comprehend links in a system. This challenge is discussed in Chapter 6.

3.5 Summary

It is a major challenge to design and implement traceability recovery techniques that extract relationships among diverse artifacts of a software system at high-levels of precision and recall. Many recovery techniques exist but none has so far been able to produce sufficiently consistent and high enough quality results that software developers require. We have developed IRETrace, an Information Retrieval Enhancement Traceability system for the extraction of traceability links between class entities and sections in documents. IRETrace incorporates three supporting techniques, RE, KP, and Clustering, with an IR model to recover traceability links. The three techniques ameliorate the key limitations of IR by taking advantage of the respective strengths of each of the three enhancement techniques.

In order to evaluate the performance of IRETrace, we set up four case studies based on four unrelated software systems: JDK1.5-SUBSET, ArgoUML, Freenet, and JMeter. For JDK1.5-SUBSET, we manually established its traceability benchmark: this is discussed in Chapter 4. For the other systems we used established traceability oracle sets. The experimental results are presented in Chapter 5. To make the extracted traceability links “useful” for maintainers of software systems, our final step is to visualize recovered links allowing users to browse and maintain these links in a natural and intuitive way. We use a hierarchical, graphical traceability link visualization to enable users to interact with large numbers of extracted relationships. Our traceability link visualization tool is discussed in Chapter 6.

Chapter 4-- Traceability Benchmark

Establishing appropriate traceability benchmarks to evaluate a traceability link recovery technique against prior studies is the second challenge we face in our software traceability research. This chapter describes a new, robust approach that we have developed and trialed for researchers to build traceability benchmarks easily and effectively.

4.1 Introduction

It is well recognized that rigorous software traceability technology plays a critical role in the software development process (Antoniol et al, 2002; Antoniol, Casazza et al, 2000; Cleland-Huang et al., 2006; Seacord et al, 2003). Such support helps developers to assess the completeness of an implementation against all stated requirements, to identify reusable software components, to support impact analysis while changes occur, and to comprehend and maintain software systems (Antoniol et al., 2002; Antoniol, Casazza et al., 2000; Seacord et al., 2003). In the last decade, researchers have put great effort into inventing new traceability recovery techniques (see Chapter 2), either automatic or semi-automatic, to retrieve traceability links in a software system with as many correct links and as few incorrect links as possible. An outstanding research question is how to examine the performance of a traceability recovery technique, validate the technique, and compare its performance with other recovery techniques. An essential element for the recovery technique evaluation process is the use of effective traceability benchmarks (Cleland-Huang et al., 2011; Dekhtyar et al., 2007; Sim et al., 2003). A traceability benchmark serves as a basis for evaluation of one technique and the comparison of multiple techniques (Cleland-Huang et al., 2011). Without such a benchmark, whether a link generated by the traceability recovery technique is correct or not remains uncertain, and whether any correct links are missed is unclear. Without

such data, the performance of a traceability recovery technique cannot be measured or compared to the performance of others. In addition, improvements to the recovery technique cannot be determined or quantified. However, it is a major challenge for researchers to obtain or establish meaningful and robust traceability benchmarks to evaluate recovery approaches due to the difficulty of obtaining or creating such benchmarks (Cleland-Huang et al., 2006).

Traceability benchmarks can be acquired or developed in three ways. First, a traceability benchmark for a project can be built by project developers during the software development process. This type of benchmark is the most attractive due to its apparent reliability and accuracy. Unfortunately in practice, traceability within the software development process has seldom been employed in most organizations. This is due to its high cost, complexity, time-consuming nature, and error-proneness (Gotel and Finkelstein, 1994; Ramesh and Jarke, 2001; Rilling et al., 2007). This results in there being almost no publicly available projects containing very robust traceability benchmarks (Charrada et al., 2011). Second, we can evaluate a new traceability recovery technique with benchmarks developed by other researchers and/or applied by them to other software projects. However, a major problem with this approach is that different traceability recovery techniques often address different issues and are applied to different software artifacts. This results in limited ability to adapt or utilize traceability benchmarks from the work of others (Dekhtyar et al., 2007). For example, a benchmark that was built to include relationships between requirements and design documents, while very useful for testing traceability techniques between these types of software artifacts, is not at all suitable to evaluate a traceability technique that captures links between source code and documentation or between source code and unit tests. Third, we can establish our own benchmarks to meet our specific traceability recovery technique evaluation needs. Due to the well-known difficulty of obtaining or using the above two types of traceability benchmarks, researchers usually create their own benchmarks to conduct evaluations of their new traceability recovery techniques (Charrada et al., 2011; Cleland-Huang et al., 2006; Dekhtyar et al., 2007; Hayes et al., 2006). Nevertheless, a major issue arises here: how do we actually go about building affordable, meaningful, and robust traceability benchmarks? Most research to date has focused on the theory, basic principles, or the establishment of all-inclusive traceability benchmarks (Charrada et al., 2011; Dekhtyar et al., 2007). Unfortunately, there are no generally agreed or applied approaches or guidelines that have been proposed to assist researchers in manually building robust

traceability benchmarks for identifying and verifying links in a system.

In this chapter, we propose a new approach and guidelines to help researchers who want to develop their own traceability benchmarks and use these benchmarks to facilitate evaluation and comparison of their own and others. The main objective of a traceability recovery technique is to trace relationships between artifacts in a software system. A new recovery technique is normally evaluated by comparing the set of its retrieved traceability links with an “oracle” i.e. the known, true traceability link set of the system, in order to compute its precision, recall, or F-measure. The oracle traceability link set of a system consists of all its actual correct, or true, traceability links. Hence, the crucial part of the development of traceability benchmarks is manually finding and verifying correct links between artifacts. We have designed and tested rigorous, manual identification and verification strategies to assist researchers to capture links and verify them. In our approach, every link is analyzed by at least two analysts before the determination of its status is made i.e. whether it is a correct/true or an incorrect/false link. We propose a formula to calculate the probability of errors (e.g. 5% or 10% erroneous links) in the created oracle link set. We have employed this approach in our own research in order to create a very robust benchmark for JDK1.5-SUBSET. This benchmark includes source code and documents produced during the software development process. We employed the formula to compute the probabilities of 5% and 10% errors in the JDK1.5-SUBSET oracle link set. The result shows that the actual probability of there being $\geq 5\%$ errors links in this oracle link set is very low, around 0.12%. Moreover, our error probability calculation shows that our rigorous manual identification and verification strategies can significantly reduce the error probability in the oracle link set. In addition, the visit to the set of traceability benchmark requirements identified by Dekhtyar et al. (2007) shows that our approach can produce benchmarks that satisfy these requirements.

This chapter is organized as follows. We firstly provide a background to traceability benchmarks and the motivation for this research. We introduce our new approach to establishing a traceability benchmark and a formula to calculate the probability of errors in a benchmark. Next we describe a benchmark that we developed using our new approach. We then carry out an evaluation of it against a set of requirements for traceability benchmarks and compute the probability of 5% or 10% errors in our created JDK1.5-SUBSET oracle link set. We also analyze the cost-quality

tradeoffs in establishing a robust benchmark. Finally, we discuss the limitations of our approach.

4.2 Background and Motivation

Sim et al. (2003) defined a benchmark in software engineering as a standard test or set of tests employed to compare the performance of alternative tools or techniques. They claimed that successful benchmarks must meet seven requirements: accessibility, affordability, clarity, relevance, solvability, portability, and scalability. Oppenheimer et al. (2003) claimed that benchmarks provide researchers with an approach to quantify design tradeoffs and a yardstick to measure and inspire progress. Because of the importance of benchmarks in the evaluation process, Cleland-Huang et al. (2006) identified building traceability benchmarks as one of the key challenge areas in their Grand Challenges in Traceability.

Few papers have been published to tackle this challenge in the software traceability area. In the Grand Challenges in Traceability, a traceability benchmark is defined in terms of a traceability task, a set of data sets on which the task is to be performed, and finally a set of metrics that will be used to evaluate the task (Cleland-Huang et al., 2006). Dekhtyar et al. (2007) established the basic principle of organization of traceability benchmarks, which comprises five components: dataset, tasks, measures, answer sets, and data representation format (the last is optional). To be successful, traceability benchmarks need to satisfy five requirements: support for traceability in multiple fields of software engineering, independence of methodology, ground truth, accuracy testing, and scalability testing (Dekhtyar et al., 2007). According to this principle, Charrada et al. (2011) proposed a traceability benchmark that includes nine types of artifacts and with end-to-end traceability links. This benchmark is developed based on AquaLush, an irrigation system. A visual experimental workbench, named TraceLab, was developed for designing and executing traceability experiments to help new researchers to establish research environments or help existing researchers to perform more rigorous evaluations and become more productive in their work (Cleland-Huang et al., 2011).

Although researchers have put some effort into building meaningful traceability benchmarks, there

are three barriers impeding the realization of this challenge. The first barrier is the lack of publicly available projects that include traceability links. In general, open source projects exclude traceability links between artifacts or contain only partial traceability. In most cases, the traceability benchmarks of commercial projects are confidential. Moreover, it is difficult to accomplish traceability in practice because tracing and maintaining inter-relationships among artifacts is arduous, time-consuming, error-prone, and costly (Gotel and Finkelstein, 1994; Ramesh and Jarke, 2001; Rilling et al., 2007). These issues make it difficult to acquire publicly available, robust traceability benchmarks (Charrada et al., 2011; Cleland-Huang et al., 2011).

The second challenge is the diversity of traceability issues tackled by traceability recovery techniques. Most traceability recovery techniques address specific traceability problems because of the researchers' expertise or project funding (Dekhtyar et al., 2007). For example, Antoniol, Canfora et al. (2002) used Probabilistic Model (PM) and Vector Space Model (VSM) approaches to recover relationships between code and documentation to assist software maintainers. Marcus and Maletic (2003) introduced Latent Semantic Indexing (LSI) to improve the performance of Information Retrieval (IR) models. Hayes et al. (2006) aimed to improve the requirements tracing process for independent verification and validation analysts. Besides targeting different traceability issues, traceability techniques utilize different artifacts to examine and evaluate their performance. For instance, Antoniol, Canfora et al. (2002) and Marcus and Maletic (2003) focused on tracing links between code and system documentation. Hayes et al. (2006) recovered links between requirements and design specifications. Bacchelli et al. (2010) sought relationships between code and emails. Moreover, researchers need to build tools implementing their traceability recovery techniques in order to perform the evaluation. These tools may accept limited programming languages. For example, the tool for our own composite traceability recovery technique, discussed in Chapter 3, only traces relationships in software projects that are written in Java. In addition, the language used to write comments and documents generated during the software development process affects the possibility of the adaptation or employment by other traceability techniques. For example, Documents in Albergate utilized by Antoniol, Canfora et al. (2002) and Marcus and Maletic (2003) are written in Italian. For researchers who are not familiar with Italian, it is difficult to process these documents. These issues lead to difficulty when adapting or applying traceability benchmarks from the work of others.

The third barrier is the difficulty of manually establishing robust traceability benchmarks (Charrada et al., 2011; Hayes et al., 2006). Traceability recovery techniques mainly involve tracing relationships between artifacts in a system (Dekhtyar et al., 2007). Hence, the most important part of a traceability benchmark during evaluation and comparison of traceability recovery techniques is the oracle, or true traceability link set, which is a set of correct/true links between artifacts. Nevertheless, manually tracing relationships from one artifact to another is arduous, time-consuming, and error-prone. Due to the first two barriers identified above, researchers often have to develop their own traceability benchmarks to meet their specific needs. For example, Hayes et al. (2006) built a traceability benchmark for the CM-1 data set to evaluate the tracing between requirements and design documents. They used a group of analysts to manually verify links retrieved by RETRO (Hayes et al., 2007), a special-purpose tool designed exclusively for requirements tracing. Bacchelli et al. (2009, 2010) established five traceability benchmarks (using ArgoUML, Freenet, JMeter, Mina, and OpenJPA) to target traceability links between code and emails. They manually annotated classes mentioned in emails and verified these annotations with a group of six participants. There are three key issues that emerge while manually creating traceability benchmarks: how to find an appropriate dataset, how to manually identify correct links between artifacts, and how to verify links to be correct or incorrect. The three issues have rarely been touched on in the software traceability community. Moreover, there are currently no guidelines or approaches that have been proposed to assist researchers to develop meaningful and robust traceability benchmarks. These challenges motivated us to develop an approach to establish affordable, meaningful, and robust traceability benchmarks easily and effectively.

4.3 Traceability Benchmark Development

As mentioned earlier, Dekhtyar et al. (2007) defined a traceability benchmark as consisting of five main components: dataset, tasks, answer sets, measures, and (optionally) data representation format. The dataset of a traceability benchmark is the collection of artifacts within a project. Tracing tasks address objectives of traceability techniques. Tasks determine the selection of artifacts. For example, if a task is to retrieve traceability links between documentation and source code, then the selected artifacts should be source code and documents, such as development guides,

design, requirements etc. Each task has an answer set. For the task of identifying traceability links between artifacts, the answer is the set of correct/true links that relate one artifact to another. This is also called the oracle/true traceability link set. Measures indicate the metrics used to evaluate the effectiveness and efficiency of traceability techniques. For example, precision, recall, and F-measure are commonly utilized to measure the accuracy and coverage of traceability link recovery techniques. (See Section 4.3.5 for definitions of these.) Time is a standard measure for scalability to quantify the time needed to execute a task. Data representation format refers to the format used to store the benchmark data. For example, the benchmark data can be stored in a XML document, a matrix or a table.

We employ this definition to design our traceability benchmarks. In our research, we are concerned with the evaluation of traceability link recovery techniques. These generally deal with the issue of the recovery of traceability links between artifacts. Hence, we define a traceability benchmark to include tasks, dataset, oracle/true traceability link sets, and measures. We replace answer sets with oracle/true traceability link sets to satisfy our concerns. Furthermore, in light of the work of Dekhytar et al. (2007), we propose five steps to establish a traceability benchmark: task identification, artifact selection, project selection, oracle/true traceability link set development, and evaluation metrics. The first step, task identification, identifies the set of traceability tasks. In the second step, artifact selection chooses which artifacts to collect based on the tasks. Third, project selection finds appropriate projects containing artifacts that are chosen in step two. Fourth, oracle/true traceability link set development manually establishes correct links between selected artifacts. The final step, evaluation metrics, defines metrics that can measure the performance of a traceability recovery technique, which is evaluated by comparing the created oracle link set with the link set retrieved by the recovery technique. The following sections describe these five steps in detail.

4.3.1 Task Identification

The first step for building a traceability benchmark is to address what tasks the benchmark has been developed to accomplish. This depends upon what issues the traceability recovery technique under evaluation is concentrating on. For example, some traceability recovery techniques focus on

tracing links between requirements and design documents while others aim to find relationships between source code and documents. The tasks must reflect these issues. For instance, if the evaluated traceability recovery technique aims to trace links between source code and documentation, then the task is to recover traceability links between code and documents.

4.3.2 Artifact Selection

Based on the tasks, the second step is to choose appropriate artifacts. If one of the tasks is the recovery of traceability links between source code and documentation, then the dataset should be the collection of source code files and documents produced during the software development process. To decide on what kinds of documents to select relies on their availability in the selected project. If the selected project provides only requirements and design documents, then we can only trace relationships between code and requirements and/or design documents. Furthermore, using a particular document detail level e.g. the class or method level for code, and the section or paragraph level for documents, is decided by the particular traceability recovery technique that needs to be evaluated. If the traceability technique retrieves links between classes and sections in documents, then the artifact dataset includes classes and sections of documents.

4.3.3 Project Selection

The next step is to search for appropriate projects from which to obtain artifacts to analyze. An appropriate project possesses four properties.

- The first property is that the project must include artifacts that are chosen in step two. For example, if the artifacts selected in step two are requirements and design documents, then the selected project must contain the two types of documents.
- The second property is that the project should be of a reasonable size. The larger the size of the project, the more resources, time, and cost are required to manually build traceability links. If the benchmark builder has limited resources, then one of the feasible ways is to choose small projects or to use only a part of larger projects. For instance, if a project contains a large number of requirements and design documents, a feasible way to build traceability links is to

choose a part of the requirements and their corresponding design documents.

- The third property is that the programming language used for programming the project is acceptable to the tool that implements the traceability recovery technique under evaluation. Some such tools have no requirements on programming languages. Hence, the chosen projects can be written in any programming language. But others may only accept certain programming languages, such as Java, C#, or C++ etc. This requires that the chosen projects must be written in a language that can be accepted by the tool. For example, our traceability tool, IRETrace, currently only accepts projects written in Java.
- The last property is that the documents and comments in the project must be written in a language that can be understood by any participant involved in the benchmark development. In other words, if the documents and comments in a project are written in French, participants who do not understand French cannot identify links in this project.

4.3.4 Oracle/True Traceability Link Set Development

After selecting the appropriate projects and artifacts, we can start building the oracle traceability link set. From the outset, we need to decide the source artifact and the target artifact. All links are bidirectional, so defining the source and target artifacts is principally to help users to more easily identify links. The source artifact should be uncomplicated, or one that can be easily divided into sets or groups. For example, when recovering links between source code and documents, source code is normally used as the source artifact. However, Bacchelli et al. (2009) used emails as the source artifact because their source code includes many versions of ArgoUML. For links between requirements and design documents, the source artifact is the requirements. Next, traceability rules need to be set up to facilitate the identification and verification of traceability links. We apply the rule defined by Charrada et al. (2011) as fundamental: an element A and an element B are related if B is derived from A or if B provides additional and useful information about A. This rule can be extended to satisfy specific concerns. For example, for defining links between code and documents, this rule can be changed to: a class A in source code and a section B in documents are related if B directly mentions A's name or identifier, or if B describes tasks that A should fulfill. Then we can start manually identifying and verifying links between selected artifacts in the selected project.

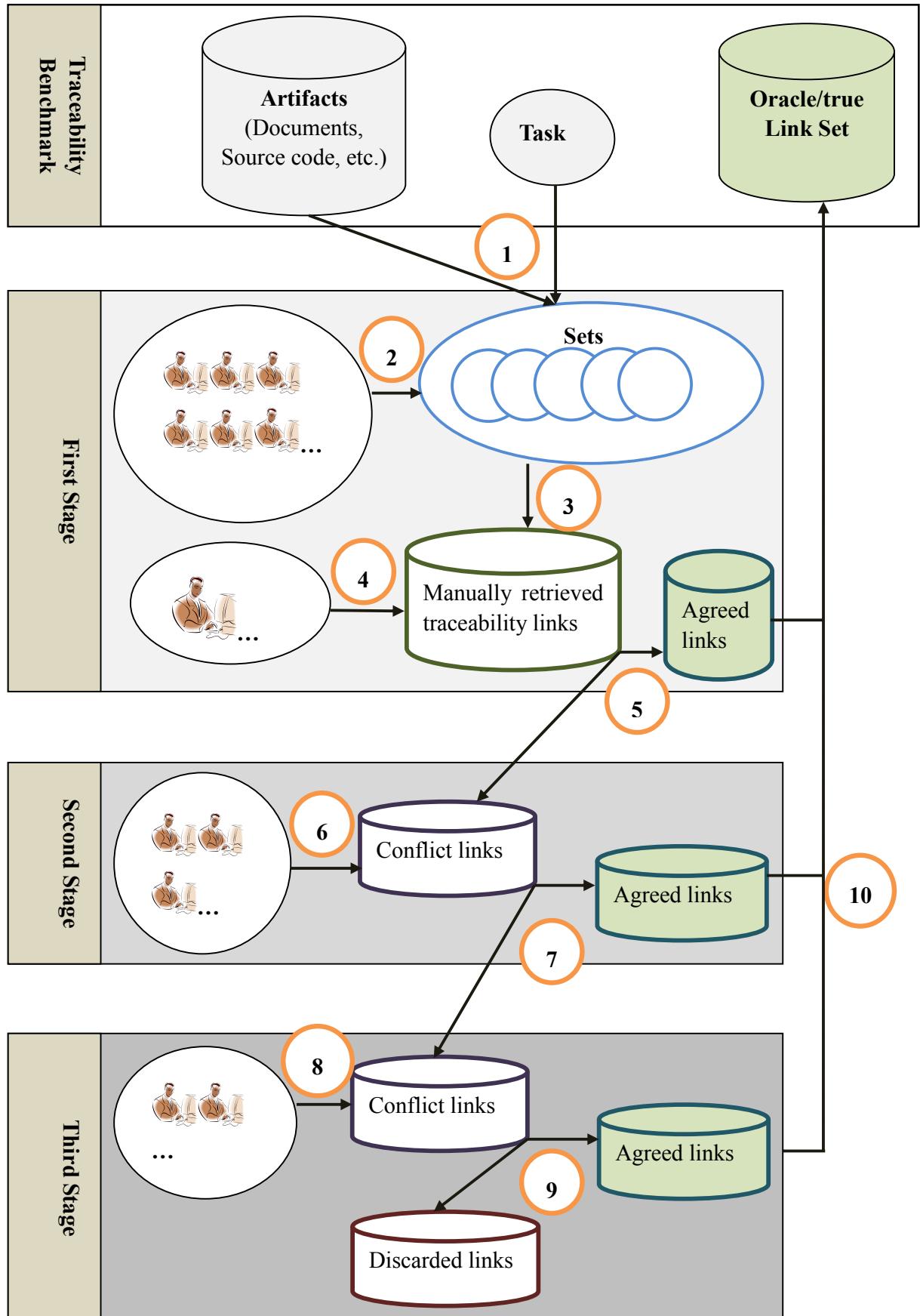


FIGURE 4.1 THE STRATEGY OF AN ORACLE TRACEABILITY LINK SET DEVELOPMENT

Figure 4.1 illustrates our strategy for oracle link set development. Our traceability link identification and verification strategies are inspired by the works of Bacchelli et al. (2009) and Hayes et al. (2006), who established their own traceability benchmarks through manually verifying traceability links by a group of participants. Our rigorous manual identification and verification of the true links is designed to remedy any potential bias that could add incorrect links to the oracle link set. Our strategies include three stages. We take the traceability between classes and sections in documents as our example.

At the first stage, the source artifact is divided into overlapping sets (1). Each element of the source artifact is at least assigned to two different sets. The number of sets depends on how much workload is assigned to each participant. The less the workload allocated to each participant, the smaller number of elements each set contains, and the more participants that need to be used. For example, we might use the class artifact as the source artifact and split all collected classes into 6 groups, each of which contains a certain number of classes. Next, according to the number of the sets, a group of participants needs to be recruited. These participants are required to have at least some knowledge about the selected project. Then every participant is allocated a set (2). They then manually identify traceability links between the artifact in the set and another selected artifact based on the traceability rule mentioned above (3). In our example, 6 junior participants might be employed; each having a background of Java programming experience. Each is assigned a set and is required to find links between classes in the set and sections in the documentation based on the traceability rule.

After these participants complete their tasks, another group of participants with good understanding of the selected project is recruited to verify these retrieved links based on the traceability rule (4). The link verification aims to identify the status of each retrieved link, whether or not it is true/correct, and to capture links missed by the previous participants. The number of participants still depends on how much workload is needed for each participant. If it is a reasonable workload for an analyst to verify all retrieved links, then using one senior analyst is enough. Otherwise, these retrieved links are split into overlapping sets. Every participant is responsible for one set. If participants all consider a link is true, then this link is in the set of agreed links; otherwise it is in the set of conflict links (5). For example, a class is assigned to two

participants in the first group and two participants in the second group. If the four participants all identify the same link related to the class, then this link is considered as an agreed link and is put into the agreed link set. Otherwise, this link is considered as a conflict link and is in the conflict link set as it is not unanimously recovered by the four participants. After the first stage, an agreed link set is generated. Each link in this set is agreed to be true by all participants who are allocated the source artifact of the link.

At the second stage, the set of conflict links generated at the first stage is randomly divided into overlapping sets, the number of which is based on how much workload each participant needs to undertake. A new group of participants is recruited based on the number of overlapping sets. Each participant is assigned a set to verify the conflict links in the set by carefully studying the content in the selected artifacts (6). At this stage, every conflict link is analyzed by two participants as each conflict link is assigned to two participants. If a link is considered to be a true link by the two participants, then this link is added to the set of agreed links, otherwise, it is a conflict link (7). In this example, every participant is required to carefully study the text of sections and the comments inside code before making the decisions. After the second stage, an agreed link set is produced. Each link in this set is agreed to be true by two participants.

At the third stage, a senior analyst is employed to verify the set of conflict links produced at the second stage (8). This analyst carefully learns the content of the selected artifacts and also consults another senior analyst to determine whether or not a conflict link is correct. In this example, the senior analyst needs to carefully learn the content of sections and the comments in classes. Each conflict link is analyzed by at least three participants, who have either identified the link at the first stage or have verified the link at the second or third stages. When three or more participants agree that a conflict link is not correct, this link is considered to be incorrect and is discarded; otherwise it is considered to be a true link (9). The third stage creates an agreed link set, in which every link is agreed to be true by at least three participants.

Finally, the three agreed link sets produced at the three stages are merged to form the oracle traceability link set for the selected artifacts in the selected project. In other words, the oracle link set is composed of three parts: the agreed link set produced at the first stage, the agreed link set

generated at the second stage, and the agreed link set confirmed at the third stage. At the first stage, a link in the agreed link set is simultaneously captured by all participants who are allocated the source artifact of the link. At the second stage, a link in the agreed link set is unanimously considered to be true by two participants who are assigned to verify this link. At the third stage, a link in the agreed link set is agreed to be true by at least three participants who have identified or verified it in the three stages. These links should be stored in a file (e.g. a XML document, a table, or a matrix) to facilitate researchers to use and interpret them.

Probability of errors

After the establishment of the oracle link set for a project, an issue arises: what is the probability of errors (e.g. 1%, 5%, or 10% errors) in this oracle link set? To analyze this probability, we build a formula to calculate the error probability based on the following three assumptions:

- 1) We assume that the probability of an error being made is dependent on the type of participant and stage. For example, we might make the following assumptions:
 - a. Junior analysts have an error probability of 20%, i.e. during link recovery, a junior analyst produces 20% incorrect links.
 - b. Senior analysts have an error probability of 10%.
 - c. If a senior analyst consults with another senior analyst, this senior analyst has an error probability of 5%; thus, he/she retrieves 5% incorrect links.
- 2) We assume that links are independent. The recovery of a link does not affect the probability of the recovery of any other link. Moreover, the status (i.e. correct or incorrect) of any individual link does not affect the probability of recovering any other link and the probability of the status of any other retrieved link. For the sake of simplicity, we assume that every link has the same likelihood of being retrieved.
- 3) We assume that errors made on links are independent; so an error made on any individual link does not affect the probabilities of errors on other links. Errors here include the errors of judging an actual correct link to be an incorrect link, or judging an actual incorrect link to be a correct link, or failing to recover an actual correct link. The probability of making an error on a link does not influence the probability of an error made on any other link.

The oracle link set comprises three parts: the agreed link set produced at the first stage, the agreed link set produced at the second stage, and the agreed link set produced at the third stage. These agreed link sets are independent. The probability of errors in the oracle link set is defined as:

$$\Pr[E] = \frac{\sum_{i=1}^k (n_i \times \Pr[e_i])}{N}, \text{ where}$$

1. N is the size of the oracle link set, i.e. the total number of links in the oracle link set.
2. k is the total number of stages, here an oracle link set is established through three stages ($k=3$).
3. n_i is the size of the agreed link set at the i^{th} stage, i.e. the number of agreed links at the i^{th} stage.
4. $\Pr[e_i]$ is the probability of errors in the agreed link set of the i^{th} stage. It is defined as:

$$\Pr[e_i] = \frac{\sum_{j=1}^{m_i} \Pr(x_j)}{m_i} \times \Pr(y_i), \text{ where}$$

- a. m_i is the number of participants at the i^{th} stage.
- b. $\Pr(x_j)$ is the probability of incorrect links captured by the j^{th} participant at the i^{th} stage.
- c. $\Pr(y_i)$ is the probability of participants simultaneously making the same mistake. For example, at the first stage, a class is assigned to three participants, all participants generate the same link for this class, and then this link goes to the agreed link set. We treat this link as a true link. But if this link is actually an incorrect link, $\Pr(y_i)$ represents the probability of the three participants all making the same mistake in capturing the incorrect link at the same time at the first stage.
- d. Errors in $\Pr(x_j)$ and $\Pr(y_i)$ are distributed in the binomial distribution. We use the following binomial probability formula (Triola, 1997) to compute $\Pr(x_j)$ and $\Pr(y_i)$.

$$P(x) = \frac{n!}{(n-x)!x!} \times p^x \times q^{n-x} \quad \text{for } x = 0, 1, 2, \dots, n, \text{ where}$$

- i. n = number of trials.
 - ❖ For $\Pr(x_j)$, n = the number of agreed links retrieved by j^{th} participant at the i^{th} stage;
 - ❖ For $\Pr(y_i)$, n = the number of participants who identify a link in the agreed link

set at the i^{th} stage.

ii. x = number of successes among n trials.

- ❖ For $Pr(x_j)$, $x = x_j$ = the number of errors (e.g. 5% or 10% errors) in the agreed link set retrieved by j^{th} participant at the i^{th} stage;
- ❖ For $Pr(y_i)$, $x = y_i$ refers to at least x participants who consider a link is true at the i^{th} stage. In other words, x = the number of participants who identify or verify a link in the agreed link set at the i^{th} stage.

iii. p = probability of success in any one trial.

- ❖ For $Pr(x_j)$, $p = 0.2$ (20%) for junior analysts, $p = 0.1$ (10%) for senior analysts, $p = 0.05$ (5%) for a senior analyst consulting another senior analyst.
- ❖ For $Pr(y_i)$, p = the average/mean error probability of n participants.

iv. q = probability of failure in any one trial ($q = 1 - p$)

The error probability of links ($Pr[E]$) in the oracle link set depends on the error probability ($Pr[e_i]$) of links in each agreed link set generated at each stage. The error probability ($Pr[e_i]$) of links at each stage largely relies on each participant's error probability ($Pr(x_j)$) and the probability of several participants ($\geq y_i$) making the same mistake ($Pr(y_i)$). In Section 4.5.2, we represent an example of calculating the error probability of links in the JDK1.5-SUBSET oracle link set. There, we initially use the 20%, 10%, and 5% error probability for the three types of analysts introduced as example assumptions above. Having demonstrated the method, we then estimate the actual average error rates for the three types of analysts based on our observations and results of the establishment of the JDK1.5-SUBSET oracle link set, and then derive more realistic estimates of the link set containing errors.

4.3.5 Evaluation Metrics

After developing the oracle traceability link set, we can perform an evaluation and comparison of the traceability recovery technique under study to other techniques. The common metrics used in the evaluation of traceability recovery techniques are precision, recall, and F-measure. These three

metrics depend on three figures: correct (or true) links retrieved, incorrect (or fault) links retrieved, and missing links.

Correct links retrieved are those that are correctly captured by the traceability recovery technique. Fault links are those that are wrongly detected by the traceability recovery technique. Total links retrieved combines these two kinds of links. Relationships that are not found by the traceability recovery technique are called missing links. Total correct links are the sum of correct links retrieved and missing links. Precision can be defined as the ratio of the number of correct retrieved links over the total number of retrieved links. If precision equals 1, it means that all the recovered links are correct, though there could be correct links that were not recovered.

$$\text{Precision} = \frac{\text{Correct links retrieved}}{\text{Total links retrieved}}$$

Recall is the ratio of the number of correct retrieved links over the total number of correct links. Recall = 1 indicates that all correct links are recovered, but there may be incorrect recovered links.

$$\text{Recall} = \frac{\text{Correct links retrieved}}{\text{Total correct links}}$$

The F-measure combines precision and recall based on their weighted harmonic mean to measure the effectiveness of retrieval. β is an adjustable weight to favour precision over recall. $\beta=1$ weights precision and recall equally. $\beta=2$ weights recall twice as much as precision. $\beta=0.5$ weights precision twice as much as recall.

$$F\text{-measure} = \frac{(\beta^2 + 1)\text{Precision} \times \text{Recall}}{(\beta^2\text{Precision}) + \text{Recall}}$$

Two sets of traceability links between selected artifacts are prepared in order to compute precision, recall, and F-measure. One set is produced by a traceability recovery system under evaluation. The other set is the oracle traceability link set for the selected artifacts. The latter is critical as it is a crucial factor in determining the number of correct and missing links. Comparison of the two sets is then conducted to determine whether a link is correct, incorrect, or missing.

4.4 Case Study

To validate the effectiveness of our approach, we have set up a case study to build a traceability benchmark for the evaluation of our traceability recovery technique introduced in Chapter 3. This benchmark comprises four components: tasks, dataset, oracle/true traceability link set, and measures.

Tasks. Our particular traceability recovery technique aims to capture traceability links between source code and documentation. Hence, the task is to trace links between code and documents.

Dataset. According to the task, the dataset is the collection of source code and documents. As our traceability recovery technique goes down to the class level in code and section level in documents, the dataset consists of classes of source code and sections of documents. The system we used for our benchmark is JDK 1.5, a free open source software system for Java developers. We chose to use three packages (java.awt, javax.naming, and javax.print) from the JDK1.5 source code and their associated documentation. (We call this case as JDK1.5-SUBSET) These three packages were chosen because of the availability and detail of corresponding natural language documents describing these parts of the system. Three documents (PDF Version of JDK Documentation, 2010) explain in detail the structure of packages. For example, JPS_PDF.pdf describes how the Java printing support works and which functions are implemented by which Java classes in the javax.print package.

TABLE 4.1 JDK1.5-SUBSET PACKAGES AND DOCUMENTS

| JDK 1.5-SUBSET | | File size | #classes/sections |
|----------------------|--|-----------|-------------------|
| Java packages | java.awt, javax.naming, and javax.print packages | 2.02 MB | 249 |
| PDF files | <i>JPS_PDF.pdf</i> : Java™ Print Service API User Guide | 505 KB | 68 |
| | <i>dnd1.pdf</i> : Drag and Drop subsystem for the Java Foundation Classes | 160 KB | 41 |
| | <i>jndispi.pdf</i> : Java Naming and Directory Interface™ Service Provider Interface(JNDI SPI) | 284 KB | 73 |
| | Total sections: | | 182 |

Table 4.1 describes the packages in JDK1.5-SUBSET and their corresponding PDF documents

used in this study, as well as the number of Java classes and the number of sections in them. We divided these PDF files into sections based on their headings. For example, if a PDF document contains 10 headings, it is split into 10 sections; the contents of each are the text between one heading and the following one. We obtained 249 Java classes and 182 sections (or small documents). The traceability task becomes that of extracting relationships between these 182 sections and the 249 Java classes in JDK1.5-SUBSET.

Oracle/true traceability link set. In order to build the oracle traceability link set for JDK1.5-SUBSET, we recruited 11 analysts: 9 analysts had at least 6 years of Java programming experience, and 2 participants had more than 9 years of Java programming experience. They include 6 Females and 5 Males. We set the class artifact as the source artifact because classes are easier than sections to group. We also set up our traceability rules to assist participants in finding and verifying a link. First, if a section directly mentions a class identifier or name, then this section is related to this class. The second rule is that if a section describes tasks that a class should fulfill, then they are related.

At the first stage, the classes were divided into 6 sets. Six participants then manually retrieved links between sections in documents and classes by following the above two rules. After they completed their task, we asked a senior participant to conduct link verification that included verifying these retrieved links and capturing links missed by them. At the end of the first stage, 408 links were identified as conflict links, and 356 links were included in the agreed link set; each link was agreed to be true by two participants.

At the second stage, 408 conflict links produced at the first stage were randomly divided into 3 overlapping sets. Three other participants verified these conflict links by carefully studying the text of documents and the comments inside code. At the end of the second stage, 75 conflict links were identified, and 333 links were included in the agreed link set; each link was verified to be true by two participants.

At the third stage, we asked a senior participant to verify those links still having conflicts. This participant carefully studied the text of documents and the comments in code. This participant also

consulted another senior participant. Each conflict link was thus analyzed by at least 3 participants. When three or more reviewers agreed that the conflict link was a fault, we considered this link to be an incorrect link and discarded it. At the end of the third stage, 4 links were considered as conflict links, and 71 links were added into the agreed link set; each link was identified to be true by at least three participants. The final oracle link set comprised 760 true links, which we then stored in a table. 110 out of 249 classes had no sections related to them.

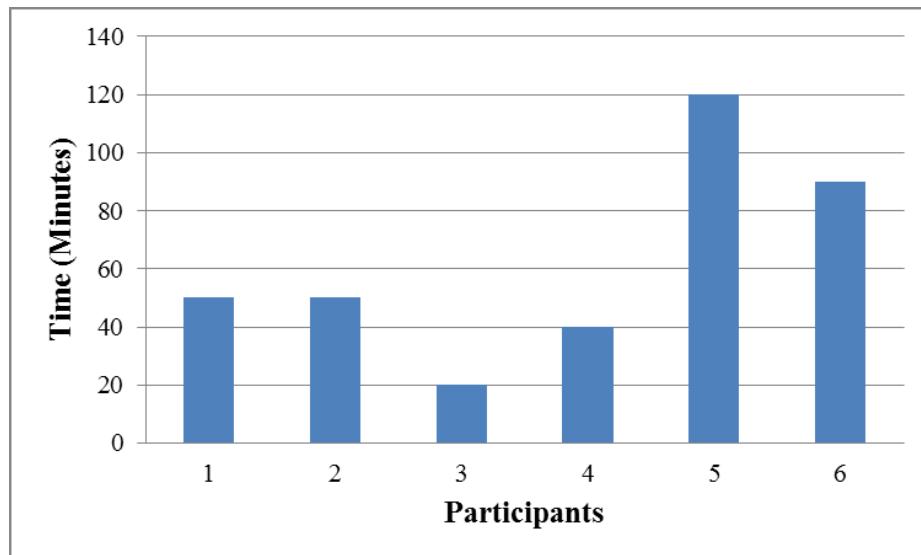


FIGURE 4.2 TIME TAKEN BY THE FIRST SIX PARTICIPANTS IN CAPTURING LINKS AT THE FIRST STAGE

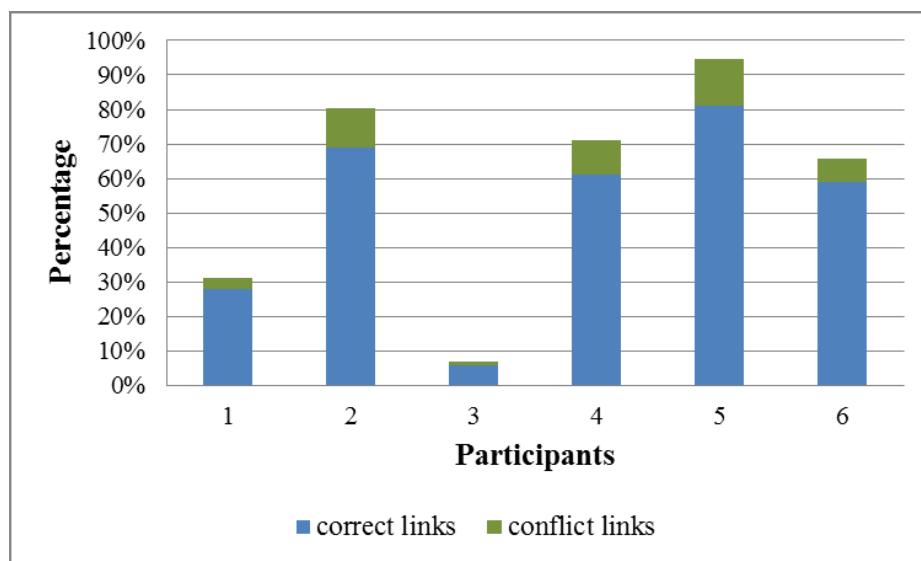


FIGURE 4.3 PERCENTAGES OF LINKS CAPTURED BY THE FIRST SIX PARTICIPANTS AT THE FIRST STAGE (DETERMINING CORRECT LINKS VS. CONFLICT LINKS)

Figure 4.2 shows the time taken by the six participants to manually capture links between 249 classes and 182 sections at the first stage. Participant 3 spent less time than others; 20 minutes

taken to retrieve links. Participant 5 took the longest time (120 minutes) to capture links. Participant 6 used 90 minutes to perform the link recovery. Participants 1 and 2 each took 50 minutes, and participant 4, 40 minutes. On average, each participant spent about one hour to identify links between 50 classes and 182 sections.

Figure 4.3 shows the link recovery performance of each participant; namely, the percentage of links captured by each participant. Participant 3 retrieved the lowest number of links, 7%. Participant 5 recovered the highest number of links, 95%. Participant 2 recovered 80% of links. Participant 4 is 71%, 66% for participant 6, and 31% for participant 1. In total, the six participants retrieved 409 links and identified 136 classes with no related sections. Figures 4.2 and 4.3 show that participant 3 used the least amount of time but retrieved the lowest number of links, while participant 5 spent the longest time but captured the highest number of links. Table 4.2 describes the comments made by the first six participants at the first stage. They commented that it was a tedious, boring, and time-consuming task. Moreover, one participant commented that he/she would favour automatic tracing technique to recover links.

TABLE 4.2 COMMENTS OF THE FIRST SIX PARTICIPANTS AT THE FIRST STAGE

| Participant | Comments |
|--------------------|---|
| 1 | it was a exhaustive and very tiring to the eyes |
| 2 | It's a very boring task – I would favour your automatic tracing technique |
| 3 | None |
| 4 | None |
| 5 | it hurted my eyes |
| 6 | tedious and time consuming |

After the first six participants identified links, these retrieved links needed to be verified. Figure 4.4 shows the time taken by the remaining five participants to manually verify these retrieved links. Table 4.3 describes the number of conflict links generated at the end of each stage. The seventh participant verified links retrieved by the first six participants and captured links missed by them. This participant took 220 minutes to verify these retrieved links and to capture new links. This participant identified 408 conflict links that included 355 new links. Figure 4.3 shows the percentage of retrieved links identified as conflict by participant 7. Links retrieved by participant 5 contained 14% conflict links, 11% for participant 2, 10% for participant 4, 7% for participant 6, and 1% for participant 3.

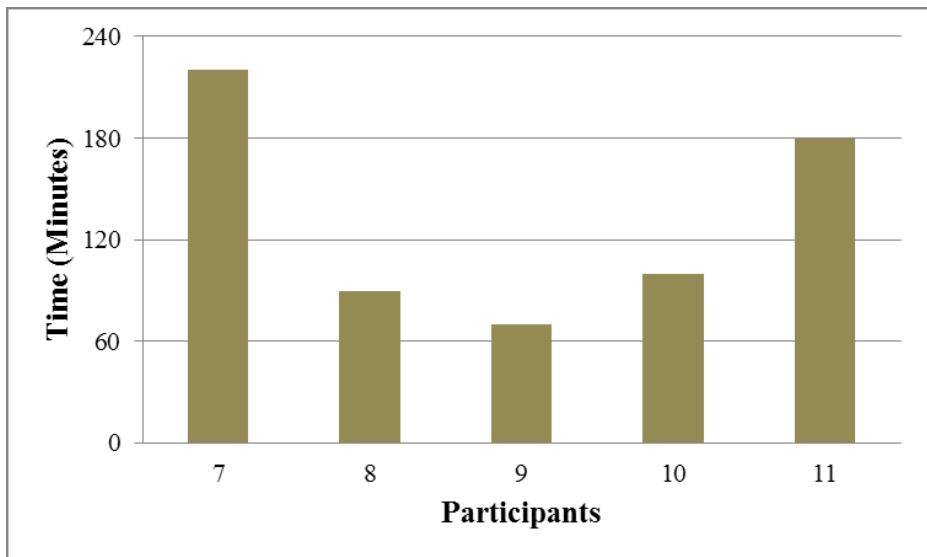


FIGURE 4.4 TIME TAKEN BY REST PARTICIPANTS IN VERIFYING RETRIEVED LINKS

TABLE 4.3 NO. OF CONFLICT LINKS PRODUCED AT THE END OF EACH STAGE

| Stage | No. of Conflict links produced at the end of each stage |
|--------------|--|
| First stage | 408 |
| Second stage | 75 |
| Third stage | 4 |

TABLE 4.4 COMMENTS OF THE REST FIVE PARTICIPANTS

| Participant | Comments |
|--------------------|---|
| 1 | it is very boringggg, desperately need tools |
| 2 | boring, good to have tools to support |
| 3 | totally boring, please tools tools.... |
| 4 | it is an absolutely tedious and boring task |
| 5 | tedious task, tools support would be so so good |

The next three participants (from 8 to 10 in Figure 4.4) were asked to verify 408 conflict links generated during the first stage. Each participant at the second stage took about 87 minutes to verify these conflict links on average. They reduced the conflict links to 75 (see Table 4.3). The last one (11 in Figure 4.4) verified the 75 conflict links produced at the second stage. This participant took 180 minutes to undertake the link verification. At the end of the process, 4 links remained in conflict (see Table 4.3). On average, each participant took around two hours to verify links. By comparison with Figure 4.2, participants spent more time in the link verification than in the link recovery on average. Table 4.4 shows the comments made by the rest five participants

when they conducted the link verification. They also commented that it was a tedious and very boring task. Four of them commented that it would be helpful to use traceability tools to support the link verification.

Measures. We use precision, recall and F-measure to evaluate the effectiveness and efficiency of a traceability recovery technique. We used this benchmark to measure our new traceability recovery technique (introduced in Chapter 3). We employed six Information Retrieval (IR) models (VSM, TF-IDF, BM25, DLH, PL2, and IFB2) to generate traceability links between the 249 classes and 182 sections in JDK1.5-SUBSET, and then compared the retrieved links with the oracle traceability link set. We obtained precision, recall, and F-measure results of six IR models using our JDK1.5-SUBSET traceability benchmark. These results form the baselines for the further evaluation and comparison. We then compared our evaluation results with the baselines to find out whether our new traceability recovery technique outperforms the six IR approaches. The evaluation results are detailed in Chapter 5.

4.5 Evaluation

We have carried out two evaluations: a review of the requirements for traceability benchmarks proposed by Dekhtyar et al. (2007), and the probability of errors (5% or 10% link errors) in the JDK1.5-SUBSET oracle link set. Furthermore, we discuss the cost-quality tradeoffs when building the traceability benchmark for a project.

4.5.1 Requirements Review

Our first evaluation was to examine to what extent traceability benchmarks developed by using our approach meet the five requirements proposed by Dekhtyar et al. (2007). These requirements concentrate on the benchmark for software traceability and include: support for traceability in multiple fields of software engineering, independence of methodology, ground truth, accuracy testing, and scalability testing.

Support for traceability in multiple fields of software engineering. The benchmark needs to support tracing methods and procedures for tasks from different areas of software engineering. Our approach is designed for the development of benchmarks for the evaluation of traceability recovery techniques. However, benchmarks built using our approach can be used for evaluating tasks from other fields of software engineering such as verification & validation, maintenance, or reverse engineering. For example, traceability links between code and other sources of information are crucial to the analysis of the impact of changes in order to accomplish any maintenance task (Dekhtyar et al., 2007). Thus the JDK1.5-SUBSET benchmark we built is able to assist in the evaluation of impact analysis techniques used by software maintenance tasks.

Independence of methodology. The benchmark needs to be independent of any specific tracing tool or approach. Our approach does not rely on any tool or traceability recovery technique to produce the oracle traceability link set. Hence, benchmarks developed by using our approach can be used to evaluate any tracing method and procedure, whether manual, semi-automated, or automated.

Ground truth. The benchmark needs to provide the true answer set for each tasks. Our approach requires producing an oracle traceability link set for every task identified within the benchmark because different tasks may need different artifacts to accomplish. Our rigorous manual identification and verification strategies ameliorate the possibility of including fault links into the oracle traceability link set.

Accuracy testing. The benchmark needs to be able to test the performance of tracing techniques. Benchmarks produced using our approach provide precision, recall and F-measure to evaluate the accuracy and coverage of traceability recovery techniques.

Scalability testing. The benchmark needs to allow assessing the scalability of tracing techniques. As our approach suggests using projects of a reasonable size, benchmarks established by using our approach are not representative of large systems. However, they can be extended to include more artifacts and/or tasks to improve the scalability. For example, our JDK1.5-SUBSET benchmark can be extended to include more classes, documents, artifacts, and/or tasks. In our example, we recruited 11 analysts to establish links between 249 classes and 182 sections. On average, each of

the first six participants took one hour to identify sections related to 50 classes, and each of the other five participants used two hours to verify these retrieved links.

4.5.2 Probability of Errors in JDK1.5-SUBSET Benchmark

We have discussed the development of the traceability benchmark for JDK1.5-SUBSET in Section 4.4. This benchmark contains an oracle link set that includes all true links in JDK1.5-SUBSET. This oracle link set consists of 356 links retrieved at the first stage, 333 links verified at the second stage, and 71 links verified at the third stage. In order to compute the probability of errors (5% or 10% errors) in this link set, we need to calculate the probability of link recovery errors made by each participant ($Pr(x_j)$) and the probability of participants simultaneously making the same mistake of capturing an incorrect link ($Pr(y_i)$). Initially we used the example error rates for different participants/stages introduced in section 4.3.4 (20% for juniors, 10% for seniors and 5% for seniors consulting one another) just to demonstrate this method.

Table 4.5 lists the results of $Pr(x_j)$ and $Pr(y_i)$ for having $\geq 10\%$ link errors. We use STATDISK calculators to produce these results of $Pr(x_j)$ and $Pr(y_i)$. STATDISK can be free downloaded at <http://www.statdisk.org>. Then the probability of $\geq 10\%$ link errors in the JDK1.5-SUBSET oracle link set is calculated as follows:

$$\Pr[E] = \frac{\Pr[e_1] \times n_1 + \Pr[e_2] \times n_2 + \Pr[e_3] \times n_3}{N} = 0.027 \text{ for } E \geq N * 10\%, \text{ where}$$

- $N = 760$
- n_i is the number of links in the agreed link set at the i^{th} stage, here $n_1 = 356$, $n_2 = 333$, and $n_3 = 71$
- $\Pr[e_i] = (\sum_{j=1}^{m_i} \Pr(x_j) / m_i) \times \Pr(y_i)$
- $\Pr[e_1] * n_1 = 0.88122235 * 0.0225 * 356 = 7.05859$
- $\Pr[e_2] * n_2 = 0.9998299 * 0.04 * 333 = 13.31773$
- $\Pr[e_3] * n_3 = 0.0641971 * 0.0266119 * 71 = 0.121297$

TABLE 4.5 PROBABILITY DISTRIBUTION FOR THE JDK1.5-SUBSET ORACLE LINK SET BASED ON THE EXAMPLE ERROR RATES OF THE THREE TYPES OF PARTICIPANTS

| Stage | Participant | Retrieved Links (R) | Conflict Links (C) | Agreed Links (n=R-C) | Example Error Probability for Participant (p) | $Pr(x_j) \text{ for } x_j \geq n * 10\% \text{ (round to the nearest integer)}$ |
|--|--|---------------------|--------------------|----------------------|---|---|
| 1st stage | 1 | 10 | 1 | 9 | 0.2 | 0.8657823 |
| | 2 | 84 | 12 | 72 | 0.2 | 0.9940038 |
| | 3 | 18 | 2 | 16 | 0.2 | 0.8592625 |
| | 4 | 35 | 5 | 30 | 0.2 | 0.955821 |
| | 5 | 145 | 21 | 124 | 0.2 | 0.9990852 |
| | 6 | 117 | 12 | 105 | 0.2 | 0.9969877 |
| | 7 | 764 | 408 | 356 | 0.1 | 0.497614 |
| $\sum_{j=1}^{m_1} Pr(x_j) / m_1 = \sum_{j=1}^7 Pr(x_j) / 7 = 0.881222357$ | | | | | | |
| $Pr(y_1) = 0.0225 \text{ for } y_1 \geq 2, \text{ where } n=2, p=(0.2+0.1)/2=0.15$ | | | | | | |
| 2nd stage | 8 | 272 | 53 | 219 | 0.2 | 0.9999765 |
| | 9 | 272 | 123 | 149 | 0.2 | 0.9996381 |
| | 10 | 272 | 87 | 185 | 0.2 | 0.9998751 |
| | $\sum_{j=1}^{m_2} Pr(x_j) / m_2 = \sum_{j=1}^3 Pr(x_j) / 3 = 0.9998299$ | | | | | |
| $Pr(y_2) = 0.04 \text{ for } y_2 \geq 2, \text{ where } n=2, p=(0.2+0.2)/2=0.2$ | | | | | | |
| 3rd stage | 11 | 75 | 4 | 71 | 0.05 | 0.0641971 |
| | $\sum_{j=1}^{m_3} Pr(x_j) / m_3 = \sum_{j=1}^1 Pr(x_j) / 1 = 0.0641971$ | | | | | |
| | $Pr(y_3) = 0.0266119 \text{ for } y_3 \geq 3, \text{ where } n=5, p=(0.2+0.1+0.2+0.2+0.05)/5=0.15$ | | | | | |

We can use the same method to calculate the probability of $\geq 5\%$ link errors in the JDK1.5-SUBSET oracle link set:

$$Pr[E] = 0.0292 \text{ for } E \geq N*5\%, \text{ where}$$

- $Pr[e_1] * n_1 = 0.994423843 * 0.0225 * 356 = 7.965335$
- $Pr[e_2] * n_2 = 1.0 * 0.04 * 333 = 13.32$
- $Pr[e_3] * n_3 = 0.4771324 * 0.0266119 * 71 = 0.9015154$

Although $Pr[E \geq N*5\%]$ is slightly larger than $Pr[E \geq N*10\%]$, their results are very close and very small. Therefore, the probability of the oracle link set having $\geq 5\%$ or 10% is very low, 0.0292 for

$\geq 5\%$ and 0.027 for $\geq 10\%$ (all around 3%). In other words, the probability of building at least 95% correct links is very high (around 97%). The above calculation reveals three features:

1. The more senior a participant, the lower probability of link errors the participant can make. For example, at the first stage, the first six participants are junior participants who have an error probability of 20%, their probabilities of making $\geq 10\%$ link errors are larger than 85.9% (see Table 4.5). But the 7th participant, who is a senior analyst with an error probability of 10%, can achieve 49.8% probability of making $\geq 10\%$ link errors.
2. The probability of participants simultaneously making the same mistake of identifying an incorrect links is very low, $Pr(y_1) = 0.0225$, $Pr(y_2) = 0.04$, and $Pr(y_3) = 0.0266$. This indicates that the chance of several participants (≥ 2 or 3) retrieving an incorrect link at the same time is very rare. This confirms that our rigorous manual identification and verification strategies can largely reduce the probability of errors in the oracle link set.
3. The more participants allocated to verify a link, the lower the probability of link errors. For example, at the first stage, each link in the agreed link set was agreed to be true by two participants, $Pr(y_1) = 0.0225$ (see Table 4.5). If we add one more participant to verify a link and the third participant is a junior participant, then $Pr(y_1) = 0.0046$ for $y_1 \geq 3$, where $n=3$, $p=(0.2+0.2+0.1)/3=0.16667$. But if the third participant is a senior participant, then $Pr(y_1) = 0.0024$ for $y_1 \geq 3$, where $n=3$, $p=(0.2+0.1+0.1)/3=0.13333$.

Instead of using the assumed example error rates for participants we can make use of the observations and results obtained from the establishment of the JDK1.5-SUBSET oracle link set to calculate the actual average error rates for the three types of participants. Table 4.6 shows the actual error rates for participants in the establishment of the JDK1.5-SUBSET oracle link set. The actual error probability for a participant is computed as the number of retrieved links / the number of incorrect links, where retrieved links are links that are recovered or verified by the participant, incorrect links are links that are recovered or verified by the participant but are excluded from the final oracle link set. The actual error rate for a participant is an approximation as the number of incorrect links is obtained by comparison between links retrieved or verified by the participant and links in the final oracle link set hence may not be completely accurate but it is a good approximation.

TABLE 4.6 ACTUAL ERROR RATES FOR THE THREE TYPES OF PARTICIPANTS DURING THE ESTABLISHMENT OF THE JDK1.5-SUBSET ORACLE LINK SET

| Stage | Participant | Retrieved Links (R) | Incorrect Links (W) | Actual Error Probability for participant ($p=W/R$) |
|-----------------------|-------------|-------------------------|-------------------------|--|
| 1 st stage | 1 | 10 | 1 | 0.1 |
| | 2 | 84 | 5 | 0.05952 |
| | 3 | 18 | 1 | 0.05556 |
| | 4 | 35 | 3 | 0.08571 |
| | 5 | 145 | 6 | 0.04138 |
| | 6 | 117 | 6 | 0.05128 |
| | 7 | 764 | 31 | 0.04058 |
| 2 nd stage | 8 | 272 | 3 | 0.01103 |
| | 9 | 272 | 8 | 0.02941 |
| | 10 | 272 | 6 | 0.02206 |
| 3 rd stage | 11 | 75 | 2 | 0.02667 |

We used nine junior participants: the first six participants at the first stage and three participants at the second stage. The average error probability for junior participants is 0.05066 (5.066%), which is much lower than the example error rate (20%) used above. The senior participant (the 7th participant in the first stage) has an error probability of 0.04058 (4.058%), which is also lower than the corresponding example error rate (10%). The senior participant (the 11th participant in the third stage) who consulted another senior participant has an error probability of 0.02667 (2.667%), which is lower than the corresponding example error rate (5%). We then apply the three actual error rates for participants to recalculate the probability of errors ($\geq 5\%$ or 10% errors) in the JDK1.5-SUBSET oracle link set.

Table 4.7 lists the results of $Pr(x_j)$ and $Pr(y_i)$ for having $\geq 10\%$ link errors based on the actual error rates for the three types participants. The probability of $\geq 10\%$ link errors in the JDK1.5-SUBSET oracle link set is calculated as follows:

$$\Pr[E] = 1.2689e-4 \text{ for } E \geq N*10\%, \text{ where}$$

- $Pr[e_1] * n_1 = 0.1247363 * 0.0020812 * 356 = 0.092418$
- $Pr[e_2] * n_2 = 0.004564633 * 0.0025664 * 333 = 0.003901$
- $Pr[e_3] * n_3 = 0.0028728 * 0.0005627 * 71 = 0.000115$

TABLE 4.7 PROBABILITY DISTRIBUTION FOR THE JDK1.5-SUBSET ORACLE LINK SET BASED ON THE ESTIMATED ERROR RATES FOR THE THREE TYPES OF PARTICIPANTS

| Stage | Participant | Agreed Links (n) | Actual Error Probability for Participant (p) | $\Pr(x_j)$ for $x_j \geq n * 10\%$ (round to the nearest integer) |
|---|---|----------------------|--|--|
| 1 st stage | 1 | 9 | 0.05066 | 0.3736804 |
| | 2 | 72 | 0.05066 | 0.0720411 |
| | 3 | 16 | 0.05066 | 0.1931093 |
| | 4 | 30 | 0.05066 | 0.1928713 |
| | 5 | 124 | 0.05066 | 0.0238598 |
| | 6 | 105 | 0.05066 | 0.0175915 |
| | 7 | 356 | 0.04058 | 0.0000007 |
| | $\sum_{j=1}^{m_1} \Pr(x_j) / m_1 = \sum_{j=1}^7 \Pr(x_j) / 7 = 0.1247363$ | | | |
| $\Pr(y_1) = 0.0020812$ for $y_1 \geq 2$, where $n=2$, $p=(0.05066+0.04058)/2=0.04562$ | | | | |
| 2 nd stage | 8 | 219 | 0.05066 | 0.0018757 |
| | 9 | 149 | 0.05066 | 0.0089458 |
| | 10 | 185 | 0.05066 | 0.0028724 |
| | $\sum_{j=1}^{m_2} \Pr(x_j) / m_2 = \sum_{j=1}^3 \Pr(x_j) / 3 = 0.004564633$ | | | |
| $\Pr(y_2) = 0.0025664$ for $y_2 \geq 2$, where $n=2$, $p=(0.05066+0.05066)/2=0.05066$ | | | | |
| 3 rd stage | 11 | 71 | 0.02667 | 0.0028728 |
| | $\sum_{j=1}^{m_3} \Pr(x_j) / m_3 = \sum_{j=1}^1 \Pr(x_j) / 1 = 0.0028728$ | | | |
| | $\Pr(y_3) = 0.0005627$ for $y_3 \geq 3$, where $n=5$, $p=(0.05066+0.04058+0.05066+0.05066+0.0028728)/5=0.03908656$ | | | |

The probability of making $\geq 10\%$ link errors is extremely low. It shows that the probability of creating at least 90% accuracy in the JDK1.5-SUBSET oracle link set is extremely close to 100%.

The probability of $\geq 5\%$ link errors in the JDK1.5-SUBSET oracle link set is computed as follows:

$$\Pr[E] = 0.0012 \text{ for } E \geq N*5\%, \text{ where}$$

- $\Pr[e_1] * n_1 = 0.563022743 * 0.0020812 * 356 = 0.417148$
- $\Pr[e_2] * n_2 = 0.595128333 * 0.0025664 * 333 = 0.508603$
- $\Pr[e_3] * n_3 = 0.1216272 * 0.0005627 * 71 = 0.004859$

This result shows that the probability of making $\geq 5\%$ link errors is 0.0012 (0.12%). In other words, the probability of building an oracle link set with accuracy of at least 95% is very high, about 99.9%. We thus conclude that our approach produces a high quality oracle link set.

4.5.3 Cost-quality Tradeoffs

The most important part of establishing a traceability benchmark is to create a high quality oracle link set (e.g. the link set with $\leq 5\%$ errors). Building a high quality oracle link set depends on the following factors:

1. The workload allocated to each participant.
2. The number of participants verifying a link.
3. The knowledge of the traced project of each participant, i.e. he/she is a junior or senior participant.
4. The time each participant spends to recover links.

In general, each participant at the same stage is allocated a similar workload. For the first stage, there are two groups of participants. The first group retrieves links between allocated artifacts. The second group verifies links retrieved by the first group and recovers links missed by them. For example, when we built the JDK1.5-SUBSET oracle link set, the first six participants at the first stage captured links between allocated classes and documents. The 7th participant at the first stage verified links retrieved by the first six participants and recovered links missed by them. Every participant in the same group was assigned a similar workload at the first stage. The more workload that is assigned to a participant, the more effort they are required to make.

From the error probability calculation discussed above, we noticed that using a different number of participants to verify a link can affect the results of the probability of errors in the oracle link set. Using more participants to verify a link can produce a more accurate oracle link set. Using at least three senior participants to verify each link can achieve better results than using at least three junior participants or the combination of junior and senior participants, or at least two senior/junior participants.

The participants' knowledge of the traced project can significantly affect the probability of errors in the agreed link set recovered by them. Based on the error probability calculation discussed above, and our observations in practice, we postulated then confirmed that the more senior a participant, the lower the probability of link errors the participant will make and that these differences have a significant impact on the overall error probability.

From the establishment of the JDK1.5-SUBSET oracle link set, we noticed that the more time a participant spends to recover links, the more links the participant identifies. For instance, with the first six participants at the first stage, participant 5 spent the longest time (120 minutes) but recovered 95% links (see Figures 4.2 and 4.3) and had the lowest error rate (4.138%) (see Table 4.6). Participant 3 spend only 20 minutes but only recovered 7% links and had the error rate of 5.556%. Therefore, spending longer time is more likely to recover all links. Otherwise, participants are more likely to identify incorrect links or miss retrieving many correct links. This can significantly damage the quality of the oracle link set.

Overall, the priority rank for the four factors is as follows. Time is the most important factor. Next is the participant's knowledge, followed by the number of participants verifying a link. The least important factor is workload. If the workload assigned to each participant is certain, the best solution for building a high quality oracle link set is to recruit all senior participants, to use at least three participants to verify a link, and to take as long as possible. Unfortunately, it is very hard to recruit senior participants in practice. Moreover, it is not easy to decide how many times are appropriate to identify or verify links, which depends on the assigned workload. In our case, we assigned approximately 50 classes to each junior participant in the first group at the first stage. On average, each took around 60 minutes to identify links between 50 classes and the documents. Each link at the first and second stages was verified by only two participants. We still achieved 99.9% probability of producing at least 95% correct oracle link set based on the actual error rates for the three types of participants. Therefore, the alternative solution for building a high quality oracle link set is:

- 1) To use junior participants for the first group at the first stage and the group at the second stage;

- 2) To use senior participants for the second group at the first stage, because they not only verify links retrieved by the first group but also recover links missed by them;
- 3) To use senior participants for the group at the third stage because they need to verify links that are still in the conflict link set after going through the two stages;
- 4) To use at least two participants to verify a link at each stage;
- 5) To take at least one hour to retrieve links if the workload allows a participant to complete in one hour and can produce an acceptable result.

4.6 Discussion

The review of requirements for traceability benchmarks shows that using our approach can develop affordable and robust benchmarks that meet the five requirements. The actual probability of making $\geq 5\%$ link errors in the JDK1.5-SUBSET oracle link set is 0.12%. Our rigorous manual identification and verification strategies significantly improve the accuracy of the each stage's agreed link set. The two evaluations illustrate that our approach can help researchers to develop a robust and high quality traceability benchmark to perform an evaluation and comparison of different traceability link recovery approaches.

However, our approach suffers from four problems that occur during the development of a traceability benchmark.

- The first problem is the difficulty of determining whether or not two elements in artifacts are in fact related. Although we provide a traceability rule to help in the identification of true links, we rely on participants' knowledge and understanding to capture links. This may lead to the capture of incorrect links. To reduce the possibility of including fault links to the oracle traceability link set, we designed a rigorous manual verification strategy to verify retrieved links, each of which are examined by at least three analysts.
- The second problem is how much workload is suitable for a participant to undertake. The greater the workload allocated to a participant, the more time and energy are required. Too much workload may make participants lose interest in participation. When we built the

JDK1.5-SUBSET benchmark, every participant took one hour to identify the related sections for 50 classes on average. But it took a longer time to do the link verification than the link recovery on average.

- The third issue is the recruitment of participants. It is not easy to recruit a good number of participants who have some knowledge of the selected project, especially for recruiting senior analysts. If a participant is new to the selected project, he/she might be more likely to capture incorrect links than someone who knows the project to some extent.
- The final problem is the scalability of benchmarks. Our approach is suitable to build benchmarks for projects of a reasonable size because of its approach of manually identifying and verifying traceability links. But benchmarks produced by using our approach can be extended to include more elements, artifacts, tasks, and/or measures.

The first threat to the validity of our approach is the traceability rules we defined to help participants find correct links may influence them to capture incorrect links or miss some correct links. We encourage traceability researchers to explore how to define traceability rules and how to validate their correctness. Second, false positive links may be included in the oracle link set. This is because a link agreed to be true by participants at each stage is put in the oracle link set even if it is actually incorrect. Third, some correct links may fail to be identified by participants. The above three threats can affect the accuracy of the actual error rate for each participant. Thus, it is important to expand our approach in the future by exploring how correct links should be defined and how to assist participants in identifying them. Fourth, some links may be harder to identify than others in practice. In that case, the binomial distribution used in our approach may not be suitable. Other probability distributions therefore need to be explored to cover this issue in the future. Fifth, the case we used is a small project that contains a small fraction of the source code and documents in the JDK1.5 system. It is not representative of large software systems. Finally, our approach also may show different probability error results when applied to recover links between artifacts in the JDK1.5-SUBSET case by using other groups of participants.

We have made our new JDK1.5-SUBSET benchmark public and we allow users to access or download it for free. Our benchmark is represented in a spreadsheet format. Anyone can review the data, apply it to evaluate their traceability approaches, and probably extend it to meet their own

needs better. Users can download it from: <http://tinyurl.com/7l3ohe4>.

In future work, we will extend this benchmark to cover more classes and documents. This benchmark could then be used to evaluate tracing approaches and procedures for a wide range of tasks from different areas of software engineering. We also will look at other probability distributions for the probability of incorrect links captured by each participant ($Pr(x_j)$) to cover the issue that links may have different probabilities of being retrieved. Because a link's recovery is highly dependent on the textual descriptions, some links may be harder than others to find due to ambiguous wording issues. This can lead to increased error rates on particular links. The binomial distribution we have applied then may become invalid as clustering may occur. However, using different probability distribution is unlikely to significantly affect the very low error rates in the oracle link set we have come up with. This is because the probability of errors in the created oracle link set heavily depends on the probability of participants simultaneously making the same mistake ($Pr(y_i)$), which is very low. Moreover, we will look at other traceability benchmarks that have been used by other researchers to compare their precision and recall results with ours. We encourage researchers to explore how to create general traceability benchmarks and share created benchmarks to evaluate the performances of traceability recovery techniques.

4.7 Summary

In this chapter, we presented an approach and guidelines to help researchers to establish affordable and robust traceability benchmarks. Our approach comprises five steps: task identification, artifact selection, project selection, oracle/true traceability link set development, and evaluation metrics. We designed rigorous identification and verification strategies to decide whether or not a link is true; every link is verified by at least two analysts. The visit to requirements for traceability benchmarks shows that our approach can develop benchmarks that satisfy these requirements. A benchmark for JDK1.5-SUBSET was built by using our approach. We built a formula to compute the probability of errors in the created oracle link set. The probability of making $\geq 5\%$ link errors in the JDK1.5-SUBSET oracle link set is around 0.1%. The accuracy of the agreed link set at each stage is significantly improved by our rigorous manual identification and verification strategies.

The error probability results show that our approach can build a high quality oracle link set for the selected project.

In the next chapter, Chapter 5, we employ the JDK1.5-SUBSET benchmark and three additional benchmarks provided by Bacchelli et al. (2010) to evaluate our new traceability recovery technique and prototype tool. In Chapter 6, we elaborate how to visualize traceability links retrieved by the traceability recovery technique.

Chapter 5 -- Evaluation of Traceability Link Recovery

This chapter presents an evaluation of our combination traceability recovery approach, **IРЕTrace**, by applying six Information Retrieval (IR) models to four case studies of varying size and context. The evaluation results are then analyzed to explore the strengths and weaknesses of IРЕTrace and to identify some potential future work.

5.1 Case Studies

To validate the effectiveness of the three enhancement techniques we added to the IR models, namely Regular Expression (RE), Key Phrases (KP), and Clustering, we have set up four case studies based on four unrelated software systems. The first system we used for an experimental case study is JDK1.5-SUBSET, which was described in Chapter 4. We chose this system because it is a widely used open source system for Java developers and is well documented, its code is well commented and detailed documentation is provided. It contains 249 classes and 182 sections. We adopted our rigorous manual identification and verification strategies (discussed in Chapter 4) to build the oracle traceability link set for JDK1.5-SUBSET. The systems used for the other three case studies are ArgoUML, Freenet, and JMeter. We chose the three systems because they are open-source free software systems written in Java, contain different types of documents, and have their own traceability benchmarks. Alberto Bacchelli (2010) kindly provided the three systems, their email archives, and their oracle traceability link sets containing true links between classes and emails. These emails were extracted from the active development mailing list of each project. These development mailing lists provide a great deal of information relating to software development. The oracle traceability link sets for the three systems were manually built by

Bacchelli et al. (2010), who read all the emails and annotated them with the class entities they contain. Table 5.1 provides details of the four case studies. Here, “sections” is used for JDK1.5-SUBSET and “emails” for other three cases.

TABLE 5.1 DETAILS OF EACH CASE STUDY

| System | Classes | Sections/Emails | Total true/correct links |
|----------------------|---------|-----------------|--------------------------|
| JDK1.5-SUBSET | 249 | 182 | 760 |
| ArgoUML | 423 | 378 | 308 |
| Freenet | 517 | 372 | 516 |
| JMeter | 372 | 348 | 563 |

We evaluated the performance of IRETrace employing two IR engines, Apache Lucene and the Terrier IR platform. Apache Lucene uses the Vector Space Model (VSM) to extract traceability links between classes and sections/emails. We also recovered links using five additional IR models, TF-IDF, BM25, DLH, PL2, and IFB2, supported by the Terrier IR platform. How the two IR engines using different retrieval IR models implement the traceability link recovery was described in Chapter 3. Their results are discussed in the next section. We applied three metrics - precision, recall, and F-measure - to measure the quality of IR models. The three metrics were described in Chapter 4.

5.2 Evaluation Results

To evaluate whether the three supporting techniques, RE, KP, and Clustering, ameliorate limitations of IR models, we compared the performances of four different combination techniques: an IR model; the combination of IR and RE (IR+RE); the combination of IR, RE and KP (IF+RE+KP); and our final approach -- IR, RE, KP and Clustering (IR+RE+KP+Clustering). The following sections describe the results produced by the four different combination techniques adopting the Lucene and Terrier engines. Every approach recovers links with a similarity score \geq the cut point. For example, if the cut point is 0.02, this denotes that all links having a similarity score ≥ 0.02 are extracted by these approaches. A cut point < 0.3 is defined to be a low cut point in the following discussion. A cut point ≥ 0.3 is defined to be a high cut point. The factor of using the cut-point scale in the following figures is to illustrate changes of precision, recall, or F-measure

results in detail between 0 and 0.1 cut points.

5.2.1 Evaluation Results from Using the Apache Lucene Engine

The Lucene engine utilizes VSM to index text and determine how relevant an artifact in a system is to a given query (Apache Lucene – Overview, 2009; Konchady, 2008). In this section, we present the evaluation results of the four combination approaches using VSM as the IR technique for the four case studies. Table 5.2 summarizes the precision and recall results of all approaches at the 0.02 and 0.9 cut points. In Figure 5.1, we illustrate the precision results of all approaches at all cut points from 0 to 0.9. Recall results of all approaches at all cut points are shown in Figure 5.2.

The Basic Retrieval Approach – VSM

First, we used VSM to recover links between documents and class entities to discover VSM's performance at different cut points. It is obvious from precision results in Figure 5.1 and recall results in Figure 5.2 that the lower the cut point is used, the lower the precision value but the higher the recall value VSM obtains. Table 5.2 shows that the four cases have low precision values (<11%) but high recall values (>81%) at the 0.02 cut point. In other words, **although VSM retrieves a majority of true links at low cut points from 0 to 0.1, many incorrect links are extracted**, especially at 0 and 0.02 cut points. VSM gets the highest precision value at the 0.9 cut point for JDK1.5-SUBSET and ArgoUML and at 0.7 cut point for Freenet and JMeter but only recovers very few true links (see Figures 5.1 and 5.2); the recall values at the 0.9 cut point in all cases are less than 4% (see Table 5.2). Therefore, **overall VSM has a low precision at low cut points and low recall at high cut points**.

TABLE 5.2 PRECISION AND RECALL RESULTS AT CUT POINTS 0.02 AND 0.9 USING VSM

| Approach | Cut point | JDK1.5-SUBSET | | ArgoUML | | Freenet | | JMeter | |
|--------------------------|-----------|---------------|--------|-----------|--------|-----------|--------|-----------|--------|
| | | Precision | Recall | Precision | Recall | Precision | Recall | Precision | Recall |
| VSM | 0.02 | 10.52% | 97.5% | 2.34% | 90.91% | 3.44% | 81.59% | 4.64% | 84.9% |
| | 0.9 | 87.88% | 3.82% | 47.62% | 3.25% | 28.57% | 0.78% | 40% | 0.71% |
| VSM+RE | 0.02 | 10.52% | 97.5% | 2.39% | 92.86% | 3.58% | 85.27% | 4.89% | 91.83% |
| | 0.9 | 93.44% | 82.5% | 56.72% | 61.69% | 65.97% | 67.64% | 51.05% | 69.27% |
| VSM+RE+KP | 0.02 | 11.49% | 96.84% | 2.46% | 92.21% | 3.6% | 85.27% | 5.16% | 92.01% |
| | 0.9 | 93.85% | 82.37% | 56.72% | 61.69% | 66.35% | 67.64% | 51.05% | 69.27% |
| VSM+RE+KP +Clustering | 0.02 | 53.29% | 88.55% | 53.33% | 62.34% | 6.82% | 80.23% | 10.07% | 79.4% |
| | 0.9 | 93.83% | 82.11% | 57.88% | 62.01% | 66.98% | 68.02% | 53.14% | 72.11% |

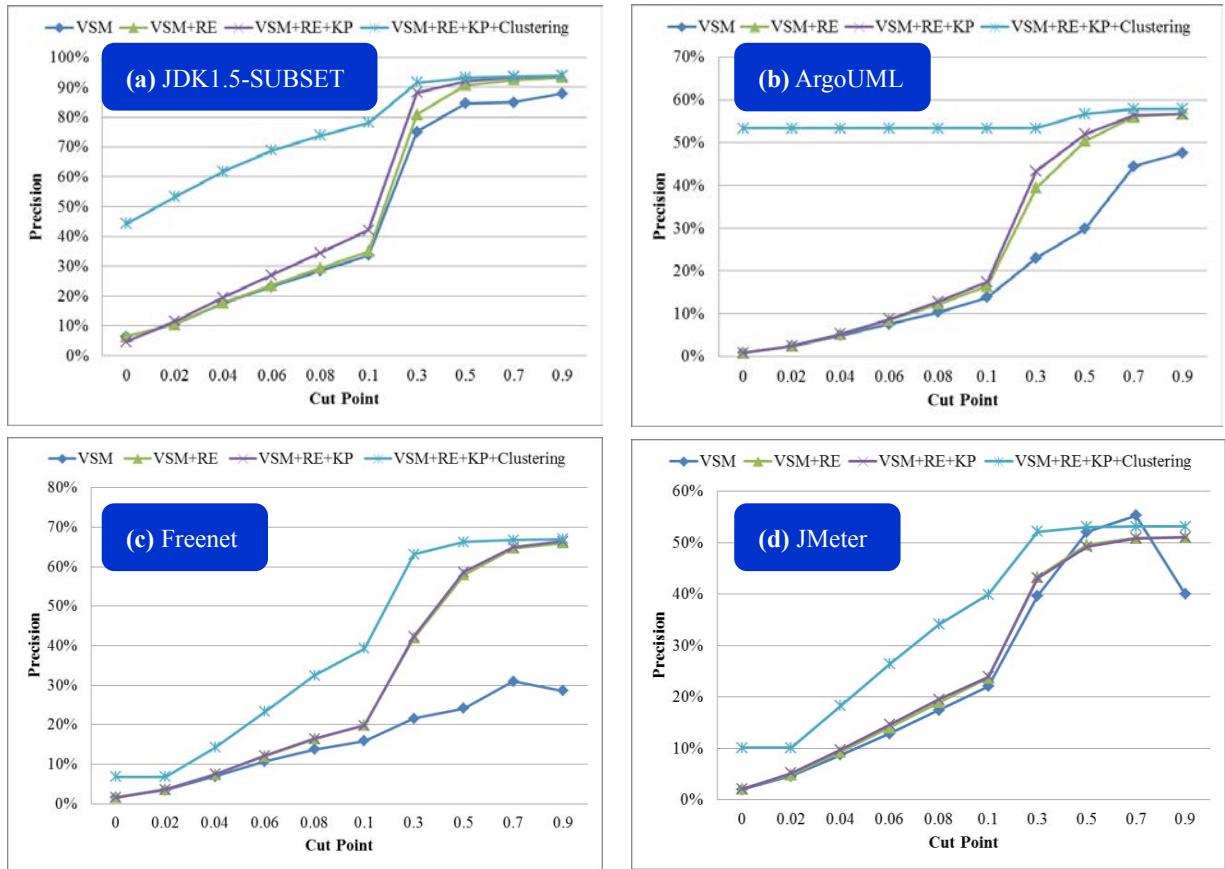


FIGURE 5.1 PRECISION RESULTS FOR THE FOUR CASES USING VSM

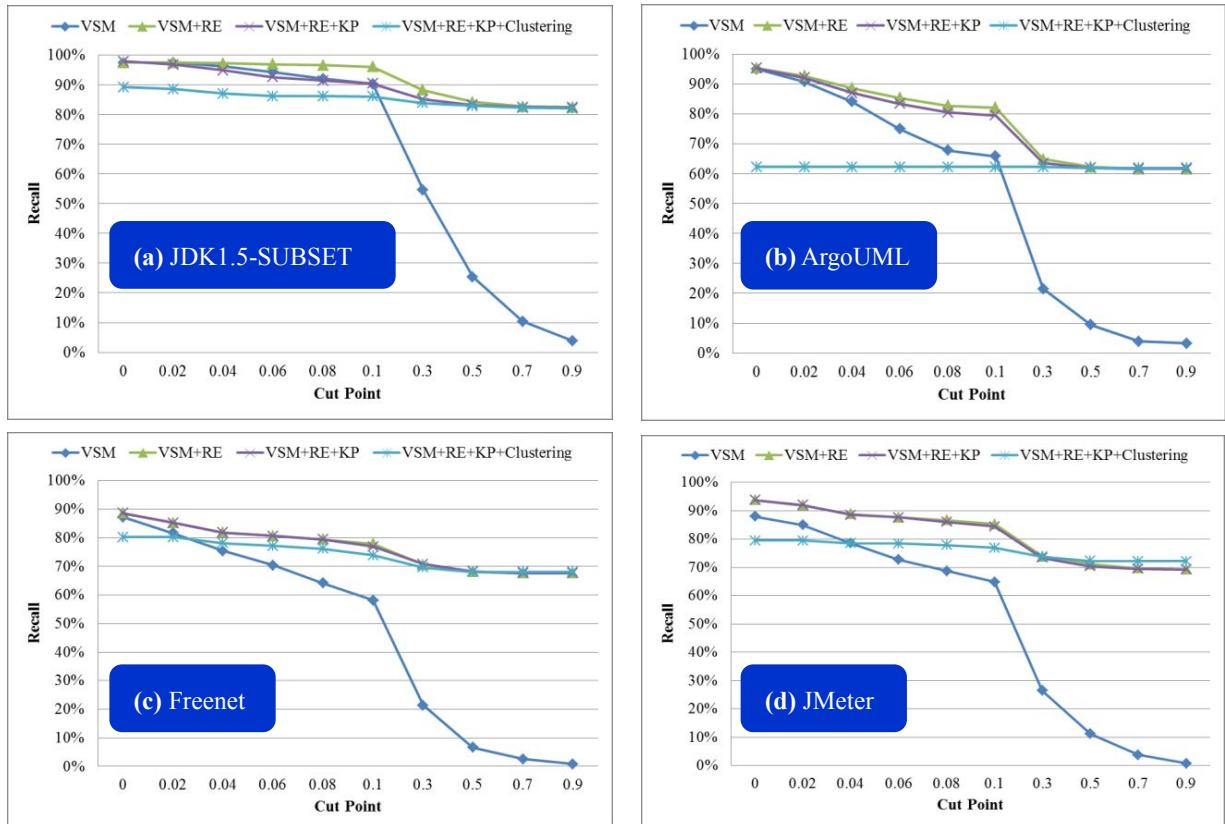


FIGURE 5.2 RECALL RESULTS FOR THE FOUR CASES USING VSM

VSM and Regular Expressions (RE) – VSM+RE

We then evaluated the combination of VSM and RE to verify whether RE can increase the number of retrieved links at high cut points, namely increase the recall values at high cut points. In Figure 5.1, precision for JDK1.5-SUBSET, ArgoUML, and Freenet is increased at all cut points especially for high cut points after adding RE to VSM. Apart from a decrease at the 0.5 and 0.7 cut points, precision for JMeter has a slight increase at all cut points. These results show that **adding RE to VSM improves precision at all cut points** except for JMeter's 0.5 and 0.7 cut points.

In Figure 5.2, we observe that **recall is largely increased at all cut points especially for high cut points**. Table 5.2 shows that the recall values at the 0.9 cut point in all cases reach at least 61%; JDK1.5-SUBSET gets the highest recall 82.5%. This indicates that adding RE to VSM can increase the recall values and retrieve more true links than VSM alone at high cut points.

VSM, RE, and Key Phrases (KP) -- VSM+RE+KP

To recover links missed by VSM, we added an additional technique, KP. From Figures 5.1 and 5.2, compared with the combination of VSM and RE, we see that **after adding KP to VSM and RE, precision at all cut points is increased** for JDK1.5-SUBSET and ArgoUML, **but recall has a slight decrease**. However, there is no significant improvement in Freenet and JMeter. Precision and recall for Freenet and JMeter have no increase or decrease at all cut points (see Figures 5.1 and 5.2). In Table 5.2, precision for all cases has a 0%-1% increase at the 0.02 and 0.9 cut points. Compared with VSM, adding KP increases precision at all cut points apart from JMeter's 0.5 and 0.7 cut points (see Figure 5.1). Moreover, recall for all cases is increased at all cut points except for a slight decrease at cut points from 0.02 to 0.08 for JDK1.5-SUBSET (see Figure 5.2). This indicates that the combination of VSM+RE+KP retrieves more true links and fewer incorrect links than VSM alone.

VSM, RE, KP, and Clustering -- VSM+RE+KP+Clustering

Incorporating Clustering into the last combination aims to reduce the number of incorrect links but not to excessively deteriorate recall. After integrating Clustering with VSM+RE+KP, **precision for all cases is largely increased at all cut points, especially for low cut points** (see Figure 5.1).

Many incorrect links are discarded at low cut points. Although our approach retrieves fewer true links than VSM alone at low cut points for JDK1.5-SUBSET and ArgoUML, at 0 to 0.02 cut points for Freenet, and at 0 to 0.04 cut points for JMeter, recall is only slightly reduced and still reaches a value $\geq 80\%$ for JDK1.5-SUBSET, Freenet and JMeter, and $> 62\%$ for ArgoUML (see Figure 5.2 and Table 5.2). These show that our approach of integrating the three supporting techniques with VSM largely reduces the number of fault links without suffering from low recall at low cut points.

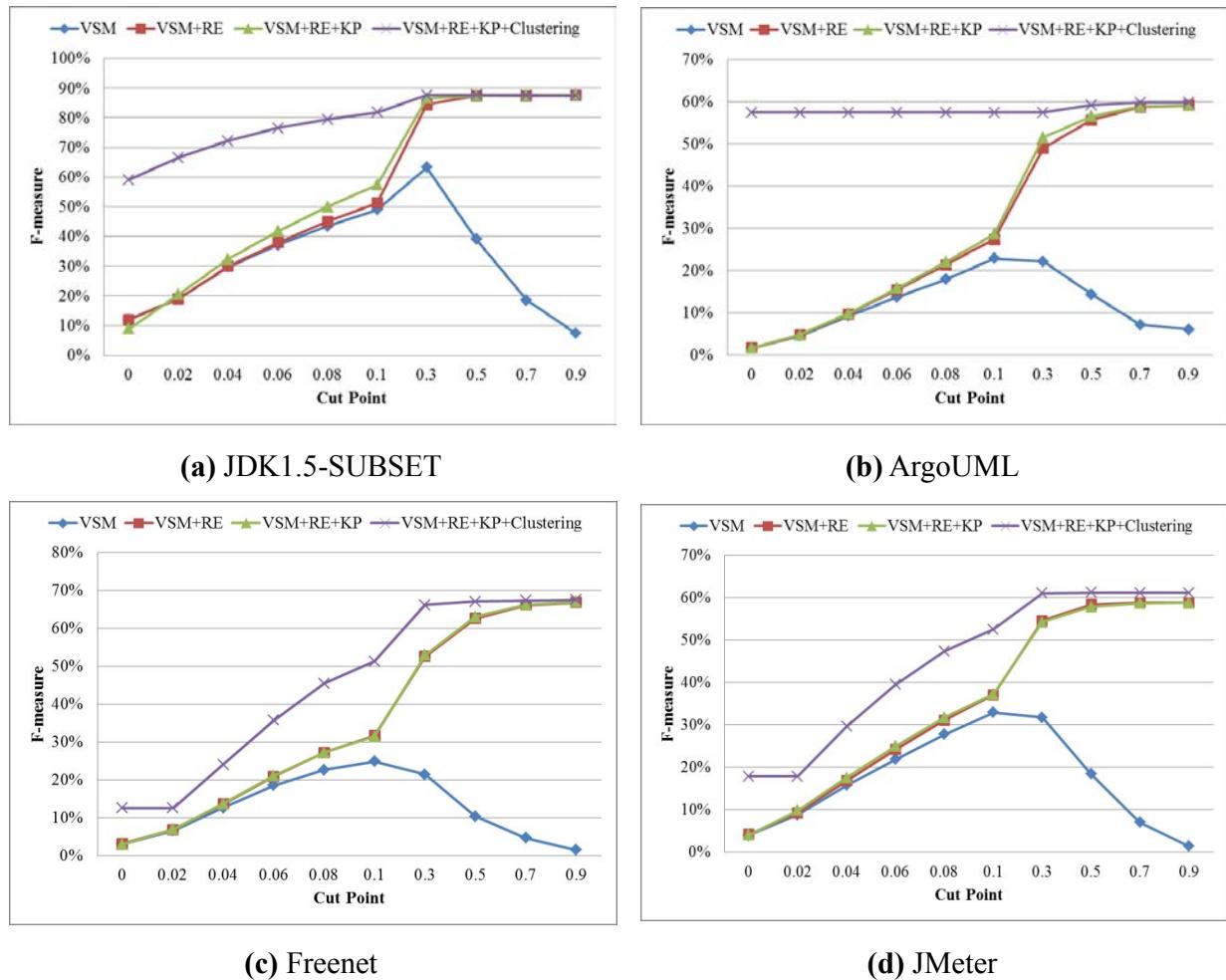


FIGURE 5.3 F-MEASURE (B=1) RESULTS FOR THE FOUR CASES USING VSM

In summary, precision is gradually improved through incrementally adding techniques into the combination approach, and is greatest when incorporating all three techniques with VSM. Adding RE to VSM increases precision at high cut points from 0.3 to 0.9 and recall at all cut points. Further adding KP increases precision at all cut points for JDK1.5-SUBSET and ArgoUML but slightly decreases recall. Freenet and JMeter are unresponsive to the KP technique. Finally,

precision at low cut points from 0 to 0.1 is greatly increased by adding Clustering. Our combination approach is able to obtain good precision at all cut points. Moreover, recall for our approach is much higher than that for VSM at high cut points, but slightly less than that for VSM at low cut points for JDK1.5-SUBSET and ArgoUML, at 0 to 0.02 cut points for Freenet, and at 0 to 0.04 cut points for JMeter. In addition, the F-measure results of all approaches in Figure 5.3 show that our approach has the biggest F-measure values than other three combination approaches. This means that our approach is the most effective among all approaches we evaluated if precision and recall are considered equally important ($\beta=1$). Even if we weight recall twice as much as precision ($\beta=2$) or precision twice as much as recall ($\beta=0.5$), our combination approach still has the best performance.

5.2.2 Evaluation Results from Using the Terrier IR Platform

Although the above evaluation results show that adding the three supporting techniques (RE, KP, and Clustering) to VSM can improve the performance of VSM, can the performance of other IR models be improved after incorporating the three techniques? To answer this question, we conducted another evaluation using other IR models. We chose the Terrier IR platform as our IR engine because it provides multiple IR models to implement indexing and text retrieval. We used five additional IR models, TF_IDF, PL2, BM25, DLH, and IFB2, to evaluate whether their performances can be ameliorated by the three supporting techniques. In this section, we present the evaluation results of the four combination approaches by using the five IR models.

TF_IDF

First, we show the evaluation results of applying TF_IDF as the basic retrieval approach. Table 5.3 summarizes the precision and recall values at the 0.02 and 0.9 cut points in the four case studies. Figure 5.4 illustrates the precision/recall results at all cut points. This figure shows recall results on the x-axis. The y-axis shows precision results and cut points from low to high, the lower a cut point is, the closer it is to the x-axis. Figure 5.4 shows that **TF_IDF only retrieves many incorrect links at low cut points and few true links at high cut points; low precision at low cut points and low recall at high cut points**. In Table 5.3, the precision values are less than 5.9% at the 0.02 cut point, and the recall values at the 0.9 cut point are less than 21.5% in all cases.

Adding RE into TF_IDF **largely increases the recall values at all cut points especially for high cut points**. The recall values at the 0.9 cut point reach at least 69% in all four cases (see Table 5.3). In other words, TF_IDF+RE recovers 49.3%-74.4% more true links than TF_IDF alone at the 0.9 cut point in every case. Moreover, there is a **slight increase in precision at all cut points** except that JMeter suffers a decrease at the 0.7 and 0.9 cut points (see Figure 5.4).

TABLE 5.3 PRECISION AND RECALL RESULTS AT CUT POINTS 0.02 AND 0.9 USING TF_IDF

| Approach | Cut point | JDK1.5-SUBSET | | ArgoUML | | Freenet | | JMeter | |
|-------------------------|-----------|---------------|--------|-----------|--------|-----------|--------|-----------|--------|
| | | Precision | Recall | Precision | Recall | Precision | Recall | Precision | Recall |
| TF_IDF | 0.02 | 5.83% | 72.94% | 0.76% | 80.84% | 1.48% | 82.75% | 1.83% | 77.44% |
| | 0.9 | 86.49% | 8.49% | 50.38% | 21.43% | 57.14% | 11.63% | 70% | 14.92% |
| TF_IDF+RE | 0.02 | 7.16% | 90.85% | 0.85% | 90.91% | 1.59% | 88.76% | 2.1% | 90.05% |
| | 0.9 | 92.32% | 82.89% | 52.28% | 70.78% | 62.57% | 68.99% | 49.69% | 71.67% |
| TF_IDF+RE+KP | 0.02 | 3.79% | 96.95% | 0.79% | 90.91% | 1.5% | 88.76% | 1.86% | 90.41% |
| | 0.9 | 86.38% | 84.08% | 50.23% | 69.48% | 62.24% | 68.99% | 49.94% | 74.07% |
| TF_IDF+RE+KP+Clustering | 0.02 | 28.52% | 89.26% | 26.99% | 73.7% | 46.48% | 74.22% | 43.26% | 76.91% |
| | 0.9 | 89.86% | 83.42% | 53.9% | 69.48% | 64.55% | 68.8% | 52.84% | 74.42% |

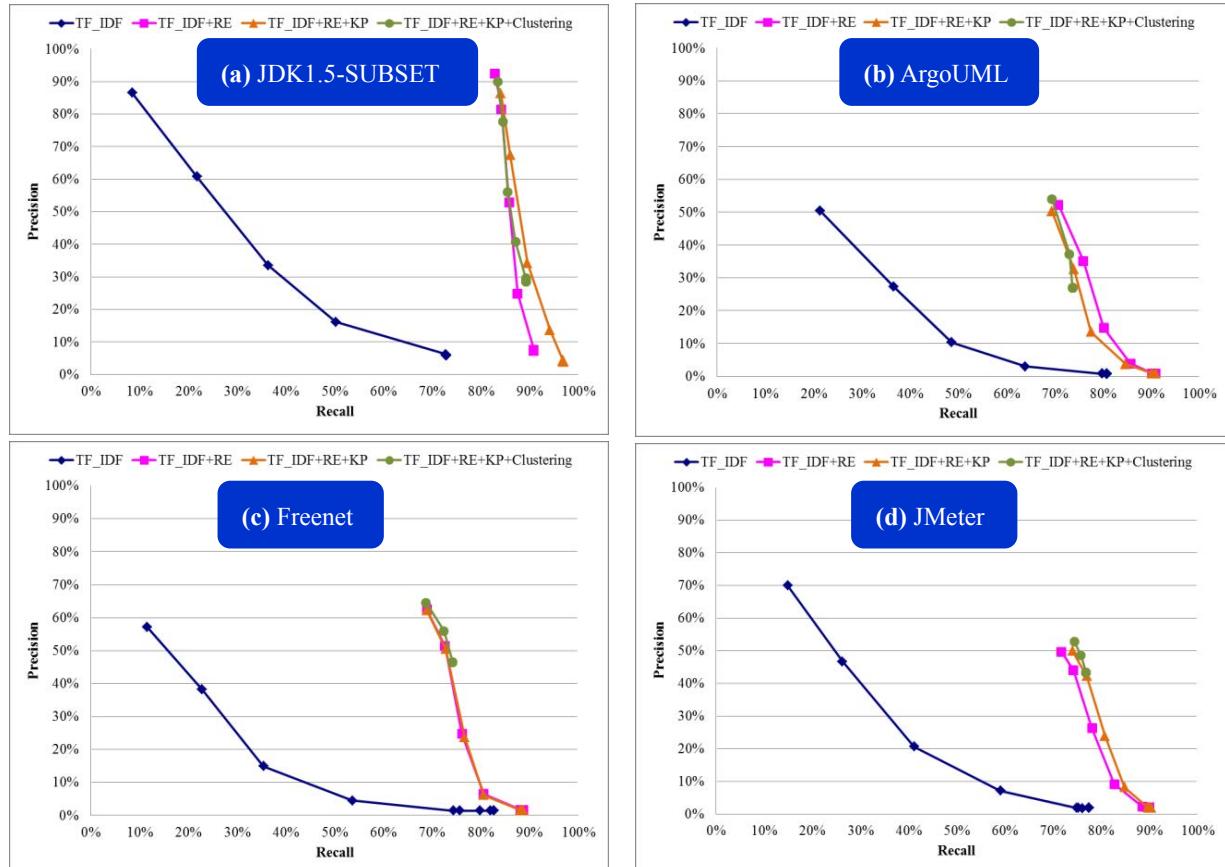


FIGURE 5.4 PRECISION/RECALL RESULTS FOR THE FOUR CASES USING TF_IDF

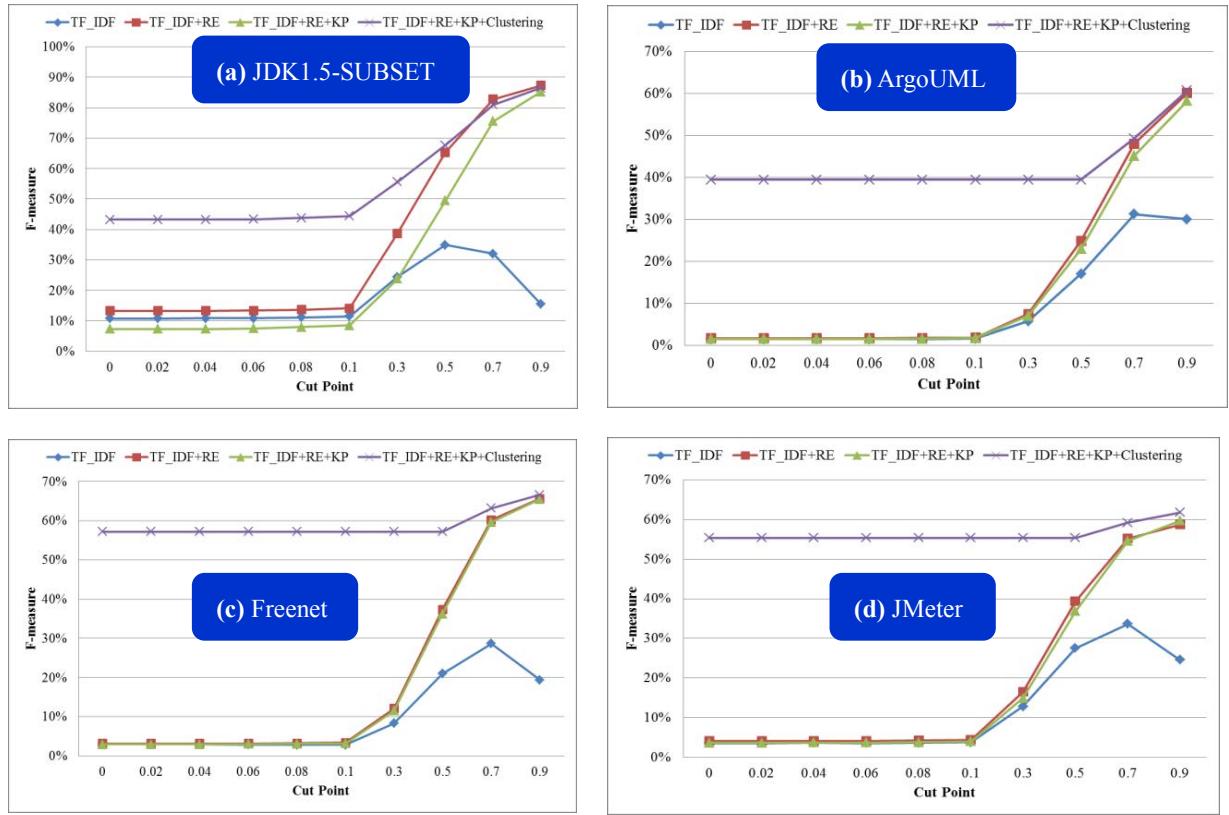


FIGURE 5.5 F-MEASURE ($B=1$) RESULTS FOR THE FOUR CASES USING TF_IDF

The results change dramatically after adding Clustering into TF_IDF+RE+KP. **Precision is significantly increased at all cut points especially for low cut points** (see Figure 5.4). All cases achieve a 24.6%-45% increase at the 0.02 cut point (see Table 5.3): namely, many incorrect links are discarded by Clustering. **Recall has a slight decrease** but still reaches at least 68.8% at all cut points.

The F-measure results of all approaches using TF_IDF as the fundamental retrieval approach in Figure 5.5 show that TF_IDF+RE+KP+Clustering reaches the highest F-measure values at all cut points. It reflects that TF_IDF+RE+KP+Clustering is the most effective among all approaches we evaluated if $\beta=1$. This combination approach still has the best performance even if $\beta=0.5$ or 2.

PL2

Second, we present the evaluation results of adopting PL2 as the basic retrieval approach. Table 5.4 summarizes the precision and recall values at the 0.02 and 0.9 cut points in the four case

studies by using the four combination approaches. In Figure 5.6, we illustrate the precision/recall results at all cut points. It is obvious in Figure 5.6 that **PL2 has low precision at low cut points and low recall at high cut points**. The precision values are less than 5.9% at the 0.02 cut point, and the recall values at the 0.9 cut point are less than 13% in all cases (see Table 5.4).

TABLE 5.4 PRECISION AND RECALL RESULTS AT CUT POINTS 0.02 AND 0.9 USING PL2

| Approach | Cut point | JDK1.5-SUBSET | | ArgoUML | | Freenet | | JMeter | |
|----------------------|-----------|---------------|--------|-----------|--------|-----------|--------|-----------|--------|
| | | Precision | Recall | Precision | Recall | Precision | Recall | Precision | Recall |
| PL2 | 0.02 | 5.84% | 72.94% | 0.76% | 80.84% | 1.29% | 72.29% | 1.8% | 76.02% |
| | 0.9 | 85.19% | 3.05% | 72.73% | 12.99% | 69.57% | 3.1% | 91.67% | 5.86% |
| PL2+RE | 0.02 | 7.17% | 90.85% | 0.85% | 90.91% | 1.57% | 88.18% | 2.11% | 90.05 |
| | 0.9 | 92.97% | 82.49% | 57.89% | 67.86% | 66.48% | 68.02% | 52.13% | 71.58% |
| PL2+RE+KP | 0.02 | 3.79% | 96.95% | 0.79% | 90.91% | 1.48% | 88.18% | 1.87% | 90.41% |
| | 0.9 | 92.05% | 82.89% | 56.4% | 67.21% | 66.48% | 68.02% | 50.9% | 70.52% |
| PL2+RE+KP+Clustering | 0.02 | 39.48% | 88.86% | 41.23% | 71.75% | 60.3% | 70.93% | 50.24% | 75.31% |
| | 0.9 | 92.86% | 82.76% | 57.85% | 68.18% | 66.98% | 68.02% | 53.45% | 73% |

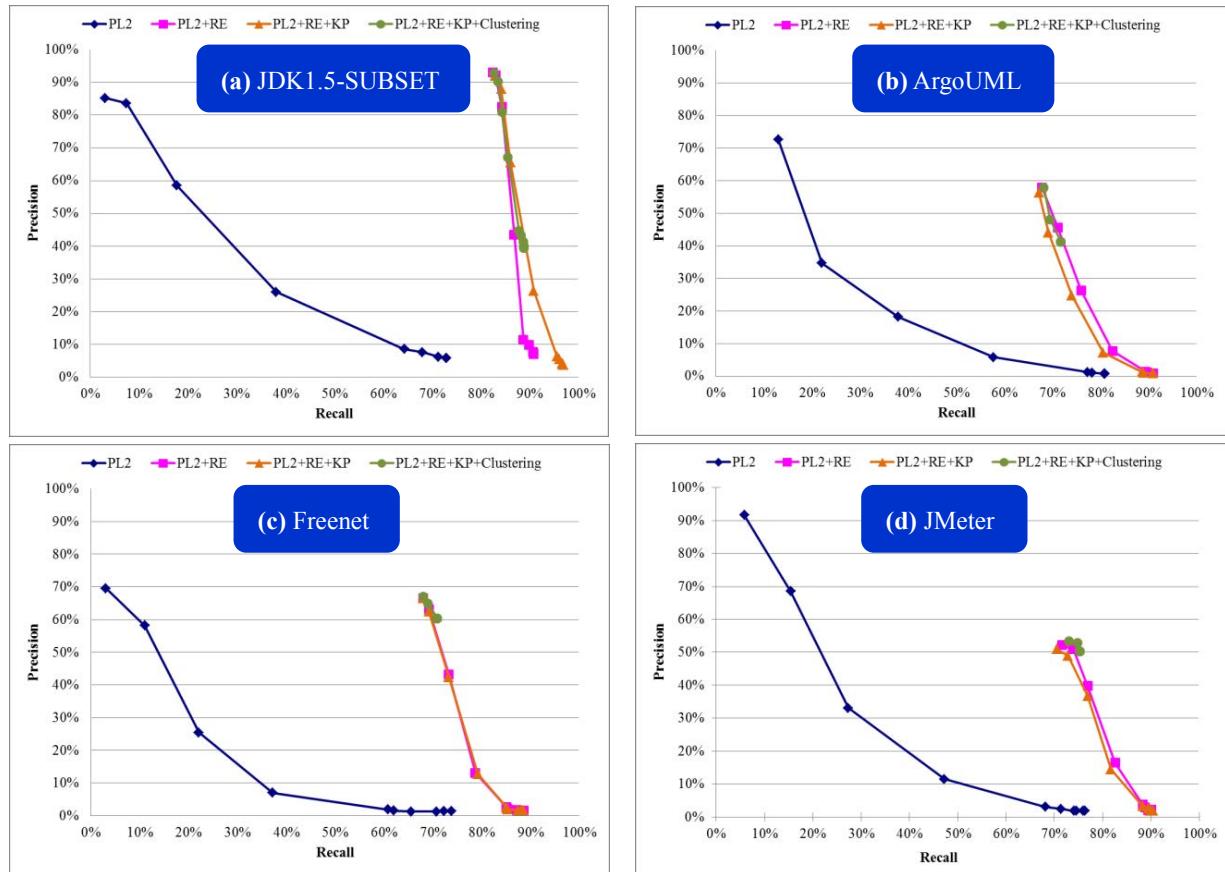


FIGURE 5.6 PRECISION/RECALL RESULTS FOR THE FOUR CASES USING PL2

Integrating RE with PL2 **largely improves the recall values at all cut points** especially for high cut points. The recall values at the 0.9 cut point reach at least 67.8% in the four cases; PL2+RE recovers 54.8%-79.4% more true links than PL2 alone at the 0.9 cut point in every case (see Table 5.4). Moreover, **there is a slight increase in precision at all cut points** except for a decrease at the 0.9 cut point for ArgoUML and Freenet, and at the 0.7 and 0.9 cut points for JMeter (see Figure 5.6).

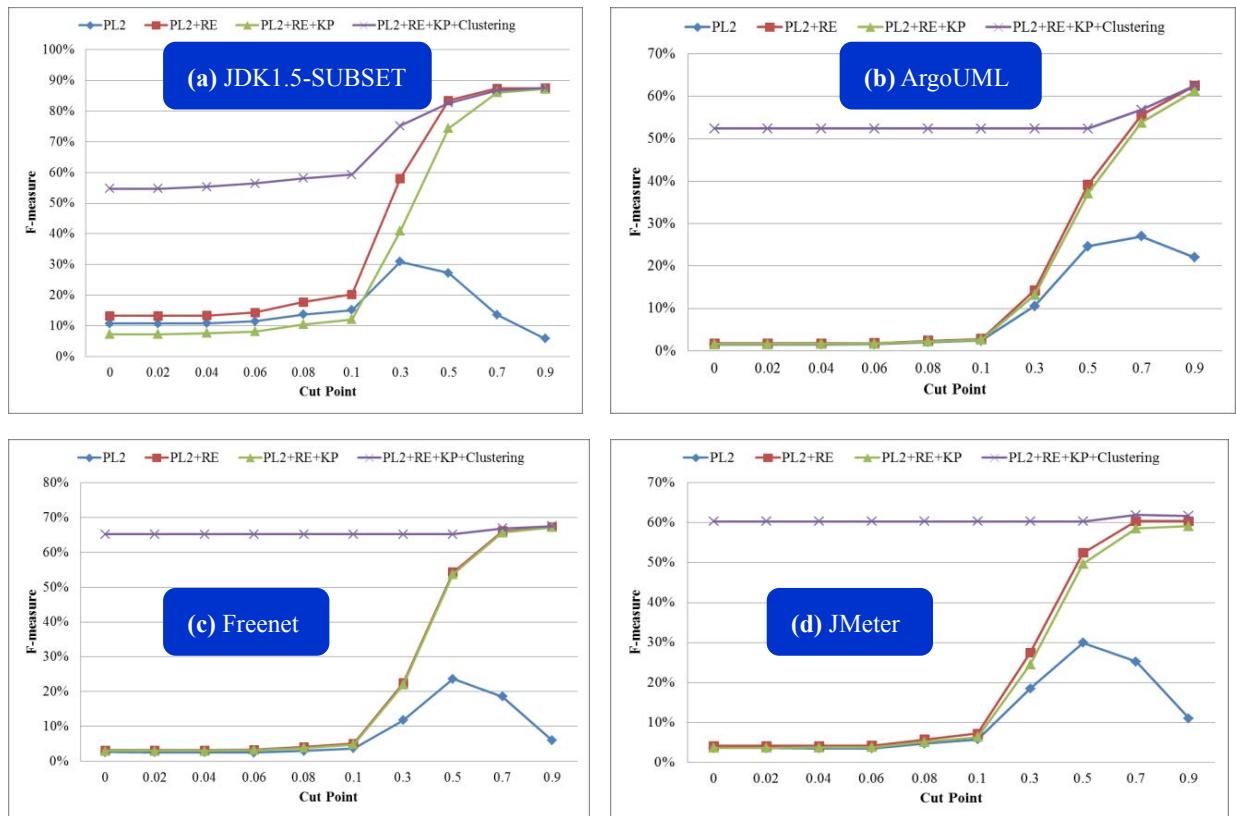


FIGURE 5.7 F-MEASURE (B=1) RESULTS FOR THE FOUR CASES USING PL2

After combining KP with PL2+RE, JDK1.5-SUBSET **obtains a slight increase in the recall at all cut points**, ArgoUML and JMeter suffer from a decrease, and Freenet remains unchanged (see Figure 5.6). However, **precision at all cut points decreases slightly** in every case.

After adding Clustering into PL2+RE+KP, precision is **dramatically increased at all cut points especially for low cut points** (see Figure 5.6). All cases achieve a 35.7%-58.8% increase at the 0.02 cut point (see Table 5.4). Recall has a slight decrease but still reaches at least 68% at all cut points. This shows that Clustering **reduces many incorrect links at low cut points**.

Figure 5.7 shows the F-measure results of all approaches using PL2 as the basic retrieval approach. It is clear that PL2+RE+KP+Clustering obtains the highest F-measure values. In other words, **it is the most effective among all approaches** we evaluated if $\beta=1$. This combination approach still has the best performance even if $\beta=0.5$ or 2.

BM25

Third, we show the evaluation results of applying BM25 as the basic retrieval approach. Table 5.5 summarizes the precision and recall values at the 0.02 and 0.9 cut points for the four case studies. It shows the precision and recall results using the four combination approaches. Figure 5.8 illustrates the precision/recall results at all cut points. BM25 possesses the same limitation as VSM, TF_IDF, and PL2 in **low precision at low cut points and low recall at high cut points** (see Figure 5.8). In Table 5.5, the precision values are less than 6.1% at the 0.02 cut point, and the recall values at the 0.9 cut point are less than 11.7% in all cases.

Adding RE into BM25 **largely increases the recall values at all cut points**, especially for high cut points. The four cases obtain high recall values ($>67.5\%$) at the 0.9 cut point (see Table 5.5). In other words, BM25+RE recovers 55.8%-80.4% more true links than BM25 alone at the 0.9 cut point in every case. Moreover, there is **a slight increase in precision at all cut points** except that JDK1.5-SUBSET and JMeter suffer from a decrease at the 0.7 and 0.9 cut points, and ArgoUML and Freenet have a decrease at the 0.9 cut point (see Figure 5.8).

TABLE 5.5 PRECISION AND RECALL RESULTS AT CUT POINTS 0.02 AND 0.9 USING BM25

| Approach | Cut point | JDK1.5-SUBSET | | ArgoUML | | Freenet | | JMeter | |
|------------------------|-----------|---------------|--------|-----------|--------|-----------|--------|-----------|--------|
| | | Precision | Recall | Precision | Recall | Precision | Recall | Precision | Recall |
| BM25 | 0.02 | 6.04% | 72.94% | 0.76% | 80.84% | 1.69% | 80.62% | 2.02% | 76.73% |
| | 0.9 | 100% | 1.99% | 80% | 11.69% | 66.67% | 2.71% | 96.88% | 5.51% |
| BM25+RE | 0.02 | 7.41% | 90.85% | 0.85% | 90.91% | 1.85% | 88.37% | 2.33% | 89.88% |
| | 0.9 | 93.52% | 82.36% | 58.59% | 67.53% | 66.48% | 68.02% | 50.52% | 69.63% |
| BM25+RE+KP | 0.02 | 4.42% | 96.68% | 0.79% | 90.91% | 1.77% | 88.37% | 2.05% | 89.88% |
| | 0.9 | 93.1% | 82.36% | 58.1% | 67.53% | 66.48% | 68.02% | 50.52% | 69.63% |
| BM25+RE+KP +Clustering | 0.02 | 44.66% | 88.73% | 45.88% | 72.4% | 62.5% | 69.77% | 51.53% | 74.78% |
| | 0.9 | 93.1% | 82.36% | 58.82% | 68.18% | 66.98% | 68.02% | 53.51% | 73.18% |

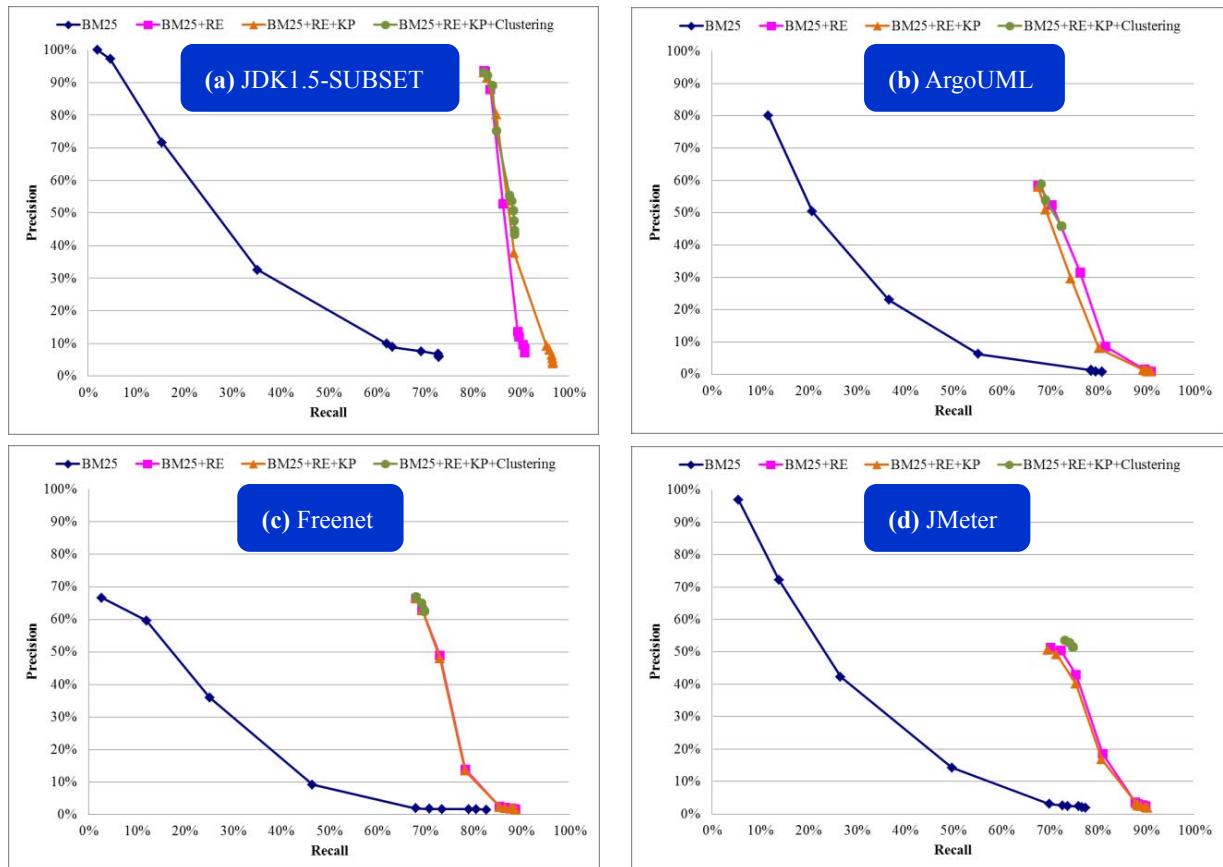


FIGURE 5.8 PRECISION/RECALL RESULTS FOR THE FOUR CASES USING BM25

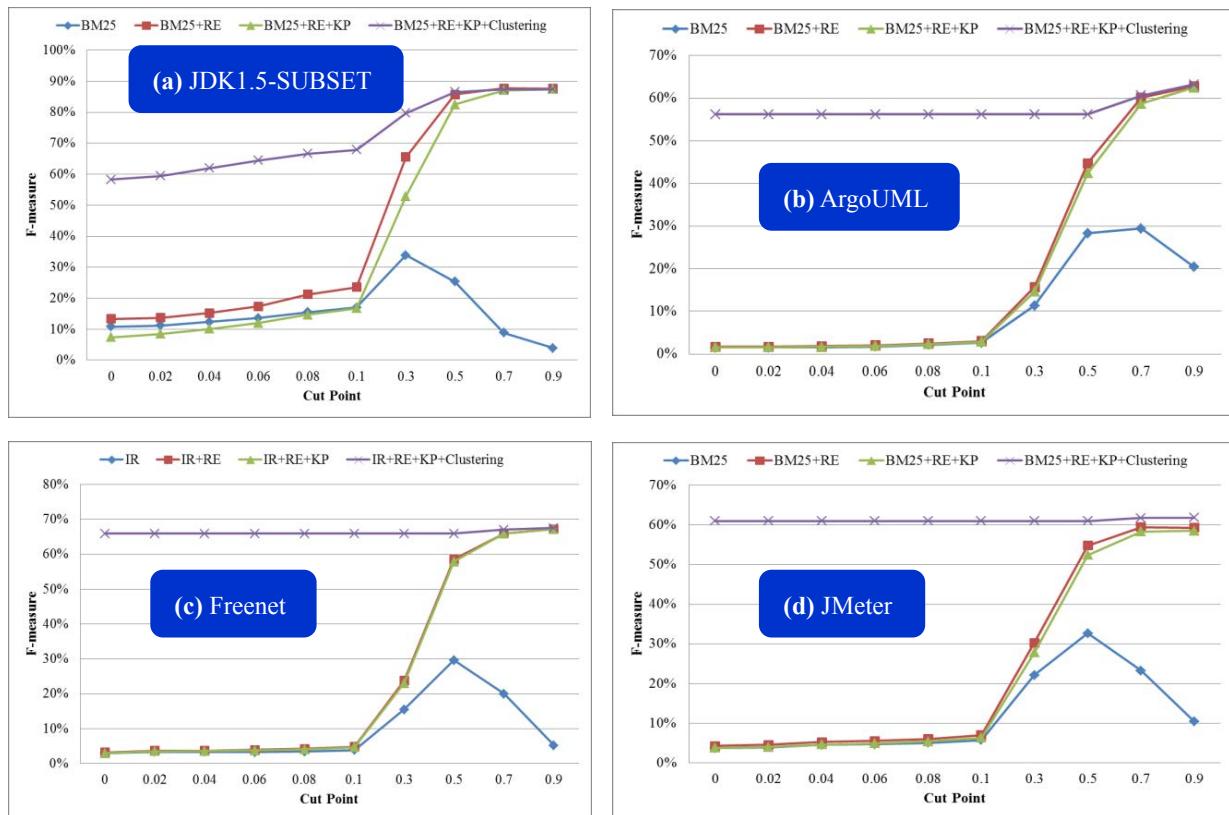


FIGURE 5.9 F-MEASURE ($B=1$) RESULTS FOR THE FOUR CASES USING BM25

After combining KP with BM25+RE, JDK1.5-SUBSET has **a slight increase in the recall at all cut points**, ArgoUML and JMeter suffer from a decrease, while Freenet shows no difference to the KP enhancement (see Figure 5.8). However, **precision at all cut points decreases slightly** in every case.

Adding Clustering into BM25+RE+KP **increases precision significantly at all cut points**, especially for low cut points; all cases achieve a 40.2%-60.7% increase at the 0.02 cut point (see Table 5.5). Recall has a slight decrease but still reaches at least 68% at all cut points (see Figure 5.8). **A large number of incorrect links are discarded** after applying Clustering.

The F-measure results of all approaches using BM25 as the basic retrieval approach in Figure 5.9 show that BM25+RE+KP+Clustering is the most effective among all approaches we evaluated if $\beta=1$ because it achieves the highest F-measure values at all cut points. This combination approach **still has the best performance** even if $\beta=0.5$ or 2.

DLH

Fourth, we present the evaluation results of adopting DLH as the basic retrieval approach. Table 5.6 summarizes the precision and recall values at the 0.02 and 0.9 cut points in the four case studies by using the four combination approaches. Figure 5.10 illustrates the precision/recall results at all cut points. It is obvious in Figure 5.10 that DLH has **low precision at low cut points and low recall at high cut points**. The precision values are less than 7.4% at the 0.02 cut point, and the recall values at the 0.9 cut point are less than 17.9% in all cases.

TABLE 5.6 PRECISION AND RECALL RESULTS AT CUT POINTS 0.02 AND 0.9 USING DLH

| Approach | Cut point | JDK1.5-SUBSET | | ArgoUML | | Freenet | | JMeter | |
|-----------------------|-----------|---------------|--------|-----------|--------|-----------|--------|-----------|--------|
| | | Precision | Recall | Precision | Recall | Precision | Recall | Precision | Recall |
| DLH | 0.02 | 7.37% | 68.3% | 0.93% | 76.95% | 1.65% | 71.51% | 2.26% | 73.36% |
| | 0.9 | 85% | 4.51% | 47.41% | 17.86% | 65.52% | 7.36% | 72.94% | 11.01% |
| DLH+RE | 0.02 | 9.48% | 90.05% | 1.07% | 89.29% | 1.99% | 86.63% | 2.7% | 88.99% |
| | 0.9 | 92.7% | 82.49% | 51.99% | 67.86% | 65.13% | 68.41% | 51.51% | 72.82% |
| DLH+RE+KP | 0.02 | 5.33% | 95.89% | 0.98% | 88.96% | 1.89% | 86.43% | 2.36% | 89.52% |
| | 0.9 | 90.68% | 82.63% | 51.48% | 67.86% | 65.13% | 68.41% | 51.06% | 72.82% |
| DLH+RE+KP +Clustering | 0.02 | 40.82% | 88.46% | 33.99% | 73.05% | 55.03% | 72.09% | 46.21% | 75.84% |
| | 0.9 | 91.89% | 82.63% | 54.12% | 68.18% | 66.6% | 68.41% | 53.12% | 74.07% |

Adding RE into DLH changes recall dramatically: **recall is largely increased at all cut points**, especially for high cut points (see Figure 5.10). The recall values at the 0.9 cut point reach at least 67.8% in the four cases; DLH+RE recovers 50%-78% more true links than DLH alone at the 0.9 cut point in every case (see Table 5.6). Moreover, **there is a slight increase in precision at all cut points** except for a decrease at the 0.9 cut point for Freenet, and at the 0.7 and 0.9 cut points for JMeter.

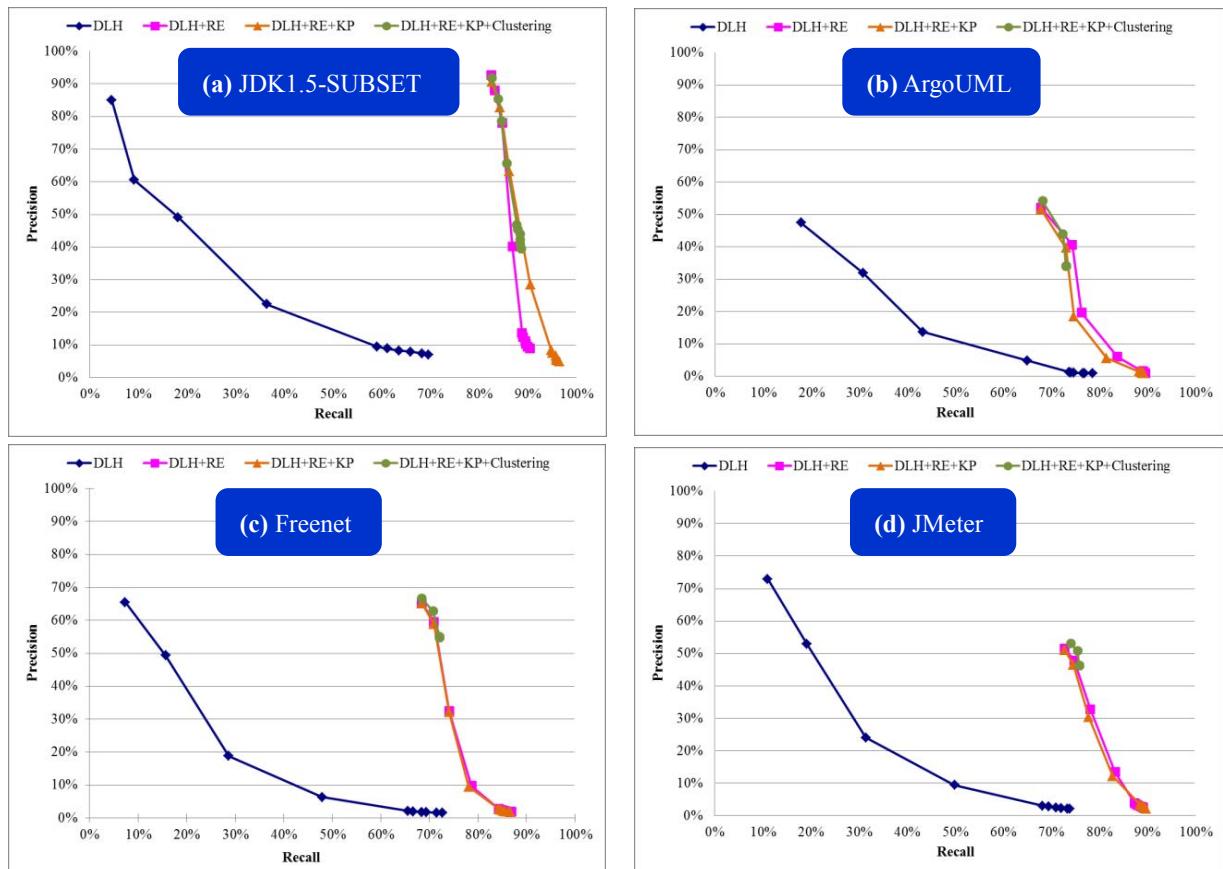


FIGURE 5.10 PRECISION/RECALL RESULTS FOR THE FOUR CASES USING DLH

After combining KP with DLH+RE, JDK1.5-SUBSET performs **better than other cases**. It obtains **a slight increase in recall at all cut points**, but ArgoUML and JMeter suffer a decrease and there is no obvious improvement in Freenet (see Figure 5.10). However, **precision at all cut points decreases slightly** in every case.

After adding Clustering into DLH+RE+KP, **precision is dramatically increased at all cut points**, especially for low cut points (see Figure 5.10). All cases achieve a 33%-53.1% increase at the 0.02

cut point (see Table 5.6). **Recall has a slight decrease** but still reaches at least 68.2% at all cut points. This indicates that many incorrect links are discarded by Clustering.

In Figure 5.11, the F-measure results of all approaches using DLH as the basic retrieval approach show that **DLH+RE+KP+Clustering is the most effective among all approaches we evaluated if $\beta=1$** because it obtains the highest values at all cut points. This combination approach still has the best performance even if $\beta=0.5$ or 2.

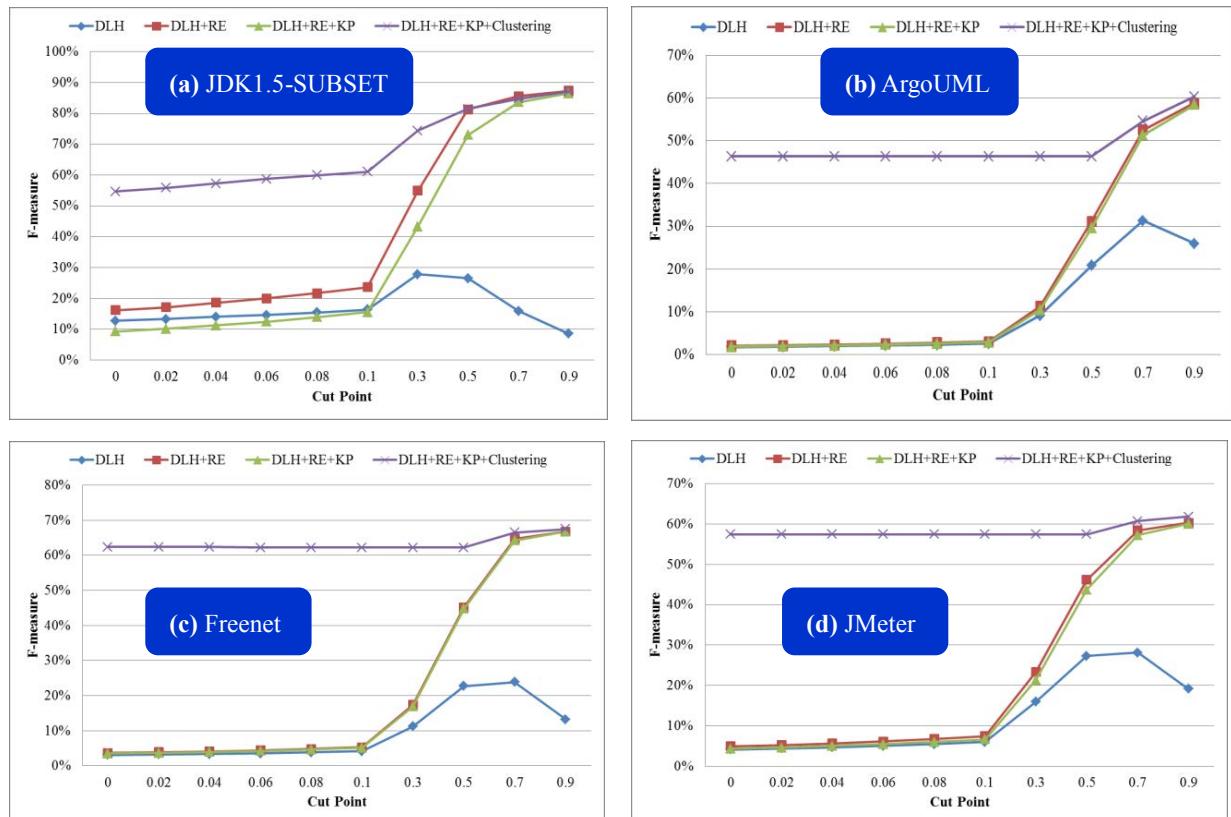


FIGURE 5.11 F-MEASURE ($\beta=1$) RESULTS FOR THE FOUR CASES USING DLH

IFB2

Finally, we present the evaluation results of using IFB2 as the basic retrieval approach. Table 5.7 summarizes the precision and recall values at the 0.02 and 0.9 cut points in the four case studies by applying the four combination approaches. Figure 5.12 illustrates the precision/recall results at all cut points. IFB2 owns the same traits shared by other IR models discussed above: **low precision at low cut points and low recall at high cut points**. The precision values are less than 5.8% at the

0.02 cut point, and the recall values at the 0.9 cut point are less than 38.6% in all cases (see Table 5.7).

TABLE 5.7 PRECISION AND RECALL RESULTS AT CUT POINTS 0.02 AND 0.9 USING IFB2

| Approach | Cut point | JDK1.5-SUBSET | | ArgoUML | | Freenet | | JMeter | |
|---------------------------|-----------|---------------|--------|-----------|--------|-----------|--------|-----------|--------|
| | | Precision | Recall | Precision | Recall | Precision | Recall | Precision | Recall |
| IFB2 | 0.02 | 5.75% | 39.79% | 1% | 75% | 1.8% | 77.91% | 2.95% | 71.23% |
| | 0.9 | 60.18% | 9.02% | 22.75% | 38.64% | 27.01% | 33.91% | 38.06% | 36.23% |
| IFB2+RE | 0.02 | 11.56% | 85.54% | 1.2% | 89.61% | 2% | 87.02% | 3.5% | 86.68% |
| | 0.9 | 88.03% | 82.89% | 30.57% | 76.62% | 37.75% | 73.45% | 39.66% | 78.33% |
| IFB2+RE+KP | 0.02 | 7.58% | 88.33% | 1.09% | 89.61% | 1.93% | 87.02% | 3.01% | 88.1% |
| | 0.9 | 86.03% | 83.29% | 29.28% | 75% | 37.19% | 73.45% | 37.23% | 76.38% |
| IFB2+RE+KP +Clustering | 0.02 | 42.34% | 84.35% | 19.64% | 74.03% | 27.67% | 74.42% | 32.91% | 77.8% |
| | 0.9 | 87.2% | 83.16% | 38.5% | 73.38% | 48.14% | 72.67% | 45.13% | 76.55% |

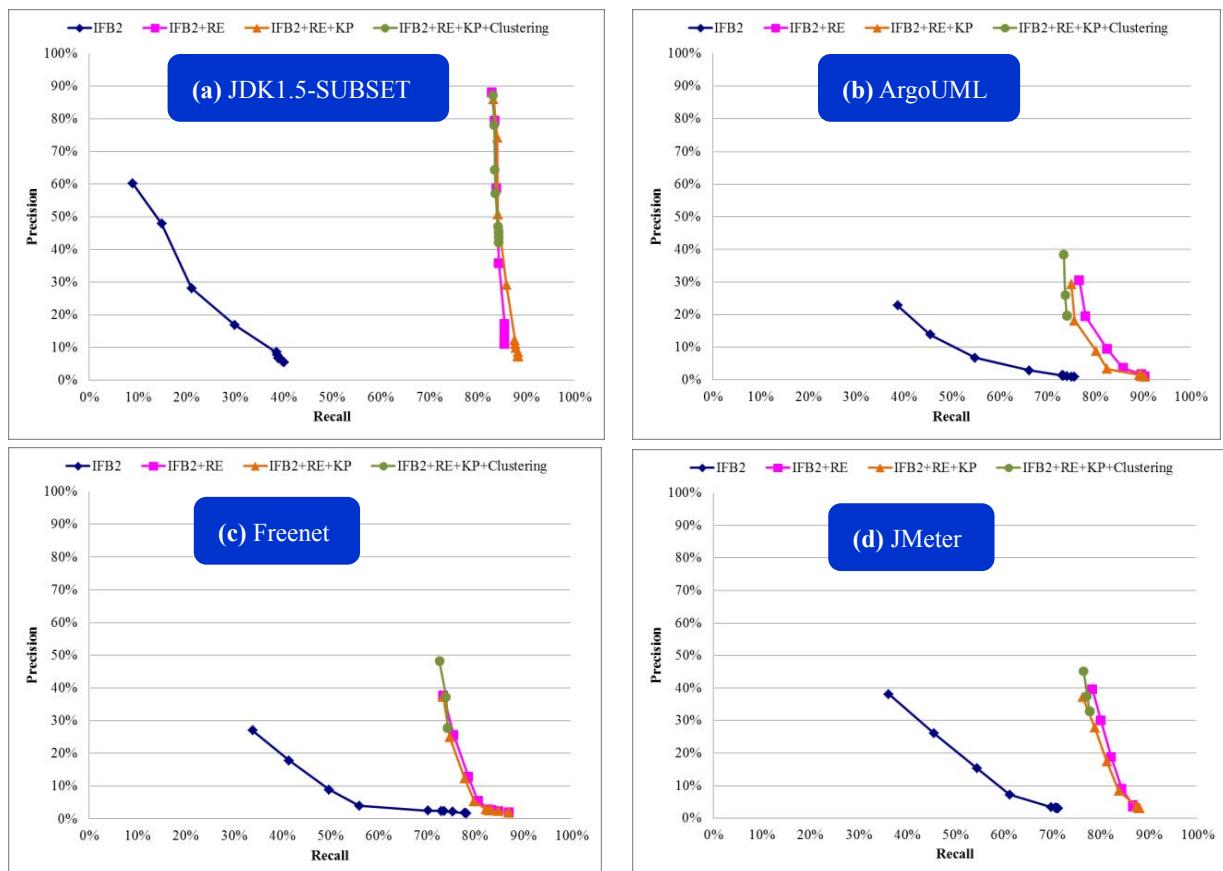


FIGURE 5.12 PRECISION/RECALL RESULTS FOR THE FOUR CASES USING IFB2

Adding RE into IFB2 significantly changes recall: **it is substantially increased at all cut points**, especially for high cut points (see Figure 5.12). The recall values at the 0.9 cut point reach at least

73.4% in the four cases. In other words, IFB2+RE recovers 38%-73.9% more true links than IFB2 alone at the 0.9 cut point in every case (see Table 5.7). Moreover, **there is a slight increase in precision at all cut points** in the four cases.

After combining KP with IFB2+RE, JDK1.5-SUBSET has better performance than other cases; it gets a **slight increase in recall at all cut points** but ArgoUML, Freenet, and JMeter suffer a decrease (see Figure 5.12). However, **precision at all cut points decreases slightly** in every case.

After adding Clustering into IFB2+RE+KP, **precision is dramatically increased at all cut points, especially for low cut points** (see Figure 5.12). All cases achieve an 18.5%-34.8% increase at the 0.02 cut point (see Table 5.7). In other words, many incorrect links are reduced at low cut points.

Recall has a slight decrease but still reaches at least 72.6% at all cut points.

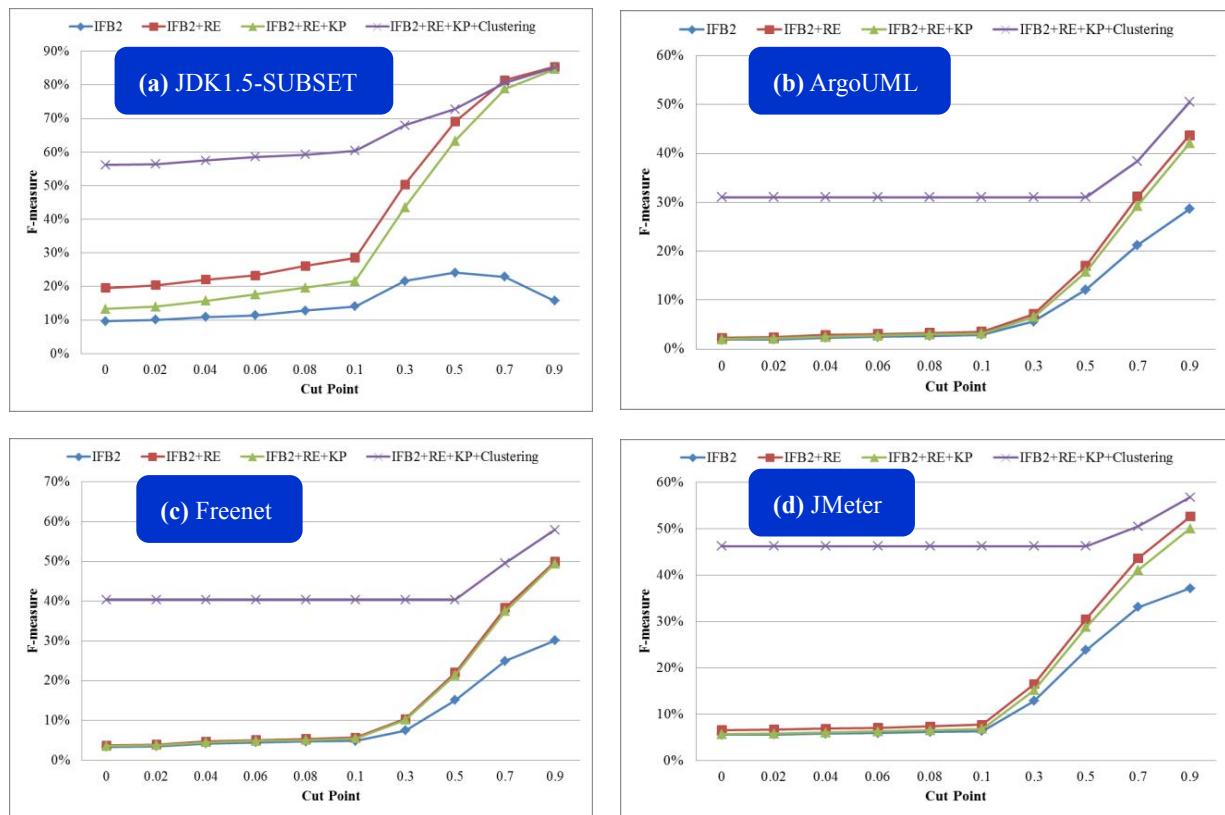


FIGURE 5.13 F-MEASURE ($B=1$) RESULTS FOR THE FOUR CASES USING IFB2

It is obvious in Figure 5.13 from the F-measure results of all approaches using IFB2 as the basic retrieval approach that IFB2+RE+KP+Clustering gains the highest F-measure values. It shows that

IFB2+RE+KP+Clustering is the most effective among all approaches we evaluated if $\beta=1$. This combination approach still has the best performance even if $\beta=0.5$ or 2.

In summary, the three enhancement techniques demonstrate different capabilities for improving the performance of IR models. The clearest conclusions from all case studies are: RE increases recall at all cut points, and Clustering improves precision at all cut points. The ArgoUML, Freenet, and JMeter cases are nearly unresponsive or have a negative response to the KP enhancement technique; only JDK1.5-SUBSET shows an improvement in recall. Nevertheless, the F-measure results show that IR+RE+KP+Clustering is more effective than the other three combination approaches (IR only, IR+RE, and IR+RE+KP).

5.2.3 Comparison Results

Figure 5.14 illustrates the comparative results among IR models and among the combination approach (IR+RE+KP+Clustering) using different IR models in each case.

Figure 5.14a shows that when using IR only to extract links in JDK1.5-SUBSET, **VSM has much better performance than other five IR models** except that BM25 achieves higher precision than VSM at the 0.7 and 0.9 cut points. The other five IR models have very similar results. However, after incorporating the three supporting techniques, RE, KP, and Clustering, with the six IR models, **they produce very similar results**. Precision is between 28% and 94% and recall is between 82% and 90% at all cut points. **Our combination approach using the six different IR models can achieve a very high recall (>82%) at all cut points.**

In Figure 5.14b, all six IR models have similar results when applying IR only to capture links in ArgoUML; **low precision at low cut points and low recall at high cut points**, but VSM has **much lower recall than the other five IR models at high cut points**. Precision and recall are changed dramatically after adding RE, KP, and Clustering to the six IR models. The precision values at all cut points are between 19% and 59%, and recall is between 62% and 74%. **Although VSM+RE+KP+Clustering achieves much better precision results (53%-58% at all cut points), it has lower recall (around 62% at all cut points).**



FIGURE 5.14 COMPARISON RESULTS BETWEEN IR ONLY AND IR+RE+KP+CLUSTERING

For Freenet, the six IR models have similar results when retrieving links (see Figure 5.14c). However, **VSM gets lower recall than the other five IR models at high cut points**. Our combination approaches (IR+RE+KP+Clustering) using different basic retrieval IR models **produce very close results**. They improve precision and recall but especially recall; the recall values are between 68% and 81% at all cut points.

When using the six IR models to extract links in JMeter, their performances are very similar; **low precision at low cut points and low recall at high cut points** (see Figure 5.14d). After combining RE, KP, and Clustering, their performances **are very close**; precision is between 10% and 54% and recall is between 72% and 80% at all cut points.

Overall, **our combination approach (IR+RE+KP+Clustering) can improve the performance of the six IR models**; precision is increased at low cut points and recall is significantly increased at high cut points. Moreover, our approach can narrow the gap between the results of applying different IR models.

5.3 Performance of Our Approach

We ran the four combination approaches (IR only, IR+RE, IR+RE+KP, IR+RE+KP+Clustering) on an iMac with a 2.4 GHz Intel Core Duo processor and 3GB of RAM. Figure 5.15a shows that our combination approach using VSM took up to 5 minutes to execute on each case. For all cases, this is 4-9 times more than VSM, 2-6 times more than VSM+RE, and up to 10 seconds more than VSM+RE+KP. 80% of the time for JDK1.5-SUBSET and at least 60% of the time for other cases are spent on KP extraction.

For the other five IR models, our combination approach took approximately 3-5 minutes to execute on each case with different IR models (see Figure 5.15b-f). For all cases, this is up to 10-25 times more than IR, 3-9 times more than IR+RE, and 1-19 seconds more than IR+RE+KP. Around 80% of the time is spent on Key Phrases extraction in every case.

Our combination approach took quite a long time in extracting key phrases from comments in source code. This is because KEA, the key phrases processor in our approach, uses an expensive machine learning algorithm for training and key phrase extraction (Witten et al., 1999). Our combination approach (IR+RE+KP+Clustering) thus produces a much better result than the other three combination techniques (IR only, IR+RE, and IR+RE+KP) but is slower. It is vastly faster than manually extracting links. When building the oracle link set, every participant spent one hour on average to identify related sections of 50 classes at the first stage.

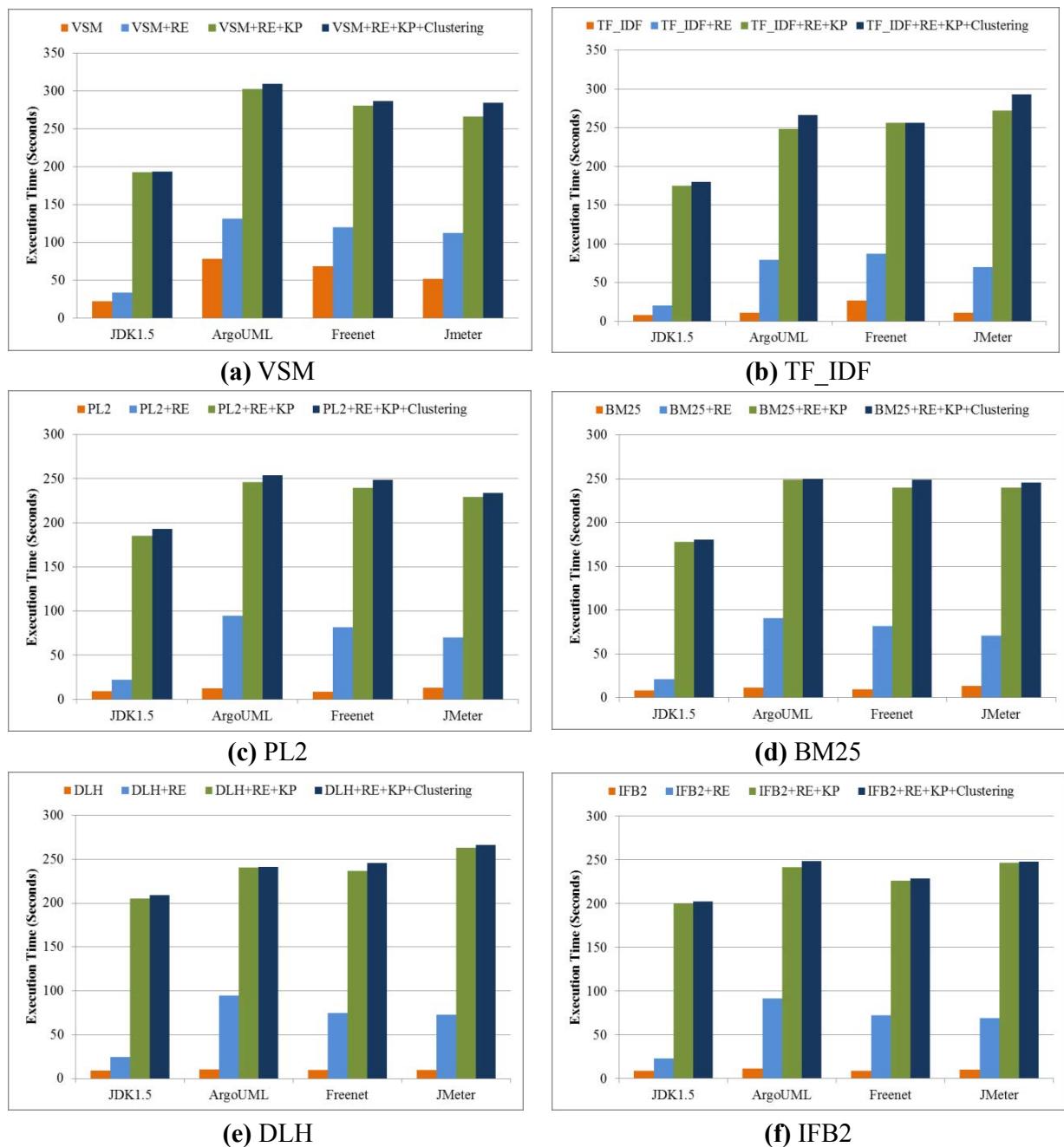


FIGURE 5.15 EXECUTION TIMES FOR DIFFERENT COMBINATIONS WITH DIFFERENT IR MODELS

5.4 Discussion

The evaluation results of using the Lucene IR engine indicate that our approach (VSM+RE+KP+Clustering) improves the precision of retrieved links and achieves high recall by utilizing the strengths of RE, KP, and Clustering to mitigate limitations of VSM. Combining RE with VSM augments the number of retrieved links at high cut points. Adding KP to this combination ameliorates the drawback of links probably missed by VSM. Finally, integrating Clustering discards many incorrect links produced by VSM.

According to the evaluation results of using the Terrier IR platform, the five additional IR models (TF_IDF, PL2, BM25, DLH, and IFB2) share the same traits as VSM: low precision at low cut points and low recall at high cut points. Adding RE largely improves recall at high cut points. There is no obvious improvement after combining KP. Nevertheless, the precision values at low cut points are significantly increased after integrating Clustering.

The three supporting techniques (RE, KP, and Clustering) demonstrate different capabilities of ameliorating the limitations of the six IR models. The most obvious conclusion from the evaluation results is that **RE increases recall at all cut points**. Analysis of the four case studies shows that documents/emails contain many class names that enable RE to match classes to documents/emails. Table 5.8 describes the percentages of sections/emails that don't contain class names in the four cases. Only 17.58% sections in JDK1.5-SUBSET mention no class names. For JMeter, no class names can be found in 22.41% emails, 56.08% for ArgoUML, and 59.95% for Freenet. Although more than half of emails in ArgoUML and Freenet have no class names mentioned, each of the remaining emails contains at least one class name.

TABLE 5.8 PERCENTAGES OF SECTIONS/EMAILS NO CLASS NAMES IN THE FOUR CASES

| Case | Sections | Sections/emails with no class names | Percentage |
|---------------|----------|-------------------------------------|------------|
| JDK1.5-SUBSET | 182 | 32 | 17.58% |
| ArgoUML | 378 | 212 | 56.08% |
| Freenet | 372 | 223 | 59.95% |
| JMeter | 348 | 78 | 22.41% |

The second obvious conclusion is that **Clustering improves precision at all cut points**. Analysis of the documents/emails in the four cases shows that documents/emails have inherent hierarchical structures that provide useful hierarchical information for Clustering to refine retrieved links. The documents in JDK1.5-SUBSET have clear and straight-forward inherent hierarchical structures based on the headings information. Although the hierarchical structures in emails are less straight-forward than that in JDK1.5-SUBSET's documents, such structures can be established based on the reply information, whether an email is original or a reply to another email.

The last key conclusion is that **the four case study software systems have various responses to the KP enhancement technique**. Freenet is nearly unresponsive to KP. ArgoUML and JMeter have a negative response to KP; precision and recall suffer a decrease except for precision in ArgoUML using VSM and recall in JMeter using TF_IDF. Only JDK1.5-SUBSET shows an improvement in precision and recall; for VSM, precision is increased at all cut points, and for the other five IR models, recall is improved at all cut points. Analysis of the source code reveals a low number of comments in source code of ArgoUML, Freenet, and JMeter. Table 5.9 describes the percentages of classes that do not contain comments in the four cases. In JDK1.5-SUBSET, only one class has no comments. 12.37% classes in JMeter do not include comments. Freenet has 33.08% classes without comments. There are no comments provided in 38.53% classes in ArgoUML. Comments in the rest of classes in JDK1.5-SUBSET are better documented than those in ArgoUML, Freenet, and JMeter.

TABLE 5.9 PERCENTAGES OF CLASSES WITHOUT COMMENTS IN THE FOUR CASES

| Case | Classes | Classes with no comments | Percentage |
|---------------|---------|--------------------------|------------|
| JDK1.5-SUBSET | 249 | 1 | 0.40% |
| ArgoUML | 423 | 163 | 38.53% |
| Freenet | 517 | 171 | 33.08% |
| JMeter | 372 | 46 | 12.37% |

Moreover, there are not many key phrases extracted from classes' comments. Table 5.10 shows the details of key phrases extracted from classes' comments in the four cases. For JDK1.5-SUBSET, there are no key phrases extracted from comments in 73 classes (29.32%). The number of extracted key phrases from comments in classes ranges from 1 to 10. For example, 41 classes in

JDK1.5-SUBSET have 2 extracted key phrases. In the other three cases, more than 80% of the classes have no extracted key phrases; 86.52% for ArgoUML, 90.52% for Freenet, and 82.99% for JMeter. Most of other classes contain only one key phrase; 25 of 57 classes in ArgoUML, 34 of 49 in Freenet, and 41 of 64 in JMeter. In addition, the key phrases extracted from classes' comments contain many key words unrelated to the purpose of classes. Table 5.11 shows how many classes contain related key phrases. JDK1.5-SUBSET has the highest number of classes (51.95%) containing related key phrases. Freenet has the lowest number of classes (6.12%) with related key phrases extracted. ArgoUML is 19.3% and 14.06% in JMeter. These results indicate that JDK1.5-SUBSET is better documented than ArgoUML, Freenet, and JMeter, the key phrases processor is more likely to extract related key phrases from comments than the other three cases.

TABLE 5.10 DETAILS OF CLASSES WITH EXTRACTED KEY PHRASES IN THE FOUR CASES

| Case | No. of extracted key phrases | | | | | | | | | | |
|---------------|------------------------------|----|----|----|----|----|---|---|---|---|----|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| JDK1.5-SUBSET | 73 | 44 | 41 | 31 | 15 | 13 | 7 | 8 | 2 | 4 | 11 |
| ArgoUML | 366 | 25 | 14 | 7 | 4 | 2 | 3 | 2 | 0 | 0 | 0 |
| Freenet | 468 | 34 | 7 | 7 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| JMeter | 308 | 41 | 11 | 4 | 4 | 3 | 0 | 0 | 1 | 0 | 0 |

TABLE 5.11 PERCENTAGES OF CLASSES WITH RELATED KEY PHRASES IN THE FOUR CASES

| Case | Classes with key phrases | Classes with related key phrases | Percentage |
|---------------|--------------------------|----------------------------------|------------|
| JDK1.5-SUBSET | 176 | 93 | 51.96% |
| ArgoUML | 57 | 11 | 19.3% |
| Freenet | 49 | 3 | 6.12% |
| JMeter | 64 | 9 | 14.06% |

The **main limitation of our combination approach is that some true links are discarded after adding Clustering.** This is because the group containing links related to a same class is totally removed when no links in the group have a similarity value larger than the threshold s value: this leads to no clusters for this group being created. True links in such groups are cut.

There are two main threats to validity in these experiments. First, we relied on human judgment to build the oracle link set and thus this set might not be 100% correct. To minimize that threat, we applied a very rigorous manual verification strategy to analyze every true link, which were verified by at least 3 analysts. Second, our traceability recovery technique may show different results when applied to other software systems with other types of documents. To alleviate this, we chose 4 unrelated open-source systems. These systems vary in the sizes of the systems, types of documents, structures of documents, and the availability of comments in source code. However, we cannot confirm that our results are similar in closed-source systems.

In future work, we will examine other key phrases extraction techniques to accelerate the execution time of our approach. We will allow users to edit or delete existing extracted key phrases in each class, or add new key phrases related to the purpose of classes. We will also allow users to edit the IR queries to delete unwanted key words or add new key words. We will also explore other techniques to cope with abbreviation, synonym, and polysemy problems by taking account of relations between terms or words. Furthermore, to increase the similarity value between a class and a section that are really relevant to each other, we will adjust the similarity score of each retrieved links based on the frequency of the class's occurrence in the section before using Clustering to refine retrieved links. For example, if the class name is mentioned n times in the section, the similarity value of them is increased by $20\% + n\%$. If the class methods/functions are mentioned in the section, the similarity between them is further increased by 20%. If the section also contains comments in the class, the similarity value takes a further increase of 20% for class comments, 10% for method/function comments, 5% for other comments. In addition, we will explore the impact of other techniques to refine the extracted links such as our visual IDE's user creation and editing of links and both user and automated ranking of relationship quality.

Finally, according to the experimental results, we examine whether our traceability recovery tool, IRETrace, meets the requirements for a successful recovery technique identified in Chapter 2

- 1) *Aim for target end users who want to get to know the target system.* Our recovery tool does not require users to have any knowledge about the traced system. Everyone who wants to learn a system can use our tool to recover links between artifacts in the system.
- 2) *Support the ability to automatically capture traceability links between artifacts.* Our

recovery tool does not require human intervention during the traceability link recovery process. It can automatically capture links in the selected system.

- 3) *Support the ability to rank the retrieved links to allow end users to select the best set of links.* Our recovery tool ranks links in descending order based on their similarity values.
- 4) *Retrieve as many correct links as possible and as few incorrect links as possible; aim to support both high precision and high recall at all cut points.* The experimental results show that our recovery tool can achieve reasonably high precision and recall at any cut point for the four cases. Integrating the three enhancement techniques, RE, KP, and Clustering, with IR models can largely increase the number of correct links retrieved at high cut points and significantly reduce the number of incorrect links recovered at low cut points.
- 5) *Minimize or mitigate the issue of dependency on cut points.* The experimental results show that our recovery tool significantly narrows the gap between the results of applying different IR models to recover links. Moreover, our tool can produce reasonably high precision and recall at all cut points and narrows the gaps among the precision and recall results generated at different cut points.
- 6) *Provide support for multi-format systems and any kinds of artifacts produced during the software development life cycle.* Currently, our recovery tool only supports systems that are written in Java and only recovers links between source code and documents. Extension to support multi-format systems and recovering links between any two artifacts is to be the subject of future work.

Overall, our recovery tool can meet all requirements for a successful traceability recovery technique except for failing to support multi-format systems and to capture links between any two artifacts.

5.5 Summary

This chapter described and analyzed the evaluation results of the four combination approaches (IR only, IR+RE, IR+RE+KP, and IR+RE+KP+Clustering) by using the six IR models (VSM, TF_IDF,

PL2, BM25, DLH, and IFB2) as the basic retrieval approaches in the four case studies. Our experimental results demonstrate that our combination recovery approach can effectively eliminate some limitations of VSM by taking advantage of the strengths of the three enhancement techniques (RE, KP, and Clustering). Adding RE can significantly augment the number of true links at all cut points. The KP enhancement can retrieve more true links than IR alone. Combining Clustering significantly reduces the incorrect links at all cut points. Furthermore, the F-measure results of all approaches show that the combination of IR, RE, KP, and Clustering is the most effective among all approaches we evaluated if $\beta=0.5$, $\beta=1$, or $\beta=2$. Our approach improves precision at all cut points, reduces incorrect links at low cut points, and increases the number of true links at high cut points. Our approach provides reasonable precision and recall at all cut points.

In the next chapter, we elaborate on how we can visualize traceability links retrieved by our traceability recovery technique. Our link visualization tool allows users to configure cut points and to select some or all techniques to apply to extract links in the traced system. Furthermore, this tool allows users to create and edit links to refine extracted links.

Chapter 6 -- Traceability Link Visualization

The third major challenge we face in software traceability research and practice is how to efficiently and effectively visualize traceability links retrieved by a traceability recovery technique to support the comprehension, browsing, and maintenance of links in a system. This chapter describes a combination visualization system that we have invented, called **DCTracVis**, to support software engineers to better recover, browse, understand, and maintain links in a natural and intuitive way.

6.1 Introduction

Many traceability recovery techniques (described in Chapter 2) have been developed to automatically or semi-automatically extract high quality traceability links between artifacts in a system, that is, to retrieve as many correct links and as few incorrect links as possible. While these link recovery techniques are very powerful, a key unsolved issue remains: how do we support software engineers to effectively and efficiently understand, browse, and maintain these retrieved traceability links? It is commonly believed that software visualization techniques can help software engineers understand complex data, support meaningful interaction between engineers, and support impact analysis (Asuncion et al., 2007; Roman & Cox, 1992). Visualizing traceability links enables users to recover, browse, and maintain inter-relationships between artifacts in a natural and intuitive way (Marcus et al., 2005). However, it is a major challenge to visualize an overwhelmingly large number of traceability links effectively and efficiently. This is because a software system with a large number of artifacts, and thus a very large number of traceability links between artifacts, quickly leads to severe scalability and visual clutter issues (Cornelissen et al., 2007; Holten, 2006; Merten et al., 2011). Moreover, the efficient visualization of both the artifact

structures themselves and the enormous number of inter-relationships between artifacts are far from trivial (Cornelissen et al., 2007; Marcus et al., 2005).

Our particular focus in this research is on traceability between classes in source code and sections in documents that are written in natural language and are produced during the software development process, e.g. requirements, design documents, tutorials, developer or user guides, and emails. The objective of our research is to provide software engineers with an effective visualization environment enabling them to retrieve, create, browse, edit, and maintain traceability links between artifacts effectively and efficiently. With this environment, engineers 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. In terms of size, we are interested in systems with potentially several hundreds to even thousands of classes, dozens if not hundreds of documents, and many tens of thousands to hundreds of thousands of traceability links between classes and document elements.

Traditionally, traceability links are stored or represented in tabular formats, e.g. a matrix. Despite their simplicity, these approaches cannot provide a global overview and they fail to support users in maintaining or interpreting links easily and conveniently (van Ravensteijn, 2011; Voytek and Nunez, 2011). Although using a graph to display links improves on these shortcomings, graphs adopted by most traceability visualization systems to date (described in Chapter 2) have suffered from visual clutter (i.e. are overcrowded) when dealing with large numbers of traceability links between artifacts. Visual clutter is caused by displaying an overwhelming number of traceability links on top of a graph structure, where artifacts are represented as nodes and traceability links as edges between related nodes (Holten, 2006). This then impedes the ability to efficiently browse, analyze, and maintain traceability links between artifacts (Holten, 2006; van Ravensteijn, 2011). These approaches simply cannot scale to the size of systems and number of traceability links we are interested in supporting.

In this chapter, we propose a combination visualization approach that combines enclosure and node-link representations to reduce visual clutter and to allow the visualization of the global structure of traces and a detailed overview of each trace, while still being highly scalable and

interactive. We adopt two visualization techniques to achieve these goals: treemap and hierarchical tree. A treemap view displays a tree structure by means of enclosure and provides an overview of inter-relationships between artifacts. In order to reduce visual clutter, we employ colours to represent the relationship status of each node in the treemap, instead of directly drawing edges between related nodes on top of the treemap. We use two hierarchical trees that can be expanded and contracted to visualize links. One hierarchical tree visualization is used to illustrate detailed link information about each trace. The other is used to display the whole project under trace and traceability links in it to communicate the hierarchical structure of the project. We adopted this general visualization approach to design and construct a traceability visualization system called **DCTracVis**. This system includes navigator, search, and filter functions to help engineers locate particular nodes and filter out uninteresting links. We have conducted a usability study to assess the usefulness of our traceability visualization system for large traceability visualization problems. The results of this evaluation show that our visualization system is both easy to use and can effectively and efficiently help software developers recover traceability links and comprehend, browse, and maintain large numbers of links.

We firstly present the key motivation for this research. We then describe the design of our traceability visualization system, followed by a description of its implementation. We report the results of a usability study and outline possible future research in Chapter 7.

6.2 Motivation

Consider manually verifying traceability links retrieved by a traceability recovery technique without the support of any visualization tools. In Chapter 4, we discussed how we managed to establish the oracle link set for JDK1.5-SUBSET. At the first stage of the development of the oracle link set, six participants manually captured links between 294 classes and 182 sections. They identified 545 links. As these retrieved links may contain incorrect links or miss some correct links, five more participants manually verified them without the support of traceability visualization tools to delete incorrect links and/or to add missing links. Each of them spent at least 70 minutes in the link verification. They commented that it was a tedious, boring, and

time-consuming task and stated that they would greatly favour traceability visualization tools to support them to conduct the verification task. For larger systems, it would be an enormous – if not an impossible – workload to verify upwards of thousands of links. Given this, what functionality should a traceability visualization tool support to ease workload and help engineers maintain complex software systems?

Consider the following scenario. You are a software engineer working on a major software development project. The project manager asks you to prepare a report about the development progress of the project. You need to know what requirements are fully implemented and tested, what requirements are fully implemented but not tested, and what requirements have not been implemented. You first create links between these artifacts and then work out these data to complete the project's development progress report. The project manager later informs you that one of the requirements has been changed. You need to know the impact of such a change: which code, supporting documents, and tests will need to be modified? Looking at the previously created links between this requirement and other artifacts, you can determine the impacted artifacts. You then need to understand those impacted artifacts through reading their documentation and then modifying them to meet the changed requirements.

The previous scenario often happens to everyone who works in the software engineering field. They can be exposed to similar situations at one point or another. In this scenario, the engineer needs to create links between artifacts, find a specific artifact, know the dependency information of the selected artifact, and read their documentation to understand and modify them. These needs can be met through the support of a suitable traceability visualization tool. Such a visualization tool provides the functionality of capturing links between artifacts automatically, editing retrieved links to remove incorrect links or add correct links, providing navigator or search to assist users in locating a specific artifact, showing the dependency information of the selected artifact, and supporting navigation of the system and its documentation to help engineers understand and modify artifacts. Then an issue arises: how to effectively and efficiently visualize these retrieved links to browse and maintain them?

In Chapter 2, we discussed some representative visualization techniques or systems to date that

have been used to display links retrieved by traceability recovery techniques. Software engineers traditionally store or represent traceability links in tabular formats using a spread-sheet, matrix, cross-references, or a database. Matrix and cross-reference techniques are very common traditional methods of representing traceability links. A traceability matrix is easy to understand and provides a quick overview of relations between two artifacts if the set of artifacts is small (van Ravensteijn, 2011). However, the matrix misses the inherent hierarchy and becomes unreadable when the set of artifacts becomes large (Voytek & Nunez, 2011). The cross-reference pattern is also easy to understand but cannot provide the overall structure of traces (van Ravensteijn, 2011). It is difficult to identify individual traceability links as they are lost in this table structure. The approach, therefore, does not scale to large numbers of classes and documents.

More recently, research has focused on displaying links in a graph or tree due to the convenience and ease of browsing and maintaining the links. Graph-based visualization techniques represent artifacts as nodes and traceability links between artifacts as edges to form a graph. Graphs can show the overall overview of relationships between artifacts and allows one to easily browse links.

ADAMS (Lucia et al., 2004) supports specifying links between pairs of artifacts. Traceability links are organized in a graph where nodes are represented by the artifacts and edges are the traceability links. After users select a source artifact, the graph is built starting from a source artifact by finding all the dependencies of a specific type that involve the source artifact either as source or target artifact (ADAMS, 2009). Within the graph, users can identify traceability paths, i.e. sets of artifacts connected by traceability links. This graph performs very well in displaying all links of a selected source artifact. However, it fails to support the display of multiple artifacts' links. Cleland-Huang and Habrat (2007) proposed a hierarchical graphical structure to visualize links, in which leaf nodes are represented by requirements while titles and other hierarchical information are represented as internal nodes. This graph visualization provides a birds-eye-view of the candidate links and their distribution across the set of traceable artifacts. It also allows the user to explore groups of candidate links that naturally occur together in the document's hierarchy (Cleland-Huang & Habrat, 2007). Unfortunately, this visualization becomes very large as the data set gets bigger. Moreover, it uses the display space inefficiently. Zhou et al. (2008) developed ENVISION, adopting a hyperbolic tree view with the enhancement of a "focus+context" approach

to facilitate software traceability understanding. The results of their empirical study show that this view allows users to maintain a global view of links as well as being able to dive deep into an interesting traceability path. However, this view is also not space-efficient.

TBreq (LDRA, 2012), a commercial application, provides end-to-end traceability from requirements to design, code, and test. It lists artifacts horizontally and draws linear edges between related items of artifacts. It cannot provide the hierarchical structure and can quickly produce severe visual clutter for a system with medium to large numbers of artifacts. TraceVis, developed by van Ravensteijn (2011), visualizes a dynamic list of hierarchies and adjacency relations. It uses icicle plots and hierarchical edge bundling (Holten, 2006) techniques to support the hierarchical structure and to reduce visual clutter. Icicle plots are used to represent hierarchies vertically. Adjacent relations are represented by drawing edges between related items. Edges are displayed using splines and are grouped using hierarchical edge bundling. TraceVis supports an overview of as well as a detailed insight into inter-related, hierarchically organized data. However, it uses space inefficiently and can result in visual clutter if the dataset is large or lateral relations visualized (van Ravensteijn, 2011).

Merten et al. (2011) utilized sunburst and netmap techniques to display traceability links between requirements knowledge elements. The sunburst visualizes the hierarchical structure of the project under trace. Nodes are arranged in a radial layout and are displayed on adjacent rings representing the tree structure. The netmap aims to represent links between requirements. The nodes in a netmap are in a circle and are segments of exactly one ring in the sunburst. Traceability links are drawn by using linear edges in the inner circle. 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. EXTRAVIS, developed by Cornelissen et al. (2007) employs a hierarchical edge bundling technique (Holten, 2006) that groups edges based on the structure of a hierarchy to reduce the visual clutter. Using a circular bundle view shows the structure of the system under trace and represents execution traces. The hierarchies are shown by using an icicle plot based on mirrored layout. A global overview of traces is provided by a massive sequence view. However, when considering a large number of traces, it becomes difficult to discern the various colors and to prevent bundles overlapping.

In addition to traditional approaches and the various graph representations similar to those reviewed above, there are several other approaches that have been used to visualize traceability links. Poirot (Cleland-Huang & Habrat, 2007; Cleland-Huang et al., 2007) displays trace results in a textual format. It uses confidence levels, user feedback checkboxes, and tabs separating likely and unlikely links to assist the analyst in evaluating candidate links. However, it cannot visualize overall structure. TraceViz (Marcus, 2005) employs a map consisting of coloured and labeled squares to display traceability links for a specific source or target artifact. It allows users to clearly visualize all links of a selected source artifact or a chosen target artifact. Unfortunately, it is unable to display links for multiple artifacts at the same time. LeanArt (Grechanik et al., 2007) utilizes an intuitive point-and-click graphical interface to enable users to navigate to program entities linked to elements of UCDs by selecting these elements, and to navigate to elements of UCDs by selecting program entities to which these elements are linked. The characteristic of LeanArt is to select a source, and it then displays targets linked to this source. It also fails to present all links at the same time. A 3D approach (Pilgrim et al., 2008) is introduced to enhance traceability visualization between UML diagrams. Artifacts are projected on layered planes. Traces between different levels of abstraction are visualized by using edges between planes. Although presenting more content at once and grouping related information together, the 3D approach adds more complexity to the graph, and still leads to visual clutter when the data set becomes large.

To varying degrees, none of traceability visualization techniques developed so far can visualize an overwhelmingly large number of traceability links effectively and efficiently without scalability and visual clutter issues. Users of such link visualizations not only need scalable, effective representations, but must also be able to navigate complex software systems and their documentation to help them recover, browse, and maintain inter-relationships between artifacts in a natural and intuitive way (Cornelissen et al., 2007; Holten, 2006; Kienle & Muller, 2007; Marcus et al., 2005; Merten et al., 2011). This leads us to the following set of requirements for a traceability visualization technique. (The more detailed requirements for a traceability visualization system was described in Chapter 2.)

- Ability to use the display space efficiently when representing a large number of artifacts in the system under trace.

- Ability to remedy visual clutter issue when visualizing traceability links in the system.
- Ability to show the whole structure of the system.
- Ability to display the global overview of traceability links in the system.
- Ability to illustrate the detailed dependency information of each link.

These issues motivated us to develop a visualization technique to enable engineers to recover, browse, modify, and maintain links effectively and efficiently. The discussion in Chapter 2 on visualization techniques showed that combining different visualization approaches can display elements efficiently. For example, combining node-link representations and enclosure offers a trade-off between an intuitive display and efficient space usage for visualizing large numbers of artifacts in a system (Graham & Kennedy, 2010; Holten, 2006; Shneiderman, 1992; van Wijk & van de Wetering, 1999); node-link representations (e.g. the hierarchical tree) communicate structure readily, and the enclosure layout (e.g. the treemap) is very effective for displaying large numbers of elements. Similar to our approach of combining several traceability recovery techniques to mitigate each other's weaknesses, our approach here is to combine several visualization techniques to provide efficient visualization.

6.3 DCTracVis -- Traceability Visualization between Documents and Source Code

In order to provide efficient traceability visualization, we have explored an approach of combining enclosure and node-link representations to display the overall structure of traceability links and provide a detailed overview of each link while still being highly scalable and interactive. We utilize two visualization techniques to achieve these goals: treemap and hierarchical tree. The treemap view is adopted to display the structure of the system under trace and the overall overview of links. We utilize colours to differentiate the relationship status of each node in the treemap instead of drawing edges directly over the treemap. The latter approach quickly leads to visual clutter. We adopt two hierarchical trees that can be expanded and contracted to visualize links. A whole hierarchical tree (the whole HT) is used to display the whole system and links in it to

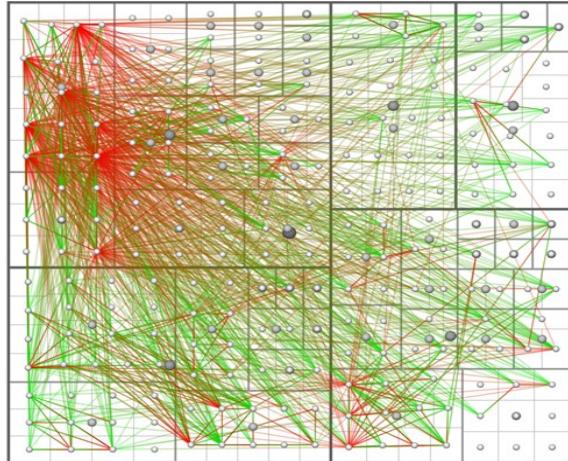
communicate the hierarchical structure of the system. When an item is selected in the treemap view or the whole HT, a detail hierarchical tree (the detail HT) is built to provide the detailed dependency information of the selected item. The detail HT is treated as a supplement to the treemap and the Whole HT. Any change to links made in the treemap is reflected in the two hierarchical trees, and vice versa. We then applied this approach to develop a novel traceability visualization system, called DCTracVis, to represent links between documents and source code. DCTracVis also contains navigation, search, and filter functions to help engineers find particular nodes and filter out unwanted links. The following sections describe the two techniques, how we support editing of links, and other functionality in detail.

6.3.1 Treemap View

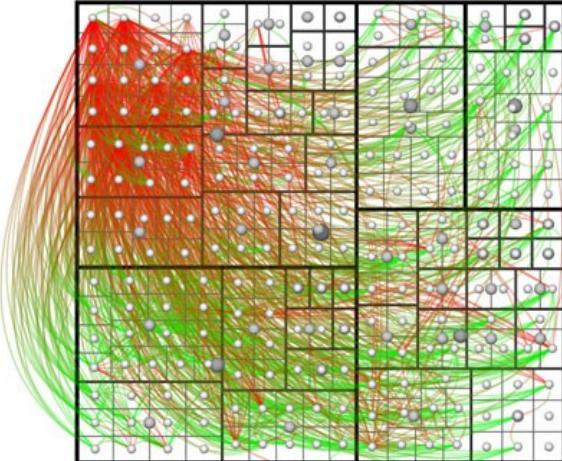
In Chapter 2, we discussed that the treemap technique adopts a space-filling layout technique to represent a tree structure by means of enclosure, which places child nodes within the boundaries of their parent nodes and encloses each group of siblings by a margin (Shneiderman, 1992). This layout makes it an ideal technique for displaying a large tree and using display space effectively (Graham & Kennedy, 2010; Holten, 2006; Shneiderman, 1992; van Wijk & van de Wetering, 1999). Although the treemap technique cannot communicate the hierarchical structure very well, it can convey the high-level, global structure of a system under trace. It is also effective in helping to answer questions such as what artifacts the system has, how many items each artifact has, which artifact contains the most numbers of items, and how artifacts are organized.

In order to display traceability links between artifacts in a treemap, the straightforward way is to add relationships between related nodes as edges over the treemap as in (Holten, 2006) (see Figure 6.1). Figure 1a shows straight/linear edges between related nodes on top of the treemap. Figure 6.1b uses curved link edges. These two approaches quickly lead to visual clutter if large numbers of edges are displayed. Using a hierarchical edge bundling technique can alleviate this issue. Figure 6.1c and d group edges based on the structure of a hierarchy (Holten, 2006). However, hierarchical edge bundling can cause bundles to overlap along the collinearity axes (see the encircled region in Figure 6.1d) if dealing with a large number of collinear nodes in the treemap. All these approaches have difficulty discerning the source and target items of a link if not using

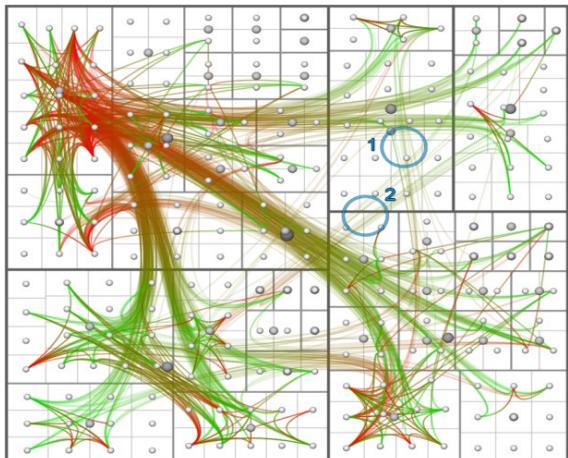
other enhancement techniques, e.g. a “focus+context” technique. For example, it is hard to know that edges circled (1 and 2) in Figure 6.1c are from where to where. Moreover, it is hard to discern the structure of the system conveyed in the treemap because of the edges drawn on top of the treemap. In addition, it is easy for it to become overcrowded when considering large numbers of links.



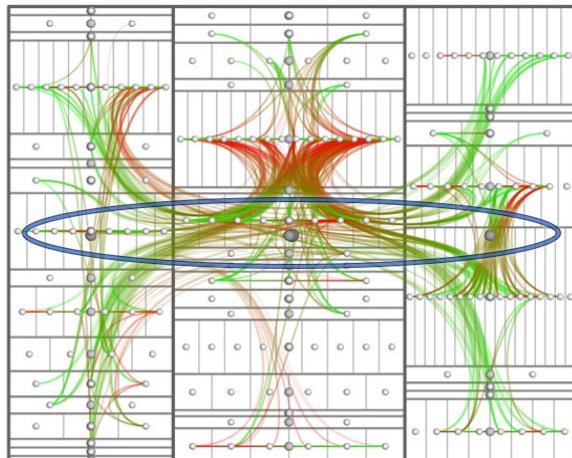
(a) Linear edges



(b) Curved edges



(c) Bundled edges



(d) Bundled edges

FIGURE 6.1 DISPLAYING TRACEABILITY LINKS BETWEEN NODES USING (A) STRAIGHT/LINEAR EDGES; (B) CURVED LINK EDGES; (C) AND (D) EDGES GROUPED BY HIERARCHICAL EDGE BUNDLING. (HOLTEN, 2006)

In order to ameliorate these issues, we introduce colours to show the relationship status of each node instead of drawing edges over the treemap. The relationship status of each node describes whether the node has links and how many links it has. We use three colour ranges to show the

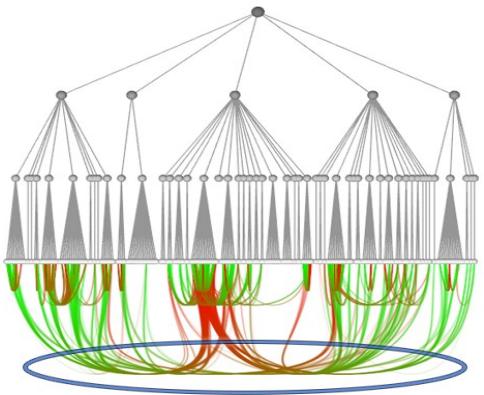
status of each node (see Table 6.1). They are arbitrarily chosen. If a node has fewer than six links, yellow-based colours are used. If the number of links is fewer than 16 but more than 5, gray-based colours are used. Otherwise, we use green-based colours. For each colour range, the shading of the colour indicates intermediate values (lighter implies fewer links, darker more links). Based on colours on each node without additional edges on top of the treemap, it is easy to discern the structure of the traced system and an overall overview of the scale of traceability links.

TABLE 6.1 THREE COLOUR RANGES INDICATING THE NUMBER OF LINKS EACH NODE HAS

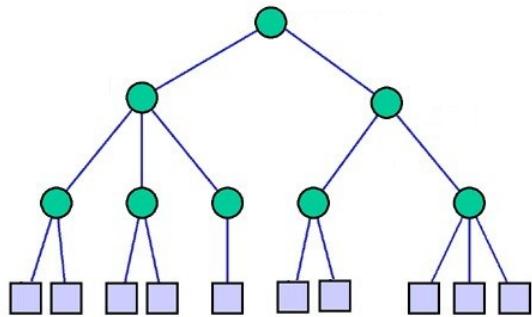
| | | | | |
|---|--------------|---|---------------|---|
| 1. $0 \leq \text{No. of links} < 6$: | Yellow-based |  | \rightarrow |  |
| 2. $6 \leq \text{No. of links} < 16$: | Gray-based |  | \rightarrow |  |
| 3. $\text{No. of links} \geq 16$: | Green-based |  | \rightarrow |  |

6.3.2 Hierarchical Tree Views

The hierarchical tree (discussed in Chapter 2) is an intuitive node-link based representation that uses lines to connect parent and child nodes to depict the relationship between them (Graham & Kennedy, 2010; Holten, 2006). This representation is easy to understand, even to a lay-person, and it communicates hierarchical structure very well (Graham & Kennedy, 2010; Holten, 2006). There are two approaches to visualize traceability links using the hierarchical tree view. The first approach is to draw edges between related children nodes (see Figure 6.2a). Edges can be grouped using the hierarchical edge bundling technique. However, the approach suffers from overlapping bundles along the collinearity axes (see the encircled region in Figure 6.2a) and hence visual clutter if dealing with rather large numbers of traceability links (Holten, 2006). The second approach is to directly add traceability links as children of leaf nodes (see Figure 6.2b). In other words, the original leaf nodes (green circle nodes in Figure 6.2b) in the hierarchical tree become inner nodes and parents of traceability links (gray rectangle nodes in Figure 6.2b). For example, if a child node is related to three other nodes, we additionally add the three nodes under the child node. The second approach can ameliorate problems with the first approach.



(a) Links as edges between nodes



(b) Links as children of nodes

FIGURE 6.2 SHOWING TRACEABILITY LINKS IN THE HIERARCHICAL TREE LAYOUT: (A) LINKS AS EDGES BETWEEN NODES (HOLTEN, 2006), (B) LINKS AS CHILDREN OF NODES

As the hierarchical structure in the treemap is difficult to perceive, we apply a left-to-right hierarchical tree (“the whole HT”) that can be expanded and contracted to display the whole system under trace. We use the second approach to display traceability links as children of artifacts in the system. We also employ the three colour ranges in Table 6.1 to differentiate the relationship status of each node; whether the node has links and how many it has. However, for nodes with no links (No. of links = 0), they are coloured white to distinguish them from other nodes that have at least one link.

We employ a left-to-right hierarchical tree layout (“the detail HT”) to show detailed information of a single item once the item is selected in the treemap or the whole HT. This second approach is adopted to display traceability links for the selected item. It illustrates two levels of dependency information. The first level is artifacts that are related to the selected item. The second level is other artifacts that are dependent on the artifacts shown in the first level. This view shows not only artifacts related to the item but also dependency information for these artifacts. Moreover, we use red-based colours to show the similarity score levels of links. The darker the colour the higher the similarity score a link has. In addition, providing the hierarchical tree with an ability to expand and contract makes it space-efficient.

Figure 6.3 shows a sequence diagram describing the visualization of links between artifacts in a traced project. When a user clicks the traced project, links between artifacts are recovered. A new

visualization view is then created to display retrieved links in the treemap and the whole HT. When the user clicks a node in the treemap or the whole HT, a new detail view is created to display the detailed link information of the selected node in the detail HT.

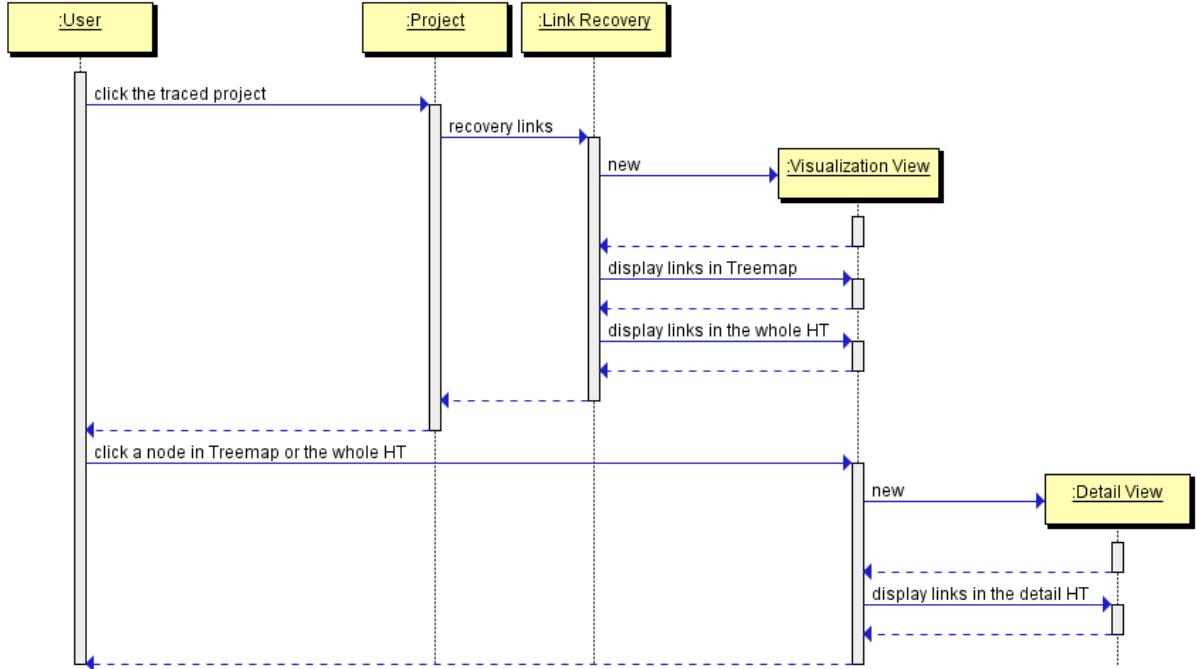


FIGURE 6.3 THE SEQUENCE DIAGRAM FOR VISUALIZING LINKS IN A PROJECT

6.3.3 Editing Traceability Links

Initially we use IRETrace (discussed in Chapter 3) to extract a candidate set of traceability links from a target system and its documentation. While our combination traceability recovery algorithm (discussed in Chapters 3 and 5) has both high precision and high recall compared to other techniques, it still suffers from recovering some incorrect trace links and misses some correct links. To address this, we allow end users of DCTRacVis to delete incorrect links and to add correct links when required.

When a node is selected in the treemap or in the whole HT, its related nodes are highlighted and a detail HT is built starting from the selected node and connecting to nodes related to it and all dependencies of these nodes. Users are then able to edit links in the treemap, the whole HT, and the detail HT views. Our visualization tool provides a popup menu allowing users to delete or change existing traceability links, add a new traceability link, and change the similarity scores ($0 \leq$

similarity score ≤ 1) of existing links.

A changed link or a newly added link is assigned the highest similarity score (=1). The three views are interactive: any change made in one view is reflected in the other two views and is saved. For instance, if an existing link is deleted in the treemap, it is deleted in the whole HT and detail HT as well, and it is not re-added if the end user runs the link extraction process again. In order to assist users in editing traceability links, we provide the full name or the similarity value when users hover the mouse over a node and the detailed content of a node when users click “Show Content” in the popup menu.

6.3.4 Other Functionality

The other functionality we provide includes navigation, search, and filter support. All artifacts in the traced system are indented when listed in the navigator. This allows users to browse the list to find a specific artifact and then to locate this artifact in the treemap and the whole HT and display the detailed link information of this artifact in a detail HT. The search function enables users to use key words to find a particular node in the treemap and the whole HT. There are three methods to filter traceability links. First, users select different traceability recovery techniques to retrieve links. Second, retrieved links can be filtered out some unwanted links by using the threshold or cut point level. Only retrieved links that have a similarity score greater than or equal to the threshold are visualized. Third, our visualization tool allows users to filter out some uninteresting artifacts according to the number of links. Only artifacts that have more than or equal to the number of links are highlighted.

6.4 Implementation

A prototype of our traceability visualization system has been developed. This prototype is seamlessly embedded within the Eclipse integrated development environment (IDE). It automatically extracts relationships between sections in documents and classes in source code and visualizes these retrieved links using the treemap and the hierarchical trees. Figure 6.4 illustrates

the architecture of our traceability visualization system, DCTracVis. First, a project under trace needs to be imported into Eclipse. If documents in the project under trace contain sections, they need to be divided into smaller documents based on headings or sections. For example, if a PDF document contains 10 headings, it is split into 10 sub-documents; the contents of each are the text between its heading and the following one. Next, source code and these smaller documents are passed to our automated traceability recovery engine, IRETrace (discussed in detail in Chapter 3) (1). This engine retrieves traceability links between classes and sections using a composite set of traceability recovery techniques (discussed in detail in Chapter 3) (2).

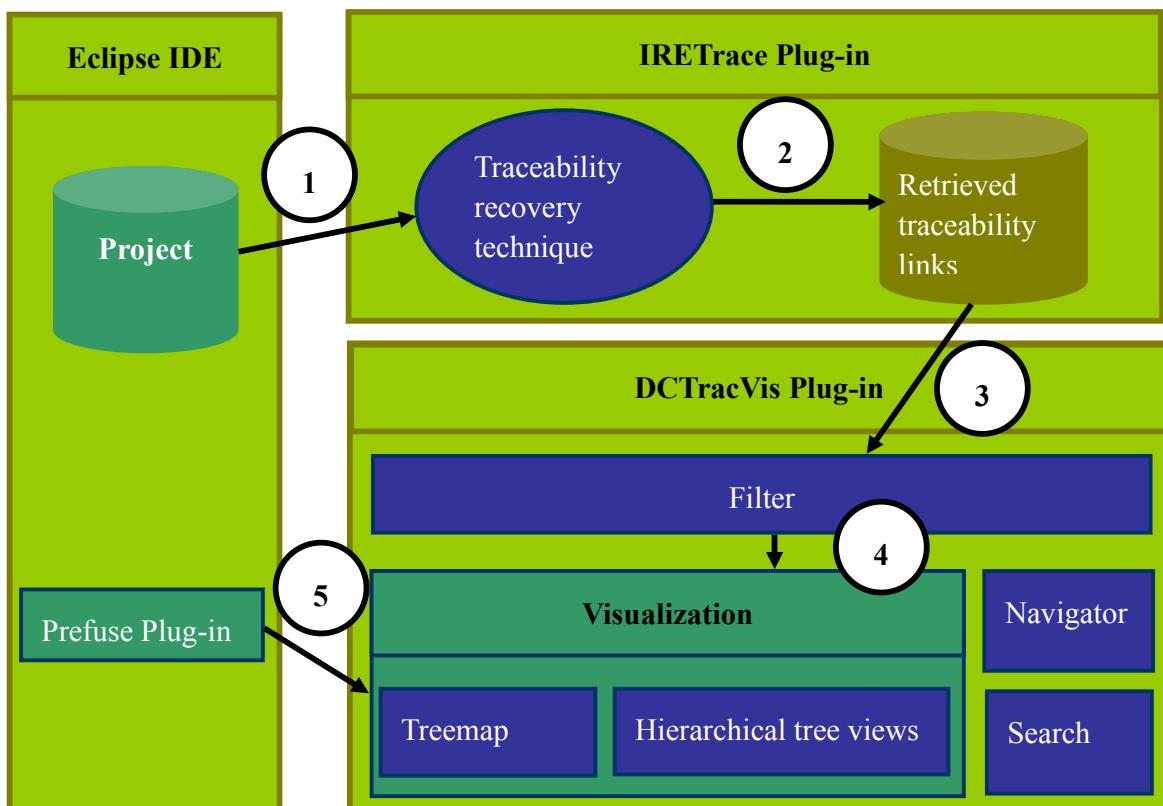


FIGURE 6.4 ARCHITECTURE OF DCTRACVIS

These retrieved traceability links are then input into our traceability visualization system, DCTracVis. They are filtered based on: (a) a threshold level, at which only links with a similarity score larger than the threshold are shown to users, and (b) the number of links level, at which only nodes having more than or equal to the number of links level are shown to users (3). After filtering, the candidate traceability links and the structure information of the project are visualized using the treemap and hierarchical tree techniques (4). Our visualization is implemented using the Prefuse Information Visualization Toolkit (Prefuse, 2011). Prefuse is an open source toolkit written in Java

and supports a rich set of features for data modeling, visualization, and interaction (Prefuse, 2011). We employ Prefuse to display artifacts and links in the treemap and the hierarchical tree (5). Navigator and search functions are provided to assist users in finding a specific node.

6.4.1 User Interface of DCTracVis

Before tracing relationships between artifacts in a system and visualizing retrieved links, artifacts in the system needs to be imported into Eclipse. In the “Traceability perspective”, users select the project in the navigation view and click the “Start Traceability” button in the popup menu (see the circled area in Figure 6.5) to start recovering links and then visualizing them.



FIGURE 6.5 HOW TO START VISUALIZING LINKS IN A PROJECT UNDER TRACE

Figure 6.6 shows an example of the user interface of our DCTracVis prototype. This screen dump shows an example of visualizing traceability links between classes and sections in the JDK1.5-SUBSET, as discussed in Chapter 4. This case contains 249 classes and 182 sections. Traceability links between them are captured using IRETrace, as discussed in Chapter 3. Our traceability perspective includes three parts: navigation view, edit area, and traceability view. The left part is the navigation view, which displays details of a project under trace, e.g. headings inside PDF documents in the JDK1.5-SUBSET. The top right area is the edit area that shows java files or documents and allows users to edit them using functions provided by Eclipse IDE. The bottom right area is the traceability view that visualizes extracted links. Our visualization prototype can provide software engineers with both IDE and traceability support.

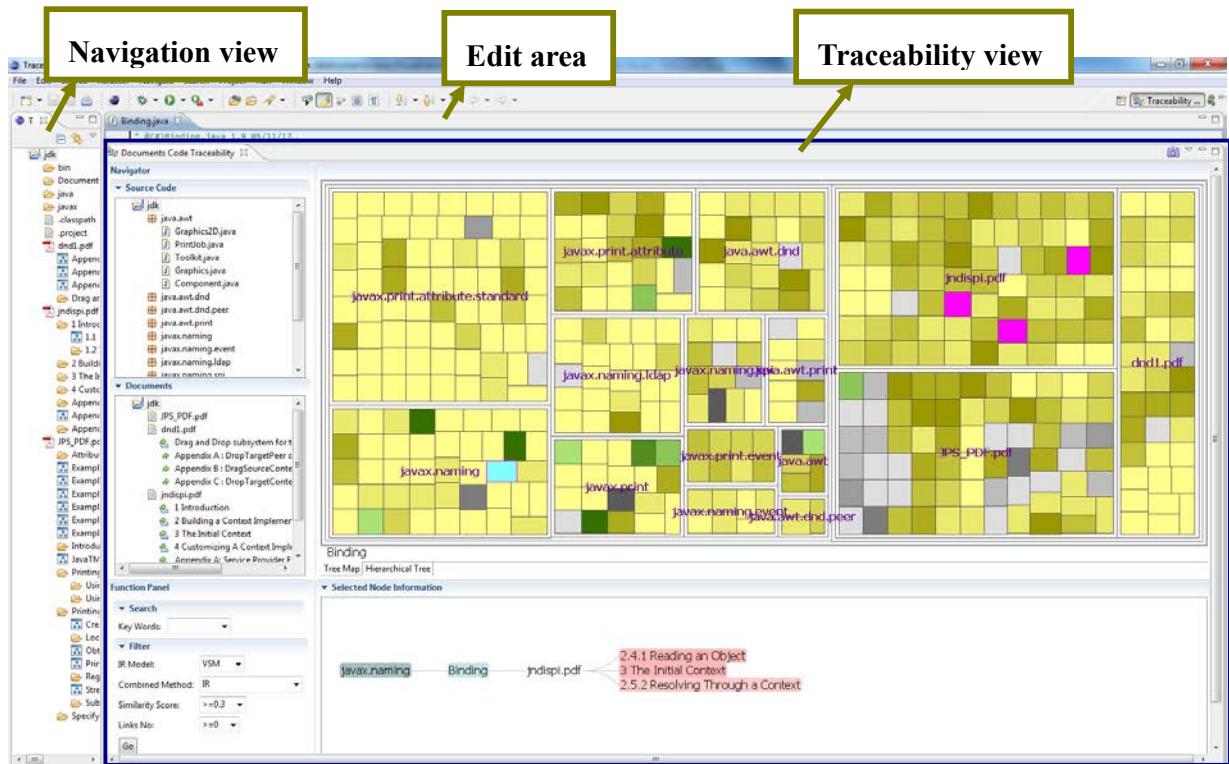


FIGURE 6.6 THE USER INTERFACE OF DCTRACVIS USING THE TREEMAP

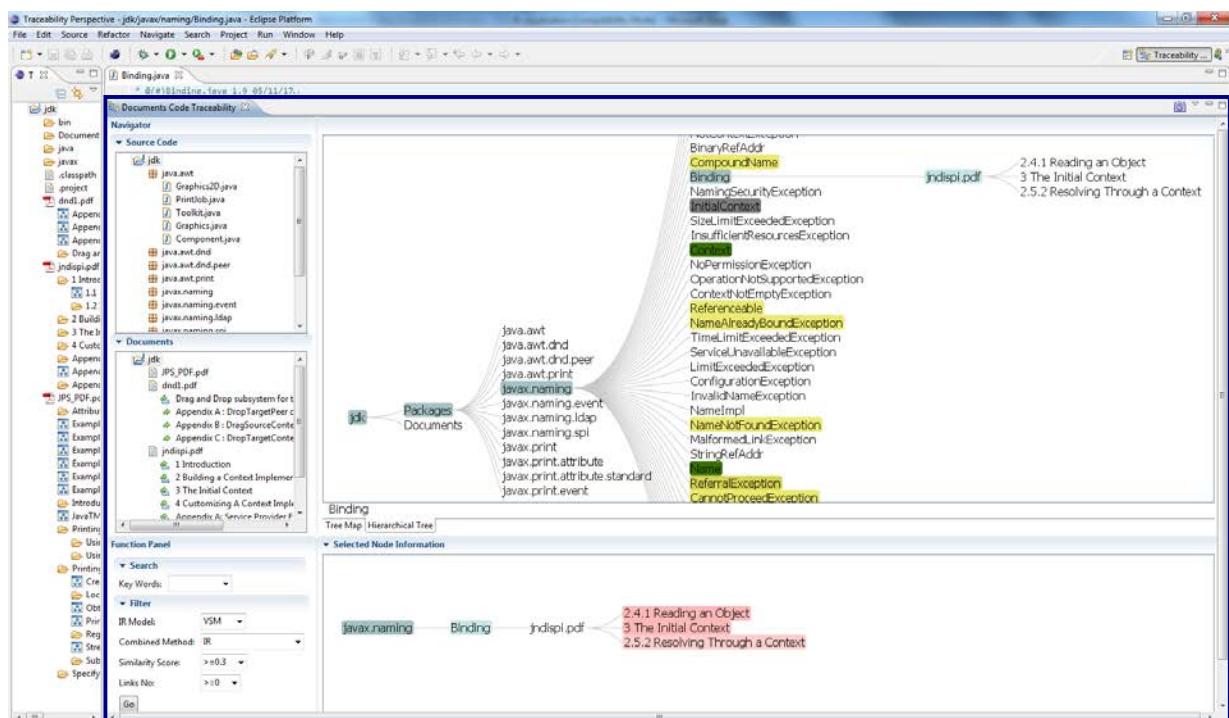


FIGURE 6.7 THE USER INTERFACE OF DCTRACVIS USING THE WHOLE HT

The traceability view includes four parts. The top left area is the Navigator that lists classes and documents in the traced project in indented form. The bottom left area is the Function Panel that includes Search and Filter functions. The top right area is the treemap view (see Figure 6.6) or the whole HT view (see Figure 6.7). The bottom right area is the detail HT view that displays the detailed information of the selected node in the treemap or the whole HT. For example, in Figure 6.6, a node named “Binding” with cyan colour in “javax.naming” package is selected. All related nodes are coloured magenta in the treemap. Detailed link information is displayed in a detail HT. Simultaneously the node “Binding” in the whole HT (see Figure 6.7) is highlighted. Its links are shown as children of this node.

6.4.2 Treemap View

The treemap in Figure 6.8 is divided into two parts: one for packages and the other for documents. Classes are displayed in the packages part and sections are in the documents part. Each node is coloured using the three colour ranges (discussed in Section 3) according to the number of traceability links they have.

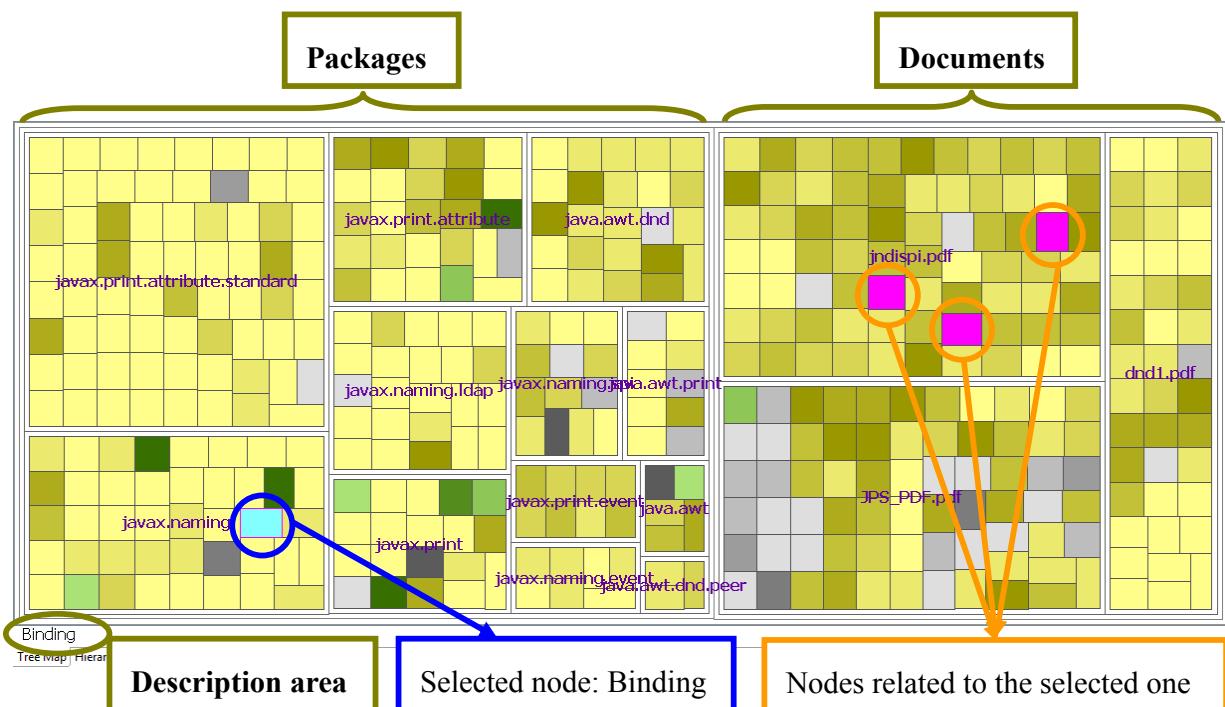


FIGURE 6.8 THE TREEMAP VIEW

When a user hovers the mouse over a node, the name of the node is described in the “Description area” at the bottom of the treemap, and all related nodes are highlighted using magenta. If the node is clicked, it is highlighted with cyan and a detail HT showing its detailed dependency information is built. For example, in Figure 6.8, the node “Binding” and coloured cyan in the “javax.naming” package is selected, and all related nodes are coloured magenta. Detailed link information is displayed in a detail HT (see Figure 6.11).

6.4.3 Hierarchical Tree Views

The whole Hierarchical Tree (HT) shows the whole project under trace and links in it (See Figures 6.9 and 6.10). Artifacts are displayed based on the hierarchical structure of the traced project. Classes and sections are coloured with the three colour ranges (discussed in Section 3) based on the number of links they have. Nodes with no links are white, to distinguish them from other nodes. The hovered-over or selected node’s name is shown in the “Description area” at the bottom of the whole HT. Traceability links of each node become their children nodes. Links are composed of two parts: the first part is names of artifacts and the second is names of items in corresponding artifacts. For example, in Figure 6.9, a class “Binding” in the “javax.naming” package is selected, its links are shown directly after this node and are also displayed in a detail HT (see Figure 6.11). Each link starts with the document’s name followed by the section’s name.

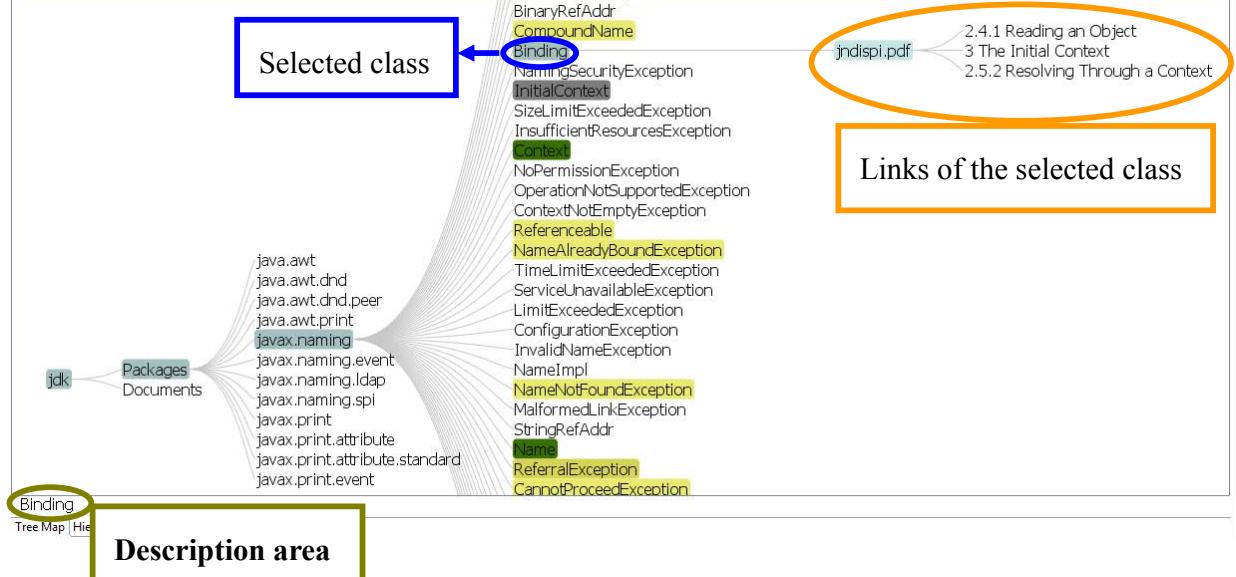


FIGURE 6.9 THE WHOLE HT VIEW WHEN A CLASS IS SELECTED

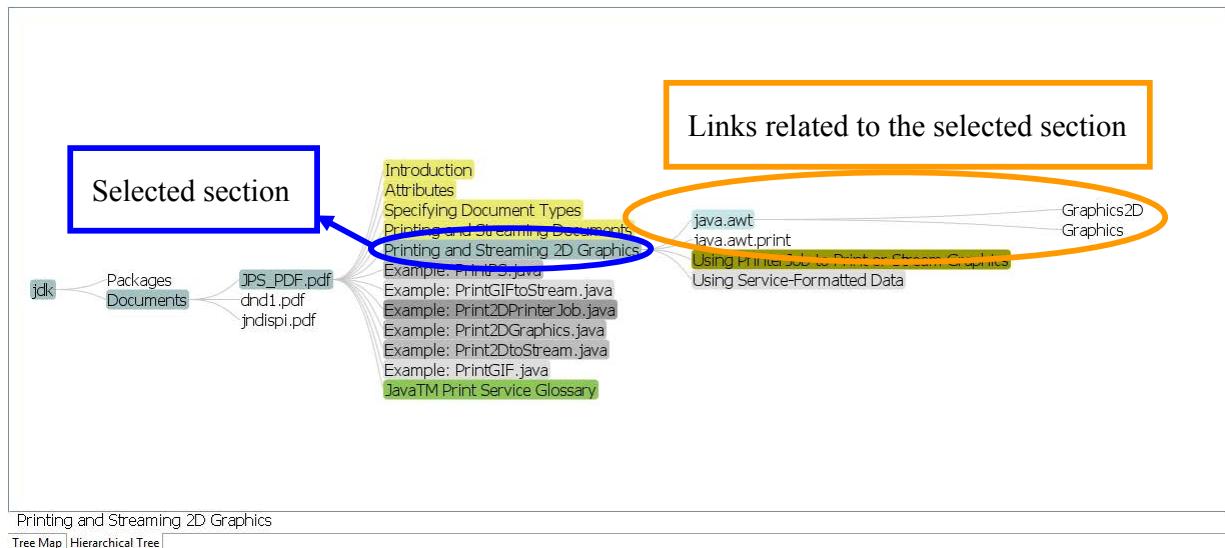


FIGURE 6.10 THE WHOLE HT VIEW WHEN A SECTION IS SELECTED

Figure 6.10 shows traceability links of a section “Printing and streaming 2D graphics” in the “JPS_PDF.pdf” document. These links display packages first, then classes.

When a node is selected in the treemap or the whole HT, a detail HT is established to display detailed link information for this node (see Figure 6.11). The detail HT can be expanded to show link information of nodes related to the selected node (see Figure 6.12). Figure 6.12 shows that the first level is sections related to the “Binding” class, and the second level is other classes dependent on these sections. These related sections and classes are coloured to differentiate their similarity value levels. The lighter the colour the lower the similarity score a node has. When a mouse hovers over the node, its similarity score is shown. In Figure 6.11, the similarity value of “2.4.1 Reading an Object” is 0.5. In Figure 6.12, the similarity score of “InitialContext” at the second level is 0.8.

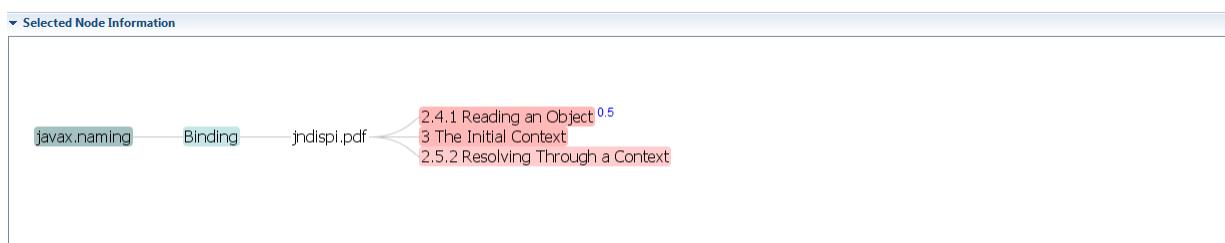


FIGURE 6.11 THE DETAIL HT CONTRACTED

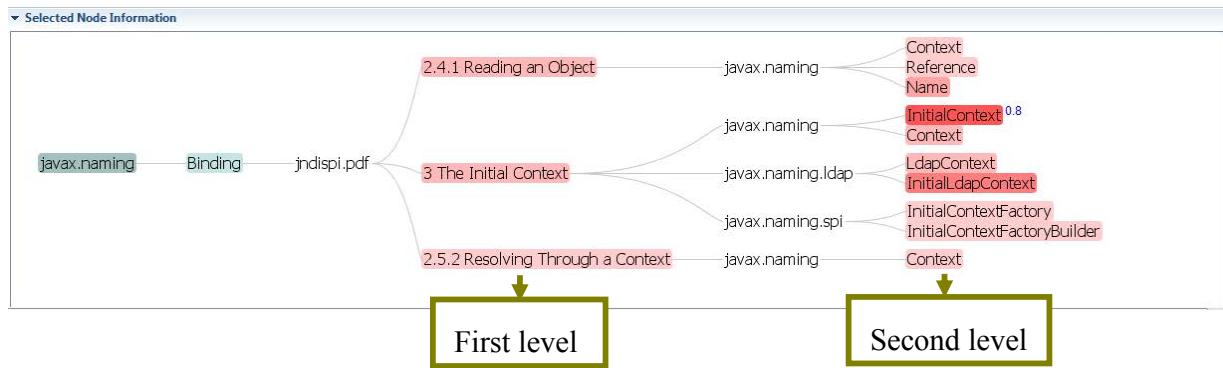


FIGURE 6.12 THE DETAIL HT EXPANDED

6.4.4 Editing Traceability Links

The popup menu contains six functions: Edit Name, Add Link, Delete, Edit Similarity Value, Show Content, and Exit. “Edit Name” allows users to change the name of an existing link. New links can be added using the “Add Link” function. “Delete” can delete an existing link. The similarity value of an existing link can be changed using “Edit Similarity Value”. “Show Content” can open a file related to the selected node in the edit area or open a content window to display the contents of the selected node. “Exit” enables users to exit from the edit model in the treemap.



FIGURE 6.13 POPUP MENU IN THE TREEMAP

The popup menu is slightly different in different views. In the treemap view, once a node is clicked, this view enters into the edit mode. In this mode, links of the selected node can be deleted or added. In other words, existing related nodes can be deleted and new nodes can be added (see Figure 6.13). After the completion of editing, users need to exit from the edit mode in order to browse link information of other nodes. There are three kinds of the popup menu. First, the popup menu for the selected node only allows using “Show Content” and “Exit” (see Figure 6.13a). Second, the popup menu at nodes related to the selected node can use “Delete” and “Show Content” (see Figure 6.13b). Third, “Add Link” and “Show Content” are active in the popup menu at other nodes (see Figure 6.13c). For example, a section (see Figure 6.13c) in the “JPS-PDF.pdf” document can be added as a new link of the “Binding” class. To prevent unwanted structural changes, names of nodes related to the selected node cannot be edited in the treemap.

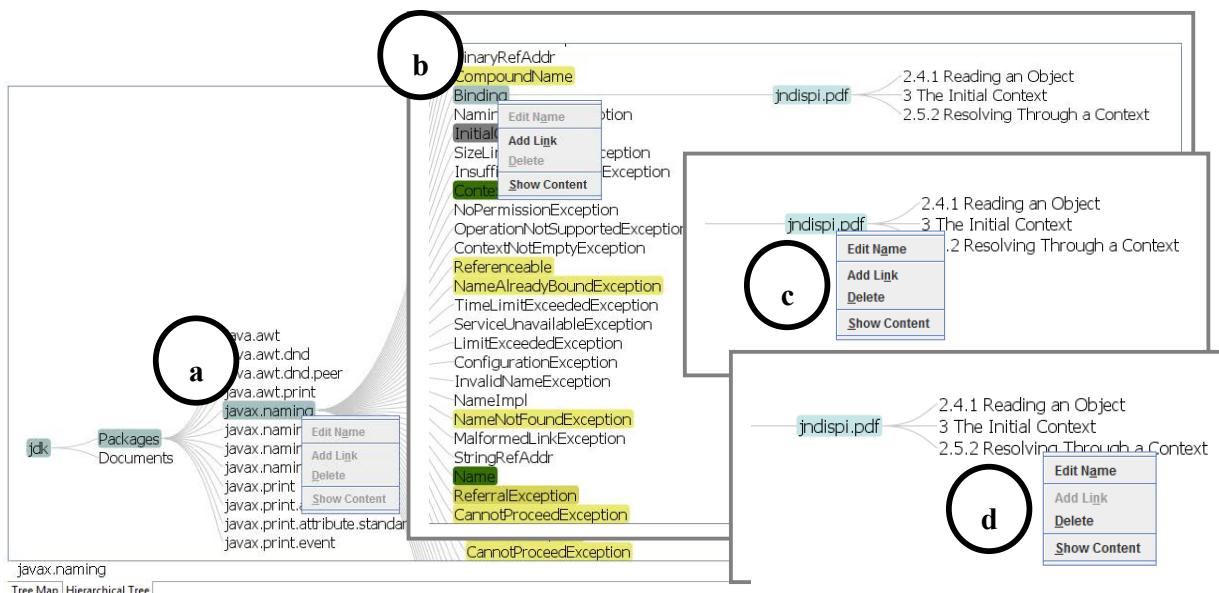


FIGURE 6.14 POPUP MENU IN THE WHOLE HT

However, they are editable in the whole HT. The name of an existing related node can be changed to become a new related node (see Figure 6.14 c and d). Links in the whole HT are represented as two parts: the first part is names of documents or packages and the second is names of sections or classes. The popup menu at the first part of a link enables users to edit the name of the document or the package, add a new section or class, and delete all existing sections or classes in this document or package (see Figure 6.14c). If the name of the document or the package is changed, then all existing links are deleted and a combo box containing sections or classes of the new

document or package is opened to allow users to add a new link. The popup menu at the second part of a link can be used to change the name of an existing link or to delete an existing link (see Figure 6.14d). To preserve the hierarchical structure of the traced project untouched, names of packages, documents, classes, and sections cannot be edited except names in traceability links. Figure 6.14b shows that only “Add Link” is active in the popup menu at the class or section level. No functions are active in the popup menu at the package or document level (see Figure 6.14a).

In the detail HT, there are three kinds of the popup menu. First, the popup menu at the selected node allows adding new links and showing its contents (see Figure 6.15a). Second, traceability links in the detail HT are organized as the same as those in the whole HT. Only “Edit Similarity Value” is inactive in the popup menu at the first part of a link (see Figure 6.15b). Third, the popup menu at the second part of a link allows editing the name of a link, deleting a link, changing the similarity score (see Figure 6.15c). As the detail HT shows two levels of traceability links (see Figure 6.12), the popup menu at the first level allows new links to be added related to items at this level (see Figure 6.15c). For example, in Figure 6.15c, new links related to “2.4.1 Reading an object” can be added.

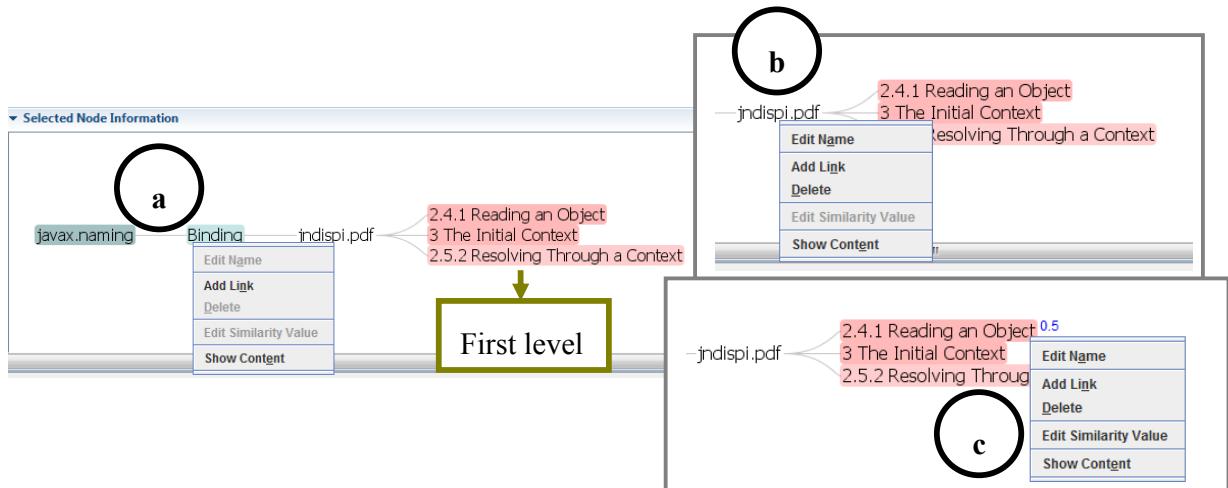


FIGURE 6.15 POPUP MENU IN THE DETAIL HT

Once a node is clicked in the treemap, the node is highlighted in the whole HT at the same time and a detail HT is built to display the detail link information of the selected node (see Figures 6.6 and 6.7). Any change to links made in the detail HT view is reflected in the treemap and the whole HT views, and vice versa. We take the “Binding” class as an example to illustrate how to add a

new link, change the similarity value of a link, and edit the name of an existing link, and how the three views interact with each other. Figure 6.16 shows how to add the “2.3 Drag source” section in the “dnd1.pdf” document as the new link of the “Binding” class. “dnd1.pdf” has to be added first (see Figure 6.16a) and then select “2.3 Drag source” in the combo box that contains sections of “dnd1.pdf” (see Figure 6.16b). A new link “2.3 Drag source” related to “Binding” is added and is assigned the highest similarity value (=1) (see Figure 6.17). At the same time, the new link is automatically added in the treemap (see Figure 6.17) and in the whole HT (see Figure 6.18).

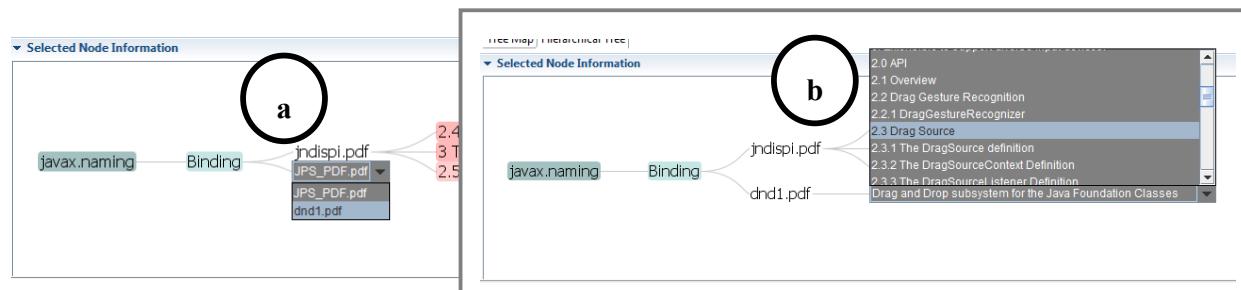


FIGURE 6.16 ADD A LINK IN THE DETAIL HT VIEW



FIGURE 6.17 A NEW LINK IS ADDED TO “BINDING” CLASS SHOWN IN THE TREEMAP



FIGURE 6.18 A NEW LINK IS ADDED TO “BINDING” CLASS SHOWN IN THE WHOLE HT

If the similarity value of an existing link is low, it can be changed using the “Edit Similarity Value” in the popup menu of the detail HT. Figure 6.19 shows that the similarity value of “2.4.1 Reading an object” is changed from 0.5 (see Figure 6.19a) to 0.9 (see Figure 6.19b).

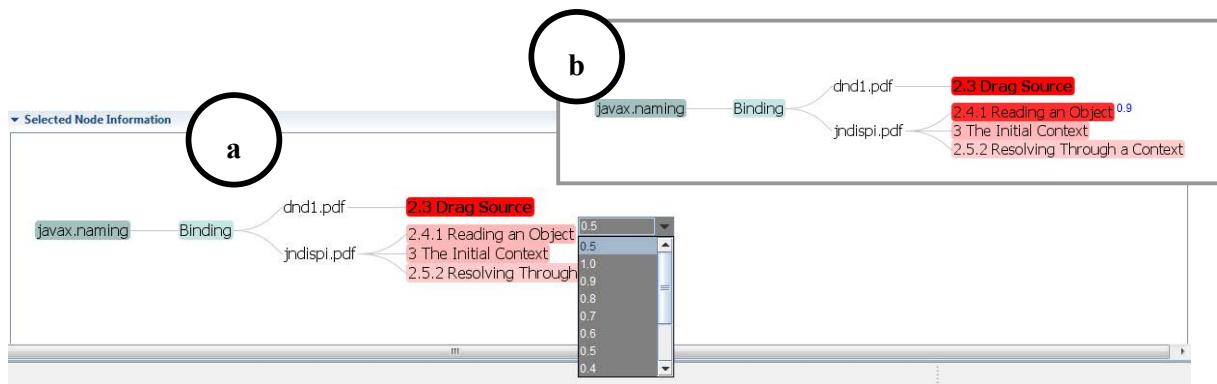


FIGURE 6.19 CHANGE THE SIMILARITY VALUE IN THE DETAIL HT

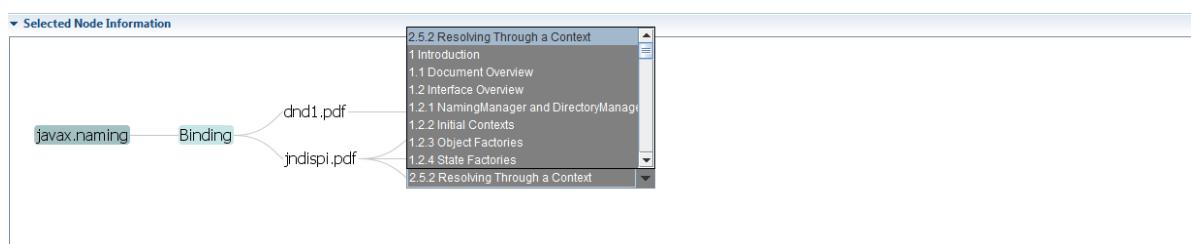


FIGURE 6.20 EDITING AN EXISTING LINK IN THE DETAIL HT



FIGURE 6.21 CHANGING AN EXISTING LINK IN THE DETAIL HT IS REFLECTED IN THE TREEMAP

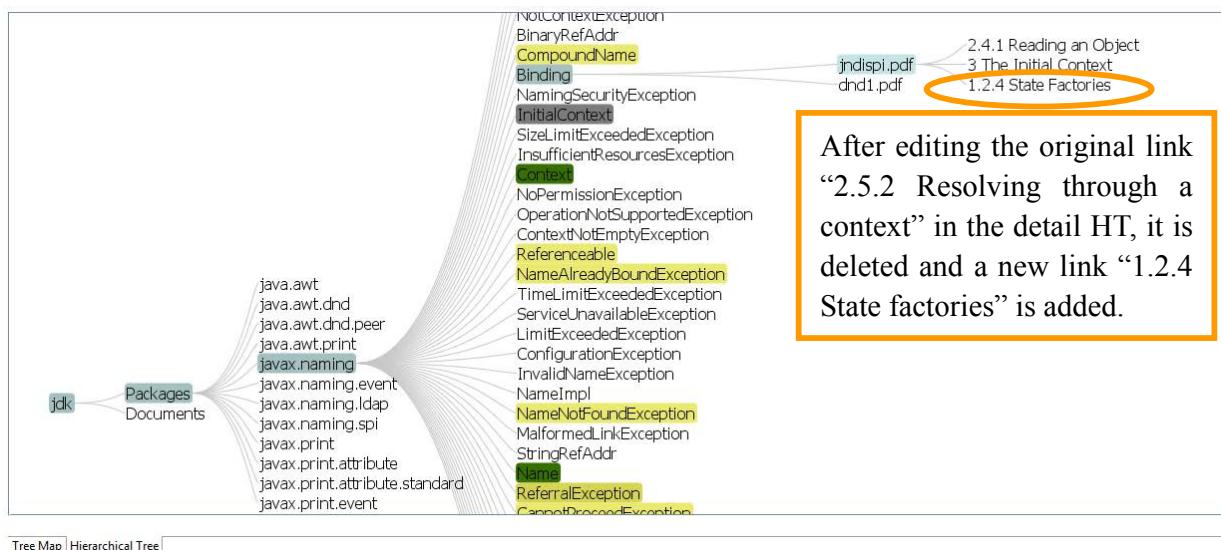


FIGURE 6.22 CHANGING AN EXISTING LINK IN THE DETAIL HT IS REFLECTED IN THE WHOLE HT

The name of an existing link can be edited using “Edit Name” in the popup menu. Figure 6.20 shows that the original link “2.5.2 Resolving through a context” of the “Binding” class is edited. It is replaced with “1.2.4 State factories”. The new link is assigned the highest similarity score (=1)

(see Figure 6.21). In the treemap (see Figure 6.21) and in the whole HT (see Figure 6.22), the original link “2.5.2 Resolving through a context” is deleted and the new link “1.2.4 State factories” is added simultaneously.

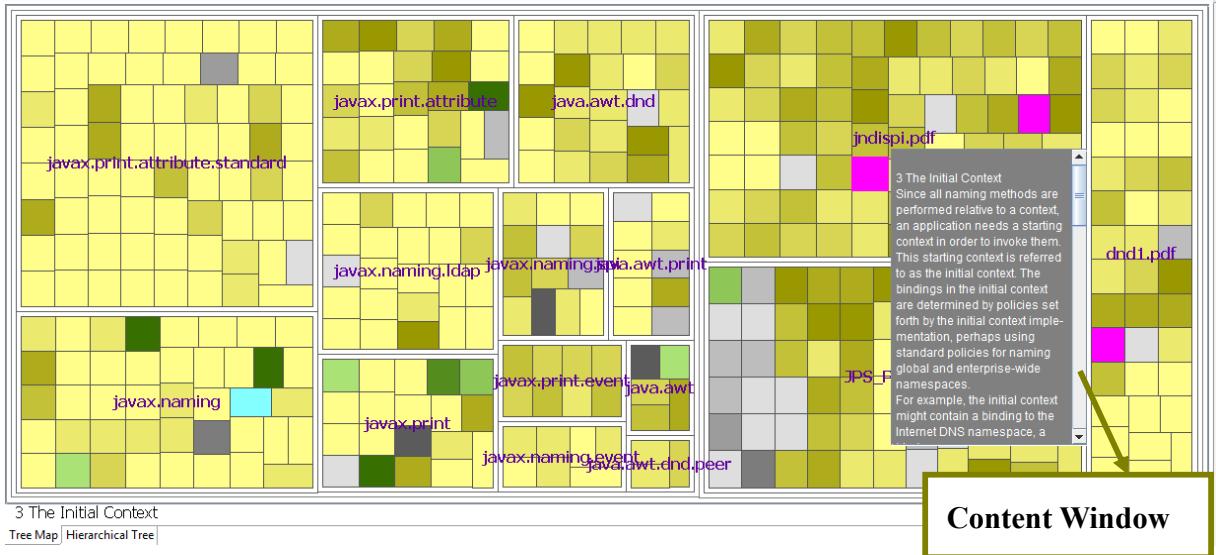


FIGURE 6.23 SHOWING CONTENTS OF A NODE IN THE TREEMAP

In the three views, we provide the contents of nodes to assist comprehension. When “Show Content” in the popup menu is selected, the file related to the node is opened in the edit area. If the node is a section, a content window is also opened to display the contents of the section (see Figure 6.23).

6.4.5 Other Functionality

In order to assist users to find a specific node and filter out unwanted links, DCTracVis provides three additional functions: navigator, search, and filter. Users can find a specific node browsing through the list in the navigator. In Figure 6.24, when the node “DragSource.java” in the “java.awt.dnd” package in the navigator is selected, it is highlighted with cyan and its related nodes are highlighted with magenta in the treemap (see Figure 6.24). This node is also highlighted in the whole HT (see Figure 6.25). Moreover, the detailed link information of this node is displayed in a detail HT (see Figure 6.24). In addition, the file related to this node is opened in the edit area.

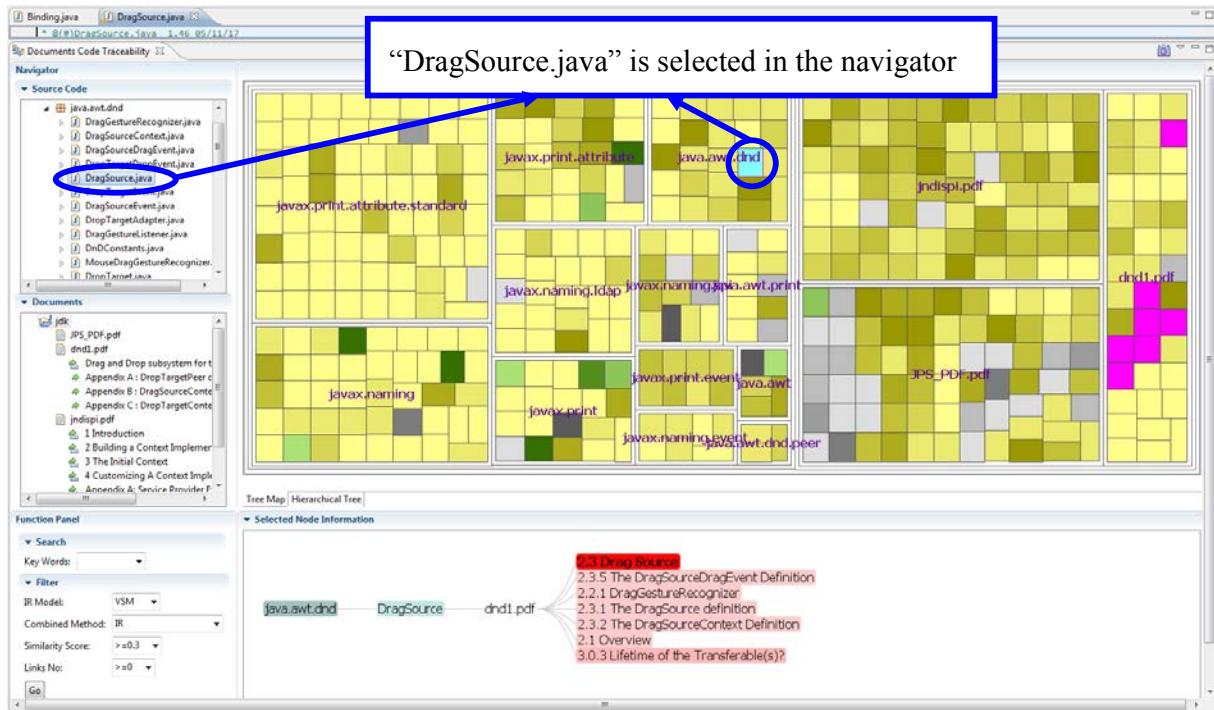


FIGURE 6.24 SELECTING A NODE IN THE NAVIGATOR IS REFLECTED IN THE TREEMAP

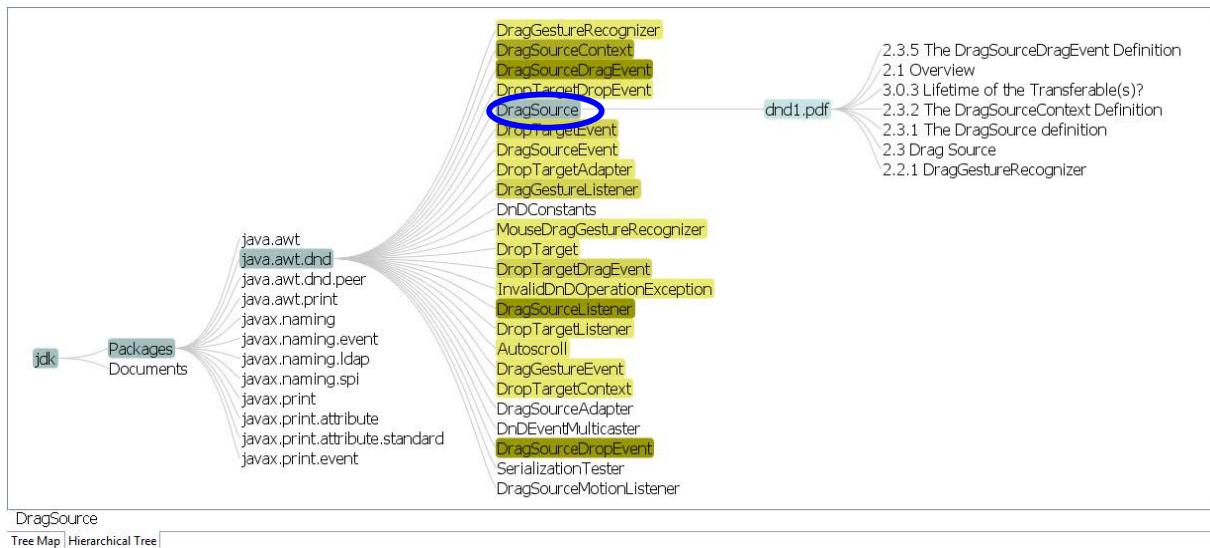


FIGURE 6.25 SELECTING A NODE IN THE NAVIGATOR IS REFLECTED IN THE WHOLE HT

The other method to find a specific node is to use the search function. Users put key words into the search box (see Figure 6.26a), and then a list of nodes matched to the key words is shown in the search combo box (see Figure 6.26b). The number of matches is shown (see Figure 6.26c) and matched nodes are highlighted in the treemap (see Figure 6.26d). For example, Figure 6.26 shows there are 16 nodes matched to the “drags” key word.



FIGURE 6.26 USE SEARCH TO FIND A NODE

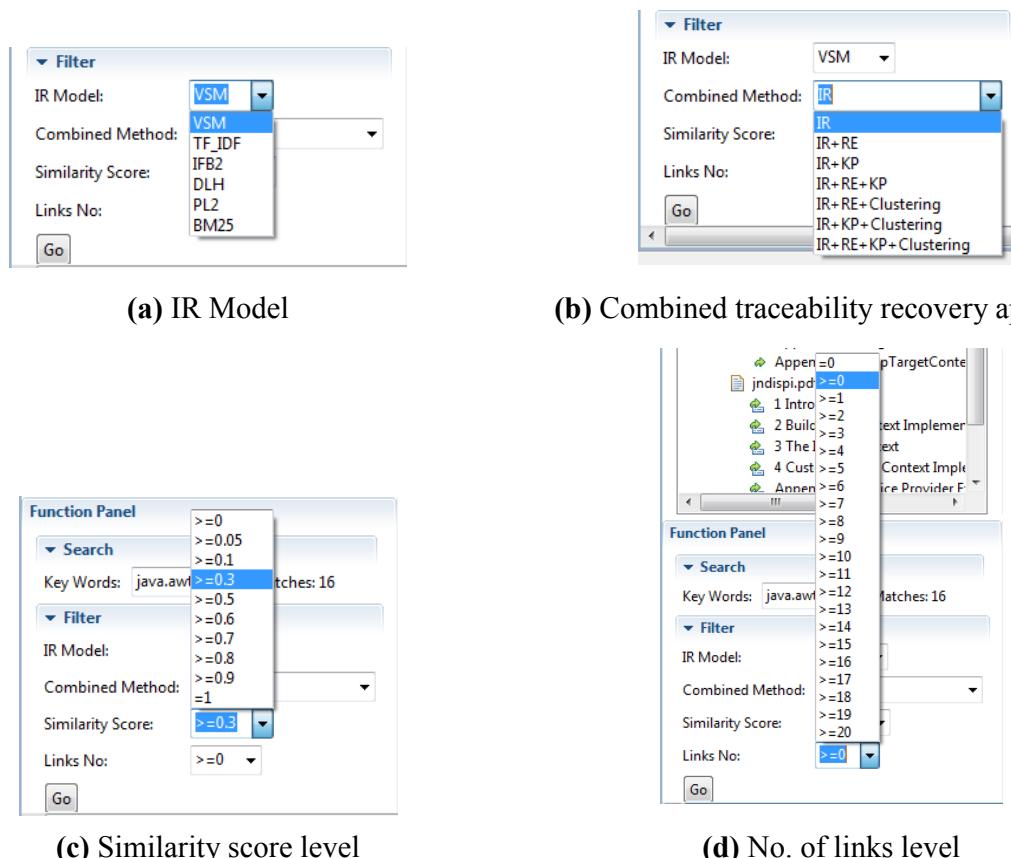


FIGURE 6.27 METHODS IN THE FILTER

There are four methods used to filter out traceability links in the filter function: IR model, combined traceability recovery approach, the similarity score level, and the number of links level. First, our visualization system employs IRETrace (discussed in Chapter 3) to recover links between artifacts. IRETrace provides six Information Retrieval (IR) models (see Figure 6.27a). Users can select an IR model to retrieve links. Second, after the selection of the IR model, users can decide whether to use the IR model alone or to combine other techniques (e.g. Regular Expression (RE), Key Phrases (KP), and Clustering) with the IR model to capture links (see Figure 6.27b). These techniques are discussed in Chapter 3. Third, users can select a similarity score level to display retrieved links that have the similarity value greater than or equal to the selected level (see Figure 6.27c). Fourth, selecting number of links level filters out nodes that have less than the selected level (see Figure 6.27d). All examples shown above are using the Vector Space Model (VSM) model alone to retrieve links between classes and sections, and only visualizing links that have a similarity value greater than or equal to 0.3. The example shown in Figure 6.28 displays links that have a similarity value greater than or equal to 0.5 (see Figure 6.28a). Figure 6.28b shows that the “DragSource” class only has one link if using a similarity score level 0.5, but it has seven links if using a similarity score level 0.3 (see Figure 6.24). Six links of this class are filtered out.



FIGURE 6.28 LINKS VISUALIZATION OF USING DIFFERENT SIMILARITY SCORE LEVEL

6.5 Summary

It is well recognized that visualizing traceability links between software artifacts in a system helps developers to recover, browse, and maintain these inter-relationships effectively and efficiently. However, it is a major challenge for researchers in the software traceability area to efficiently visualize traceability links for big software systems because of scalability and visual clutter issues. We have presented a new visualization approach that integrates enclosure and node-link visualization representations to support an overview of traceability in the system and the detailed overview of each link while still being highly scalable and interactive. The treemap and hierarchical tree visualization techniques are applied to display traceability links in a system. The treemap view provides the overall structure of the system and the overall overview of traceability links. Our approach reduces visual clutter through adopting colours to represent the relationship status of each node instead of directly drawing edges between related nodes on top of the treemap. Two hierarchical trees are employed to visualize links. The whole HT view is to represent the whole system under trace and links in it to communicate the hierarchical structure of the system. The detail HT view can be treated as a supplement to the treemap and the whole HT. When a node is selected in the treemap or the whole HT, the detail HT view displays all nodes that are related to the selected node and other dependency information of these nodes. These traceability links can be modified (add, delete, edit) and their similarity scores can also be changed. The three views are interactive - changes made in one view can be reflected in the other two views, and vice versa. We adopted this visualization approach to build a traceability visualization system, DCTracVis. This system provides the support of the navigator, search, and filter functions to assist users in locating a specific node or filtering out some uninterested links.

Finally, we examine whether our traceability visualization system, DCTracVis, meets the requirements for a successful visualization system identified in Chapter 2.

- 1) *Need to seamlessly integrate the link visualization support within an IDE (e.g. Eclipse) or other software management tools.* Our visualization system, DCTracVis, is integrated with the Eclipse IDE. Users can enjoy both the Eclipse and DCTracVis support.
- 2) *Ability to integrate with traceability recovery tools to automatically capture traceability*

links between artifacts in the traced system. Our traceability recovery tool, IRETrace (discussed in Chapter 3), is integrated with DCTracVis automatically retrieving links between artifacts in the traced system.

- 3) Provide efficient and effective visualization techniques to represent retrieved links. Technical challenges for successful visualization techniques are as follows:

- ◆ *Use the display space efficiently when representing a large number of artifacts in the traced system.* In DCTracVis, links are visualized in a treemap and a hierarchical tree with the ability to expand and collapse. Treemap is an ideal approach to display a large number of items (Shneiderman, 1992). The ability to expand and collapse within the hierarchical tree makes the use of the display space more efficient.
- ◆ *Remedy the visual clutter issue when visualizing links in the system.* We adopted three colour ranges to represent link statuses of nodes in the treemap instead of directly drawing edges between related nodes on top of the treemap. For the hierarchical tree, traceability links are displayed as children of artifacts in the system.
- ◆ *Show the whole hierarchical structure of the system.* The hierarchical tree is to communicate the hierarchical structure of the system in DCTracVis.
- ◆ *Show the overall overview of traces in the system.* Treemap is to covey the overall structure of the system and the overall overview of traces in the system in DCTracVis.
- ◆ *Illustrate detailed dependency information for a specific artifact.* Once a node is in the treemap and the hierarchical tree, another hierarchical tree is created to display the detailed dependency information of the selected node.

- 4) *Provide detailed information about each link or artifact to assist users to understand them.* In DCTracVis, the contents of a selected node can be opened in the content window (for documents) or at the top of the edit area (for code) while the “Show Content” in the popup menu is clicked.
- 5) *Allow end users to add new links, delete or modify existing links and their similarity values, and edit the properties of links and their connected artifacts.* In DCTracVis,

existing links can be deleted or modified and new links can be added. The similarity value of each link can be modified. In future work, we can add a property window to show the properties of a selected link, such as related artifacts, who created the link, when the link is created, the similarity value.

- 6) *Allow end users to browse through artifacts to locate a specific item in the traceability visualization view.* In DCTracVis, a navigator is provided to allow users to browse through the system and find a specific artifact.
- 7) *Allow end users to search a specific link or artifact through keywords or query.* We provide the search function to find a specific item using keywords.
- 8) *Filter out some uninterested links or artifacts.* In DCTracVis, users can apply different combination recovery approach to capture links, filter the retrieved links according to the similarity score level and/or the number of links level.
- 9) *Capture and record browsing history.* In DCTracVis, no browsing history is recorded. This can be among our future works.

Overall, our traceability visualization system, DCTracVis, meets all the requirements for a successful visualization system except for capturing and recording browsing history. We have conducted a usability study to answer the question: does our visualization system help to support and improve the comprehension, browsing, and maintenance of traceability links in a system? The results are discussed in Chapter 7.

Chapter 7 -- Evaluation of Traceability Link Visualization

This chapter presents a usability evaluation of our traceability link visualization tool, DCTracVis, and discusses the evaluation results. Ethics approval was obtained from the University of Auckland Human Participants Ethics Committee for this usability evaluation; details can be found in Appendix A.

7.1 Introduction

Usability evaluation is a common technique used in the design and development of a system to verify the quality and feasibility of the system and to identify weaknesses by evaluating the usability or user-friendliness of the system (Dumas & Redish, 1999; Gena & Weibelzahl, 2007). We undertook a usability evaluation to test how natural and intuitive DCTracVis is to recover, browse, and maintain traceability links in a system under trace. The main objective of this usability evaluation was to obtain qualitative information on user perceptions of DCTracVis and evaluate the tool's usability and effectiveness in comprehending, recovering, browsing, and maintaining traceability links in a traced system. We wanted to learn whether DCTracVis supports and improves the comprehension, browsing, and maintenance of traceability links in a system, and which visualization view provides the best support for comprehension, browsing, and maintenance of links.

To answer these questions a usability evaluation was designed as detailed in the following section. First, the evaluation method is described, then followed by an analysis of the evaluation results. Finally, we discuss limitations of DCTracVis and propose possible future research to improve it.

7.2 Evaluation Design

This usability evaluation is structured into three parts: tasks, observation, and questionnaire. The tasks part lists the tasks to be accomplished by participants. The observation part involves the collection of observation data while participants are performing the tasks. The questionnaire part provides a questionnaire to be answered after the completion of the tasks. The three parts are discussed in detail in the following sections.

7.2.1 Tasks

In order to understand real user reactions to the functionality of DCTracVis, we designed three tasks (the details of which are described in Appendix A).

- The first task was to understand the traced system; the structure of the system and to get an overview of links between artifacts in the system.
- The second task was to understand how an artifact works; how a class works in order to fix a bug related to it, where the documentation of this class can be found, and what other classes are related to this class.
- The third task was to modify traceability links of an artifact. Links for a class retrieved by IR recovery techniques may contain incorrect links, or miss correct links, or have low similarity score of correct links. These retrieved traceability links of the class need to be edited to contain only correct links, delete incorrect links or add missing links.

7.2.2 Observation

There were two types of observation carried out: unobtrusive observation and obtrusive observation. With unobtrusive observation, participants were observed while they were performing the tasks based on the following aspects:

- How participants managed to complete the tasks;
- How participants explored the tool to browse and locate links;

- How participants navigated different functions of the tool;
- Verbal responses and facial expressions of participants while using the tool.

With obtrusive observation, participants were asked to express aloud what they thought while using the tool in order to learn more about the usefulness and the acceptance of the tool. Perceptions and comments from participants were collected. The observation data was recorded anonymously and no personal information was collected.

7.2.3 Questionnaire

Our questionnaire is composed of three sections: background information, DCTracVis usability, and comparative questions about the visualization views. The first section contains four questions to collect some background information of participants' positions, software development experience, frequency of using Eclipse for programming, and frequency of using other traceability tools.

Questions in the second section were designed to reflect user perceptions of DCTracVis. This section has two parts. The first part has seven questions to gather user perceptions of the functionality of DCTracVis; whether link recovery, browsing, maintaining and understanding are supported or improved as perceived by the participants. The seven questions ask about user opinions and experiences about the following: effective link extraction, easy to extract links, easy to maintain links, easy to browse links, easy to find links, support for comprehension, and support for maintenance and development. The second part of this section of the questionnaire collects user perceptions about the overall performance of DCTracVis; the usefulness, ease of use, ease of learning, and satisfaction of this tool. This part contains eight questions: functions well integrated, functions all present, user friendly, easy to use, learn quickly, easy to learn, like to use it in future, and recommend to friends. All questions we designed for this section were based on the questions from the System Usability Scale (SUS) (Brooke, 1996) and the USE questionnaire (Lund, 2001). The SUS is a simple, ten-item scale giving a global view of subjective assessments of usability (Brooke, 1996). The USE questionnaire stands for usefulness, satisfaction, and ease of use (Lund, 2001). Each question in this section was recorded using a five point Likert scale: 2=Strongly agree,

1=Agree, 0=Neutral, -1=Disagree, -2=Strongly disagree.

The third section contains five questions to obtain user perceptions towards the three traceability link visualization views we used to represent links: the treemap view and the whole hierarchical tree view (both discussed in Chapter 6) and the tree view (the initial visualization view we used in IRETrac discussed in Chapter 3). The aim of these questions was to find out which visualization view participants preferred with regard to browsing traceability links, maintaining traceability links, supporting them in understanding the system, or to support them in maintaining and developing the system.

The answers to the questionnaire were collected anonymously. A sample of our evaluation survey appears in Appendix A.

7.3 Evaluation Method

The example documents and code used in this study is the JDK1.5-SUBSET described in Chapter 4. We invited potential participants who had a computer science or software engineering background. There were no requirements for participants to have some knowledge about JDK1.5-SUBSET. Before any participants carried out the study, informal pilot testing was conducted to test the suitability of the questions and to make changes as appropriate. We then recruited a group of 20 participants for the evaluation of DCTracVis. At the beginning, a brief introduction and a demonstration were provided to help participants to gain familiarity with our tool. The participants then practiced our tool to familiarize themselves with its interface and functionality. Once the participants felt comfortable with our tool, they performed the three tasks. How long each participant took to complete each task was recorded. After the completion of the tasks, the participants answered a set of questions on our tool, as well as some general questions regarding their background. Finally, the participants were requested to provide open-ended comments on our tool. During the evaluation, we observed and recorded how participants used our tool to complete the tasks and their verbal responses and facial expression. In general, each participant in this study took approximately 40 minutes. The results of the study are analyzed in

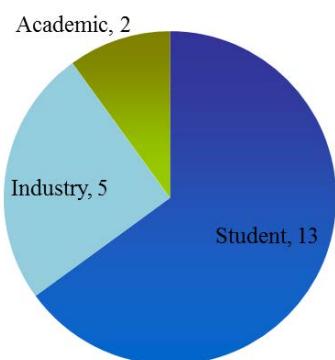
the following section. Consent Forms and Participant Information Sheets are in Appendix A.

7.4 Evaluation Results

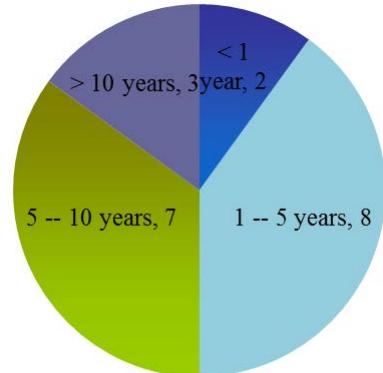
In this section, we present the results and analysis of the usability evaluation.

7.4.1 Background of Participants

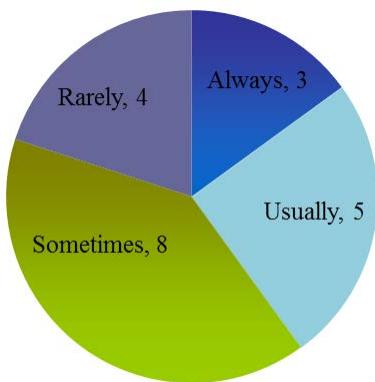
We recruited 20 participants for the evaluation of DCTracVis. The participants were 13 students and 2 academics of the University of Auckland and 5 from industry (see Figure 7.1a). Figure 7.1b shows the software development experience each participant had. Among 20 participants, 3 had more than 10 years of development experience, 7 had fewer than 10 years but more than 5 years, 8 had fewer than 5 years but more than 1 year, and 2 had less than 1 year.



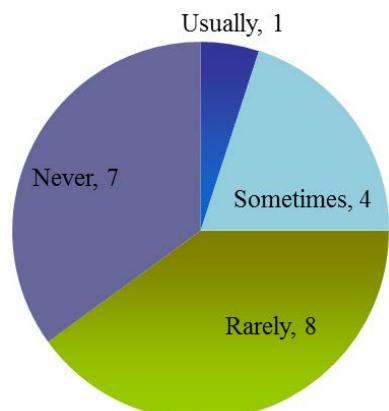
(a) Participants position



(b) Software development experience



(c) Frequency of using Eclipse for programming



(d) Frequency of using other traceability tools

FIGURE 7.1 THE BACKGROUND INFORMATION OF PARTICIPANTS

All of the participants were at least a little bit experienced with the use of Eclipse for programming Java systems (see Figure 7.1c). Among them, 3 always used Eclipse for software development, 5 usually used Eclipse, 8 sometimes used it, and 4 rarely used it. Only one participant usually applied traceability tools to assist in comprehending or maintaining or programming software systems. 4 participants sometimes used traceability tools, 8 rarely used, and 7 never used such tools (See Figure 7.1d).

7.4.2 Tasks and Observation

All 20 participants completed the three tasks with times varying from 5 minutes to 19 minutes. Figure 7.2 shows the time taken by each participant to complete each task. Participant 20 spent the longest time (19 minutes) to accomplish all tasks. Participant 14 took less time (5 minutes) than others to complete the tasks. On average, the first task needed 1.85 minutes to complete, the second task required 3.15 minutes, and 4.35 minutes for the third task.

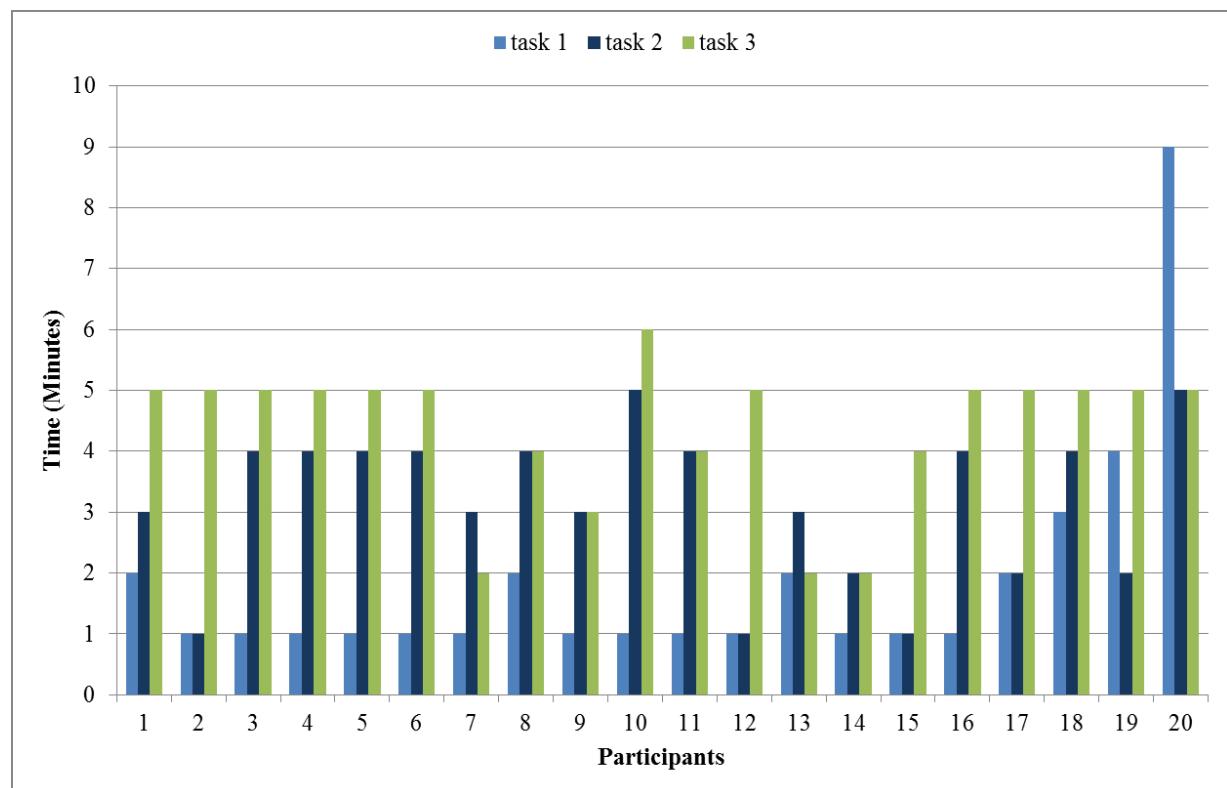


FIGURE 7.2 TIMES TAKEN BY PARTICIPANTS IN COMPLETING EACH OF THE THREE TASKS

Table 7.1 summarizes what DCTracVis functions were used by participants to complete each task. The detail hierarchical tree in this table displays the detailed traceability link information of a node selected in the treemap, in the whole hierarchical tree, or in the tree. Branches in the two hierarchical trees can be expanded and contrasted, but those in the tree cannot. (The tree is discussed in Chapter 3; others are discussed in Chapter 6.)

TABLE 7.1 DCTRACVIS FUNCTIONS USAGE STATISTIC

| Function | Task1 | Task2 | Task3 |
|--------------------------|-------|-------|-------|
| Navigator | 9 | 12 | 12 |
| Search | 0 | 6 | 8 |
| Filter | 20 | 0 | 0 |
| Show contents | 0 | 20 | 12 |
| Treemap | 6 | 1 | 4 |
| Whole hierarchical tree | 10 | 1 | 3 |
| Detail hierarchical tree | 0 | 20 | 18 |
| Tree | 0 | 0 | 0 |

The first interesting result was revealed when performing the first task. 13 of the 20 participants completed this in less than 1 minute, 4 spent 2 minutes, 1 for 3 minutes, 1 for 4 minutes, and 1 took the longest time (9 minutes) to complete this task (see Figure 7.2). Based on our observation, the participant who took 9 minutes spent most of their time getting familiar with our tool as he/she had done little practice before performing the three tasks. However, they strongly agreed that DCTracVis clearly illustrated the hierarchical structure of the system and provided a good overview of links in the system. They accomplished this task by using the navigator, filter, treemap, and/or whole hierarchical tree functions (See Table 7.1).

All participants completed the second task with times varying from 1 minute to 5 minutes (see Figure 7.2). All participants looked relaxed and happy while conducting this task. 12 participants used the navigator function to find a specific node, 6 used the search function, 1 directly located the node in the treemap, and 1 used the whole hierarchical tree (see Table 7.1). All participants used the show content function to help them understand links and utilized the detail hierarchical tree to accomplish this task. They agreed that the detailed dependency information provided in the detail hierarchical tree view was a good supplement to both the treemap view and the whole

hierarchical tree view while performing the second task.

The third task was completed between 2 and 6 minutes (see Figure 7.2). In Table 7.1, 12 participants located a node using the navigator function, and 8 used the search function. The majority of participants (18 of 20) undertook the modification of traceability links of a node using the detail hierarchical tree view. Because they thought this view was more intuitive and straight-forward for this task. 2 used either the treemap or the whole hierarchical tree to modify links of a node. 3 participants edited links of a node in the treemap and the detail hierarchical tree, and 2 completed the link modification in the whole hierarchical tree and the detail hierarchical tree. 12 participants used the show content function to support them in comprehending the modified links. 19 participants looked relaxed and happy when editing links. However, one participant looked frustrated and stressed while doing the link modification. He/she complained that the edit menu was not easy to use.

Most participants used the navigator function to seek a specific node in the second and third tasks as they thought it was easier and more natural to browse artifacts in the navigator to find the node they were interested in. We also noticed that no one used the tree to complete any task (see Table 7.1). They commented that it was hard to browse in the tree view as it was too cluttered and messy. Moreover, our observations indicated that the participants encountered difficulties in directly finding a specific node in the treemap. However, with the support of the navigator and search functions, they easily and quickly found an item they were interested in. In addition, three participants felt confused after applying the filter. They suggested it would be better to provide a summary about how many links recovered and how many links filtered out after using the filter.

7.4.3 DCTracVis Usability Questions

The main results analysis performed after this evaluation was on the set of questions answered by participants based on their experiences of using DCTracVis in comparison to other software tools they have used. We start with the analysis of the evaluation in the functionality of DCTracVis. The results can be seen in Figure 7.3. The diagram shows the seven questions (effective link recovery, easy to extract links, easy to maintain (add, delete, edit) links, easy to browse links, easy to find

links, support comprehension, and support maintenance/development) on the x-axis. The y-axis shows the number of participants; how much they agreed (strongly agree, agree, or neutral, disagree, or strongly disagree) that our tool helped recover, browse, and maintain traceability links in a system and supported users in comprehending, maintaining, or developing the system.

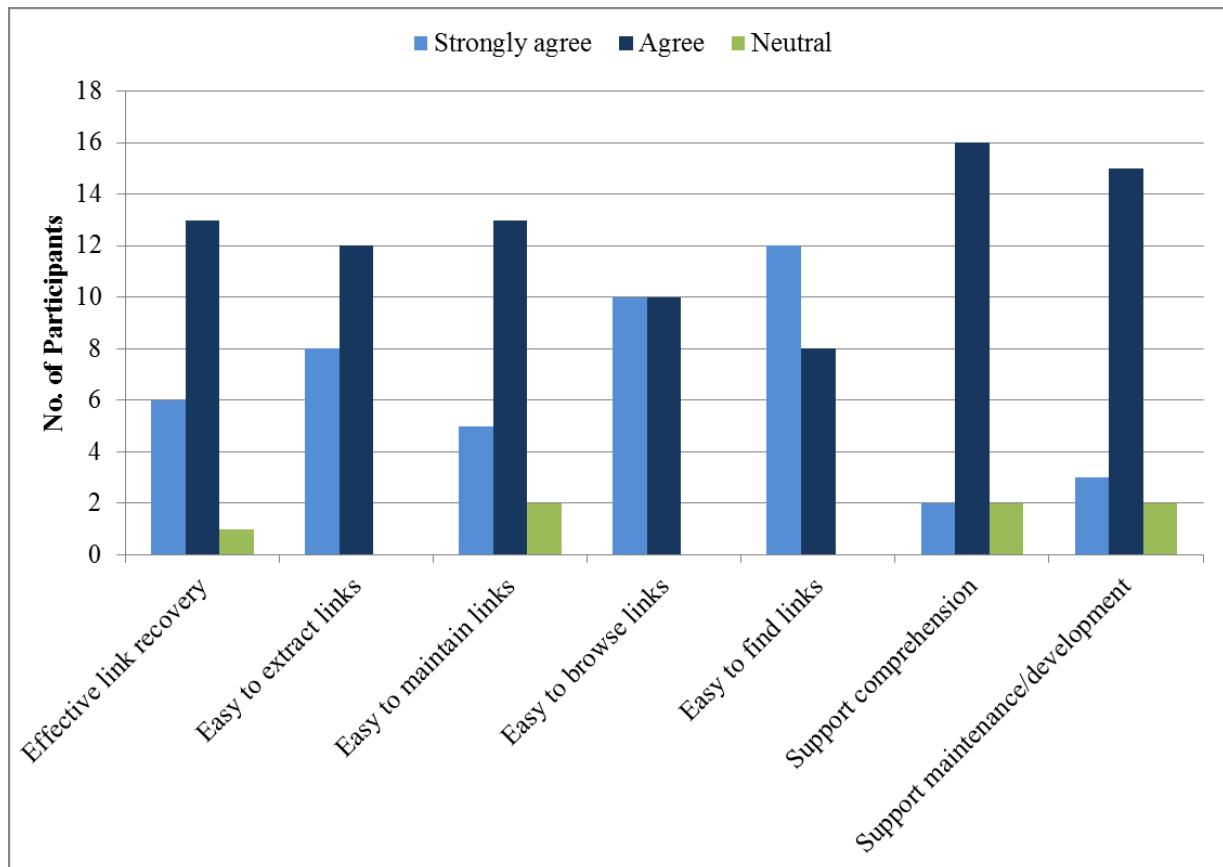


FIGURE 7.3 RESULTS OF THE EVALUATION IN THE FUNCTIONALITY OF DCTRACVIS

No participants made a negative response to any of the seven questions. All participants strongly agreed or agreed that our tool was easy to use to extract traceability links, easy to browse links, and easy to find links. 19 participants (strongly) agreed that it helped them more effective in extracting links between artifacts within systems. 18 of them (strongly) agreed that our tool was easy to maintain links. 18 participants (strongly) agreed that it was useful to support them in the comprehension and maintenance/development of a system. Several participants gave a neutral answer to questions for effective link recovery, link maintenance, comprehension, and maintenance and development. They responded this way because they could not undertake the comparison as they had never used other traceability tools.

Figure 7.4 shows the average ratings for the seven questions on the functionality of DCTracVis. The participants largely agreed that our tool made it easier to find traceability links (1.6) and to browse links (1.5). The remaining questions have average scores of at least 1, indicating that the participants agreed that our tool could recover links effectively, was easy to extract links and easy to modify links, and supported the system's comprehension and maintenance and development. Overall, the results clearly show that the participants agreed that our tool could help them to recover, understand, browse, and maintain traceability in the system.

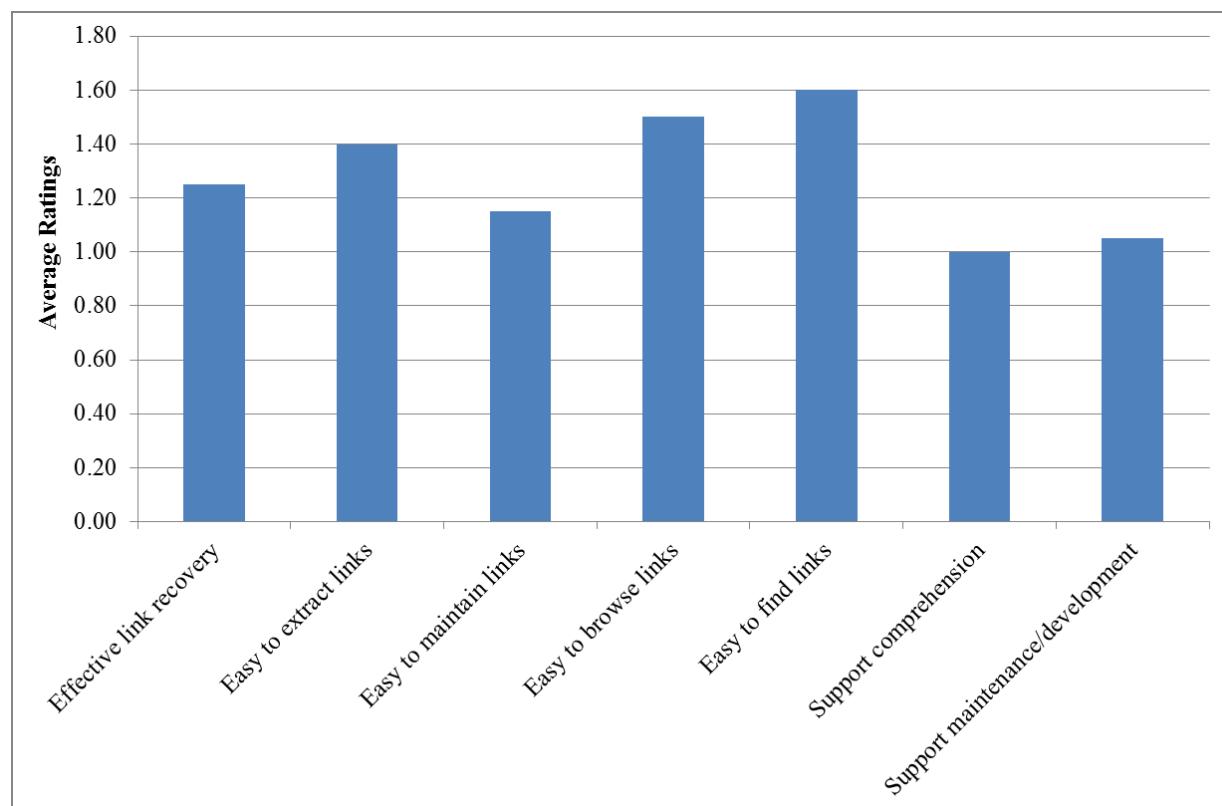


FIGURE 7.4 AVERAGE RATINGS OF THE FUNCTIONALITY OF DCTRACVIS

Next, we analyzed the evaluation of the overall performance of DCTracVis. Figure 7.5 shows the evaluation results. The x-axis displays the eight questions (functions well integrated, functions all present, user friendly, easy to use, learn quickly, easy to learn, like to use it in the future, and recommend to friends) about our tool's overall performance. The y-axis shows the number of participants; how much they agreed (strongly agree, agree, or neutral, disagree, or strongly disagree) that our tool contained all necessary functions, and was ease of use and learning, and satisfied the participants.

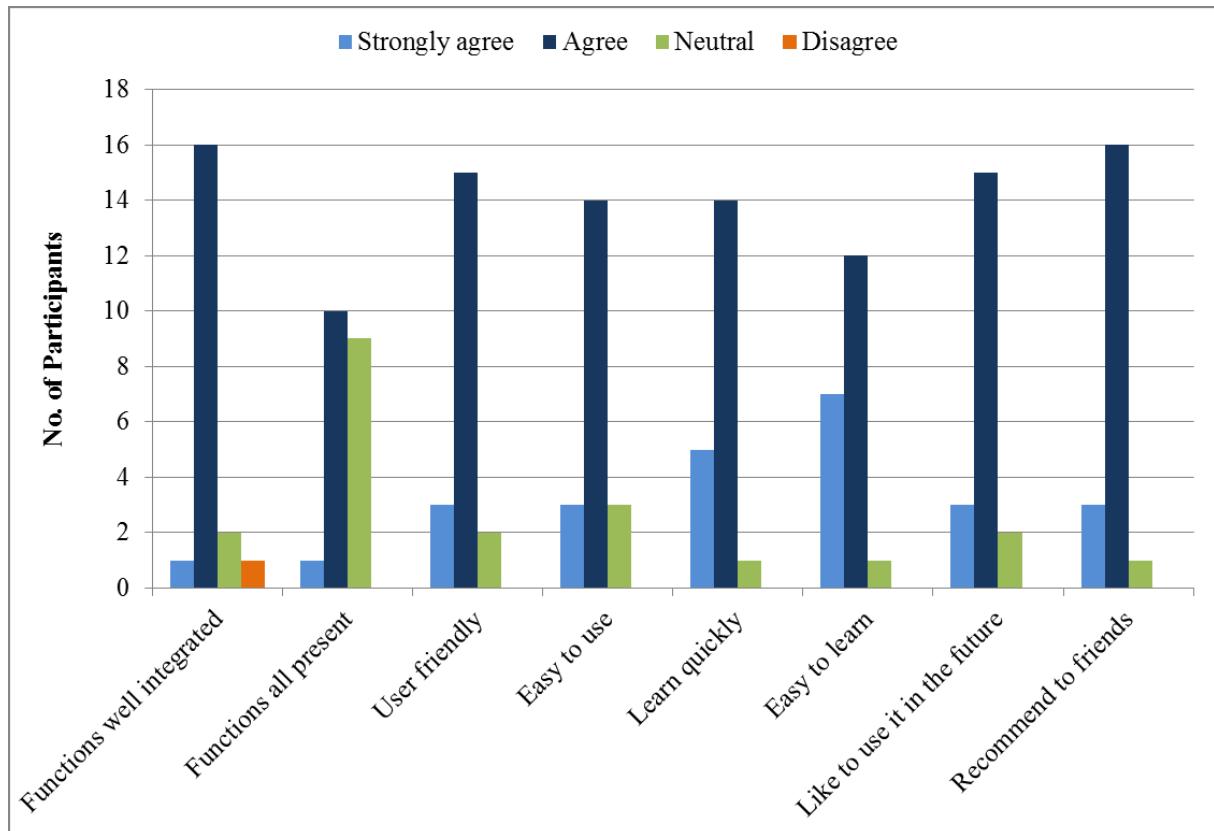


FIGURE 7.5 RESULTS OF THE EVALUATION IN THE OVERALL PERFORMANCE OF DCTracVis

One participant disagreed that the various functions in our tool were well integrated and easy to find. He/she commented this way because it was a little bit hard to modify links of a node using the edit menu. 19 participants (strongly) agreed that they learned to use our tool quickly, it was easy to learn how to use it, and they would recommend it to friends. 18 of them (strongly) agreed that our tool was user friendly, and they would like to use it in the future. 17 participants (strongly) agreed that it was easy to use, and functions in it were well integrated and easy to find. 11 agreed that all the functions they expected were all present, but 9 participants gave a neutral answer to this question. 5 participants thought that there was room to improve although expected functions were all present. 4 participants responded this way because they could not undertake the comparison as they had never used other traceability tools. Several participants provided a neutral feedback to other seven questions. They commented that they were unable to conduct the comparison because they had no experience of using other traceability tools.

Figure 7.6 shows the average ratings of the eight questions in the overall performance of DCTracVis. The participants were inclined to agree that expected functions in our tool were well

integrated and all present. The average ratings for the two questions are less than 1. The other questions have average scores of at least 1. This suggests that the participants agreed that our tool was user friendly, easy to use, and easy to learn, and they learned to use it quickly, would like to use it in the future, and would recommend it to friends. Overall, the results show that the participants agreed that our tool provided all expected functions, was useful, easy to use, easy to learn, and satisfied them.

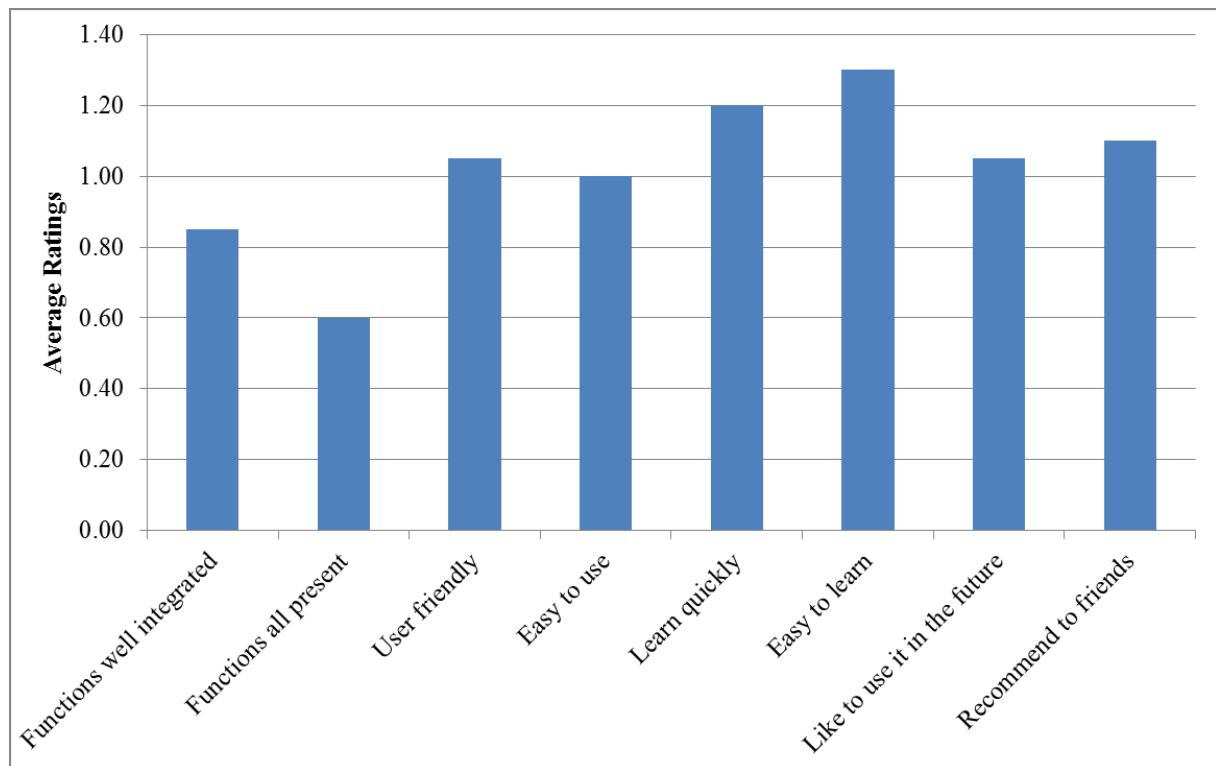


FIGURE 7.6 AVERAGE RATINGS OF THE OVERALL PERFORMANCE OF DCTRACVIS

The participants also reported many valuable comments on our tool. Table 7.2 summarizes the comments that the participants made to DCTracVis. These comments include the following:

- One participant pointed out that it was not feasible for colour-blind users to discern nodes if we adopted inappropriate colours to represent the number of links that each node has in the treemap or the whole hierarchical tree and to differentiate the similarity value level of each link in the detail hierarchical tree.
- Five participants commented that colours used to identify the number of links in the treemap might confuse users, and it was hard to remember them.

TABLE 7.2 COMMENTS OF PARTICIPANTS MADE TO DCTRACTVIS

| Participant | Comments |
|-------------|--|
| 1 | Zooming needs to be improved; Color used may confuse users |
| 2 | None |
| 3 | It is hard to remember colors in treemap; The highlight items are not easy to detect. |
| 4 | There are rooms to improve the content window; may use word size to tell the similarity score |
| 5 | None |
| 6 | Good tool for traceability; But colors in treemap may confuse users; Change to use the size of node in treemap to display the number of links that the node has |
| 7 | Big window would be better to view |
| 8 | The contents of sections should highlight the related words to corresponding classes; Make the size of each square in treemap different to represent the number of links for each nodes |
| 9 | Each window can be moved around |
| 10 | More user friendly |
| 11 | None |
| 12 | None |
| 13 | None |
| 14 | It will have better visual effects on a bigger screen; Using colors that are not friendly/easily comprehensible to color blind/partially color blind people, may use different sizes of squares; A system documentation/manual around the system to assist learning is usage |
| 15 | None |
| 16 | More user friendly; more fancy |
| 17 | Explore novel method for measuring the similarity between the classes and the docs; The treemap uses color to identify the links, which is hard to remember. |
| 18 | The edit menu is a bit hard to use (right click on the node is not easy); Divide the treemap into 2 sections (maybe with different colours or divider) |
| 19 | Place a legend; Bigger view; Number of related link in the status |
| 20 | The heatmap coloring for both the map and hierarchical tree could be improved. Also the highlight color needs to be different to heatmap colors. Size of classes could represent actual lines of code. |

- Five participants suggested that it would be helpful to use different sizes of nodes to represent the number of links that each node has, or to reflect the sizes of classes and sections in the system, or to differentiate the similarity score levels.
- Two participants commented that it was not easy to quickly notice the selected node and its related nodes.
- Two participants suggested that words related to a selected node should be highlighted

when showing the contents of the related node.

- Five participants commented that it would have better visual effect if it improved the zooming, used a big screen or big view windows, or allowed view windows to be editable.
- Two participants thought that it would be helpful to provide summary documents about the traced system's information, such as how many packages, documents, classes, retrieved links, related links of each node, and so on.
- One participant suggested that the treemap should be divided into two sections using different colours.
- Two participants commented that it needed to be more user friendly or fancier.

7.4.4 Comparative Questions of Visualization Views

We initially employed the tree view to display a traced system and traceability links in it (discussed in Chapter 3). In order to ameliorate the performance of the initial link visualization view, we adopted the treemap view and the whole hierarchical tree view to visualize links in the system (discussed in Chapter 6). We wanted to learn which visualization view the participants preferred with regard to browsing links, maintaining links, and supporting them in the comprehension and maintenance and development of a system.

Figure 7.7 shows the evaluation results of a comparison made between the three visualization views. No participants liked to use the tree view. One participant commented that it was hard to use the tree view because it was too cluttered and messy. More participants preferred the whole hierarchical tree to the treemap in browsing links (12 vs. 8). 11 participants preferred the treemap to maintain links and to support them in understanding the traced system, but 9 participants preferred the whole hierarchical tree. For the support of maintenance and development of a system, 10 preferred the treemap and the others preferred the hierarchical tree. Overall, 12 participants thought that the treemap was the best one, while the others (8) preferred the hierarchical tree.

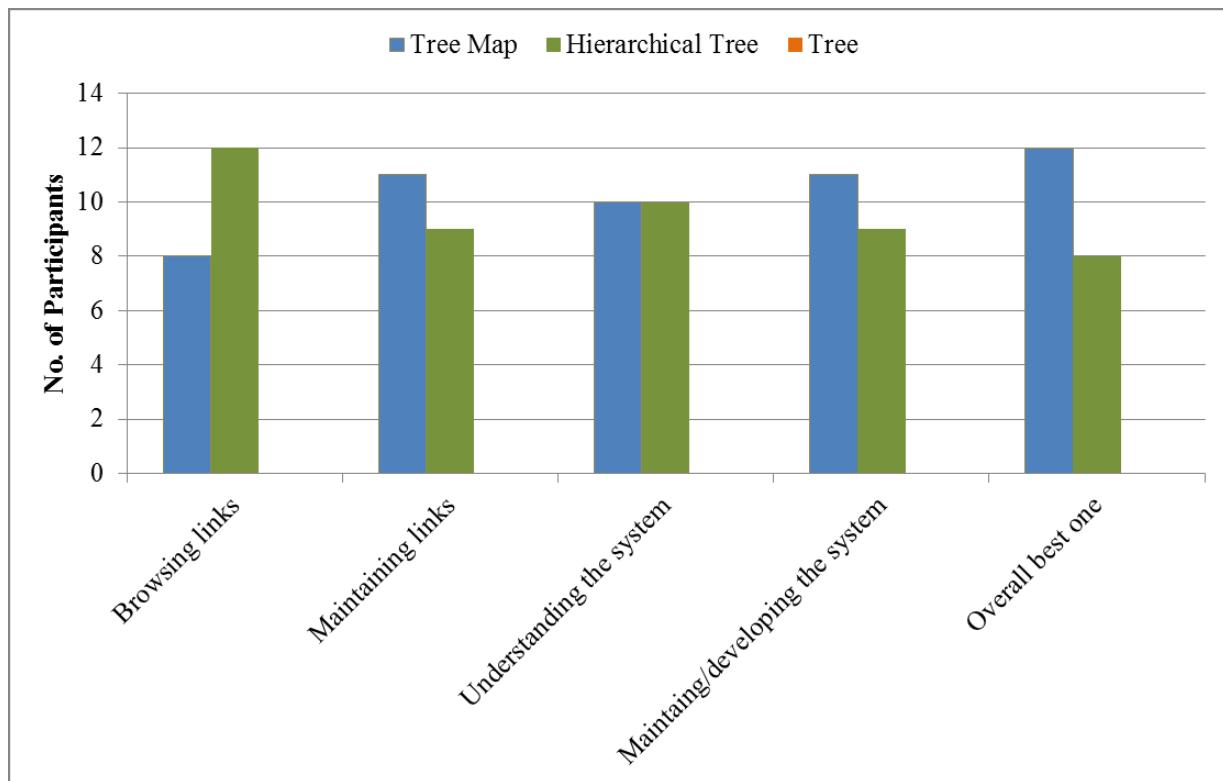


FIGURE 7.7 RESULTS OF COMPARISON AMONG THE THREE VISUALIZATION VIEWS

Table 7.3 summarizes the reasons given by each participant in their determination of which visualization view is the best. The main reasons that the participants thought the treemap was the best visualization view included the following.

- 4 participants pointed out that the detail hierarchical tree was a good supplement to the treemap, and their combination was easy to use.
- 2 participants reasoned that the treemap was easy to move around and easy to map the relationships between documents and classes.
- 2 participants commented that the treemap presented more information than the whole hierarchical tree without being as cluttered as the tree view, and that it showed the complete system and allowed them to look into details.
- 2 participants ranked the treemap as the best based on colour and overall view, and it was more useful for maintaining as well as browsing than the whole hierarchical tree.

The main reasons that the participants disliked the treemap included the following:

- 1 participant commented that the treemap was compact but classes' names were not shown inside the map, and also the maps for the packages and document were identical, making it difficult to remember which map was for packages or documents.
- 1 participant mentioned that the treemap was a quite good-looking, rich-user-experience but it was not easy to perform tasks such as navigate to a class, or edit or delete a link.

TABLE 7.3 REASONS GIVEN BY PARTICIPANTS IN DECIDING THE BEST VISUALIZATION VIEW

| Participant | Reasons |
|-------------|---|
| 1 | Treemap and the selected information are good combination. They support to each other. |
| 2 | It is more obvious to use hierarchical tree to browse links |
| 3 | It is easy to browse links in hierarchical tree although users have to go through it step-by-step. But navigator, search make it easy to find links |
| 4 | None |
| 5 | They are all good, but hierarchical tree is more intuitive |
| 6 | None |
| 7 | Hierarchical tree is also good, but it needs go step by step to find a node |
| 8 | Treemap plus hierarchical tree for details are easy to use |
| 9 | Treemap is easy to move around and the detailed tree is a great support to treemap |
| 10 | Hierarchical tree is easy to browse but the combination of treemap and hierarchical tree are very easy to use |
| 11 | Treemap presents more information than the hierarchical tree without being as cluttered as the tree view |
| 12 | None |
| 13 | The treemap is compact but the name of the classes are not shown inside the map. Also the map for the packages and document are quite identical making it difficult to remember which map is for packages, which map is for document. The tree view is too clutter and messy. |
| 14 | Both treemap and hierarchical tree are good but treemap is more useful for maintaining as well as browsing. |
| 15 | Treemap is easy to map the relationship between doc and classes |
| 16 | The treemap is quite good-looking rich-user-experience but not easy to operate tasks such as navigate a class, edit or delete a attribute. The hierarchical tree is easy to use & observable. |
| 17 | The hierarchical tree is more comprehensive. |
| 18 | Hierarchical tree seems to be easier to use, you can see which node (class) you are trying to edit and it is more organized than tree. |
| 19 | Tree map is the best based on colour and overall view |
| 20 | Treemap shows the complete system and allows to look into details |

The reasons that the participants who chose the whole hierarchical tree as the best included:

- 4 participants commented that the whole hierarchical tree was easier to use and easy to browse links.
- 2 participant though that the whole hierarchical tree was more intuitive and comprehensive than others.

One participant commented that a disadvantage of the whole hierarchical tree was that users had to browse through the hierarchical tree step-by-step to find a node. Overall, both the treemap and the whole hierarchical tree have their strengths and weaknesses.

TABLE 7.4 COMMENTS OF PARTICIPANTS MADE TO THE VISUALIZATION VIEWS

| Participant | Comments |
|-------------|---|
| 1 | More fancy, animation would be better |
| 2 | None |
| 3 | None |
| 4 | Animation |
| 5 | None |
| 6 | None |
| 7 | Fancy interface |
| 8 | None |
| 9 | More fancy |
| 10 | None |
| 11 | None |
| 12 | None |
| 13 | None |
| 14 | None |
| 15 | There is room to improve for visualization. |
| 16 | Animations & highlighting item. Class diagram is the traditional way that software developer understand the system & easy to make sense what relationship between classes |
| 17 | None |
| 18 | None |
| 19 | None |
| 20 | None |

Table 7.4 summarizes the suggestions participants gave to the visualization views. 6 participants suggested that there was room to improve the visualization view to be fancier and to provide more animations. One participant also commented that software developers preferred to use a class

diagram as it was the traditional way for developers to understand the system, and it was easy to make sense what relationships between classes.

7.5 Discussion

The usability evaluation obtained positive results. The participants could recover traceability links in a traced system effectively and efficiently. The participants also could easily browse links and find a specific node. Furthermore, the participants could easily and conveniently modify links of a node. In addition, our tool supported the comprehension of links.

The usability evaluation showed that both the treemap and the hierarchical tree have their advantages and disadvantages. 12 participants were satisfied with the performance of the treemap. 8 thought that the hierarchical tree outperformed the treemap. We combined the treemap and the hierarchical tree to visualize links in the system and adopted another hierarchical tree to display the detailed link information of a node. Our visualization approach took advantage of the strengths of each of them to ameliorate limitations of each of them. Our tool provided multiple approaches to visualize links to meet different participants' needs. Our tool allowed the participants to easily gain the structure of the traced system and the overall overview of links in it.

However, the usability evaluation exposed some weaknesses with our tool. We propose some possible solutions and improvements for these weaknesses.

- 4 participants experienced difficulty in recognizing/remembering colors used in the detail hierarchical tree. It would be more intuitive to employ different font sizes and/or colors of nodes in the detail hierarchical tree to display their similarity value levels and to make the important links more visible.
- 2 participants felt that the selected node and its related links were not noticeable. It would be more noticeable to make the selected node and its related nodes stand out from other nodes in the treemap by enlarging these nodes.
- 2 participants felt that contents in the content window were hard to read. In the contents window, it would be more readable to highlight words that are related to the selected node.

- 6 participants encountered difficulty in remembering colors that differentiated the number of links level. To apply or combine other methods to represent the relationship status of each node in the treemap and the whole hierarchical tree would make the view more intuitive.
- 11 participants appeared to have high expectations regarding the ability to edit using the windows of the navigator, the filter, the treemap, the whole hierarchical tree, and the detail hierarchical tree, and the ability to move them around. It would be more visually effective to separate them into different independent and editable view windows.
- 2 participants appeared to be disappointed that our tool didn't provide a summary report about the traced system after adopting the filter. It would be more understandable to supply a summary report to briefly introduce the traced system whenever the filter is used.

These propositions all represent potential future work for refining our tool. Some other interesting future work includes the following: (1) In order to retrieve more correct links and fewer fault links, users are allowed to edit the regular expressions used to match words in documents; (2) Users are allowed to add new key phrases or edit/delete existing key phrases to refine extracted key phrases to recover more related links; (3) To include a history navigation, which allows users to learn the history of their movements and activities and to undo or redo previous activities. (4) Any modifications made to traceability links need to be saved.

There are two limitations of our tool. (1) The size of each node in the treemap becomes small in order to display a system with large numbers of artifacts in one screen. (2) The three color ranges used in the treemap and the whole hierarchical tree may need to be extended to clearly distinguish nodes if the range of numbers of links that nodes have becomes large. Allowing user configuration of colours and colour gradients may be helpful, especially for colour-blind users.

7.6 Summary

This chapter described the usability evaluation that assessed the effectiveness of DCTracVis. The overall feedback from the evaluation participants was that our tool performed well and was both

helpful and useful. Our tool was able to extract traceability links in a system easily and effectively. Our tool also allowed users to easily browse links and to quickly locate a specific link. Moreover, it allowed users to easily and conveniently maintaining links. In addition, it supported the comprehension of links and provided the hierarchical structure of the system and the overall overview of links.

The study showed that both the treemap and the hierarchical tree have their strengths and weaknesses in visualizing the traced system and traceability links in it. Our combined visualization approach took the advantages to reduce the limitations of each of them to meet different users' needs. This study explored some weaknesses of our tool. We proposed some solutions and improvements to address these deficiencies.

The next chapter presents conclusions together with the contributions of this research.

Chapter 8 – Conclusions

This chapter concludes this thesis by summarizing the presented research in responding to the research questions described in Chapter 1, listing the research contributions, discussing the limitations of our research, and suggesting some possible future work to extend our research.

8.1 Thesis Summary

This thesis describes a traceability link recovery and visualization system, DCTracVis. This system is intended to provide users with an electronic environment for recovering and visualizing traceability links in a traced system in Eclipse. It provides users with both Eclipse IDE and traceability support. Users can not only use the functionality provided within the Eclipse IDE but can also use our visualization prototype as a stand-alone tool.

A traceability recovery tool, IRETrace, is integrated with DCTracVis to automatically retrieve links between artifacts in the traced system. IRETrace adopts a combination recovery technique that incorporates three enhancement strategies, Regular Expression (RE), Key Phrases (KP), and Clustering, into Information Retrieval (IR) models to ameliorate the key limitations of IR by taking advantage of the respective strengths of each of the three enhancement techniques. We evaluated our combination recovery approach using four case studies and six IR models. Our experimental results demonstrated that our recovery approach can effectively eliminate some limitations of IR models. Adding RE can significantly augment the number of true links at all cut points. The KP enhancement can retrieve more true links than IR alone. Combining Clustering significantly reduces the incorrect links at all cut points. Our approach improves precision at all cut points, reduces incorrect links at low cut points, and increases the number of true links at high

cut points. Furthermore, our approach provides reasonable precision and recall at all cut points. We thus conclude that our experimental results clearly answered the first research question: *the performance of an IR-based traceability recovery technique can be improved to retrieve high quality links at all cut points through incorporating supporting strategies with it to remedy the limitations of IR.*

In order to assist users in manually building affordable and robust traceability benchmarks to evaluate a traceability recovery technique, we presented an approach/guideline that comprises five steps: task identification, artifact selection, project selection, oracle/true traceability link set development, and evaluation metrics. This approach guides users to build their own benchmarks efficiently and effectively. We designed rigorous identification and verification strategies to assist researchers in the link recovery and verification. Before the determination of whether or not a link is true, every link is verified by at least two analysts. A benchmark for JDK1.5-SUBSET was created using our proposed guideline. The probability of $\geq 5\%$ link errors made in the JDK1.5-SUBSET oracle link set is around 0.1%. The analysis of error probability results show that our rigorous identification and verification strategies can help researchers to build a high quality oracle link set for the selected project. Moreover, the visit of the set of traceability benchmark requirements identified by Dekhtyar et al. (2007) shows that our proposed guideline can produce benchmarks that satisfy these requirements. We therefore conclude that the second research question was addressed: *researchers can manually build a robust, effective traceability benchmarks by following the five steps and the rigorous identification and verification strategies.*

After being retrieved by IRETrace, links are displayed as well as the structure of the system. DCTracVis utilizes a combination traceability visualization approach that integrates treemap and hierarchical tree 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. The treemap view illustrates the overall structure of the system and the overall overview of traceability links. In order to reduce visual clutter, colours are adopted to represent the relationship status of each node instead of directly drawing edges between related nodes on top of the treemap. Two hierarchical trees (HTs) are employed to visualize links. The whole HT view represents the complete system under trace and links in it to convey the hierarchical structure of the system. The detail HT view

supplements the treemap and the whole HT and displays the detailed dependency information of a node selected in the treemap or the whole HT. These traceability links can be modified (add, delete, edit). Their similarity scores also can be changed. The three views are interactive; changes made in one view can be reflected in the other two views, and vice versa. DCTracVis also provides the support of the navigator, search, and filter functions to assist users in locating a specific node or filtering out some uninterested links. We conducted a usability evaluation to evaluate DCTracVis. The overall feedback from the participants was that our tool performed well and was both helpful and useful. Our tool was able to extract traceability links in a system easily and effectively. It also allowed users to easily browse links and to quickly locate a specific link. Moreover, it allowed users to easily and conveniently maintain links. In addition, our tool supported the comprehension of links and provided the hierarchical structure of the system and the overall overview of links. We then can reach a conclusion that the third research question was addressed: *a traceability visualization system that can effectively and efficiently visualize the structure and links of a traced system and provide supporting functionality (e.g. navigator, search, or filter) can assist users in comprehending, browsing, and maintaining traceability links in the system.*

Finally, we can draw a conclusion that *effective and efficient traceability between artifacts in a system can help software engineers to understand, maintain, and manage the system.*

The limitations of our research, as revealed by the evaluations, include the following aspects.

- 1) The main limitation of our combination recovery approach in IRETrace is that some true links are discarded after adding Clustering. This is because the group containing links related to a particular class is totally removed when no clusters for this group are created. True links in such groups are cut.
- 2) The guideline of manual establishment of traceability benchmarks suffers from four issues.
 - a. It is difficult to determine whether or not two elements in artifacts are in fact related because we rely on participants' knowledge and understanding to capture links.
 - b. It is not easy to decide how much workload is suitable for a participant to undertake. The more workload is allocated to a participant, the more time and energy are required. Too much workload may make participants lose interest in participation.

- c. It is not easy to recruit a good number of participants who have some knowledge of the selected project, especially when recruiting senior analysts. If a participant is new to the selected project, he/she might be more likely to capture incorrect links than someone who knows the project to some extent.
 - d. The approach has scalability issues due to the need to manually identify and verify traceability links.
- 3) Limitations of DCTracVis are as follows:
- a. The size of each node in the treemap shrinks in order to display a system with large numbers of artifacts in one screen.
 - b. The three colour ranges used in the treemap and the whole hierarchical tree may need to be extended to clearly distinguish nodes if the range of numbers of links that nodes have becomes large.
 - c. Using colour to differentiate the number of links level may cause difficulty in recognizing/remembering their corresponding indications.
 - d. The selected node and its related links from other nodes are not distinct.
 - e. The contents of an artifact in the content window are hard to read.
 - f. The windows of the navigator, the filter, the treemap, the whole hierarchical tree, and the detail hierarchical tree cannot be edited or moved around.
 - g. Our tool fails to provide a summary report about the traced system after applying a filter.

8.2 Research Contributions

Our research presented and discussed in this thesis contributes to the knowledge on software traceability. The main contributions of this thesis include the following:

- 1) The traceability recovery tool, IRETrace, which adopts a combination recovery approach to improve the automated traceability recovery between artifacts in a system. This combination approach integrates IR models with three supporting techniques, RE, KP, and Clustering. This tool provides an accessible, easy-to-use environment for automatically recovering links

between artifacts using different combination approaches. Due to its ability to automatically capturing high quality links, this tool significantly reduces the time and effort needed for users to establish links between artifacts. Using four case studies and six IR models, we proved that our combination recovery approach can produce high quality links at any cut point.

- 2) The guidelines for manually building traceability benchmarks, which provides step by step guidelines in manually establishing an affordable and robust benchmark. It adopts rigorous strategies to identify and verify links. Using a case study and a formula that calculates the probability of errors in created benchmarks, we proved that our guidelines can build high quality benchmarks. We believe that our guidelines will be useful to other researchers to build their own benchmarks manually.
- 3) The benchmark for JDK1.5-SUBSET built by following the guidelines. This can be accessed or downloaded free from: <http://tinyurl.com/7l3ohe4>. The probability of making $\geq 5\%$ errors in this benchmark is very low, around 0.1%. We believe that this benchmark will also be useful for other researchers to apply when evaluating their traceability approaches, and to extend to meet their own needs better.
- 4) The traceability visualization system, DCTracVis, which adopts treemap and hierarchical tree techniques to represent the structure and traceability links of a traced system without scalability and visual clutter issues. It utilizes IRETrace to automatically retrieve links between artifacts. This system provides an environment for recovering, browsing, and maintaining links. It also provides navigation, filter, and search functions to assist users in locating a specific node and filtering out uninterested links. Due to its ability to automatically recover high quality links and visualizing links in a natural and intuitive way, this system significantly reduces the time and effort for users to maintain links. Using a usability evaluation, we established that DCTracVis can assist users in the comprehension, browsing, maintenance of traceability links.

8.3 Future Work

Several areas for possible further research are as follows:

- 1) Some future work for IRETrace:

- a. Adopting other key phrases extraction techniques to accelerate the execution time of our approach.
 - b. Allowing users to edit or delete existing extracted key phrases in each class, or add new key phrases related to the purpose of classes.
 - c. Allowing users to edit the IR queries to delete unwanted key words or add new key words.
 - d. Exploring other techniques to cope with abbreviation, synonym, and polysemy problems through taking account of relations between terms or words.
 - e. Before applying Clustering to refine retrieved links, reweighting the similarity value of each retrieved link based on the frequency of the class occurrence in the section to increase similarity values of links that are really relevant to each other. For example, if the class name is mentioned n times in the section, the similarity value is increased by $20\% + n\%$. If the class methods/functions are mentioned in the section, the similarity between them is further increased by 20%. If the section also contains comments in the class, the similarity value takes a further increase of 20% for class comments, 10% for method/function comments, and 5% for other comments.
 - f. Exploring the impact of other techniques to refine the extracted links such as our visual IDE's user creation and editing of links and both user and automated ranking of relationship quality.
- 2) Some future work for the guidelines in building traceability benchmarks.
- a. Extending the JDK1.5-SUBSET benchmark to cover more classes and documents. This benchmark could then be used to evaluate tracing approaches and procedures for a wide range of tasks in different areas of software engineering.
 - b. Extending the probability formula by applying other probability distributions to cover the issue that links may have different probabilities for being retrieved. Retrieving a link highly relies on the textual descriptions: some links may be more difficult than others to capture due to the way in which they are written and hence a lower or higher probability of being in error. The binomial distribution we have applied then may become invalid as clustering may occur. However, using different probability distribution is unlikely to significantly affect the low error rates computed using the binomial distribution. Because the

probability of errors in the created oracle link set depends heavily on the probability of participants simultaneously making the same mistake ($Pr(y_i)$), which is very low.

- c. Exploring how correct links should be defined, how to define traceability rules to assist participants in identifying correct links, and how to validate rules correctness.
 - d. Using other benchmarks that have been used by other researchers to compare their precision and recall results with ours.
 - e. Exploring how to create general traceability benchmarks and share them with other researchers.
- 3) Some future work for DCTracVis.
- a. Employing different font sizes and/or colours of nodes in the detail hierarchical tree to display their similarity value levels and to make the important links more visible.
 - b. Making the selected node and its related nodes stand out from other nodes in the treemap by enlarging these nodes.
 - c. Highlighting words that are related to the selected node in the contents window.
 - d. Applying or combining other methods to represent the relationship status of each node in the treemap and the whole hierarchical tree.
 - e. Separating the windows of the navigator, the filter, the treemap, the whole hierarchical tree, and the detail hierarchical tree into different, independent and editable view windows.
 - f. Supplying a summary report to briefly introduce the traced system whenever the filter is used.
 - g. Allowing the editing of the regular expressions used to match words in documents to retrieve more correct links and fewer fault links.
 - h. Allowing users to add new key phrases or edit/delete existing key phrases to refine extracted key phrases to recover more related links.
 - i. Including a history navigation to allow users to learn the history of their movements and activities and to undo or redo previous activities.
 - j. Any modifications made to traceability links need to be saved.

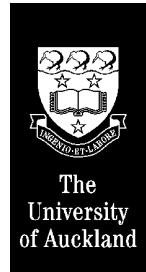
8.4 Summary

Our research arose from the need to retrieve high quality traceability links between artifacts in a system and to visualize these retrieved links for users to understand, browse, and maintain them in an effective and easy way. A combination traceability recovery approach that combined RE, KP, and Clustering with IR models was developed to ameliorate the limitations of IR and demonstrated via four case studies and six IR models. Furthermore, a combination traceability visualization technique that uses treemap and hierarchical tree techniques was designed and developed to convey the structure and links of a system. A formal usability evaluation was conducted to evaluate and proved the usefulness of our traceability system – DCTracVis. In addition, a guideline was proposed to provide guidance for manually creating affordable and robust traceability benchmark.

Appendix A – Surveys

The following pages include the following:

1. The Participant Information Sheet (Head of Department)
2. The Consent Form (Head of Department)
3. The Participant Information Sheet (Researchers)
4. The Consent Form (Researchers)
5. The Participant Information Sheet (Management)
6. The Consent Form (Management)
8. The Participant Information Sheet (Industries)
9. The Consent Form (Industries)
10. The Invitation Email
- 11 Survey Questionnaires



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

PARTICIPANT INFORMATION SHEET (HEAD OF DEPARTMENT)

Title: DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code

My name is Xiaofan Chen and I am a PhD student in the Department of Computer Science at the University of Auckland under the supervision of Prof. John Hosking and Prof. John Grundy. I am conducting research on automatically extracting traceability links between documents and source code inside software systems and visualizing the retrieved links. I am investigating how and whether traceability links visualization can support users in browsing and maintaining traceability links, and comprehending, programming and managing software systems. I have developed a prototype called **DCTracVis** of such a traceability tool. Part of our research involves including potential users in the design, usability testing, evaluation of the prototype's effectiveness and accuracy for traceability links extraction and in providing visualization support for the retrieved links.

We are seeking the participation of candidates with a Computer Science or Software Engineering background for this evaluation study. As the Head of Department of Computer Science, we would like to ask your permission to allow us to have access to students and staff members who have a background of software development and/or programming and to support and permit the students and staff members to participate voluntarily in our study. Your support is of the utmost importance to this research and is highly and deeply appreciated by us.

Participation in this study takes approximately 40 minutes. Participants are given the Participant Information Sheet and Consent Form that explain this study and the terms and conditions of participation. If they agree to participate, they need to sign the Consent Form. Both documents will be collected immediately after they agree to participate in the evaluation and before they start with the evaluation. Next they are asked to perform a number of tasks on paper, using a computer. The tasks will be fully explained and demonstrated. They will then be asked to fully explore DCTracVis by browsing, finding and modifying traceability links. The mouse movements they undertake to complete tasks on the compute and the time they spend working on each task will be digitally recorded. They will not be audio-taped or recorded by any other electronic means such as Digital Voice Recorders. They will be observed based on the following aspects: (a) how they manage to complete tasks given to them; (b) how they explore the tool to browse and find links; (c) how they navigate different functions of the tool; and (d) their verbal responses while using the

tool. The observation data will be recorded anonymously. They will be asked to fill in a short questionnaire to note their status, existing experience with the tasks and technology and complete a questionnaire on their experience of using DCTracVis. Once they completed the questionnaire, they need to submit it in a sealed envelope that will be provided to them after they agree to participate. There will be no coding to their questionnaire as it is treated anonymously.

The observation and questionnaire data will be compiled and analysed, and the results will be used for a PhD thesis and for other academic publications. The digital recordings, with their specific consent, may be used in research reports on this project. The participant consent forms will be held in a secure file for 6 years, at the end of this time they will be properly disposed of. Participants' names will not be used in any reports arising from this study. The information collected during this study may be used in future analysis and publications and will be kept indefinitely. When it is no longer required all copies of the data will be destroyed. At the conclusion of the study, a summary of the findings will be available from the researcher upon request.

If a person we approach do not want to participate, they don't have to give any reason for their decision. If they do decide to participate, they may withdraw at any time during the session without explanation and without penalty. Completing the required tasks in the survey and submitting the evaluation is an indication of consent but as the evaluation is anonymous, no answers can be withdrawn once the evaluation is submitted. If the participant is a student or staff member at The University of Auckland choosing not to participate, or to withdraw or their information, their grades or relationship with the University or other members of staff will not be affected.

This project is partly supported by funding from the Foundation for Research, Science and Technology.

Contact Details:

Thank you very much for your time and help in making this study possible. If you have any questions at any time you can contact my supervisor Prof. John Hosking: Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland 1142. Email: john@cs.auckland.ac.nz. Tel: 3737599 ext 88297.

For any queries regarding ethical concerns, please contact The Chair, The University of Auckland Human Participants Ethics Committee, The University of Auckland, Office of the Vice Chancellor, Private Bag 92019, Auckland 1142. Tel. 3737599 ext 83711.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651.



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

CONSENT FORM (HEAD OF DEPARTMENT)
This consent form will be held for a period of at least six years

Title: DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code

Researcher: Xiaofan Chen

I have been given and understood an explanation of this research project. I have had an opportunity to ask questions and have them answered. I understand that at the conclusion of the study, a summary of the findings will be available from the researcher upon request.

I agree to support this evaluation study.

I agree to allow the researcher to have access to the students and staff members who have a background of software development and/or programming in my Department.

I agree to permit the students and staff members to participate voluntarily in the study.

I understand that the data collected from the study will be held indefinitely and may be used in future analysis.

I understand that the participation for participants in this study will take about 40 minutes.

I understand that participants may withdraw their participations during the session at any time. But no answers can be withdrawn once the evaluation is submitted.

I agree to provide the assurance that the students and staff members' grades and/or relationships with The University of Auckland or members of staff will be unaffected whether or not they participate in this study or withdraw their participation during it.

I understand that digital recordings are taken while participants manage to complete tasks during the session and results from them will be used in research reports on this project.

Signed:

Name:

(please print clearly)

Date:

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

PARTICIPANT INFORMATION SHEET (RESEARCHERS)

Title: **DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code**

To participants:

My name is Xiaofan Chen and I am a PhD student in the Department of Computer Science at the University of Auckland under the supervision of Prof. John Hosking and Prof. John Grundy. I am conducting research on automatically extracting traceability links between documents and source code inside software systems and visualizing the retrieved links. I am investigating how and whether traceability links visualization can support users in browsing and maintaining traceability links, and comprehending, programming and managing the software systems. I have developed a prototype called **DCTracVis** of such traceability tool. Part of our research involves including potential users in the design, usability testing, evaluation of the prototype's effectiveness and accuracy for traceability links extraction and in providing visualization support for the retrieved links.

You are invited to participate in our research as your computer science or software engineering background and we would appreciate any assistance you can offer us, although you are under no obligation to do so.

Your participation in this study takes approximately 40 minutes. You are given the Participant Information Sheet and Consent Form that explain this study and the terms and conditions of participation. If you agree to participate, you need to sign the Consent Form. Both documents will be collected immediately after you agree to participate in the evaluation and before you start with the evaluation. Next you are asked to perform a number of tasks on paper, using a computer. The tasks will be fully explained and demonstrated. You will then be asked to fully explore DCTracVis by browsing, finding and modifying traceability links. The mouse movements you undertake to complete tasks on the computer and the time you spend working on each task will be digitally recorded. You will not be audio-taped or recorded by any other electronic means such as Digital Voice Recorders. You will be observed based on the following aspects: (a) how you manage to complete tasks given to you; (b) how you explore the tool to browse and find links; (c) how you navigate different functions of the tool; and (d) your verbal responses while using the tool. The observation data will be recorded anonymously. You will be asked to fill in a short questionnaire to note your status, existing experience with the tasks and technology and complete a questionnaire on your experience of using DCTracVis. Once you completed the questionnaire, you need to submit it in a sealed envelope that will be provided to you after you agree to participate. There will be no coding to your questionnaire as it is treated anonymously.

The observation and questionnaire data will be compiled and analysed, and the results will be used for a PhD thesis and for other academic publications. The digital recordings, with your specific consent, may be used in research reports on this project. Your consent form will be held in a secure file for 6 years, at the end of this time it will be properly disposed of. Your name will not be used in any reports arising from this study. The information collected during this study may be used in future analysis and publications and will be kept indefinitely. When it is no longer required all copies of the data will be destroyed. At the conclusion of the study, a summary of the findings will be available from the researcher upon request.

If you don't want to participate, you don't have to give any reason for your decision. If you do decide to participate, you may withdraw at any time during the session without explanation and without penalty. Completing the required tasks in the survey and submitting the evaluation is an indication of consent but as the evaluation is anonymous, no answers can be withdrawn once the evaluation is submitted. If you are a student or staff member at The University of Auckland choosing not to participate, or to withdraw yourself or your information, your grades or relationships with the University or members of staff will not be affected. This assurance is given by the Head of Department of Computer Science.

This project is partly supported by funding from the Foundation for Research, Science and Technology.

Contact Details:

Thank you very much for your time and help in making this study possible. If you have any questions at any time you can contact my supervisor Prof. John Hosking: Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland 1142. Email: john@cs.auckland.ac.nz. Tel: 3737599 ext 88297.

For any queries regarding ethical concerns, please contact The Chair, The University of Auckland Human Participants Ethics Committee, The University of Auckland, Office of the Vice Chancellor, Private Bag 92019, Auckland 1142. Tel. 3737599 ext 83711.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651.



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

CONSENT FORM (RESEARCHERS)

This consent form will be held for a period of at least six years

Title: DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code

Researcher: Xiaofan Chen

I have been given and understood an explanation of this research project. I have had an opportunity to ask questions and have them answered. I understand that at the conclusion of the study, a summary of the findings will be available from the researcher upon request.

I understand that the data collected from the study will be held indefinitely and may be used in future analysis.

I understand that participation will take about 40 minutes.

I understand that I may withdraw my participation during the session at any time. But no answers can be withdrawn once the evaluation is submitted.

I understand that my grades and relationships with The University of Auckland will be unaffected whether or not I participate in this study or withdraw my participation during it. This assurance is given by the Head of Department of Computer Science.

I understand that digital recordings are taken during the session and results from them will be used in research reports on this project.

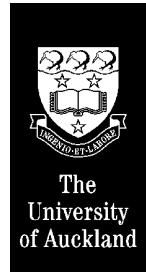
I agree to take part in this research by completing the session.

Signed:

Name:
(please print clearly)

Date:

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651.



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

PARTICIPANT INFORMATION SHEET (MANAGEMENT)

Title: **DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code**

My name is Xiaofan Chen and I am a PhD student in the Department of Computer Science at the University of Auckland under the supervision of Prof. John Hosking and Prof. John Grundy. I am conducting research on automatically extracting traceability links between documents and source code inside software systems and visualizing the retrieved links. I am investigating how and whether traceability links visualization can support users in browsing and maintaining traceability links, and comprehending, programming and managing software systems. I have developed a prototype called *DCTracVis* of such a traceability tool. Part of our research involves including potential users in the design, usability testing, evaluation of the prototype's effectiveness and accuracy for traceability links extraction and in providing visualization support for the retrieved links.

We are seeking the participation of candidates with software programming background for this evaluation study. As a manager of the company, we would like to ask your permission to allow us to have access to your staff members who have a background of software development and/or programming and to support and permit the staff members to participate voluntarily in our study. Your support is of utmost important to this research and is highly and deeply appreciated by us.

Participation in this study takes approximately 40 minutes. Participants are given the Participant Information Sheet and Consent Form that explain this study and the terms and conditions of participation. If they agree to participate, they need to sign the Consent Form. Both documents will be collected immediately after they agree to participate in the evaluation and before they start with the evaluation. Next they are asked to perform a number of tasks on paper, using a computer. The tasks will be fully explained and demonstrated. They will then be asked to fully explore DCTracVis by browsing, finding and modifying traceability links. The mouse movements they undertake to complete tasks on the compute and the time they spend working on each task will be digitally recorded. They will not be audio-taped or recorded by any other electronic means such as Digital Voice Recorders. They will be observed based on the following aspects: (a) how they manage to complete tasks given to them; (b) how they explore the tool to browse and find links; (c) how they navigate different functions of the tool; and (d) their verbal responses while using the tool. The observation data will be recorded anonymously. They will be asked to fill in a short questionnaire to note their status, existing experience with the tasks and technology and complete a questionnaire on their experience of using DCTracVis. Once they completed the questionnaire,

they need to submit it in a sealed envelope that will be provided to them after they agree to participate. There will be no coding to their questionnaire as it is treated anonymously.

The observation and questionnaire data will be compiled and analysed, and the results will be used for a PhD thesis and for other academic publications. The digital recordings, with their specific consent, may be used in research reports on this project. The participant consent forms will be held in a secure file for 6 years, at the end of this time they will be properly disposed of. Participants' names will not be used in any reports arising from this study. The information collected during this study may be used in future analysis and publications and will be kept indefinitely. When it is no longer required all copies of the data will be destroyed. At the conclusion of the study, a summary of the findings will be available from the researchers upon request.

If a person we approach does not want to participate, they don't have to give any reason for their decision. If they do decide to participate, they may withdraw at any time during the session without explanation and without penalty. Completing the required tasks in the survey and submitting the evaluation is an indication of consent but as the evaluation is anonymous, no answers can be withdrawn once the evaluation is submitted. If they choose not to participate, or to withdraw themselves or their information, their relationships with the company or other members of staff will not be affected.

This project is partly supported by funding from the Foundation for Research, Science and Technology.

Contact Details:

Thank you very much for your time and help in making this study possible. If you have any questions at any time you can contact my supervisor Prof. John Hosking: Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland 1142. Email: john@cs.auckland.ac.nz. Tel: 3737599 ext 88297.

For any queries regarding ethical concerns, please contact The Chair, The University of Auckland Human Participants Ethics Committee, The University of Auckland, Office of the Vice Chancellor, Private Bag 92019, Auckland 1142. Tel. 3737599 ext 83711.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651.



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

CONSENT FORM (MANAGEMENT)

This consent form will be held for a period of at least six years

Title: DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code

Researcher: Xiaofan Chen

I have been given and understood an explanation of this research project. I have had an opportunity to ask questions and have them answered. I understand that at the conclusion of the study, a summary of the findings will be available from the researcher upon request.

I agree to support this evaluation study.

I agree to allow the researcher to have access to my staff members who have a background of software development and/or programming.

I agree to permit the staff members to participate voluntarily in the study.

I understand that the data collected from the study will be held indefinitely and may be used in future analysis.

I understand that the participation for participants in this study will take about 40 minutes.

I understand that participants may withdraw their participations during the session at any time. But no answers can be withdrawn once the evaluation is submitted.

I agree to provide the assurance that the staff members' relationships with the company or other members of staff will be unaffected whether or not they participate in this study or withdraw their participation during it.

I understand that digital recordings are taken while participants manage to complete tasks during the session and results from them will be used in research reports on this project.

Signed:

Name:

(please print clearly)

Date:

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651.



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

PARTICIPANT INFORMATION SHEET (INDUSTRIES)

Title: **DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code**

To participants:

My name is Xiaofan Chen and I am a PhD student in the Department of Computer Science at the University of Auckland under the supervision of Prof. John Hosking and Prof. John Grundy. I am conducting research on automatically extracting traceability links between documents and source code inside software systems and visualizing the retrieved links. I am investigating how and whether traceability links visualization can support users in browsing and maintaining traceability links, and comprehending, programming and managing the software systems. I have developed a prototype called **DCTracVis** of such traceability tool. Part of our research involves including potential users in the design, usability testing, evaluation of the prototype's effectiveness and accuracy for traceability links extraction and in providing visualization support for the retrieved links.

You are invited to participate in our research as your computer science or software engineering background and we would appreciate any assistance you can offer us, although you are under no obligation to do so.

Your participation in this study takes approximately 40 minutes. You are given the Participant Information Sheet and Consent Form that explain this study and the terms and conditions of participation. If you agree to participate, you need to sign the Consent Form. Both documents will be collected immediately after you agree to participate in the evaluation and before you start with the evaluation. Next you are asked to perform a number of tasks on paper, using a computer. The tasks will be fully explained and demonstrated. You will then be asked to fully explore DCTracVis by browsing, finding and modifying traceability links. The mouse movements you undertake to complete tasks on the compute and the time you spend working on each task will be digitally recorded. You will not be audio-taped or recorded by any other electronic means such as Digital Voice Recorders. You will be observed based on the following aspects: (a) how you manage to complete tasks given to you; (b) how you explore the tool to browse and find links; (c) how you navigate different functions of the tool; and (d) your verbal responses while using the tool. The observation data will be recorded anonymously. You will be asked to fill in a short questionnaire to note your status, existing experience with the tasks and technology and complete a questionnaire on your experience of using DCTracVis. Once you completed the questionnaire, you need to submit it in a sealed envelope that will be provided to you after you agree to participate. There will be no coding to your questionnaire as it is treated anonymously.

The observation and questionnaire data will be compiled and analysed, and the results will be used for a PhD thesis and for other academic publications. The digital recordings, with your specific consent, may be used in research reports on this project. Your consent form will be held in a secure file for 6 years, at the end of this time it will be properly disposed of. Your name will not be used in any reports arising from this study. The information collected during this study may be used in future analysis and publications and will be kept indefinitely. When it is no longer required all copies of the data will be destroyed. At the conclusion of the study, a summary of the findings will be available from the researcher upon request.

If you don't want to participate, you don't have to give any reason for your decision. If you do decide to participate, you may withdraw at any time during the session without explanation and without penalty. Completing the required tasks in the survey and submitting the evaluation is an indication of consent but as the evaluation is anonymous, no answers can be withdrawn once the evaluation is submitted. If you choose not to participate, or to withdraw yourself or your information, your relationships with your company or other members of staff will not be affected. This assurance is given by the manager of the company.

This project is partly supported by funding from the Foundation for Research, Science and Technology.

Contact Details:

Thank you very much for your time and help in making this study possible. If you have any questions at any time you can contact my supervisor Prof. John Hosking: Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland 1142. Email: john@cs.auckland.ac.nz. Tel: 3737599 ext 88297.

For any queries regarding ethical concerns, please contact The Chair, The University of Auckland Human Participants Ethics Committee, The University of Auckland, Office of the Vice Chancellor, Private Bag 92019, Auckland 1142. Tel. 3737599 ext 83711.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651.



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

CONSENT FORM (INDUSTRIES)

This consent form will be held for a period of at least six years

Title: DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code

Researcher: Xiaofan Chen

I have been given and understood an explanation of this research project. I have had an opportunity to ask questions and have them answered. I understand that at the conclusion of the study, a summary of the findings will be available from the researcher upon request.

I understand that the data collected from the study will be held indefinitely and may be used in future analysis.

I understand that participation will take about 40 minutes.

I understand that I may withdraw my participation during the session at any time. But no answers can be withdrawn once the evaluation is submitted.

I understand that my relationships with my company or other members of staff will be unaffected whether or not I participate in this study or withdraw my participation during it. This assurance is given by the manager of the company.

I understand that digital recordings are taken during the session and results from them will be used in research reports on this project.

I agree to take part in this research by completing the session.

Signed:

Name:
(please print clearly)

Date:

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651.



Department of Computer Science
The University of Auckland
Science Centre Building 303S
38 Princes St
Auckland 1142
New Zealand
Phone: 09 373 7599 ext 88260

Invitation Email

My name is Xiaofan Chen and I am a PhD student in the Department of Computer Science at the University of Auckland under the supervision of Prof. John Hosking and Prof. John Grundy. I am conducting research on automatically extracting traceability links between documents and source code inside software systems and visualizing the retrieved links. I am investigating how and whether traceability links visualization can support users in browsing and maintaining traceability links, and comprehending, programming and managing software systems. I have developed a prototype called **DCTracVis** of such a traceability tool. Part of our research involves including feedback from potential users in the design, usability testing and evaluation of the prototype's effectiveness and accuracy for traceability links extraction and in providing visualization support for the retrieved links.

In this study we are testing the usability of a prototype implementation of **DCTracVis**. The studies are being conducted either in the Computer Science Lab at the University of Auckland or at your offices. They will take a maximum of 40 minutes.

You are invited to participate in our research and we would appreciate any assistance you can offer us, although you are under no obligation to do so. The studies will take place between 01/10/2011 and 01/06/2012. If you would like to participate please email/phone me to arrange a time, xche044@aucklanduni.ac.nz or 09 3737599 ext 88260

Regards

Xiaofan Chen

This research has been approved by The University of Auckland Human Participants Ethics committee on 16 November 2011 for a period of 3 years from 16 November 2011. Reference 2011/7651.



The
University
of Auckland

Department of Computer Science
The University of Auckland
Private Bag 92019
Auckland Mail Centre
Auckland 1142
New Zealand

Tel: +64 (9) 373 7599 x 88260

Survey: Evaluation of DCTracVis: Extraction and Visualization of Traceability Links between Documents and Source Code

Note: This survey is structured into Three parts. Part one provides brief description of the prototype "Documents Code Traceability Visualization Tool" (**DCTracVis**). Part two is the three tasks that participants need to perform. An observation data will be collected by PhD student Xiaofan Chen. After completing the tasks, a list of questions in Part three need to be answered.

Part One: Introduction of DCTracVis

Source code alone is not sufficient to capture all information about a software system. Software requirements, architectural decisions, detailed design, tutorials and user documentation, and various types of technical system documentation (e.g. deployment configuration) are important artifacts produced while engineering software systems. Tracing and maintaining interrelationships between these various forms of software documentation and source code enables users to better understand systems, undertake improved maintenance of systems, and ultimately to produce higher quality systems [1, 2, 3].

DCTracVis (Figure 1) we developed is a tool that can automatically retrieve traceability links between documents and source code, and visualize the recovered links. Traceability links are retrieved by using an approach that combines three supporting techniques, Regular Expression, Key Phrases, and Clustering, with Information Retrieval (IR) models. Then the recovered links are visualized in the three visualization views in the Eclipse IDE to help users easily browse these relationships. The three visualization views include Tree Map (Figure 2), Hierarchical Tree (Figure 3), and Tree (Figure 4).

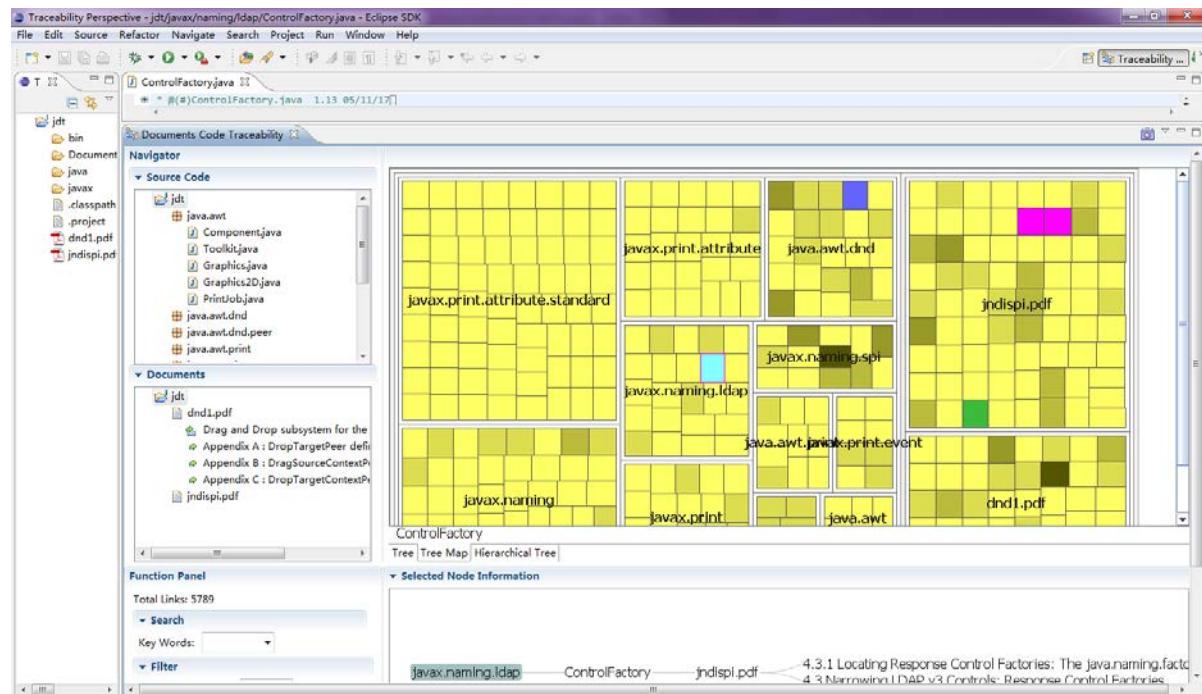
References:

- [1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. "Recovering traceability links between code and documentations". *TSE* 28 (10), Oct. 2002, pp. 970-983
- [2] G. Antoniol, G. Casazza, and A. Cimitile. "Traceability recovery by modelling programmer behavior". *7th WCRE*, Queensland, Australia, Nov. 2000, pp. 240-247
- [3] R. Seacord, D. Plakosh, and G. Lewis. "*Modernizing legacy systems: software technologies, engineering processes, and business practices*". 2003, Addison-Wesley

Figure 1 is the user interface of **DCTracVis**. It includes four parts: Navigator, Visualization window, Function Panel, and Selected Node Information window.

- Navigator allows users to navigate through the tree to find a specific artefact.
- Visualization window includes Tree Map view, Hierarchical Tree view, and Tree view. The three views show the structure of the system and allow users to browse and maintain links.
- Function Panel allow users to do Search and filter traceability links
- Once users select a node in Navigator or visualization views, detailed information of the specific artefact will be shown in the Selected Node Information window show

Figure 1: the user interface of DCTracVis



The following are the three visualization views: Tree Map (Figure 2), Hierarchical Tree (Figure 3), and Tree (Figure 4).

Figure 2: Tree Map:

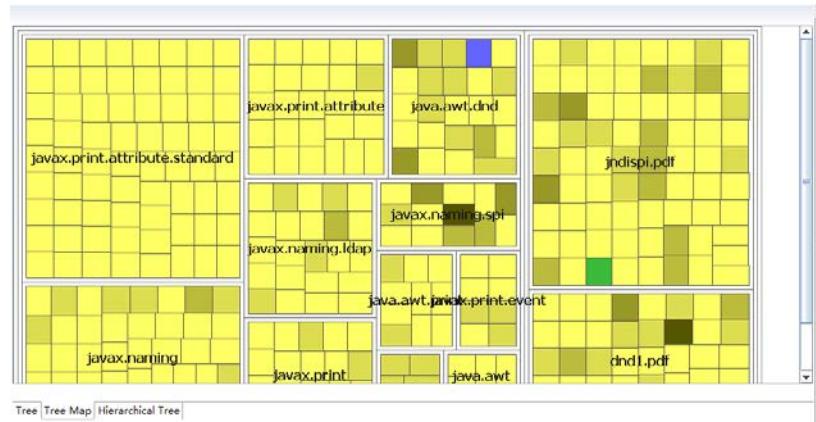
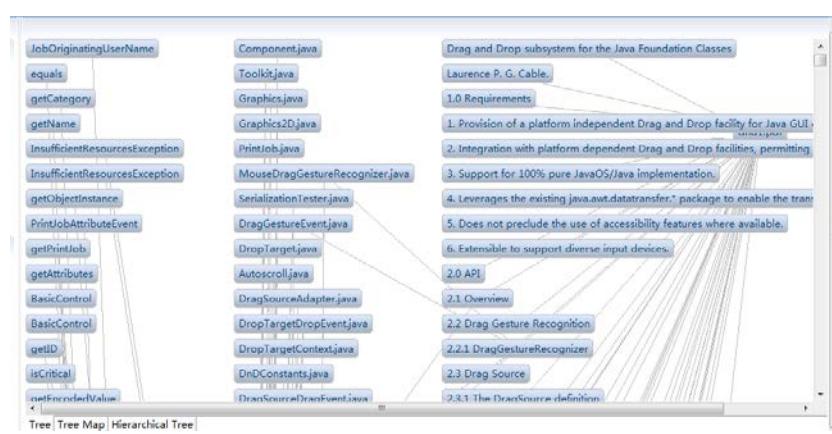


Figure 3: Hierarchical Tree:



Figure 4: Tree:



The case study we use to evaluate **DCTracVis** is JDK1.5-SUBSET, a free software system for Java developers. The following Table describes the packages in JDK 1.5 and their corresponding PDF documents used in this study, as well as the number of Java classes and the number of sections in them. We divide these PDF files into sections based on their headings.

| JDK 1.5-SUBSET | | #classes/ sections |
|--------------------------|--|-------------------------------|
| Java packages | java.awt, javax.naming, and javax.print packages | 249 |
| PDF files | JPS_PDF.pdf: Java™ Print Service API User Guide | 68 |
| | dnd1.pdf: Drag and Drop subsystem for the Java Foundation Classes | 41 |
| | jndispi.pdf: Java Naming and Directory Interface™ Service Provider Interface(JNDI SPI) | 73 |
| | Total sections: | 182 |

Part Two: Tasks

*Please perform the three tasks using the "Documents Code Traceability Visualization Tool" (**DCTracVis**) as described below.*

Task 1: Understand the JDK1.5-SUBSET system

Image you are new to the JDK1.5-SUBSET system. You want to get to know how this system is organized, how artefacts inside this system are connected to each other.

Start time: _____

- How is JDK1.5-SUBSET organized?
 - ✧ Use visualization views or Navigator to find out the number of packages, number of documents, types of documents, structure of documents, etc.
- How are artefacts inside JDK1.5-SUBSET connected to each other?
 - ✧ Use visualization views to browse links between classes and sections in documents
 - ✧ Use Filter to find out what links have similarity score ≥ 0.7 , what classes/sections have more than 1 links etc.

Finish time: _____

Task 2: Understand how a class works

Image you find a bug related to **InitialLdapContext.java** in the package of **javax.naming.ldap**. You want to find out how this class works in order to fix it. You need to find out where the documentation of this class can be found and what other classes are related to this class.

Start time: _____

- Find this class and its links
 - ✧ Use Navigator or Search or visualization views to find this class
- Find sections in documents that describe this class
 - ✧ Use the right click menu to show the content of related sections
- Find classes that are probably related to this class
 - ✧ Use “Selected Node Information” or visualization views to find other classes related to these sections

Finish time: _____

Task 3: Modify traceability links of a class

Image you have the five traceability links information of **InitialContextFactoryBuilder.java** in the package of **javax.naming.spi**. This class is related to the following sections (these links are recognized as **true/correct links**):

1. jndispi.pdf → 1.2 Interface overview
2. jndispi.pdf → 1.2.1 Namingmanager and directorymanager
3. jndispi.pdf → 1.2.2 Initial contexts
4. jndispi.pdf → 3 The initial context
5. jndispi.pdf → 3.3.2 Removing all policy

Start time: _____

- Find this class and look at links of this class provided by **DCTracVis**
 - ◊ Use Navigator or Search or visualization views to find this class
- If links provided by **DCTracVis** are not correct, you can simply delete them, or edit them to be true links
 - ◊ Use the right click menu to delete or edit links
- Add some true links if they are missed by **DCTracVis**
 - ◊ Use the right click menu to add links
- Change the similarity score of some links provided by **DCTracVis** to 1, if the link is true and the score calculated by **DCTracVis** is very low.
 - ◊ Use the right click menu to change the similarity score of links in “Selected Node Information”.

Finish time: _____

Part Three: Questionnaire

Please complete the following questions.

Section A: Background Information

Below is a list of questions about your background.

1. I have read the **Participant Information Sheet** and have understood the nature of the research and I agree to take part in this research.
 - a. Yes
 - b. No
 2. I am:
 - a. A Student
 - b. An Academic
 - c. An Industry Participant
 3. How many years of software development experience do you have?
 - a. < 1 year
 - b. 1 -- 5 years
 - c. 5 -- 10 years
 - d. > 10 years
 4. I use Eclipse for programming Java systems.
 - a. Always
 - b. Usually
 - c. Sometimes
 - d. Rarely
 - e. Never
 5. I use traceability tools to aid me in comprehending or maintaining or programming software systems.
 - a. Always
 - b. Usually
 - c. Sometimes
 - d. Rarely
 - e. Never

Section B: DCTracVis Usability

Below is a list of questions that describe your perception of the DCTracVis.

1. Do you feel that you successfully completed all tasks on the task sheet?
a. Yes b. No

*For the following questions, please select the answer you feel most accurately relates to **DCTracVis** in comparison to other software you have used, other approaches/tools that can be used to extract/find links e.g. manually extracting/finding links.*

2. It is easy to use.
a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree

 3. It helps me more effective in extracting traceability links between artefacts within systems
a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree

 4. It makes it easier to extract traceability links.
a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree

5. It makes it easier to maintain traceability links (add, delete, edit).
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
6. It makes it easier to browse traceability links.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
7. It makes it easier to find traceability links.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
8. It is useful to support me in comprehending the system.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
9. It is useful to support me in maintaining/developing the system.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
10. The various functions in this tool are well integrated and easy to find.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
11. All the functions I expected are all present.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
12. It is user friendly.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
13. I learned to use it quickly.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
14. It is easy to learn how to use it.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
15. I would like to use it in the future.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
16. I would recommend it to friends.
 a. Strongly agree b. Agree c. Neutral d. Disagree e. Strongly disagree
17. What other comments or suggestions would you like to make to the **DCTracVis**?

Section C: Comparative Questions of visualization views

*Below is a list of questions that describe your perception towards the three link visualization views provided in the **DCTracVis**.*

1. Which visualization view do you prefer with regard to browsing traceability links?
a. Tree Map b. Hierarchical Tree c. Tree
2. Which visualization view do you prefer with regard to maintaining traceability links?
a. Tree Map b. Hierarchical Tree c. Tree
3. Which visualization view do you prefer with regard to supporting you in understanding the system?
a. Tree Map b. Hierarchical Tree c. Tree
4. Which visualization view do you prefer with regard to supporting you in maintaining/developing the system?
a. Tree Map b. Hierarchical Tree c. Tree
5. Overall, which view is the best one?
a. Tree Map b. Hierarchical Tree c. Tree

Why?

6. What other comments or suggestions would you like to make to visualization views?

Bibliography

- Abadi, A., Nisenson, M., and Simionovici, Y. 2008. A traceability technique for specifications. *16th IEEE International Conference on Program Comprehension*, pp. 103-112
- ADAMS 2009. Overview. Data accessed: February 2009, <http://adams.dmi.unisa.it/adams-2009/Overview.html>
- Amati, Giambattista. 2003. *Probability models for information retrieval based on divergence from randomness*. PHD thesis. University of Glasgow. <http://theses.gla.ac.uk/1570/>
- Antoniol, G., Canfora, G., Casazza, G., and Lucia, A. D. 2000a. Information retrieval models for recovering traceability links between code and documentation. *In Proc. Of IEEE Intl. Conf. On Software Maintenance 2000*, San Jose, USA
- Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D., and Merlo, E. 2002. Recovering traceability links between code and documentations. *TSE*, Vol. 28, No. 10, Oct., pp. 970-983
- Antoniol, G., Casazza, G., and Cimitile, A. 2000b. Traceability recovery by modelling programmer behavior. *7th WCRE*, Queensland, Australia, Nov. 2000, pp. 240-247
- Apache Lucene – Overview, 2009, Extracted on 4 December 2009 from <http://lucene.apache.org/java/docs/>
- Asuncion, H. U., Francois, F., and Taylor, R. N. 2007. An end-to-end industrial software traceability tool. *ESEC-FSE 2007*, Sep. 3-7, Cavtat near Dubrovnik, Croatia, pp. 115-124
- Bacchelli, A., D'Ambros, M., and Lanza, M. 2010. Extracting source code from E-mails. *18th ICPC'10*, pp. 24-33
- Bacchelli, A., D'Ambros, M., Lanza, M., and Robbes, R. 2009. Benchmarking lightweight techniques to link e-mails and source code. *In Proceeding of WCRE 2009*, pp. 205-214
- Bacchelli, A., Lanza, M., and Robbes, R. 2010. Linking E-mails and source code artifacts. *ICSE'10*, May, pp.375-384
- Boehm, B. 2003. Value-based software engineering. *Software Engineering Notes*, 28(2), pp. 1-12
- Brooke, John. 1996. SUS – A quick and dirty usability scale. *Usability Evaluation in Industry*, London: Taylor and Francis. From <http://www.itu.dk/courses/U/E2005/litteratur/sus.pdf>
- Capobianco, G., Lucia, A. D., Oliveto, R., Panichella, A., and Panichella, S. 2009. Traceability recovery using numerical analysis. *16th Working Conference on Reverse Engineering*, pp. 195-204
- Charrada, E.B., Caspar, D., Jeanneret, C., and Glinz, M. 2011. Towards a benchmark for traceability. *IWPSE-EVOL 11*, Sep. 5-6, Szeged, Hungary, pp. 21-30
- Cleland-Huang, J., Czauderna, A., Dekhtyar, A., Gotel, O., Hayes, J.H., Keenan, E., Leach, G., Maletic, J., Poshyvanyk, D., Shin, Y., Zisman, A., Antoniol, G., Berenbach, B., Egyed, A., and Maeder, P. 2011. Grand challenges, benchmarks, and TraceLab: developing infrastructure for the software traceability research community. *TEFSE 2011*, May 23, Waikiki, Honolulu, HI, USA

- Cleland-Huang, J., Dekhtyar, A., Hayes, J.H., Antoniol, G., Berenbach, B., Eyged, A., Ferguson, S., Maletic, J., and Zisman, A. 2006. Grand challenges in traceability. *Technical Report COET-GCT-06-01-0.9, Center of Excellence for Traceability*, 2006
- Cleland-Huang, J. and Habrat, R. 2007. Visual support in automated tracing. *REV 2007*, IEEE Computer Society, pp. 4-8
- Cleland-Huang, J., Settimi, R., Duan, C., and Zou, X. 2005. Utilizing supporting evidence to improve dynamic requirements traceability. *RE '05*, Paris, Aug. 2005, pp.135-144
- 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
- Cockburn, A. 2008. Using both incremental and iterative development. *The Journal of Defense Software Engineering*, May 2008, pp. 27-30
- Configuring retrieval in Terrier, 2010. Extracted from http://terrier.org/docs/v3.0/configure_retrieval.html
- Conklin, J. and Begeman, M. 1988. gIBIS: a hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6, pp. 303-331
- CORE. 2012, extracted from <http://www.vtcorp.com>
- Cornelissen, B., Holten, D., Zaidman, A., Moonen, L., van Wijk, J.J., and van Deursen, A. 2007. Understanding execution traces using massive sequence and circular bundle views. *15th ICPC 2007*, Alberta, BC, pp. 49-58
- Cover, T. M. and Thomas, J. A. 1991. *Elements of information theory*. Wiley-Interscience
- Dagenais, B. and Hendren, L. 2008. Enabling static analysis for partial Java programs. *23rd Conference on Object-Oriented Programming Systems Languages and Applications*, pp. 313-328
- Dagenais, B. and Robillard M. P. 2012. Recovery traceability links between an API and its learning resources. *ICSE 2012*, Zurich, Switzerland, pp. 47-57
- Dekhtyar, A., Hayes, J.H., and Antoniol, G. 2007. Benchmarks for traceability? *TEFSE 2007*, March, Lexington, KY
- Divergence From Randomness (DFR) framework, 2011. Extracted from http://terrier.org/docs/v3.5/dfr_description.html
- Domges, R. and Pohl, K. 1998. Adapting traceability environments to project specific needs. *CACM*, 1998, 41(12), pp. 54-62
- Domingos, P. and Pazzani, M. 1997. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29 (2/3), pp. 103-130
- Dumas, J. And Redish, C. 1999. *A practical guide to usability testing (Revised Edition)*. Intellect Books, pp. 21-39, ISBN 1841500208
- Easterbrook, S. M., Singer, J., Storey, M-A., and Damian, D. 2007. Selecting empirical methods for software engineering research. *Guide to Advanced Empirical Software Engineering*, Springer
- Eclipse Java Development Tools (JDT), 2010, Extracted on 7 April 2010 from <http://www.eclipse.org/jdt>

- Edwards, M. and Howell, S. 1992. *A Methodology for Requirements Specification and Traceability for Large Real-Time Complex Systems*. Technical Report, Naval Surface Warfare Center
- Egyed, A. 2003. A scenario-driven approach to trace dependency analysis. *TSE*, 29(2), pp. 116-132
- Egyed, A. 2006. Trace Analyzer tool: a mini tutorial. http://www.alexander-egyed.com/tools/trace_analyzer_tool.html
- 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
- Egyed, A., Binder, G. and Grunbacher, P. 2007. STRADA: a tool for scenario-based feature-to-code trace detection and analysis. *29th International Conference on Software Engineering (ICSE'07 Companion)*, 2007 IEEE
- Fjeldstad, R. and Hamlen, W. 1983. Application program maintenance-report to our respondents. *Tutorial on Software Maintenance*, 13-27. Parikh, G. & Zvezintzov, N. (Eds.). IEEE Computer Soc. Press.
- GATE Information Extraction, 2010, Extracted on 4 August 2010 from <http://gate.ac.uk/ie/>
- Gena, C., & Weibelzahl, S. 2007. Usability Engineering for the Adaptive Web. In P. Brusilovsky, A. kobsa & W. Nejdl (Eds.), *The Adaptive Web, LNCS* (Vol. 4321, pp. 720-762): Springer-Verlag Berlin Heidelberg
- Gibiec, M., Czauderna, A., and Cleland-Huang, J. 2010. Towards mining replacement queries for hard-to-retrieve traces. *ASE 2010*, September, Antwerp, Belgium, pp. 245-254
- Gotel, O.C. and Finkelstein, A. C. W. 1994. An analysis of the requirements traceability problem. *1st RE*, pp. 94-101
- Graham, M. and Kennedy, J. 2010. A survey of multiple tree visualization. *Information Visualization*, 2010, Vol. 9(4), pp. 235-252
- Grechanik, M., McKinley, K. S., and Perry, D. E. 2007. Recovering and using use-case-diagram-to-source-code traceability links. *ESEC/FSE'07*, Cavtat, Croatia, pp. 95-104
- Greenspan, S.J. and McGowan, C.L. 1978. Structuring Software Development for Reliability. *Microelectronics and Reliability*, vol. 17, 1978
- Gupta, C. and Grossman, R. 2004. GenIc: a single pass generalized incremental algorithm for clustering. *2004 SIAM International Conference on Data Mining*, pp. 137-153
- Hamilton, V.L. and Beeby, M.L. 1991. Issues of Traceability in Integrating Tools. *Proc. IEE Colloquium Tools and Techniques for Maintaining Traceability during Design*, Dec. 1991
- Hayes, J. H., Dekhtyar, A., and Osborne, J. 2003. Improving requirements tracing via information retrieval. *Proc. Int'l Conf. Requirements Eng. (RE)*, pp. 151-161, Sept. 2003
- Hayes, J. H., Dekhtyar, A., and Sundaram, S. K. 2006. Advancing candidate link generation for requirements tracing: the study of methods. *TSE*, Vol. 32, No. 1, January, pp. 4-19

- Hayes, J. H., Dekhtyar, A., Sundaram, S. K., Holbrook, E. A., Vadlamudi, S., and April, A. 2007. Requirements Tracing On target (RETRO): improving software maintenance through traceability recovery. *Innovations Syst Softw Eng* (2007) 3, pp. 193-202
- He, B. and Ounis, I. 2005. Term frequency normalisation tuning for BM25 and DFR model. *27th European Conf. On Information Retrieval (ECIR)*, 2005, pp. 200-214
- He, B. and Ounis, I. 2007. Combining fields for query expansion and adaptive query expansion. *Information Processing and Management*, 43(2007), pp. 1294-1307
- Herman, I., Melancon, G., and Marshall, M. 2000. Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics*, Vol.6, Issue 1, January, pp. 24-43
- Hofmann, T. 1999. Probabilistic latent semantic indexing. In *SIGIR*, pages 50-57
- Holten, D. 2006. Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 5, 2006, pp. 741-748
- IBM Rational DOORS: Teleologic DOORS. 2012, extracted from [heep://www.teleologic.com/products/doors](http://www.teleologic.com/products/doors)
- Jin, D. and Cordy, J. 2005. Ontology-based software analysis and reengineering tool integration: the OASIS service-sharing methodology. *21st ICSM 2005*. pp.613-616
- Jirapanthong, W. and Zisman, A. 2005. Supporting product line development through traceability. *12th APSEC'05*, IEEE Computer Society, pp.506-514
- Jirapanthong, W. and Zisman, A. 2009. XTraQue: traceability for product line systems. *Software and System Modeling Journal*, Vol. 8, No. 1, pp. 1619-1366
- Kaindl, H. 1992. The missing link in requirements engineering. *Software Engineering Notes*, June, pp. 498-510
- KEA: keyphrase extraction algorithm. 2010. Extracted on 1 May 2010 from <http://www.nzdl.org/Kea/>
- Kienle, H. M. and Muller, H. A. 2007. Requirements of software visualization tools: a literature survey. *4th VISSOFT 2007*, pp. 2-9
- Kim, S. N. and Kan M. 2009. Re-examining automatic keyphrase extraction approaches in scientific articles. *2009 Workshop on Multiword Expressions, ACL-IJCNLP 2009*, August, Singapore, pp. 9-16
- Konchady, M. 2008. *Building search applications: Lucene, LingPipe, and Gate*, Musstru Publishing, Oakton, Virginia
- Lang, J. P. 2011. Redmine, from <http://www.redmine.org>
- Larman, C. 2005. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development (3rd Edit)*. Published by Prentice Hall
- Larman, C. and Basili, V. R. 2003. Iterative and incremental development: a brief history. *Computer*. June 2003, Vol. 36, No. 6, pp. 47-56
- LDRA, LDRA Product Brochure v7.2. 2012. From <http://www.ldra.com>

- Lindval, M. and Sandahl, K. 1996. Practical implications of traceability. *Software Practice and Experience*, Vol. 26, No. 10, pp. 1161-1180
- Lucia, A. D., Fasano, F., Francese, R., and Tortora, G. 2004. ADAMS: an artifact-based process support system. *16th Int. Conf. on Software Engineering and Knowledge Engineering*, 2004, Alberta, Canada, pp. 31-36
- Lucia, A. D., Fasano, F., Oliveto, R., and Tortora, G. 2005. ADAMS Re-Trace: a traceability recovery tool. *9th European Conference on Software Maintenance and Reengineering*, pp. 32-41
- Lucia, A. D., Fasano, F., Oliveto, R., and Tortora, G. 2007. Recovering traceability links in software artifact management systems using information retrieval methods. *TOSEM*, Vol. 16, No. 4, Article 13
- Lucia, A. D., Fasano, F., Oliveto, R., and Tortora, G. 2010. Fine-grained management of software artefacts: the ADAMS system. *Software Practice and Experience*, Vol. 40, pp. 1007-1034
- Lucia, A. D., Oliveto, R., and Sgueglia, P. 2006. Incremental approach and user feedback: a silver bullet for traceability recovery? *22nd ICSM 2006*, pp. 299-309
- Lucia, A. D., Oliveto, R., and Tortora, G. 2008. ADAMS Re-Trace: traceability link recovery via Latent Semantic Indexing. *ICSE 2008*, May, Germany, pp. 839-842
- Lucia, A. D., Penta, M. D., Oliveto, R., Panichella, A., and Panichella, S. 2011. Improving IR-based traceability recovery using smoothing filters. *19th IEEE International Conference on Program Comprehension*, pp. 21-30
- Lund, A. M. 2001. Measuring usability with the USE questionnaire. *STC Usability SIG Newsletter, Usability Interface*, October, Vol 8, No. 2, from http://www.stcsig.org/usability/newsletter/0110_measuring_with_use.html
- MacQueen, J. B. 1967. Some methods for classification and analysis of multivariate oberservations. *5th Berkeley Symp. On Math. Stat. and Prob.* pp. 281-297
- Marcus, A. and Maletic, J. I. 2003. Recovering documentation-to-source-code traceability links using latent semantic indexing. *25th ICSE'03*, pp. 125-135
- Marcus, A., Xie, X., and Poshyvanyk, D. 2005. When and how to visualize traceability links? *TEFSE 2005*, Nov. 8, California, USA, pp. 56-61
- Martin, R. C. 1999. Iterative and incremental development (IID). *Engineering Notebook Column, C++ Report*, from <http://www.objectmentor.com/resources/articles/IIDII.pdf>
- Merten, T., Juppner, D., and Delater, A. 2011. Improved representation of traceability links in requirements engineering knowledge using Sunburst and Netmap visualization. *4th MARK*, 2011, Trento, pp. 17-21
- Nielsen, J. 1993. *Usability engineering*. Boston: AP Professional
- Oliveto, R., Antoniol, G., Marcus, A., and Hayes, J. 2007. Software artefact traceability: the never-ending challenge. *ICSM 2007*, pp. 485-488
- Oppenheimer D., Brown, A. B., Traupman, J., Broadwell, P., and Patterson, D. A. 2002. Practical issues in dependability benchmarking. In *Evaluating and Architecting System Dependability*, October, 2002

- Palmer, J.D. 1997. Traceability. *Software Requirements Eng.*, R.H. Thayer and M. Dorfman, eds., pp. 364-374, 1997
- Penta, M. D., Gradara, S., and Antoniol, G. 2002. Traceability recovery in RAD software systems. *IWPC 2002*, pp. 207-216
- PDF Version of JDK Documentation, 2010, Extracted on 10 Feb. 2010 from <http://java.sun.com/j2se/1.5.0/download-pdf.html>
- Pfleeger, S.L. 1998. *Software Engineering: theory and practice*. Prentice Hall
- Pfleeger, S.L. and Bohner, S.A. 1990. A framework for software maintenance. *Proceeding of Conference on Software Maintenance*, Nov. 1990, pp. 320-327
- Pilgrim, J. von, Vanhooff, B., Schulz-Gerlach, I., and Bervers, Y. 2008. Constructing and visualizing transformation chains. *4th ECMDA-FA '08*, 2008, Heidelberg, pp. 17-32
- Pinheiro, F.A.C. and Goguen, A. 1996. An object-oriented tool for tracing requirements. *IEEE Software*, 13(2), pp. 52-64
- Pohl, K. 1996. *Process-centered requirements engineering*. John Wiley Research Science Press
- Prefuse, the prefuse information visualization toolkit, 2011, <http://prefuse.org/>
- Pulham, J. and Wills, S. 2008. *A new way to visualize requirements information and its traceability*. Telelogic white paper, An IBM Company
- Ramesh, B. and Jarke, M. 2001. Toward reference models of requirements traceability. *IEEE Trans. Software Eng.*, 27(1), pp. 58-93, 2001
- Ramesh, B., Stubbs, C., Powers, T., and Edwards, M. 1997. Requirements traceability: theory and practice. *Annals of Software Engineering 3 (1997)*, 1997, pp. 397-415
- Ramos, Juan. 2003 Using TF-IDF to determine word relevance in document queries. *First International Conference on Machine Learning*, 2003
- Rilling, J., Charland, P. and Witte, R. 2007. Traceability in Software Engineering—Past, Present and Future. *CASCON Workshop*, IBM Technical Report: TR-74-211, October 25
- Robertson, S. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*. Vol 60, No 5, pp. 503-520
- Robertson, S. and Walker, S. 1994. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. *In ACM Conf. On Research and Development in Information Retrieval (SIGIR)*, 1994, pp. 345-354
- Robey, D., Welke, R., and Turk, D. 2001. Traditional, iterative, and component-based development: a social analysis of software development paradigms. *Information Technology and Management*. Vol. 2, pp. 53-70
- Roman, G. C. and Cox, K. C. 1992. Program visualization: the art of mapping programs to picture. *Proc. Of Int. Con. on Software Engineering*, 1992, pp. 412-420.
- RTM, Integrated Chipware. 2012, extracted from <http://www.chipware.com>
- Salton, G. and Buckley, C. 1988. Term-weighting approaches in automatic text retrieval. *In Information Processing & Management*, 24(5), pp. 513-523
- Salton, G. and McGill, M. 1983. *Introduction to modern information retrieval*. McGraw-Hill

- Seacord, R., Plakosh, D., and Lewis, G. 2003. *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley
- Settimi, R., Cleland-Huang, J., Ben Khadra, O., Mody, J., Lukasik, W., and DePalma, C. 2004. Supporting software evolution through dynamically retrieving traces to UML artifacts. *7th IWPSE*, 2004, Kyoto, Japan, pp. 49-54
- Shneiderman, B. 1992. Tree visualization with tree-maps: 2d space-filling approach. *ACM Transactions on Graphics (TOG)*, 11(1), 1992, pp. 92-99
- Sim, S. E., Easterbrook, S., and Holt, R.C. 2003. Using benchmarking to advance research: a challenge to software engineering. *ICSE'03*, pp. 74-83
- Singhal, A. 2001. Modern information retrieval: a brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, Vol. 24, No. 4, pp. 35-42
- Spanoudakis, G. and Zisman, A. 2005. Software traceability: a roadmap. *Advances in Software Engineering and Knowledge Engineering*, Vol. 3: Recent Advances, (ed) S.K Chang, World Scientific Publishing, August 2005
- Sparck Jones, Karen. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28 (1): pp 11-21
- Sparck Jones, K., Walker, S., and Robertson, S. E. 2000. A probabilistic model of information retrieval: development and comparative experiments. *Information Processing and Management*, 36(6), pp. 779-840
- Standish, T. 1984. An essay on software reuse. *IEEE Transactions on Software Engineering* SE-10 (5), 494-497
- Stranieri, A. and Zeleznikow, J. 2005. *Knowledge discovery from legal databases (Law and Philosophy Library)*. Volume 69. Published by Springer Netherlands
- Tan P., Steinbach, M., and Kumar, V. 2005. *Introduction to data mining*. 1st edition, Addison Wesley
- TBreq, Requirements traceability with TBreq, 2012, extracted from <http://www.ldra.com/solutions/by-software-lifecycle/94-tbreq>
- 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
- Terrier IR platform, 2010, extracted from <http://terrier.org/>
- The Standish Group Report: Chaos, 2007. Extracted on 1 May 2010 from http://www.projectsmart.co.uk/docs/chaos_report.pdf
- Triola, M. F. 1997. *Elementary statistics (7th ed.)*. Addison Wesley Longman, Inc.
- Turney, P. 1999. Learning to extract keyphrases from text. *Technical Report ERB-1057*, National Research Council Canada, Institute for Information Technology, <http://arxiv.org/ftp/cs/papers/0212/0212013.pdf>
- van Ravensteijn, W.J.P. 2011. *Visual traceability across dynamic ordered hierarchies*. Master's thesis, Eindhoven University of Technology, August 2011

- van Wijk, J. J. and van de Wetering, H. 1999. Cushion treemaps: visualization of hierarchical information. *INFOVIS 99*, San Francisco, Oct. 25-26, 1999, pp. 1-6
- Voytek, J. B. and Nunez, J. L. 2011. Visualizing non-functional traces in student projects in information systems and service design. *CHI 2011*, May, 2011, Vancouver, Canada
- 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
- Watkins, R. and Neal, M. 1994. Why and how of requirements tracing. *5th Inter. Conf. Applications of Software Measurement*, Nov. 1994, California, pp.104-106
- Weiss, S. M., Indurkhya, N., Zhang, T., and Damerau, F. J. 2005. *Text mining: predictive methods for analyzing unstructured information*. Springer New York, 2005
- Winkler, S. and Pilgrim, J. 2010. A survey of traceability in requirements engineering and model-driven development. *Software System Model*, Vol. 9, September, pp. 529-565
- Witte, R., Li, Q., Informatic, F. F., Zhang, Y., and Rilling, J. 2008. Text mining and software engineering: an integrated source code and document analysis approach. *IET Software Journal*, Vol. 2, No. 1, 2008, pp. 1-19
- Witten, I. H., Paynter, G. W., Frank, E., Gutwin, C., and Nevill-Manning, C. G. 1999. Kea: practical automatic keyphrase extraction. *4th ACM DL*, Berkeley, pp. 254-255
- Yin, R. K. 2009. *Case study research: design and methods (4th edition)*. Sage Publications.
- Yoshikawa, T., Hayashi, S., and Saeki, M. 2009. Recovering traceability links between a simple natural language sentence and source code using domain ontologies. *Proc. ICSM 2009*, Edmonton, Canada, pp. 551-55
- Zest: the eclipse visualization toolkit. 2009. Extracted on 3 December 2009 from <http://www.eclipse.org/gef/zest>
- Zhang X., Hu, T., Dai, H., and Li X. 2010. Software development methodologies, trends and implications: a testing centric view. *Information Technology Journal*, Vol. 9, pp. 1747-1753
- Zhou, X., Huo, Z., Huang, Y., and Xu, J. 2008. Facilitating software traceability understanding with ENVISION. *COMPSAC 2008*, pp. 295-302
- Zou, X., Settimi, R., and Cleland-Huang, J. 2006. Phrasing in dynamic requirements trace retrieval. *30th COMPSAC 2006*, pp. 265-272
- Zou, X., Settimi, R., and Cleland-Huang, J. 2008. Evaluating the use of project glossaries in automated trace retrieval. *Software Engineering Research and Practice (SERP) 2008*, July, USA, 2 Volumes, pp. 157-163