DCTracVis: a system retrieving and visualizing traceability links between source code and documentation

Xiaofan Chen • John Hosking • John Grundy • Robert Amor

Abstract It is well recognized that traceability links between software artifacts provide crucial support in comprehension, efficient development, and effective management of a software system. However, automated traceability systems to date have been faced with two major open research challenges: how to extract traceability links with both high precision and high recall, and how to efficiently visualize links for complex systems because of scalability and visual clutter issues. To overcome the two challenges, we designed and developed a traceability system, DCTracVis. This system employs an approach that combines three supporting techniques, Regular Expressions, 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 the 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 correct links than IR alone. After having retrieved high-quality traceability links, DCTracVis then utilizes 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. Usability evaluation results show that our approach can effectively and efficiently help software developers comprehend, browse, and maintain large numbers of links.

This research is financially supported by the Foundation for Research, Science and Technology, MBIE Software Process and Product Improvement project, and University of Auckland

---

Xiaofan Chen
School of Computer Science and Engineering, Nanjing University of Science and Technology, China
e-mail: xiaofanchen@hotmail.com

John Hosking
Dean of Science, University of Auckland, New Zealand
e-mail: j.hosking@auckland.ac.nz

John Grundy
Senior Deputy Dean for the Faculty of Information Technology, Monash University, Australia
e-mail: john.grundy@monash.edu

Robert Amor
Department of Computer Science, University of Auckland, New Zealand
e-mail: trebor@cs.auckland.ac.nz

Keywords Software traceability · Traceability recovery· Traceability visualization

1 Introduction

It is well recognized that traceability links between artifacts play a critical role in program comprehension, maintenance, requirements tracing, impact analysis, reuse and management of a software system (Antoniol et al., 2000; Gotel and Finkelstein, 1994; Settimi et al., 2004; Watkins and Neal, 1994). Unfortunately, it is also painstaking, error-prone, complex and time-consuming work to manually retrieve and maintain traceability links between artifacts. These efforts can be significantly reduced by applying traceability recovery approaches to automatically obtain high quality relationships and 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 links in a natural and intuitive way (Asuncion et al., 2007; Roman and Cox, 1992). High quality links represent a link set containing as many as possible correct links and as few as possible incorrect links. Moreover, high quality links 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.

Most automatic traceability recovery approaches (Antoniol et al., 2002; Cleland-Huang et al., 2005; Hayes et al., 2003; Lucia et al., 2007; Marcus and Maletic; 2003; Wang et al., 2009) 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 (Hayes et al., 2003; Spanoudakis and Zisman, 2005). The precision of the extracted traceability links heavily depends on a threshold that decides which links can be recovered. There are two ways to determine the threshold (Lucia et al., 2007; Marcus and Maletic; 2003). One way is to determine a threshold for the similarity value that identifies which documents are linked. Only links that have a similarity value greater than the threshold will be captured. Another way is to impose a threshold 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 threshold. However, the lower the threshold is the greater the number of incorrect links that are retrieved. Conversely, the higher the threshold is the lesser the number of correct links that are retrieved. This means that many potentially useful and important links are missed at high thresholds. Similarly, many incorrect or unuseful links are extracted at low thresholds and may confuse developers. Furthermore, the same threshold may or may not be best suited for different systems.

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 and Cox, 1992). However, displaying a great many 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 a far from trivial problem (Cornelissen et al., 2007; Marcus et al., 2005). Many traceability visualization techniques (Cleland-Huang and Habrat, 2007; Cornelissen et al., 2007; Merten et al., 2011; van Ravensteijn, 2011; Zhou et al., 2008) have been designed and developed to represent traceability links. To date, however, no traceability visualization techniques can visualize a great many traceability links effectively and efficiently without scalability and visual clutter issues.

In order to remedy these issues, we designed and developed a traceability system, called DCTracVis, to capture and visualize traceability links between artifacts efficiently and effectively. Our traceability system employs a new traceability recovery approach that can automatically recover high quality links between artifacts in the traced system at all cut points, and a new traceability visualization technique to visualize retrieved links in a natural and intuitive way.

Our traceability recovery approach combines Information Retrieval (IR) models with three supporting techniques: Regular Expressions (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 recovery approach attempts to take advantage of the different strengths of the three enhancement techniques to increase precision at low levels of threshold and recall at high levels of threshold of links recovered by IR. We have conducted a detailed experiment to evaluate our recovery approach by applying six IR models to four case studies varying in size and context. Analysis of the experimental results illustrates that a combination of the three enhancement techniques can be used effectively to improve precision and recall of links retrieved by IR at all thresholds.

Our traceability visualization technique 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 have adopted 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. Our traceability system also includes navigation, 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 system for large traceability visualization problems. The results of this evaluation show that our 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.

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.

The rest of this paper is organized as follows. Related work is discussed in Section 2. Section 3 describes approaches for retrieving traceability links and visualizing them. A description of our traceability system and its implementation are described in Sections 4 and 5. Section 6 presents the evaluation results, followed by the analysis of these results in Section 7. Finally, we draw conclusions in Section 8.

2 Related work

Due to the importance of software traceability, extensive effort in the software engineering research community has been put into improving the precision and recall of recovered traceability links between artifacts through various traceability recovery techniques and the visualization of retrieved links in a natural and intuitive way through traceability visualization techniques.

2.1 Traceability link recovery

The most studied and often used techniques in automated traceability link recovery to date are Information Retrieval (IR) models (Antoniol et al., 2002; Cleland-Huang et al., 2005; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009). Early IR systems were Boolean models which use a complex combination of Boolean ANDs, ORs, and NOTs to specify users' needs (Hayes et al., 2003; 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. Most of them assign a similarity value to every document and rank documents by this value. (Hayes et al., 2003; Singhal, 2001)

Antoniol et al. (2002) applied two different IR models, Probabilistic Model (PM) and Vector Space Model (VSM), to extract links between code and documentation. Their results show that IR provides a practical solution for automated traceability recovery, and the two IR models have similar performance when terms in artifacts perform a preliminary morphological stemming. However, PM and VSM produce links at low levels of precision and reasonable levels of recall. Marcus and Maletic (2003) introduced Latent Semantic Indexing (LSI), an extension of the VSM, to recover links between documentation and source code. Their results show that although LSI achieves very good performance without the need for stemming, as required for PM and VSM, it suffers from the problem of low precision and high recall at low levels of thresholds or high precision and low recall at high levels of thresholds.

In order to improve the performance of IR models, many strategies have been developed. A traceability recovery tool based on PM was developed to explore how the

retrieval performance can be improved by modeling programmer behavior (Antoniol et al., 2000). Their results show that improvement in recall is achieved (Antoniol et al., 2000; Lucia et al., 2007). 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. Nevertheless, the three strategies have varying abilities to enhance link retrieval and are unable to work in all cases.

Settimi et al. (2004) investigated the effectiveness of VSM and VSM with a general thesaurus for generating links between requirements, code, and UML models. The comparison results show that precision and recall are not improved by the use of the 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. The results show that improvements in recall and sometimes in precision are achieved. Nishikawa et al. (2015) proposed the Connecting Links Method (CLM) to recover transitive traceability links. This approach first employs VSM to extract links between X and Z and between Y and Z, then connects these links to recover links between X and Y. Their results show that CLM is more effective than VSM alone as CLM can recover links in cases where VSM does not.

Wang et al. (2009) presented four enhanced strategies to improve LSI, namely, source code clustering, identifier classifying, similarity thesaurus, and hierarchical structure enhancement. Their comparison results indicate that this approach has higher precision than LSI and PM, but has lower recall. To improve IR-based traceability recovery, namely from the VSM, LSI and Jensen-Shannon models (JS), Lucia et al. (2013) applied a smoothing filter to remove "noise" from the textual corpus of artifacts. Their study indicates that a smoothing filter can remove "noise" that simple stop word filters cannot remove and can significantly improve the performances of traceability recovery. Falessi et al. (2017) proposed a new approach called ENRL which employs Machine Learning classifiers to estimate the number of remaining positive links in a ranked list of candidate traceability links produced by an NLP-based recovery approach (e.g. VSM, TF-IDF, or Latent Semantic Analysis etc.). Their results indicate that ENRL can provide accurate estimates of the number of remaining positive links but depends on the choice of the NLP technique. Kuang et al. (2017) combined IR techniques with closeness analysis to improve IR-based traceability recovery. The closeness analysis is to quantify the degree of interaction based on direct and indirect code dependencies among classes. Their results show that this approach outperforms IR-based approaches (e.g. VSM, JS, and LSI etc.).

Although various strategies have been applied to enhance the performance of IR techniques, no approaches can significantly decrease incorrect (fault) links at low levels of threshold and significantly increase correct (true) links at high levels of threshold (Antoniol et al., 2002; Cleland-Huang et al., 2005; Marcus and Maletic, 2003; Settimi et al., 2004; Wang et al., 2009). Our hypothesis is that augmenting IR with several complementary techniques - Regular Expressions, Key Phrases and Clustering - will significantly enhance both precision and recall.

2.2 Traceability link visualization

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; Li and Maalej, 2012). However, the matrix misses the inherent hierarchy and becomes unreadable when the set of artifacts becomes large (Voytek and 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 understanding the links (Li and Maalej, 2012). 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 and 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. Kamalabalan et al. (2015) developed a tool that visualizes relationships as a graph with nodes and edges. This tool uses the Neo4j Graph Database for modeling relationships. It allows users to overview the overall structure and to get more in-depth details using cluster views of filtered artefacts or relationships. Though the cluster views reduce visual clutter, the overview graph is getting larger and more complex when displaying a larger volume of data. Nakagawa et al. (2017) visualized the traceability links between specifications and test case descriptions in macro and micro views. In the macro view, nodes correspond to specifications and edges represent relatively high similarities between two specifications. The size of a node represents the number of assigned test case descriptions. The micro

view contains a specification list; each specification item is linked to a page that lists its related test case descriptions. However, the views have visual clutter issues.

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 (van Ravensteijn, 2011; van Amstel et Al., 2012), 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 are visualized (van Ravensteijn, 2011; van Amstel et Al., 2012). Rocco et al. (2013) used TraceVis to visualize the dependencies between artefacts and their related metamodel, and to help understanding how and where changes affect the system.

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 a 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. Multi-VisioTrace (Rodrigues et al., 2016) supports multiple visualization techniques: matrix view, tree view, sunburst and graph view. It allows users to choose the most appropriate to their tasks. This tool still leads to visual clutter when displaying a large volume of data.

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; 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 et al., 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 the traceability visualization techniques developed so far can visualize a great many 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; Marcus et al., 2005; Merten et al., 2011).

These issues motivated us to develop a visualization technique to enable engineers to recover, browse, modify, and maintain links effectively and efficiently. The discussion above 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 and Kennedy, 2010; Holten, 2006; Shneiderman, 1992); node-link representations (e.g. the hierarchical tree) that communicate structure readily, and the enclosure layout (e.g. the treemap) which 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 visualization approach combines several visualization techniques to provide more effective and efficient link visualization.

3 Our approach

We have developed a traceability system, called DCTracVis, to support software engineers to recover, browse, understand, and maintain links efficiently and effectively. Our traceability system employs a new traceability recovery technique to retrieve traceability links between artifacts and a new traceability visualization technique to display these retrieved links. Section 3.1 describes our traceability recovery technique. Section 3.2 presents our traceability visualization technique. Other functions provided in the DCTracVis are described in Section 3.3

3.1 Traceability link recovery

In order to improve the accuracy of retrieved traceability links to a reasonably high level at all levels of threshold, 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. In our previous work (Chen and Grundy, 2011), we focused on generating traceability links by combining the Vector Space Model (VSM) with RE, KP and Clustering techniques. We showed that this combination approach provided better performance than VSM alone. In this paper, we broaden our previous work to investigate whether IR models can recover traceability links with high precision and recall at any level of threshold by combining a range of different IR models with these three enhancement techniques.

Our basic retrieval technique uses different IR models to recover links between class entities and sections. Table 1 shows the six IR models used in this paper: VSM (Chen and Grundy, 2011), TF_IDF, PL2, BM25, DLH, and IFB2. (For more details on the other five IR models, we refer the reader to work on the Terrier tool (Terrier IR platform, 2010).)

Table 1 A Description of IR Retrieval Models

IR models	Description
VSM	Vector space model by constructing vector representations for documents
TF_IDF	The *tf***idf* weighting function, where *tf* is given by Robertson's *tf* and *idf* is given by the standard Sparck Jones' *idf*.
PL2	Poisson estimation for randomness, Laplace succession for first normalization, and Normalization 2 for term frequency normalization.
BM25	The BM25 probabilistic model ranks documents based on query terms appearing in each document, regardless of the interrelationship between query terms within a document.
DLH	The DLH hyper-geometric Divergence From Randomness (DFR) model, parameter-free weighting model.
IFB2	Inverse Term Frequency model for randomness, the ratio of two Bernoulli's processes for first normalization, and Normalization 2 for term frequency normalization.

As many papers have extensively discussed these IR models (Antoniol et al., 2002; 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 IR queries are built. IR queries, to find text relevant to a class name, include class names and their constituent words if a class name is formed by compound words. A class name (or identifier) composed of two or more words is split into separate words. An IR query string is established by using the OR operator to combine the name and the separate words. For example, PrinterName is split into the words printer and name, then the query string is "PrinterName OR printer name OR printer OR name". The query is case-insensitive. IR extracts from a collection a subset of sections that are deemed relevant to a given query and assigns a similarity score ($0 \leq$ similarity score ≤ 1) to each retrieved section based on frequency and distribution of key words in the query. This can result in some accurate links having a very low similarity score (Antoniol et al., 2002; Cleland-Huang et al., 2005; Hayes et al., 2003; Lucia et al., 2007; Marcus and Maletic; 2003; Wang et al., 2009). The lower the threshold that is used, the more possible links are retrieved but the more fault links are captured as well. In other words, at a high threshold, IR captures few links with few positive links.

In order for us to augment the number of retrieved links at high threshold levels, the RE technique is used to find all of the occurrences of class names in documents. It uses two regular expressions (using the class "Control" for the example): (.*)(^a-zA-Z0-9\-)<C-?o-?n-?t-?r-?o-?l>(^a-zA-Z0-9\-)(.*) for matching class names in documents; (.*)(^a-zA-Z0-9\-)<each part of package name>(^a-zA-Z0-9\-)(.*) for matching each part of package names. The two REs are automatically built to tailor the different data sets. 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 add these to IR queries to augment our IR model link recovery. We utilize the Clustering technique to reduce fault links by using the inherent hierarchical structure in documents. We modify the K-mean clustering algorithm (MacQueen, 1967) to discard links that are not assigned in clusters. For a detailed discussion of the three enhancement techniques, please refer to Chen and Grundy (2011).

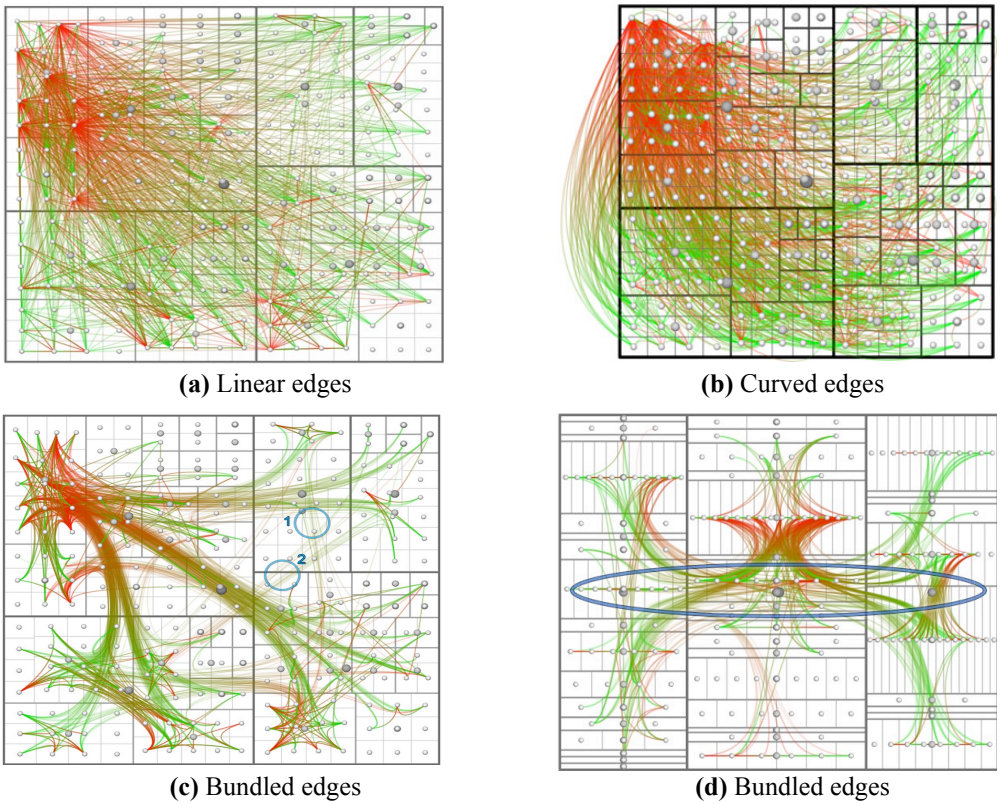
Our approach is intended to overcome the limitations of IR techniques by taking advantage of the strengths of RE, KP, and Clustering. Combining RE with IR models allows extraction of more correct links at high thresholds. As long as class names are retrieved correctly and refined regular expressions 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 thresholds 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 thresholds and reduces the number of incorrect links at any threshold.

3.2 Traceability link visualization

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 detailed 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. The previous work presented in Chen et al. (2012) focused on visualizing links using Treemaps and Hierarchical trees. In this paper, we extend our previous prototype to include three more functions: Navigator, Search and Filter. Navigator and Search functions are provided to assist users to find a specific node. For the filter function, four methods are used to filter out traceability links: IR model, combined traceability recovery approach, the similarity score level, and the number of links level. The following sections describe the visualization techniques and how we support editing of links.

3.2.1 Treemap view

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 and Kennedy, 2010; Holten, 2006; Shneiderman, 1992). 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.



(a) Linear edges **(b)** Curved edges

(c) Bundled edges **(d)** Bundled edges

Figure 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 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 1). Figure 1a shows straight and linear edges between related nodes on top of the treemap. Figure 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 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 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 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.

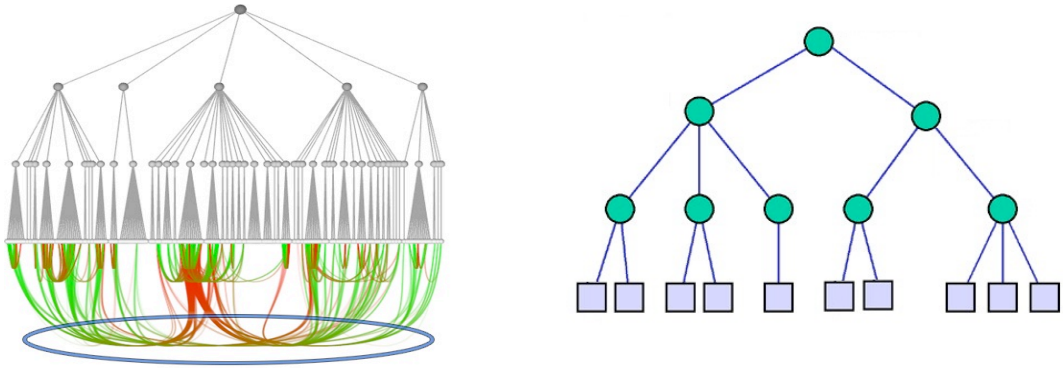
In order to ameliorate these issues, we introduce colours to show the relationship status of each node, instead of drawing edges over the treemap. The relationship status of each node describes whether the node has links and how many links it has. We use three colour ranges to show the status of each node (see Table 2 and its application in Figure 4). 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 the colours of each node without the distraction of the 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 at the expense of understanding the connectivities.

Table 2 Three Colour Ranges Indicating the Number of Links Each Node has

1. 0 ≤ No. of links < 6:	Yellow-based	→	
2. 6 ≤ No. of links < 16:	Gray-based	→	
3. No. of links ≥ 16:	Green-based	→	

3.2.2 Hierarchical tree views

The hierarchical tree 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 (Jackson & Wilkerson, 2016; 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 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 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 2b). In other words, the original leaf nodes (green circle nodes in Figure 2b) in the hierarchical tree become inner nodes and parents of traceability links (gray rectangle nodes in Figure 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 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 connectivity in the treemap is difficult to perceive, we supplement it with a left-to-right hierarchical tree (the whole HT) that can be expanded and contracted to display the whole system under trace (see Figure 5 top). We use this approach to display traceability links as children of artifacts in the system. We also employ the three colour ranges in Table 2 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.

A second left-to-right hierarchical tree layout ("the detail HT") shows detailed information of a single item once the item is selected in the treemap or the whole HT (see bottom of Figures 4 or 5). This further approach is adopted to display traceability links for the selected item. It illustrates two levels of dependency information. The first level contains artifacts that are related to the selected item. The second level contains 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 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.

3.2.3 Editing Traceability Links

We allow end users of DCTracVis to delete incorrect links and to add correct links when required, recognizing that the heuristics used to construct the links can be usefully supplemented with other information. 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. DCTracVis provides a popup menu allowing users to delete or change existing

traceability links, add a new traceability link, and change the similarity scores ($0 \leqslant$ similarity score $\leqslant 1$) of existing links.

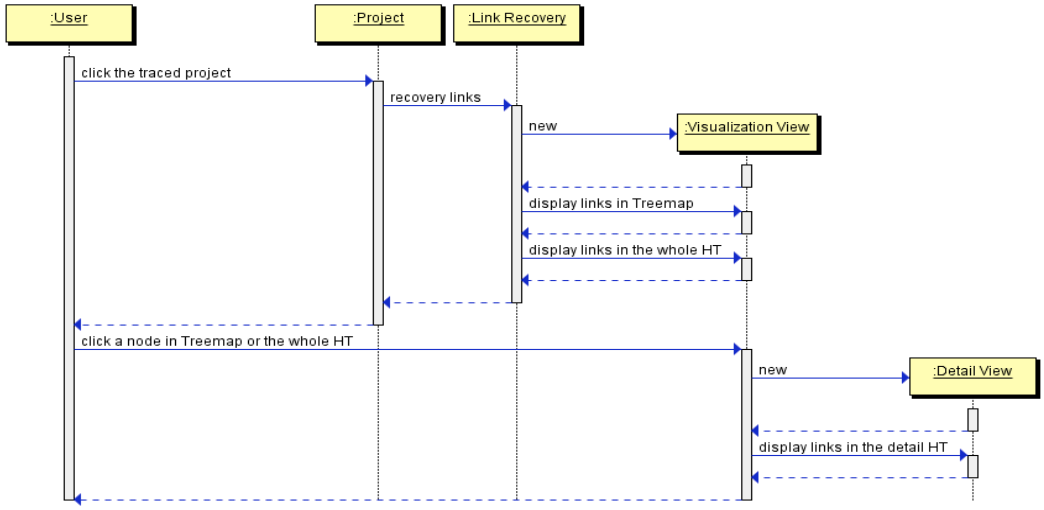


Figure 3 the Sequence Diagram for Visualizing Links in a Project

A changed link or a newly added link is assigned the highest similarity score (=1). The three views dynamically update: 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.

3.3 Other Functionality

Additional functionality in DCTracVis 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, users choose a threshold or cut point level to filter out uninteresting links. Only 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.

4 Usage Example

Figure 4 shows an example of the user interface of our DCTracVis prototype. Before tracing relationships between artifacts in a system and visualizing retrieved links, the DCTracVis plug-in must be installed in Eclipse, and then artifacts in the system must 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 to start recovering links and then visualizing them. This screen dump (Figure 4) shows an example of retrieving and visualizing traceability links between 249 classes and 182 documentation sections in the JDK1.5 (Chen et al., 2013). 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. 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.

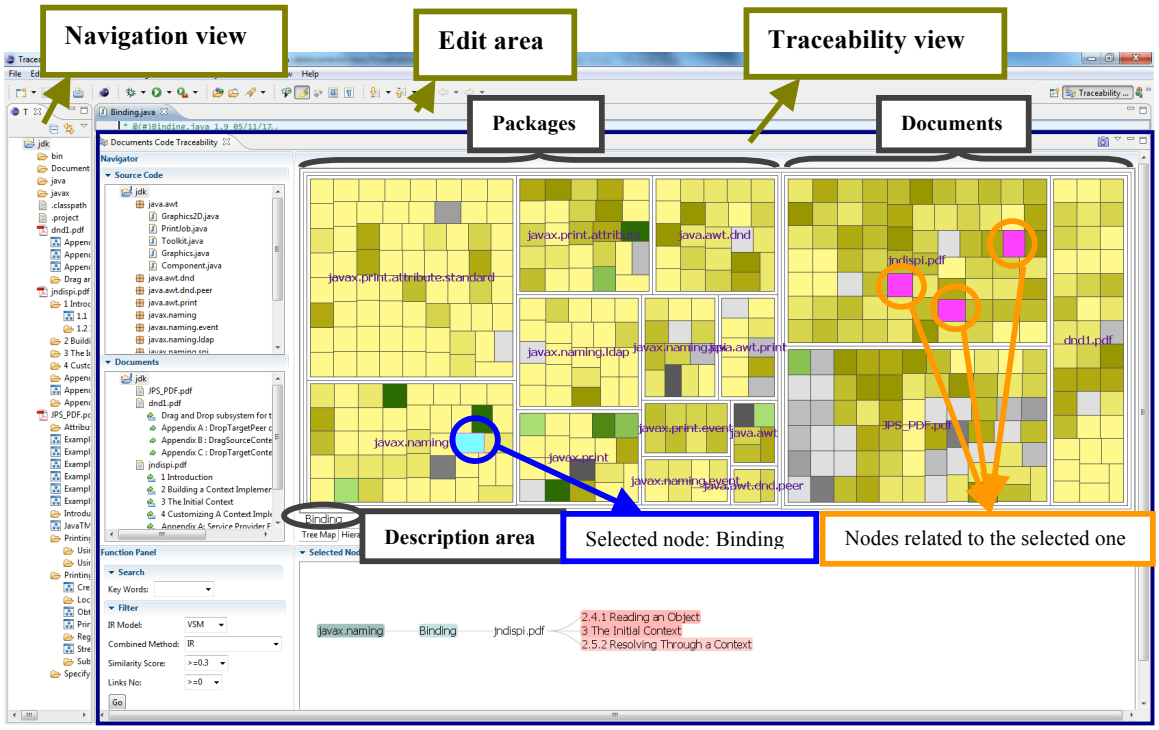


Figure 4 The User Interface of DCTracVis Using the Treemap

The traceability view includes four parts (see Figure 5). The top left area is the Navigator that lists classes and documents in the traced project in indented form to help users to find a specific item by browsing through the lists. Once an item is selected in the Navigator, its related artifacts are highlighted in the treemap and hierarchical trees. The bottom left area is the Function Panel that includes Search and Filter functions. The search function allows users to search for a specific item by key words. In Filter, users can decide whether to use the IR model alone or to combine other techniques (e.g. RE, KP and Clustering) with the IR model to capture links. Users can also select a similarity score level and a number of links level to filter out unwanted links. The top right area is the treemap view (see Figure 4) or the whole HT view (see Figure 5). 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 4, 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 5) is highlighted. Its links are shown as children of this node.

The treemap in Figure 4 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 3.2) according to the number of traceability links they have. 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 4, 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).

The whole Hierarchical Tree (HT) shows the whole project under trace and links in it (See Figure 5). 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 3.2) 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 5, 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). Each link starts with the document's name followed by the section's name.

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). The detail HT can be expanded to show link information of nodes related to the selected node (see Figure 7). Figure 7 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, the similarity value of "2.5.2 Resolving Though a Context" is 0.4. In Figure 7, the similarity score of "InitialContext" at the second level is 0.8.

Once a node is clicked in the treemap, a user can edit its links in both views. In the treemap, existing related nodes can be deleted and new nodes can be added. To prevent unwanted structural changes, names of nodes related to the selected node cannot be edited in the treemap. However, they are editable in the whole and detail hierarchical trees (see the popup menu in Figure 6); the name of an existing related node can be changed to become a new node, and their similarity scores can be changed. In all three views, we provide the contents of nodes to assist comprehension. When "Show Content" in the popup menus 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. Moreover, both views are interactive; changes made in one view are reflected in the other view. For example, if an existing related node is deleted in the treemap, it is deleted in the hierarchical tree too.

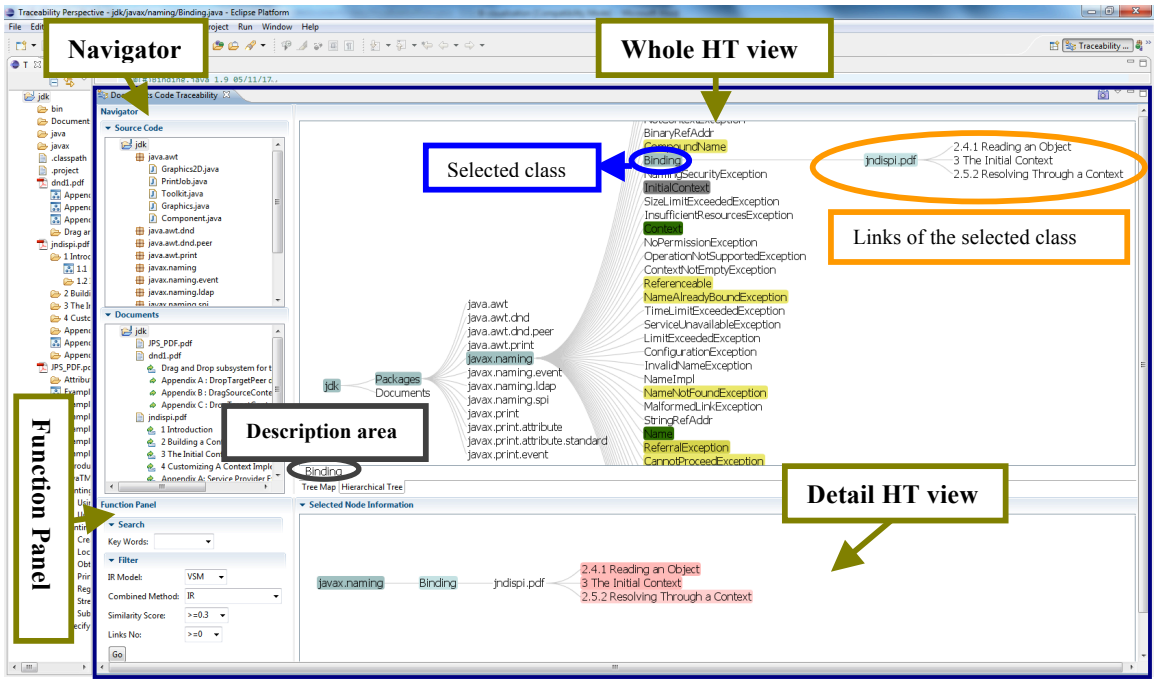


Figure 5 The User Interface of DCTracVis Using the Whole HT

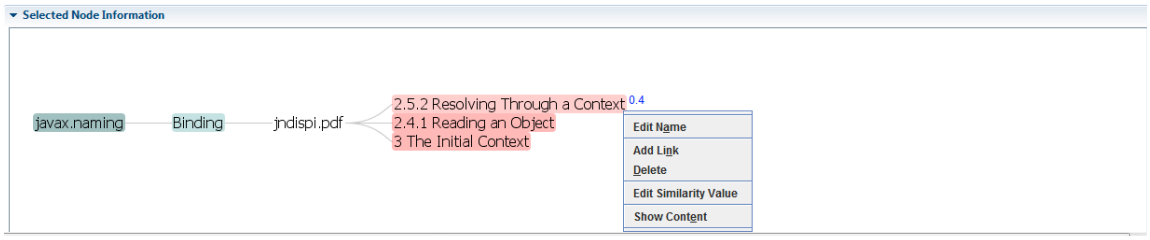


Figure 6 The Detail HT Contracted

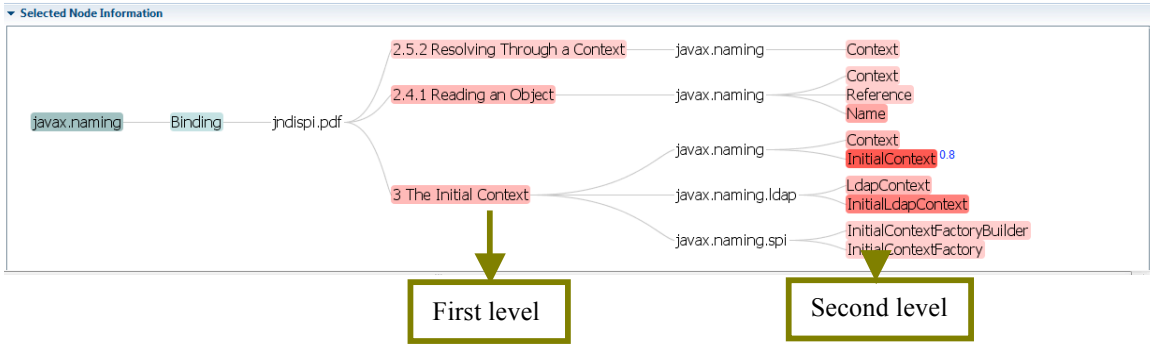


Figure 7 The Detail HT Expanded

5 Implementation

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

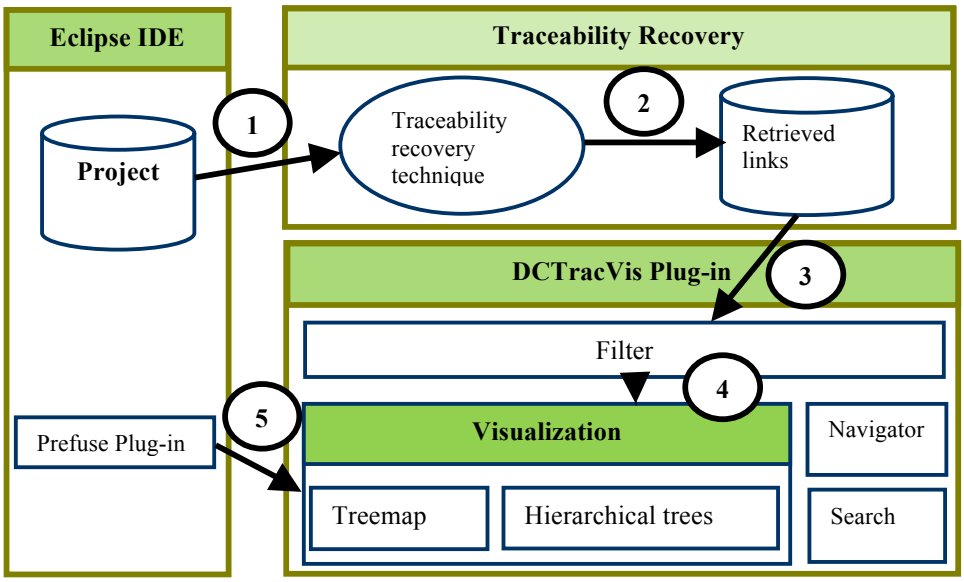


Figure 8 Architecture of DCTracVis

Figure 8 illustrates the architecture of our traceability 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 (1). This engine retrieves traceability links between classes and sections using a composite set of traceability recovery techniques (2).

These retrieved traceability links are then input into our traceability visualization system. They are filtered based on: (a) a threshold level, where only links with a similarity score larger than the threshold are shown to users, and (b) the number of links level, where only nodes having greater than or equal to the specified number of links 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 Evaluation

We have conducted two evaluations to assess DCTracVis's usability and effectiveness in comprehending, recovering, browsing, and maintaining traceability links in a traced system. Section 6.1 describes the first evaluation using four case studies and six different IR models to validate the effectiveness of our traceability recovery technique. Section 6.2

discusses the second evaluation, usability evaluation, to learn whether our traceability system supports the comprehension, browsing, and maintenance of traceability links in a system.

6.1 Evaluation for our recovery approach

6.1.1 Test cases

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: JDK1.5, ArgoUML, Freenet, and JMeter. We adopted rigorous manual identification and verification strategies (Chen and Grundy, 2011; Chen et al., 2013) to build a JDK1.5 oracle traceability link set that contains 760 correct links between classes and sections in documents. Table 3 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 divided these PDF files into sections based on their headings. The traceability benchmarks of ArgoUML, Freenet, and JMeter were kindly provided by Alberto Bacchelli (2010) comprising the three systems, their email archives, and their oracle traceability link sets that include correct links between classes and emails (Chen and Grundy, 2011). These emails were extracted from active development mailing lists of each project. Table 4 provides details of the four case studies. Here, "sections" is used for JDK1.5 and "emails" for other three cases.

Table 3 JDK1.5 Packages and Documents

JDK 1.5		#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

Table 4 Classes, Lines of Code, Sections/Emails, Size for Documents and Total True Links Per System

System	Classes	Lines of Code	Sections/ Emails	Size for Sections/Emails (MB)	Total true/correct links
JDK1.5	249	59,219	182	0.97	760
ArgoUML	423	417,811	378	1.8	308
Freenet	517	177,742	372	1.9	516
JMeter	372	172,131	348	1.7	563

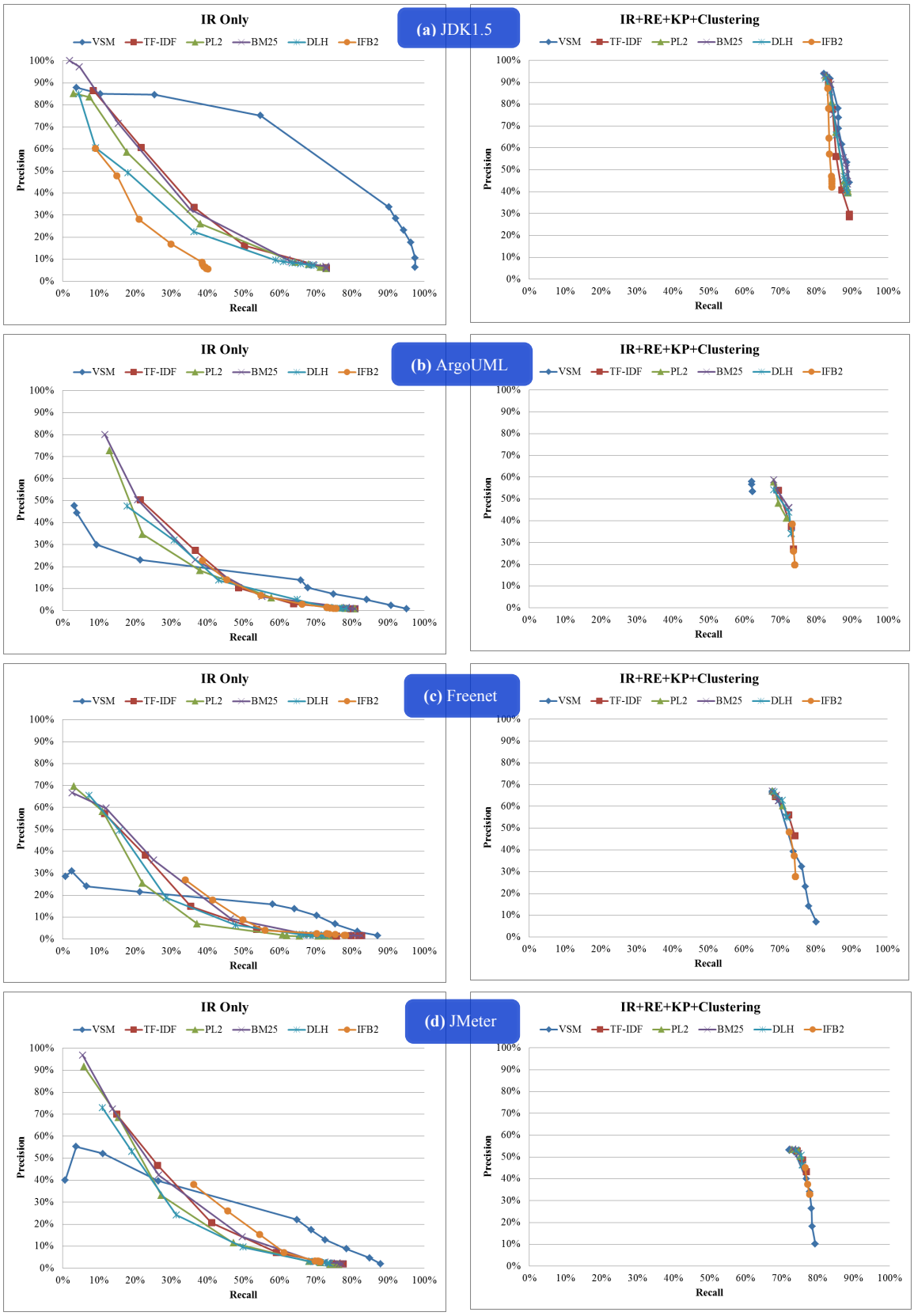


Figure 9 Comparison Results between IR Only and IR+RE+KP+Clustering

6.1.2 Evaluation results

We evaluated the performance of our traceability recovery technique employing two IR engines, Apache Lucene and the Terrier IR platform. Apache Lucence uses the 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. We applied two standard metrics – precision and recall - to measure the quality of IR models. Precision measures number of correct links over total links retrieved. Recall measures number of correct link received over total correct links.

Figure 9 illustrates the comparative results among IR models and among the combination approach (IR+RE+KP+Clustering) using different IR models in each case. We employ the same threshold scale in Chen and Grundy (2011) to illustrate precision and recall results. It contains 10 thresholds: 0, 0.02, 0.04, 0.06, 0.08, 0.1, 0.3, 0.5, 0.7, and 0.9. A threshold < 0.3 is defined to be a low threshold in the following discussion; otherwise, it is a high threshold. The factor of using this threshold scale is to illustrate changes of precision and recall results in detail between 0 and 0.1 cut points.

Figure 9a shows that when using IR only to extract links in JDK1.5, VSM has much better performance than the other five IR models except that BM25 achieves higher precision than VSM at the 0.7 and 0.9 thresholds. 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 thresholds. Our combination approach using the six different IR models can achieve a very high recall (>82%) at all thresholds.

In Figure 9b, all six IR models have similar results when applying IR only to capture links in ArgoUML; low precision at low thresholds and low recall at high thresholds. However, VSM has much lower recall than the other five IR models at high thresholds. Precision and recall are changed dramatically after adding RE, KP, and Clustering to the six IR models. The precision values at all thresholds 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 thresholds), it has lower recall (around 62% at all thresholds).

For Freenet, the six IR models have similar results when retrieving links (see Figure 9c). However, VSM has lower recall than the other five IR models at high thresholds. 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 thresholds.

When using the six IR models to extract links in JMeter, their performances are very similar; low precision at low thresholds and low recall at high thresholds (see Figure 9d). 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 thresholds.

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

Table 5 reports the differences between the precision and recall values and the differences of the number of incorrect links (false positives) achieved with using VSM

alone versus using our approach (VSM+RE+KP+Clustering) at different levels of threshold. The results showed in Table 5 highlight that precision and recall percentages are improved at positive thresholds while recall has a slight decrease at very low thresholds. Our approach removes around 92-99% of false positives at the threshold of 0, e.g. 34604 (99%) incorrect links are reduced in ArgoUML.

Table 5 Improvement of precision, recall and reduction of number of false positives (FP) at different levels of threshold when using VSM alone versus using VSM+RE+KP+Clustering.

Improvement	Threshold 0			Threshold 0.5			Threshold 0.9		
	Precision %	*Recall %*	*FP*	*Precision %*	*Recall %*	*FP*	*Precision %*	*Recall %*	*FP*
JDK1.5	+37.79	-8.29	-9960	+8.55	+57.5	+11	+5.95	+78.29	+37
ArgoUML	+52.49	-32.79	-34604	+26.78	+52.59	+78	+10.26	+58.76	+128
Freenet	+5.24	-6.98	-22385	+42.12	+61.43	+72	+38.41	+67.24	+163
JMeter	+8.06	-8.52	-20111	+0.99	+61.1	+302	+13.14	+71.4	+352

We measure average precision to see whether our recovery approach has better performance than LSI. The results in Table 6 demonstrate that the precision value improves on average when compared to LSI results. It is noticeable that VSM outperforms LSI. These results are consistent with earlier findings (Abadi et al., 2008; Aswani and Srinivas, 2009). One possible reason for this is that there are high degrees of term overlap between queries formed by class names and relevant documents in the four cases, JDK1.5, ArgoUML, Freenet and JMeter. VSM handles well the relevant documents with high number of terms matching query terms, leaving little room for LSI to improve. However, LSI can improve relevant documents for a query with little or no term overlap. (Aswani and Srinivas, 2009)

Table 6 Average precision results for IR-alone and VSM+RE+KP+Clustering.

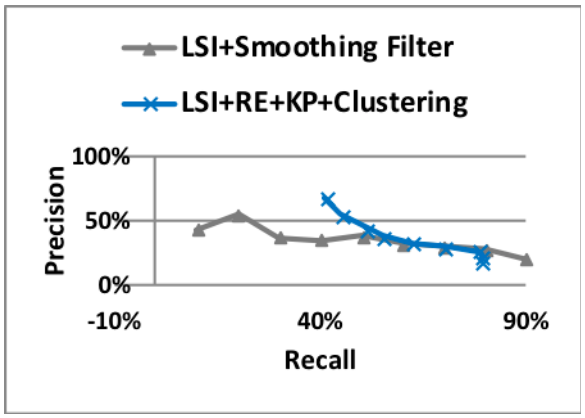
Average Precision (%)	JDK1.5	ArgoUML	Freenet	JMeter
LSI	14.87	0.31	0.25	4.93
VSM	69.21	20.22	16.59	30.21
VSM+RE+KP+Clustering	91.07	58.15	60.18	50.80

6.1.3 Comparison results

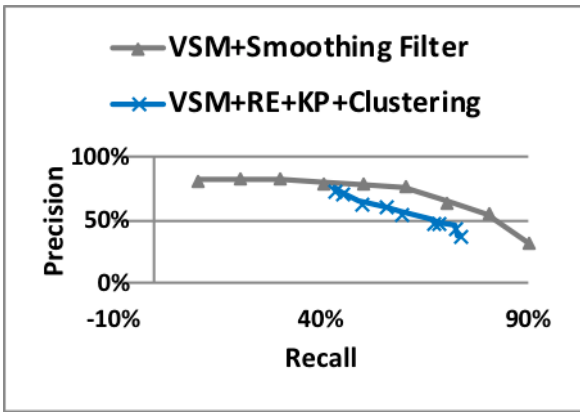
To find out whether our combination approach outperforms other approaches proposed in the literature, we compared our results to those produced in Bacchelli et al. (2010), Lucia et al. (2013), and Nishikawa et al. (2015). As ArgoUML, Freenet and JMeter have been used by Bacchelli et al. (2010), we compare our recovery approach with their light-weight methods, which are based on regular expressions. Table 7 shows that our approach provides better results than light-weight methods overall. Light-weight with an entity name that matches class entity names in emails tends to provide higher recall, light-weight with mixed approach that is matching the class file and package names in emails has higher precision, whereas our approach tends to find a better balance between precision and recall in all cases.

Table 7 Comparison results between light-weight methods in Bacchelli et al. (2010) and our combination approach at threshold 0.9.

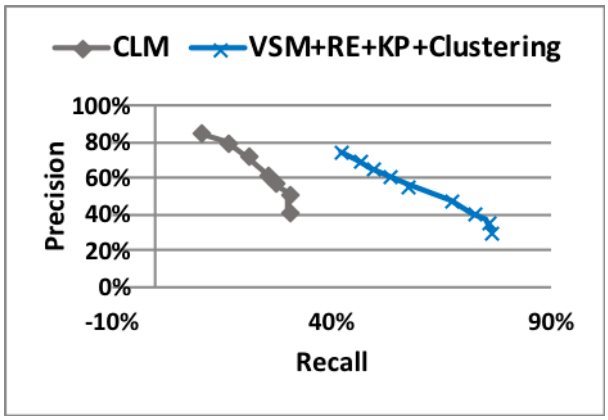
Case	Recovery Approach	Precision (%)	Recall (%)	F-measure (%)
ArgoUML	Light-weight – Entity name	27	68	38
	Light-weight – Mixed approach	64	61	63
	VSM+RE+KP+Clustering	58	62	60
Freenet	Light-weight – Entity name	17	70	27
	Light-weight – Mixed approach	59	59	59
	VSM+RE+KP+Clustering	67	68	68
JMeter	Light-weight – Entity name	15	73	25
	Light-weight – Mixed approach	59	65	62
	VSM+RE+KP+Clustering	53	72	61



(a) EasyClinic: CC-TC (Smoothing Filter vs Our Approach)



(b) EasyClinic: CC-UC (Smoothing Filter vs Our Approach)



(c) EasyClinic: CLM (CC-TC(ID)) vs Our Approach (CC-TC)

Figure 10 Comparison between our Approach, the Smoothing Filter in Lucia et al. (2013), and CLM in Nishikawa et al. (2015).

To compare between our approach, the smoothing filter used in Lucia et al. (2013), and CLM used in Nishikawa et al. (2015), we use EasyClinic as the test case. EasyClinic is a software system to manage a medical doctor's office, containing four artifacts: use cases (UC), test cases (TC), code classes (CC), and UML interaction diagrams (ID). Figure 10a compares precision/recall results between our approach and a smoothing filter

by using LSI to retrieve links between CC and TC in EasyClinic. Thisshows that our approach provides better precision and recall than the smoothing filter. Figure 10b is for using VSM to recover links between CC and UC; our approach produces better recall though the smoothing filter has higher precision. Figure 10c is the comparison between our approach and CLM. The result indicates that our approach outperforms CLM, producing higher recall. In summary, our approach can provide better recall than the smoothing filter and CLM.

6.2 Usability evaluation

We undertook a usability study of DCTracVis to answer the question: whether our approach of combining treemap and hierarchical tree views help to support the comprehension, browsing, and maintenance of traceability links in a system. The case used in this study is the JDK1.5 (see Tables 3 and 4), which contains 760 true links between 249 classes and 182 sections/emails.

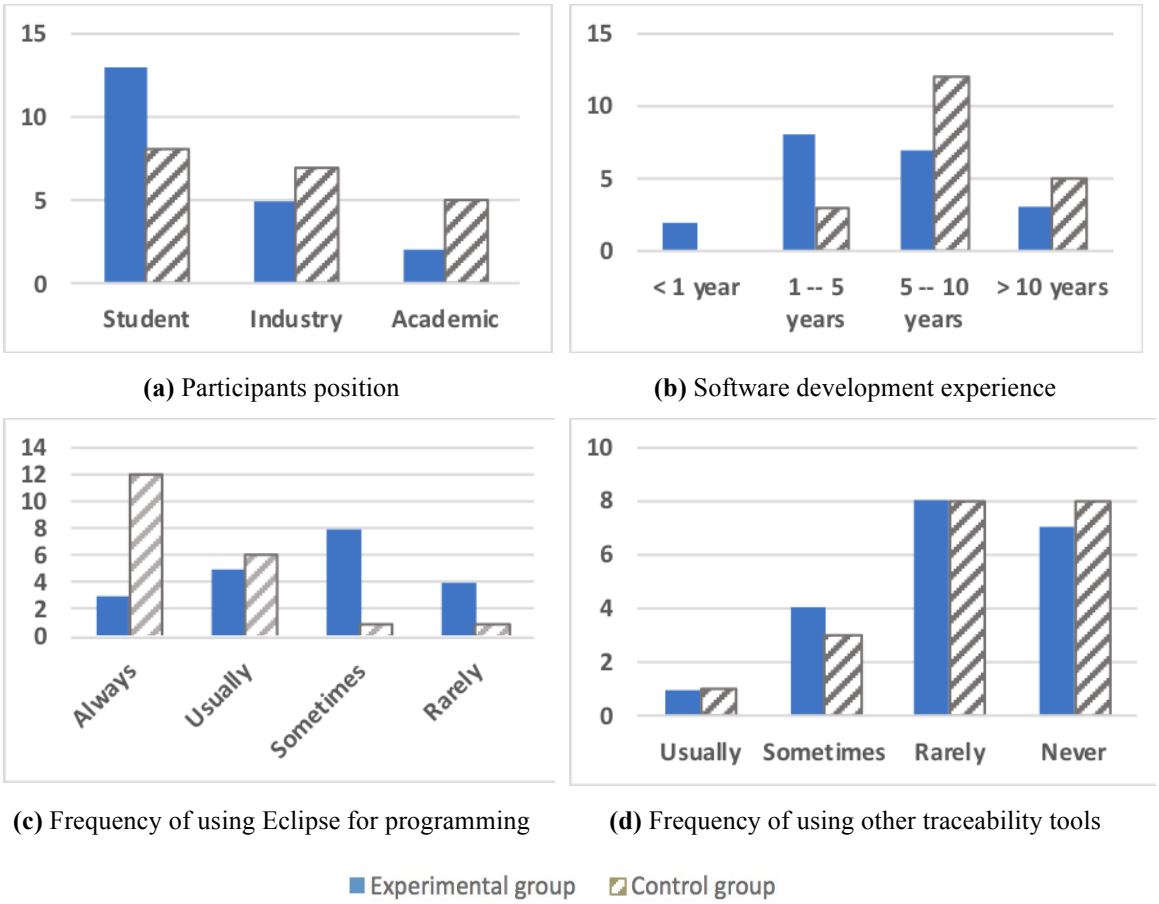
6.2.1 Study design

To answer the question, this study contained two parts. Part 1 was using Eclipse and the PDF Reader tool to complete three tasks. Part 2 was to employ our tool (DCTracVis) to complete the same three tasks. The three tasks were: 1) to understand the JDK1.5 system; the structure of the system and the overview of links between artifacts in the system. 2) 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. 3) to modify traceability links of an artifact; links of a class retrieved by IR recovery techniques may contain incorrect links or may miss correct links or may have low similarity scores for correct links; these retrieved traceability links of the class need to be edited to contain only correct links by deleting incorrect links or adding missing links. During each part, a brief introduction and a demonstration were provided to help participants to gain familiarity with Eclipse, the PDF reader and our tool at the beginning. The participants then performed the three tasks. In Part 2, after completion of the study tasks, our participants also needed to answer questions that were designed to reflect user perceptions of our tool.

6.2.2 Study participants

We recruited 40 participants for the evaluation of DCTracVis. We randomly divided them into two groups: a control group for Part 1 and an experimental group for Part 2. In the experimental group, the participants were 13 university students and 2 academics and 5 from industry (see Figure 11a). The Control group has more academics (5) and industry (7) but less students (8). Figure 11b shows the software development experience each participant had. Among 20 participants in the experimental group, 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. But the majority of participants in the control group have more than 5 years of development experience. All participants in the experimental group had at least some experience with the use of Eclipse for programming Java systems (see Figure 11c). Among them, 3 always used

Eclipse for software development, 5 usually used Eclipse, 8 sometimes used it, and 4 rarely used it. For the control group, most of them were familiar with Eclipse. In the experimental group, only one participant usually applied traceability tools to assist in comprehending or maintaining or programming software systems (See Figure 11d). This was also the case in the control group.



(a) Participants position

(b) Software development experience

(c) Frequency of using Eclipse for programming

(d) Frequency of using other traceability tools

Figure 11 The Background Information of Participants

6.2.3 Study results

Table 8 summarizes what features were used by participants to complete each task. The first interesting result was revealed when performing the first task (understanding the JDK1.5 system). In the experimental group, 13 of the 20 participants completed this in less than 1 minute, 4 spent 2 minutes, 1 spent 3 minutes, 1 spent 4 minutes, and the longest took 9 minutes to complete this task. Based on our observations, the participant who took 9 minutes spent most of their time gaining familiar with our tool as they 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 8). In the control group, the participants spent around 7 minutes on average to complete the first task. Each

participant used navigator, search and code reading and documents to understand the JDK1.5 system.

Table 8 Features usage statistics in experimental and control groups

Function	Group	Task1	Task2	Task3
Navigator	Experimental	9	12	12
	Control	20	20	20
Search	Experimental	0	6	8
	Control	20	20	20
Filter	Experimental	20	0	0
	Control	/	/	/
Class contents	Experimental	0	20	10
	Control	20	20	20
PDF contents	Experimental	0	20	12
	Control	20	20	20
Treemap	Experimental	6	1	4
	Control	/	/	/
Whole hierarchical tree	Experimental	10	1	3
	Control	/	/	/
Detail hierarchical tree	Experimental	0	20	18
	Control	/	/	/

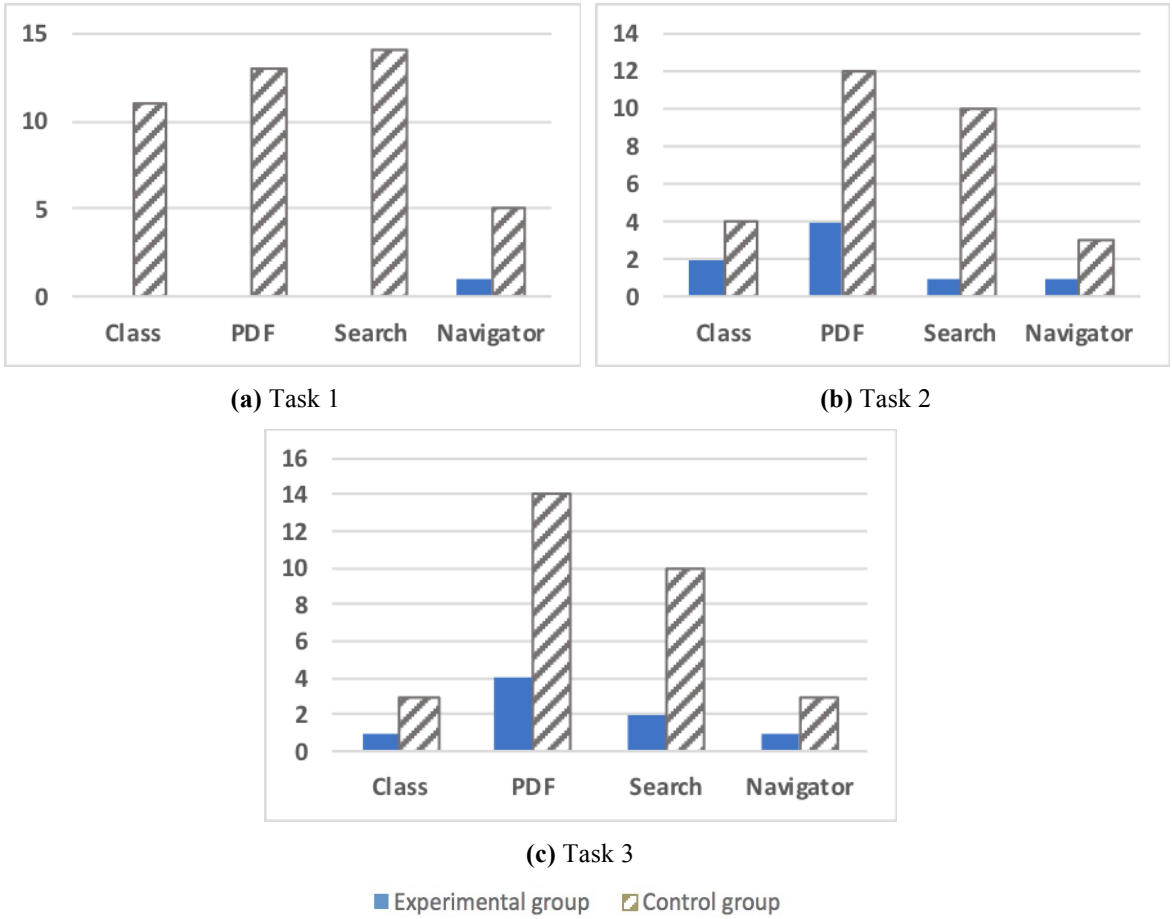
All participants of the experimental group completed the second task (bug fixing) with times varying from 1 minute to 5 minutes, whereas, in the control group, participants took 4 to 13 minutes to complete. We noticed that participants in the control group took great effort in detecting links by taking notes and frequently switching between Eclipse and PDF reader to understand code and PDF contents. On the contrary, all participants in the experimental group completed this task quickly and easily. 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 8). All participants used the show class and PDF 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 (link modification) was completed within 2 and 6 minutes for the experimental group, while the control group took 5 to 15 minutes. In the experimental group, 12 participants located a node using the navigator function, and 8 used the search function (see Table 8). 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 could quickly modify links. However, one participant looked frustrated and stressed while doing the link modification. They complained that the edit menu was not easy to use. This tough situation happened to all participants in the control group because they had to go through all PDF files to locate the correct links.

In the experimental group, 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 noticed 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. Our observation also indicated that three participants felt confused after applying the filter. They suggested it would be better to provide a summary of how many links were recovered and how many links filtered out after using the filter. In the control group, participants had to put in more effort to accomplish the three tasks. They used navigator in Eclipse to find classes, used the search function to find links in PDF documents, and had to read code and documents to edit links, which is a time-consuming and inefficient process. They commented that for "tedious tasks, tool support would be so so good."

6.2.4 Information load results

Information load refers to the amount of information available for judgement; participants can view the information in one interface or in more than one interface (Kardes et al., 2004). We compared the information load of the two groups from two factors: information density and context switching.



(a) Task 1

(b) Task 2

(c) Task 3

Figure 12 Average Participant Actions for Various Features in Experimental and Control Group

Information density examines the compactness of an interface in terms of the amount of information conveyed (Woodruff and Stonebraker, 1997). We observed that participants using our tool took less actions in performing the three tasks compared to participants using Eclipse and PDF reader. Figure 12 shows the average number of actions participants took during the completion of the three tasks. The results indicate that participants took less actions hence less effort with our tool. Our tool can provide higher volumes of information effectively in a single view.

Context switching involves storing the old state and retrieving the new state, switching from one interface to another (Li et al., 2007). To complete the three tasks, all participants in the control group frequently switched between Eclipse and PDF reader to read code and documents. Participants in the experimental group just stayed in our tool to view links and the content of code and documents. We also noticed that all participants in the control group had to take notes to keep a mental track of all links they found. By comparison, participants in the experimental group could browse links in the treemap or the hierarchical tree easily. Moreover, participants in the control group spent more time and cognitive efforts in finding correct links. However participants in the experimental group could find and edit links quickly and easily. Most participants in the control group commented that it would be good to have an automatic traceability tool to help in easy link maintenance.

6.2.5 Questions results

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 that they have used. We start with the analysis of the evaluation in the functionality of DCTracVis. The results can be seen in Figure 13. 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.

No participant gave 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. Results in Chen et al. (2013) concerning manually building the oracle link set further reinforce the claim that our tool can effectively and easily extract links. Our tool only takes up to 3.5 minutes to recover links between 249 classes and 182 sections in JDK1.5 when running on an iMac with a 2.4GHz Intel Core Duo processor and 3GB of RAM, and combining different IR models. Our tool achieves high recall, between 82% and 90%, at all thresholds (See Figure 9). However, when we manually built the JDK1.5 benchmark, every participant spent one hour to identify the related sections for 50 classes on average during the link recovery. Furthermore, it took a longer time to do the link verification than the link recovery on average. (Chen et al. 2013)

For link maintenance, 18 (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.

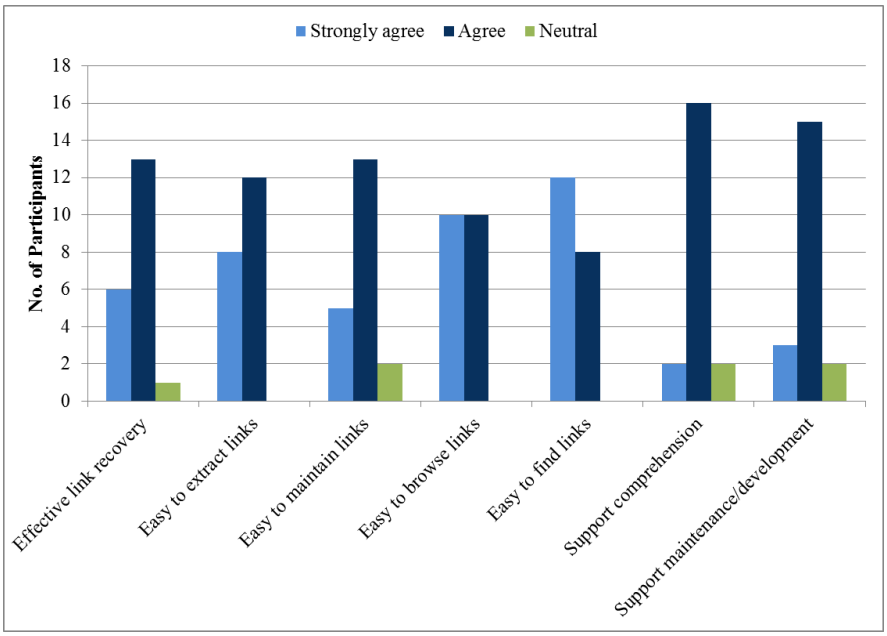


Figure 13 Results of the Evaluation in the Functionality of DCTracVis

Next, we analyzed the evaluation of the overall performance of DCTracVis. Figure 14 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) concerning our tool's overall performance. The y-axis, as previously, shows the number of participants and their agreement level.

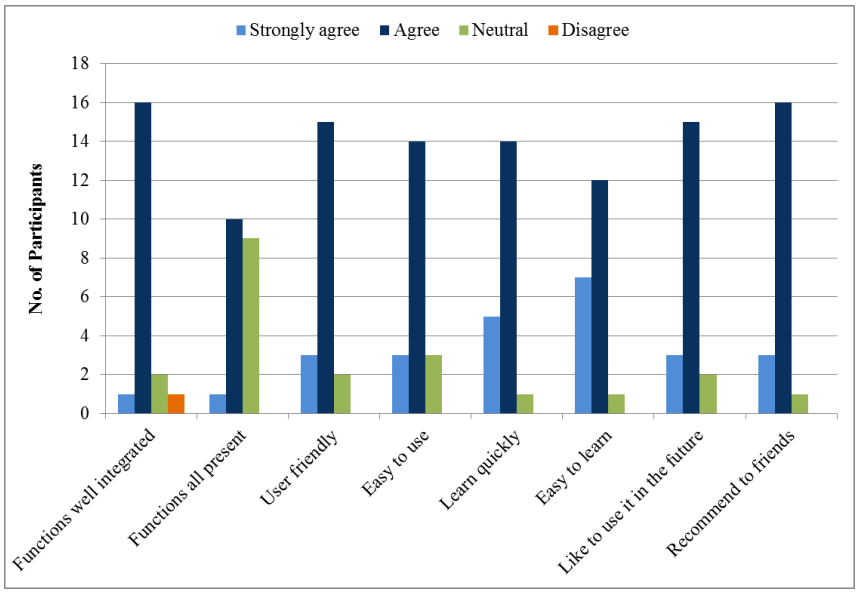


Figure 14 Results of the Evaluation in the Overall Performance of DCTracVis

One participant disagreed that the various functions in our tool were well integrated and easy to find. They found it was a little difficult 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 the other seven questions. They commented that they were unable to conduct the comparison because they had no experience of using other traceability tools.

The participants also reported many valuable comments about our tool via the questionnaire's open answer questions. These comments include the following:

- **Browsing and Maintenance:** In terms of colors, 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 also commented that colours used to identify the number of links in the treemap might confuse users, and it was hard to remember them. One participant suggested that the treemap should be divided into two sections using different colours. Moreover, two participants commented that it was not easy to quickly notice the selected node and its related nodes. In terms of other representations, five participants commented that it would have a better visual effect if the zooming was improved, and the tool made use of a large screen or large view windows, or allowed view windows to be editable. 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 needed to be more user friendly or fancier.
- **Comprehension:** Two participants suggested that words related to a selected node should be highlighted when showing the contents of the related node. Two participants thought that it would be helpful to provide summary documents about the meta-data of the traced system, such as how many packages, documents, classes, retrieved links, related links of each node, and so on.

Finally, 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 15 shows the evaluation results of a comparison made between the treemap view and the whole hierarchical tree view. 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 selected treemap as the best visualization, while the others (8) preferred the hierarchical tree.

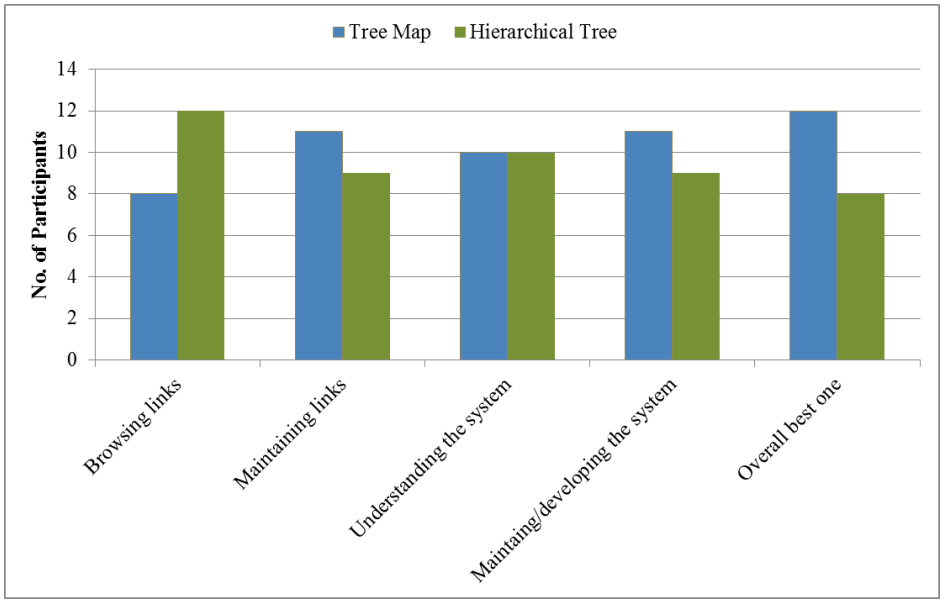


Figure 15 Results of Comparison among the Two Visualization Views

The main reasons that the participants thought the treemap was the best visualization view included the following.

- **Browsing and Maintenance:** 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 also reasoned that the treemap was easy to move around and easy to map the relationships between documents and classes. 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.
- **Comprehension:** 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.

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

- **Browsing and Maintenance:** 1 participant mentioned that the treemap was a quite good-looking, rich-user-experience but it was not easy to perform tasks such as navigating to a class, or editing or deleting a link.
- **Comprehension:** 1 participant commented that the treemap was compact but class 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.

The reasons that participants preferred the whole hierarchical tree included:

- **Browsing and Maintenance:** 4 participants commented that the whole hierarchical tree was easier to use and easy to browse links.

- **Comprehension:** 2 participants thought 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. 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 of relationships between classes.

7 Discussion

According to the evaluation results, the three enhancement techniques demonstrate the capability of improving the performance of IR models. The most obvious observations from the all of the case studies are: precision at low thresholds is increased, recall is significantly improved at high thresholds, and the results of applying different IR models become less differentiated. Our approach produces a much better result than an IR model alone but it takes a longer time to process. We ran our approach on an iMac with a 2.4 GHz Intel Core Duo processor and 3GB of RAM. Our approach took up to 5 minutes to execute on each case with different IR models. Around 80% of the time is spent on Key Phrases extraction. 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 approach is much faster than manually capturing links: when building the oracle link set for JDK1.5, participants spent one hour to identify related sections of 50 classes on average (Chen and Grundy, 2011; Chen et al., 2013).

Our usability evaluation of DCTracVis 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 each of 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.

- **Browsing and Maintenance:** 4 participants had trouble 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. This could be addressed by enlarging selected node and their related nodes to make them stand out from other nodes in the treemap. 2 participants also felt that contents in the content window were hard to read. Highlighting words that are related to the selected node could address this. Moreover, 6 participants encountered difficulty 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. In addition, 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.

- **Comprehension:** 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, allow users to edit the regular expressions used to match words in documents; (2) Allow users to add new key phrases or edit/delete existing key phrases to refine extracted key phrases to recover more related links; (3) Include a history navigation, which allows users to learn the history of their movements and activities and to undo or redo previous activities. (4) Allow modifications made to traceability links to be saved. (5) Examine other key phrase extraction techniques to improve execution time. (6) Other enhancement techniques to improve precision and recall of IR models. (7) Use other IR models to evaluate and validate our recovery approach to improve the performance of IR models.

There are four main 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. (3) Some correct links are discarded after adding clustering because correct links that are not included in clusters are cut out (Chen and Grundy, 2011).

8 Threats to Validity

There are some threats that could affect the validity of the evaluation of our traceability link recovery approach and the usability study of our tool, DCTracVis.

For the evaluation of our link recovery technique, there are two threats. First, we relied on human judgment to build the oracle link set and thus this set might not 100% correct. To alleviate this, in the case of the JDK1.5 oracle, we applied a very rigorous manual verification strategy to analyze every true link, which were verified by at least 2 analysts

(Chen et al., 2013). The other three benchmarks, ArgoUML, Freenet and JMeter, have been used by other authors (Bacchelli et al., 2010). 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 size of the systems, the types of documents, the structures of documents, and the availability of comments in source code. However, we cannot confirm that our results are similar in closed-source systems.

For the usability study of DCTracVis, we used only one system, JDK1.5. Hence, we cannot claim that these results would work for all systems. To mitigate this threat, we made sure we used an open source, widely used system. The second threat is our participants were limited to a small subset of students and developers who volunteered, and therefore might be biased towards assessing DCTracVis. However, we made sure to recruit participants who had different experience and familiarity in traceability and software development. We also recruited participants from industry and academia, since practices in the system development and maintenance may vary from those used by students. The final threat to validity is caused by using participants that were unfamiliar with the JDK1.5 and the concept of traceability. However, in a real-world scenario, system maintenance is often performed by developers who are not as familiar with the system as the original developer. To resolve this, at the beginning of the study we gave a detailed description about the JDK1.5 and traceability, provided a demo and demonstrated our tool, and gave participants sufficient time to be familiar with JDK1.5, traceability and our tool.

9 Conclusions

It is a major challenge for automated IR traceability recovery techniques to extract links between artifacts of a system at high-levels of precision and recall no matter which threshold is chosen. The shortcoming of commonly used IR models is their low precision with high recall at low thresholds or high precision with low recall at high thresholds. We have developed a new recovery approach by incorporating three enhancement techniques, RE, KP, and Clustering, with IR to extract links between documents and class entities. Our approach ameliorates the limitations of IR by taking advantage of the strenghs of these three enhancement techniques. Our experimental results based on four case studies and applying six different IR models provides a demonstration that the limitations of IR can be effectively mitigated by combining these three enhancement techniques. Our approach provides reasonble precision and high recall at all thresholds.

After capturing traceability links, it is a large challenge to visualize them effectively and efficiently because of scalability and visual clutter issues. We present an approach that integrates enclosure and node-link visualization representations to support the overall overview of traceability in the system and the detailed overview of each link while still being highly scalable and interactive. 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 colors 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 dynamically updated, changes made in one view are reflected in the other two views, and vice versa.

We adopted this composite recovery approach and composite visualization approach to build a novel link traceability system, DCTracVis. This provides support including navigation, search, and filter functions to assist users in locating a specific node or filtering out some uninteresting links. Our usability study shows that our traceability system performed well and was both helpful and useful. Our system was able to extract traceability links in a system easily and effectively. Our system 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, it supported the comprehension of links and provided the hierarchical structure of the system and the overall overview of links.

Acknowledgement

The authors gratefully acknowledge Alberto Bacchelli for agreeing to share his exemplar test sets and oracles of ArgoUML, Freenet and JMeter.

References

Abadi, A., Nisenson, M., and Simionovici, Y.: A traceability technique for specifications. 16th IEEE International Conference on Program Comprehension, pp. 103-112 (2008)

ADAMS 2009. Overview. Data accessed: February 2009, http://adams.dmi.unisa.it/adams-2009/Overview.html

Antoniol, G., Casazza, G., and Cimitile, A.: Traceability recovery by modelling programmer behavior. 7th WCRE, Queensland, Australia, Nov., pp. 240-247 (2000)

Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D., and Merlo, E.: Recovering traceability links between code and documentations. TSE, Vol. 28, No. 10, Oct., pp. 970-983 (2002)

Asuncion, H. U., Francois, F., and Taylor, R. N.: An end-to-end industrial software traceability tool. ESEC-FSE, Cavtat near Dubrovnik, Croatia, pp. 115-124 (2007)

Aswani Kumar, Ch. and Srinivas, S.: On the performance of Latent Semantic Indexing-based Information Retrieval. Journal of Computing and Information Technology – CIT 17, 2009, 3, pp. 259-264 (2009)

Bacchelli, A., Lanza, M., and Robbes, R.: Linking E-mails and source code artifacts. ICSE'10, May, pp.375-384 (2010)

Cleland-Huang, J. and Habrat, R.: Visual support in automated tracing. REV 2007, IEEE Computer Society, pp. 4-8 (2007)

Cleland-Huang, J., Settimi, R., Duan, C., and Zou, X.: Utilizing supporting evidence to improve dynamic requirements traceability. RE'05, Paris, pp.135-144 (2005)

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

Chen, X. and Grundy, J.: Improving automated documentation to code traceability by combining retrieval techniques. ASE 2011, pp. 223-232 (2011)

Chen, X., Hosking J., and Grundy, J.: Visualizing Traceability Links between Source Code and Documentation. VL/HCC 2012, Innsbruck (2012)

Chen, X., Hosking, J., Grundy, J., and Amor, R.: Development of robust traceability benchmarks. 22nd ASWEC, June 2013, Melbourne, Australia (2013)

Cornelissen, B., Holten, D., Zaidman, A., Moonen, L., van Wijk, J.J., and van Deursen, A.: Understanding execution traces using massive sequence and circular bundle views. 15th ICPC, Alberta, BC, pp. 49-58 (2007)

Falessi, D., Di Penta, M., Canfora, G., and Cantone, G.: Estimating the number of remaining links in traceability recovery. Empirical Software Engineering (EMSE), June 2017, Volume 22, Issue 3, pp. 996-1027 (2017)

Gotel, O.C. and Finkelstein, A. C. W.: An analysis of the requirements traceability problem. 1st RE, pp. 94-101 (1994)

Graham, M. and Kennedy, J.: A survey of multiple tree visualization. Information Visualization, Vol. 9(4), pp. 235-252 (2010)

Grechanik, M., McKinley, K. S., and Perry, D. E.: Recovering and using use-case-diagram-to-source-code traceability links. ESEC/FSE'07, Croatia, pp. 95-104 (2007)

Hayes, J. H., Dekhtyar, A., and Osborne, J.: Improving requirements tracing via information retrieval. Proc. Int'l Conf. Requirements Eng. (RE), pp. 151-161 (2003)

Holten, D.: Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. IEEE Transactions on Visualization and Computer Graphics, Vol. 12, No. 5, pp. 741-748 (2006)

Jackson, M. and Wilkerson, M.: MBSE-driven visualization of requirements allocation and traceability. Aerospace Conference, MT, USA, pp. 1-17 (2016)

Kamalabalan, K., Uruththirakodeeswaran, T., and Balasubramaniam, D.: Tool support for traceability of software artefacts. Moratuwa Engineering Research Conference (MERCon), Moratuwa, Sri Lanka, pp. 318-323 (2015)

Kardes, F. R., Cronley, M. L., Kellaris, J. J., and Posavac, S. S.: The role of selective informtiaon processing in price-quality inference. Journal of Consumer Research, 31(2), pp. 368-374 (2004)

Kuang, H., Nie, J., Hu, H., Rempel, P., Lu, J., Egyed, A., and Mader, P.: Analyzing closeness of code dependencies for improving IR-based traceability recovery. IEEE 24th Int. Conf. on Software Analysis, Evolution and Reengineering (SANER), Feb. 2017, Klagenfurt, Austria, pp. 68-78 (2017)

LDRA, LDRA Product Brochure v7.2. 2012. From http://www.ldra.com

Li, C., Ding, C., and Shen, K.: Quantifying the cost of context switch. Proceedings of the 2007 workshop on experimental computer science (ExpCS'07), San Diego, California, June (2007)

Li, Y. and Maalej, W.: Which traceability visualization is suitable in this context? A comparative study. REFSQ 2012, LNCS 7195, pp. 194-210, (2012)

Lucia, A. D., Fasano, F., Francese, R., and Tortora, G.: ADAMS: an artifact-based process support system. 16th Int. Conf. on Software Engineering and Knowledge Engineering, Alberta, Canada, pp. 31-36 (2004)

Lucia, A. D., Fasano, F., Oliveto, R., and Tortora, G.: Recovering traceability links in software artifact management systems using information retrieval methods. TOSEM, Vol. 16, No. 4, Article 13 (2007)

Lucia, A. D., Penta, M. D., Oliveto, R., Panichella, A., Panichella, S.: Applying a smoothing filter to improve IR-based traceability recovery processes: an empirical investigation. Information and Software Technology, 55(2013), pp. 741-754 (2013)

MacQueen, J. B.: Some methods for classification and analysis of multivariate oberservations. 5th Berkeley Symp. On Math. Stat. and Prob., pp. 281-297 (1967)

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

Marcus, A., Xie, X., and Poshyvanyk, D.: When and how to visualize traceability links? TEFSE 2005, Nov. 8, California, USA, pp. 56-61 (2005)

Merten, T., Juppner, D., and Delater, A.: Improved representation of traceability links in requirements engineering knowledge using Sunburst and Netmap visualization. 4th MARK, Trento, pp. 17-21 (2011)

Nakagawa, H., Matsui, S. and Tsuchiya, T.: A visualization of specification coverage based on document similarity. IEEE/ACM 39th Int. Conf. on Software Engineering Companion (ICSE-C), pp. 136-138 (2017)

Nishikawa, K., Washizaki, H., Oshima, K., and Mibe, R.: Recovering transitive traceability links among software artifacts. ICSME 2015, Bremen, Germany, pp. 576-580 (2015)

Penta, M. D., Gradara, S., and Antoniol, G.: Traceability recovery in RAD software systems. IWPC 2002, pp.207-216 (2002)

Pilgrim, J. von, Vanhooff, B., Schulz-Gerlach, I., and Bervers, Y.: Constructing and visualizing transformation chains. 4th ECMDA-FA '08, Heidelberg, pp. 17-32 (2008)

Prefuse, the prefuse information visualization toolkit, 2011, http://prefuse.org/

Rocco, J.D., Ruscio, D.D., Iovino, L., and Pierantonio, A.: Traceability visualization in metamodel change impact detection. GMLD'13, Montpellier, France, pp. 51-62 (2013)

Rodrigues, A., Lencastre, M., Filho, G. A. de A. C.: Multi-VisioTrace: traceability visualization tool. 10th Inter. Conf. on the Quality of Information and Communication Technology, pp. 61-66 (2016)

Roman, G. C. and Cox, K. C.: Program visualization: the art of mapping programs to picture. Proc. Of Int. Con. on Software Engineering, pp. 412-420 (1992)

Settimi, R., Cleland-Huang, J., Ben Khadra, O., Mody, J., Lukasik, W., and DePalma, C.: Supporting software evolution through dynamically retrieving traces to UML artifacts. 7th IWPSE, Kyoto, Japan, pp. 49-54 (2004)

Shneiderman, B.: Tree visualization with tree-maps: 2d space-filling approach. ACM Transactions on Graphics (TOG), 11(1), pp. 92-99 (1992)

Singhal, A.: Modern Information Retrieval: a brief overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol. 24, No. 4, pp. 35-42 (2001)

Spanoudakis, G. and Zisman, A.: Software traceability: a roadmap. Advances in Software Engineering and Knowledge Engineering, Vol. 3: Recent Advances, (ed) S.K Chang, World Scientific Publishing (2005)

Terrier IR platform, 2010, extracted from http://terrier.org/

Van Amstel, M. F., van den Brand, M.G.J. and Serebrenik, A.: Traceability visualization in model transformations with TraceVis*. ICMT 2012, LNCS 7307, pp. 152-159 (2012)

van Ravensteijn, W.J.P.: Visual traceability across dynamic ordered hierarchies. Master's thesis, Eindhoven University of Technology (August 2011)

Voytek, J. B. and Nunez, J. L.: Visualizing non-functional traces in student projects in information systems and service design. CHI 2011, Vancouver, Canada (2011)

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

Watkins, R. and Neal, M.: Why and how of requirements tracing. 5th ASM, California, pp.104-106 (1994)

Witten, I. H., Paynter, G. W., Frank, E., Gutwin, C., and Nevill-Manning, C. G.: Kea: practical automatic keyphrase extraction. 4th ACM DL, Berkeley, pp. 254-255 (1999)

Woodruff, A. and Stonebraker, M.: VIDA: visual information density adjuster. Tech. Report CSD-97-968, Univ. of California, Berkeley, Calif., Sep. (1997)

Zhou, X., Huo, Z., Huang, Y., and Xu, J.: Facilitating software traceability understanding with ENVISION. COMPSAC 2008, pp. 295-302 (2008)