In Proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing, Innsbruk, Austria, Sep 30-Oct 3 2012. © IEEE 2012

Visualizing Traceability Links between Source Code and Documentation

Xiaofan Chen[1], John Hosking[2], John Grundy[3]
[1]Department of Computer Science, University of Auckland, Auckland, New Zealand
xche044@aucklanduni.ac.nz
[2]College of Engineering & Computer Science, Australian National University, Canberra, Australia
john.hosking@anu.edu.au
[3]Centre for Computing & Engineering Software Systems, Swinburne University of Technology, Melbourne, Australia
jgrundy@swin.edu.au

Abstract—**It is well recognized that visualizing traceability links between software artifacts helps developers to recover, browse, and maintain these inter-relationships effectively and efficiently. However, it is a major challenge for researchers to efficiently visualize traceability links for big software systems because of scalability and visual clutter issues. In this paper we present a new approach that combines treemap and hierarchical tree visualization techniques to provide a global structure of traces and a detailed overview of each trace. These both reduce visual clutter while still being highly scalable and interactive. Our usability study shows that our approach can support comprehension, browsing, and maintenance of traceability links.**

Keywords-Software traceability; Traceability visualization; Treemap; Hierarchical tree

I. Introduction

Traceability between software artifacts has been recognized as a critical success factor for effective development, management, and comprehension of a complex software system throughout the software development life cycle [3, 7, 20]. The larger and more complex a software system is, the larger number of traceability links between artifacts exist, especially as software systems become more and more complex. Many traceability recovery techniques [2, 4, 9, 16] have been developed to automatically or semi-automatically extract high quality traceability links between artifacts in a system, namely to extract as many correct links and as few incorrect links as possible. While these link extraction techniques are very powerful, a key unsolved issue remains: how do we support software engineers to effectively and efficiently understand, browse, and maintain these retrieved traceability links? It is commonly believed that software visualization techniques can help understand the complex data, interact between engineers at a high degree, and support impact analysis [3, 21]. Visualizing traceability links enables users to recover, browse, and maintain inter-relationships between artifacts in a natural and intuitive way [17]. However, it is a big challenge to visualize an overwhelmingly large number of traceability links effectively and efficiently. This is 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 [6, 12, 18]. Moreover, the efficient visualization of both the artifact structures themselves and the enormous number of inter-relationships between artifacts is far from trivial [6, 17].

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, emails and so on. The objective of our research is to provide users with an effective visualization enabling them to create, browse, edit, and maintain traceability links between artifacts effectively and efficiently. With this visualization users can trace relationships between various documents and source code, easily create and change links as well as conveniently browse and maintain links. In terms of size, we are interested in systems with potentially several hundreds to even thousands of classes, dozens if not hundreds of documents, and many tens of thousands to hundreds of thousands of traceability links between classes and document elements.

Traditionally, traceability links are stored or represented in tabular formats, e.g. a matrix. Despite their simplicity, these approaches cannot provide a global overview and fail to support users to maintain or interpret links easily and conveniently [23, 24]. Although using a graph to display links improves these shortcomings, graphs adopted by most traceability visualization systems to date [6, 17, 18, 23, 26] have suffered from visual clutter (i.e. are overcrowded) when dealing with large numbers of traceability links between artifacts. Visual clutter is caused by displaying the overwhelming number of traceability links on top of a graph structure where artifacts are represented as nodes and traceability links are edges between related nodes [12]. This then impedes the ability to efficiently browse, analyze, and maintain traceability links between artifacts. These approaches simply cannot scale to the size of systems and number of traceability links we are interested in supporting.

In this paper, we present a visualization approach that combines enclosure and node-link representations to reduce visual clutter and to allow the visualization of the global structure of traces and a detailed overview of each trace, while still being highly scalable and interactive. We adopt two visualization techniques to achieve these goals: treemap and hierarchical tree. A treemap view displays a tree structure by means of enclosure and provides an overview of inter-relationships between artifacts. In order to reduce visual clutter, we employ colors to represent the relationship status of each node in the treemap, instead of directly drawing edges between related nodes on top of the treemap. A hierarchical tree visualization that can be expanded and contracted is treated as a supplement of the treemap to illustrate the detailed information about each trace. We have conducted a usability study to assess the usefulness of our new combined visualization approach for large traceability visualization problems. The results of this evaluation show that our visualization approach is both easy to use and is able to effectively and efficiently help software developers in comprehension, browsing, and maintenance of large numbers of traceability links.

In Section 2, we review and critique key related work. Section 3 describes the design of our visualization approach, followed by a description of its implementation in Section 4. We report the results of a usability study and outline possible future research in Section 5. We draw conclusions in Section 6.

II. Related Work

Software engineers traditionally store or represent traceability links in tabular formats using a spread-sheet, matrix, cross-references, or database. More recently, research has focused on displaying links in a graph or tree due to the convenience, and ease of browsing and maintaining links. These methods are discussed in the following three sections.

A. *Traditional Approaches*

Matrix and cross-reference techniques are very common traditional methods of representing traceability links. A traceability matrix is a two-dimensional grid that displays artifacts in rows and columns and represents traceability links as marks between row artifacts and column artifacts. It is easy to understand and provides a quick overview of relations between two artifacts if the set of artifacts is small [23]. The Trace/Analyzer tool developed by Egyed [8] uses such a matrix to visualize the trace links among models, code, and test scenarios. This matrix depicts the artifacts on both axes and uses colors or symbols to indicate whether two artifacts are related or not. However, the matrix misses the inherent hierarchical structure in artifacts and becomes unreadable when the set of artifacts becomes large [24]. The cross-reference pattern lists each artifact using natural language and gives a list of related links for each artifact [23]. It is easy to understand but cannot provide the overall structure of traces. It is difficult to identify individual traceability link as they are lost in this table structure. The approach, therefore, does not scale to large numbers of classes and documents.

B. *Graph-based Traceability Link Visualization*

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 can be used to easily browse links. ADAMS [15] supports specifying links between pairs of artifacts. Traceability links are organized in a graph where nodes represent 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 [1]. Within the graph, users can identify traceability paths and 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 [5] propose 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 visualization graph provides a birds-eye-view of the candidate links and their distribution across the set of traceable artifacts, and allows the user to explore groups of candidate links that naturally occur together in the document's hierarchy [5]. Unfortunately, this visualization becomes very large as the data set gets bigger. Moreover, it uses the display space inefficiently. Zhou et al [26] adopt a hyperbolic tree view with the enhancement of a "focus+context" approach to facilitate software traceability understanding. The results of their empirical study show that this view allows users to maintain a global view of links as well as being able to dive deep into an interesting traceability path. However, this view is also not space-efficient.

TBreq [14], a commercial application, provides end-to-end traceability from requirements to design, code, and test. It lists artifacts horizontally and draws linear edges between related items of artifacts. It cannot provide the hierarchical structure and can quickly produce severe visual clutter for a system with medium to large numbers of artifacts. TraceVis, developed by van Ravensteijn [23], visualizes a dynamic list of hierarchies and adjacency relations. It uses icicle plots and hierarchical edge bundling [12] techniques to support the hierarchical structure and to reduce visual clutter. Icicle plots are used to represent hierarchies vertically. Adjacency relations are represented by drawing edges between related items. Edges are displayed using splines, and are grouped using hierarchical edge bundling. TraceVis supports an overview of as well as a detailed insight into inter-related, hierarchically organized data. However, it uses space inefficiently and can result in visual clutter if the dataset is large or lateral relations visualized [23].

Merten et al [18] utilize sunburst and netmap techniques to display traceability links between requirements knowledge elements. The sunburst is to visualize 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. Cornelissen et al [6] employ a hierarchical edge bundling technique [12] 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 the mirrored layout. The 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.

C. *Other Approaches*

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 [5] 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 [17] employs a map consisting of colored 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 [11] 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 then it displays targets linked to this source. It also fails to represent all links at the same time. A 3D approach [19] is introduced to enhance traceability visualization between UML diagrams. Artifacts are projected on layered planes. Traces between different levels of abstraction are visualized by using edges between planes. Although presenting more content at once and grouping related information together, the 3D approach adds more complexity to the graph, and still leads to visual clutter when the data set becomes large.

To varying degrees, none of traceability visualization techniques developed so far can visualize an overwhelmingly large number of traceability links effectively and efficiently. 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.

III. Our Approach

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. The overview of traces provides users with information about the distribution of traces in the system and whether or not an artifact has links. As a result users don't need to check one by one to see which artifacts have no links. 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 colors to differentiate the relationship status of each node in the treemap instead of drawing edges directly over the treemap. The later approach quickly gives visual clutter. The hierarchical tree is used to provide the detailed dependency information of a single item when the item is selected in the treemap view. Any change to links made in the treemap is reflected in the hierarchical tree, and vice versa. The following sections describe the two techniques and how we support editing of links in detail.

A. *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 [22]. This layout makes it an ideal technique for displaying a large tree and using display space effectively [22, 25]. 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.

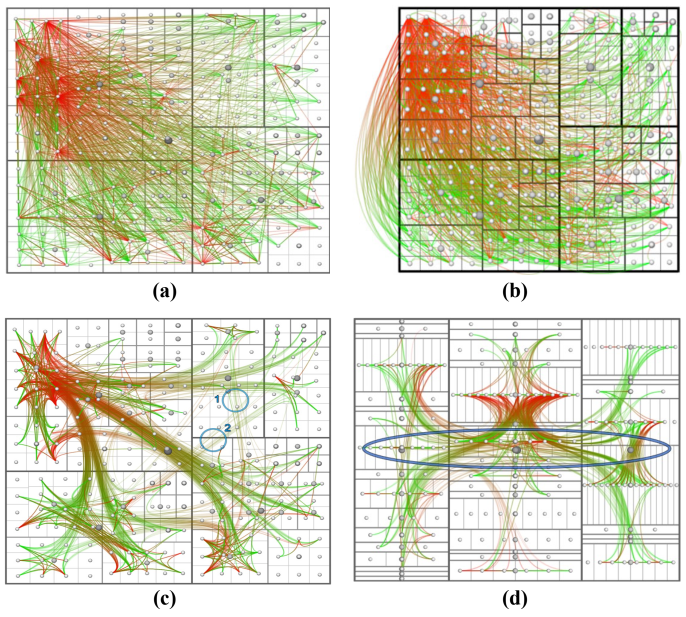


Figure 1. Displaying traceability links between nodes using (a) straight/linear edges; (b) curved link edges; (c) and (d) edges grouped by hierarchical edge bundles. [12]

In order to display traceability links between artifacts in a treemap, the straight-forward way is to add relationships between related nodes as edges over the treemap as in [12] (see Figure 1). Figure 1a shows straight/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 [12]. However, hierarchical edge bundling can cause bundles to overlap along the collinearity axes (see the encircled region in Figure 1d) if dealing with the large amount 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 encircled (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 colors 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 color ranges to show the status of each node (see Table 1). If a node has less than six links, yellow-based colors are used. If the number of links is less than 16 but more than 5, gray-based colors are used. Otherwise, we use green-based colors. For each color range, the shading of the color indicates intermediate values (lighter implies less links, darker more links). Based on colors on each node without additional edges on top of the treemap, it is easy to discern the structure of the traced system and an overall overview of the scale of traceability links.

TABLE I. THREE COLOR RANGES INDICATING THE NUMBER OF LINKS EACH NODE HAS

1. $0 \le$ No. of links < 6:	Yellow-based	🟨	→	🟨
2. $6 \le$ No. of links < 16:	Gray-based	⬜	→	⬛
3. No. of links ≥ 16:	Green-based	🟩	→	🟩

B. *Hierarchical Tree View*

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 [10, 12]. This representation is easy to understand, even to a lay-person, and it communicates hierarchical structure very well [10, 12]. There are two approaches to visualize traceability links using a hierarchical tree visualization. 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 [12]. 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.

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

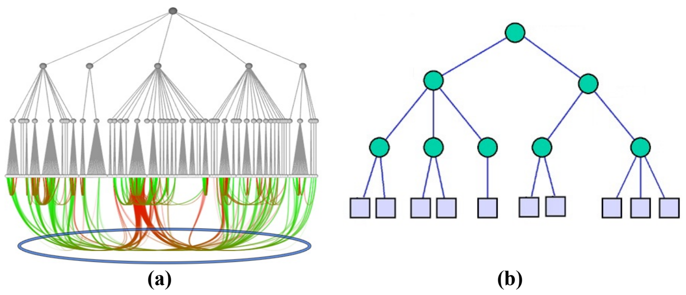


Figure 2. Showing traceability links in the hierarchical tree layout: (a) links as edges between nodes [12], (b) links as children of nodes

C. *Editing Traceability Links*

Initially we use an automated algorithm to extract a candidate set of traceability links from a target system and its documentation [4]. While our algorithm has both high precision and high recall compared to other techniques, it still suffers from displaying some incorrect trace links and misses some correct trace links. To address this, incorrect links can be deleted by an end user and correct links added when required.

When a node is selected in the treemap, its related nodes are highlighted and a hierarchical tree is built starting from the selected node and connecting to nodes related to it and all dependencies of these nodes. After this, users are able to edit links in both the treemap and hierarchical tree views. Our visualization tool provides a popup menu allowing users to delete or change existing traceability links, add a new traceability link, and change the similarity scores ($0 \le$ similarity score ≤ 1) of existing links.

A changed link or a newly added link is assigned the highest similarity score (=1). Both views are interactive; any change made in one view is reflected in the other view. 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.

IV. IMPLEMENTATION

Figure 3 illustrates the traceability visualization process used by our approach. First, if documents in the project under trace contain sections, they need to be divided into small 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 small documents are passed to our automated traceability recovery engine (discussed in detail in [4]) (1). This engine retrieves traceability links between classes in source code and sections in documents using a composite set of traceability recovery techniques (discussed in detail in [4]) (2). Then these retrieved traceability links are filtered based on a threshold level - only links with a similarity score that is bigger than the threshold 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).

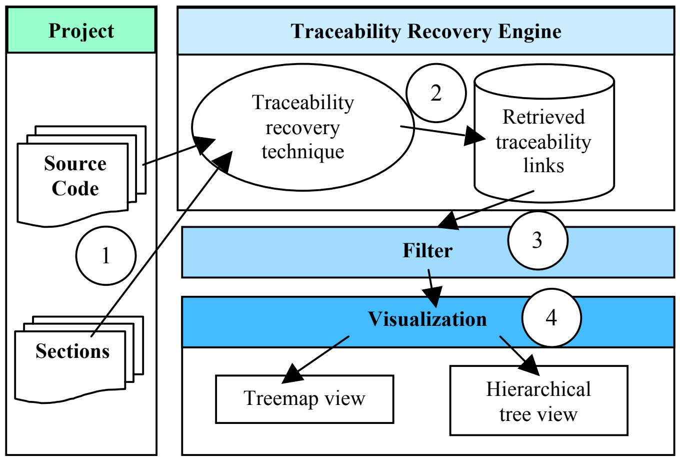

Project
Source Code
1
Sections
Traceability Recovery Engine
Traceability recovery technique
2
Retrieved traceability links
Filter
3
Visualization
4
Treemap view
Hierarchical tree view

Figure 3. Traceability visualization process of our approach

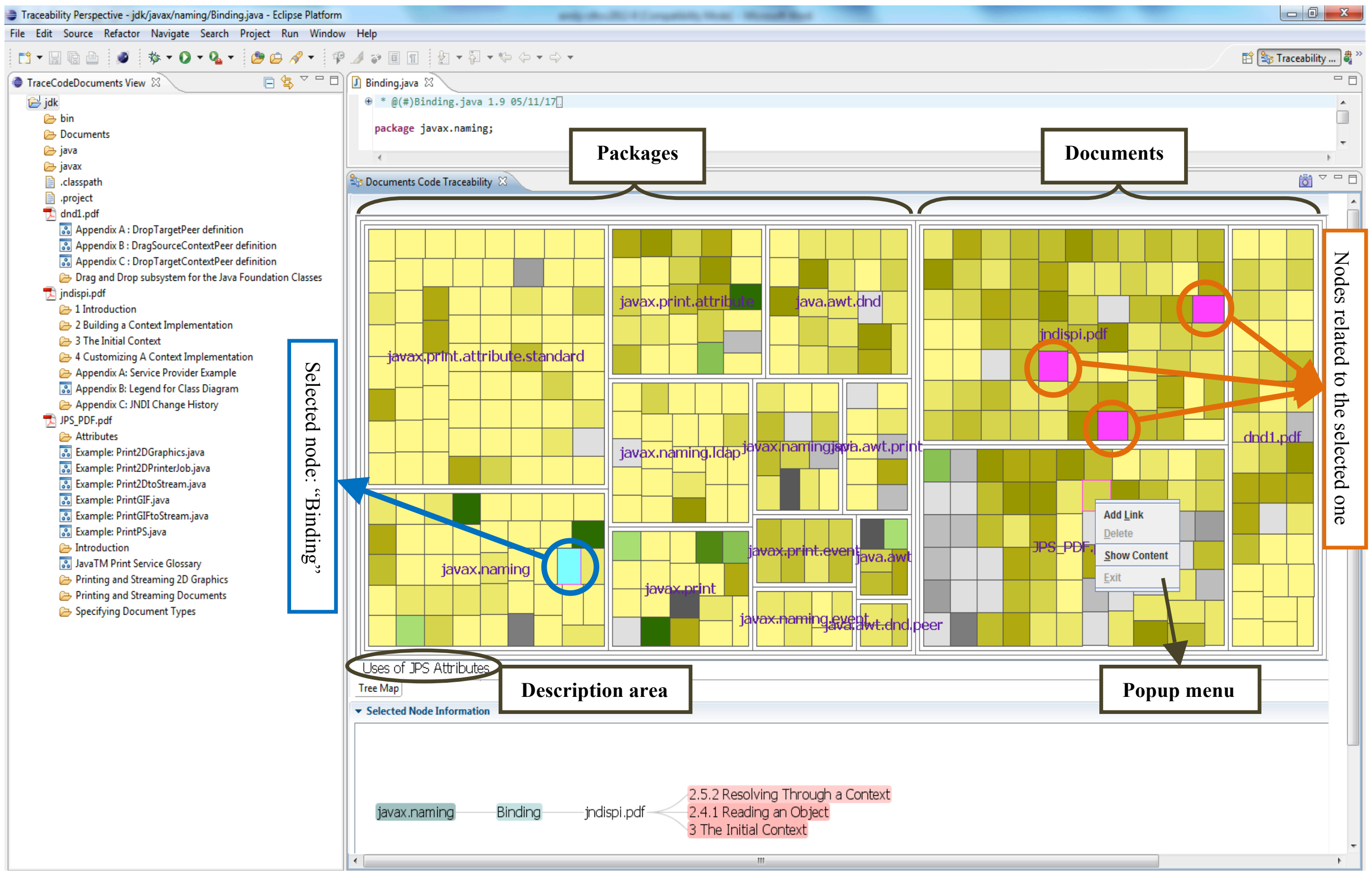
A prototype of our combination traceability visualization approach has been developed. This prototype is seamlessly embedded within the Eclipse integrated development environment (IDE). It automatically extracts relationships between sections in documents and classes in source code and visualizes these retrieved links. Figure 4a is the user interface of our visualization prototype. It shows an example of visualizing traceability links between classes and sections in the JDK1.5, which is discussed in [4]. This case contains 249 classes and 182 sections. Traceability links between them are captured using Information Retrieval (IR) recovery techniques discussed in [4]. 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. The bottom right area is the traceability view that visualizes extracted links. The traceability view includes two parts: the top area is the treemap view, and the bottom area is the hierarchical tree view that displays the detailed information of the selected node. Our visualization prototype can provide software engineers with both IDE and traceability support. Users can use the functionality provided not only within the Eclipse IDE but can also use our visualization prototype as a stand-alone tool.

The treemap in Figure 4a is divided into two parts: one for packages and the other for documents. Each node is colored using the three color ranges (discussed in Section 3) 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 a magenta color. If the node is clicked, it is highlighted with cyan and a hierarchical tree showing its detailed dependency information is built. For example, in Figure 4a, the node "Binding" with cyan color in "javax.naming" package is selected, all related nodes are colored with magenta. Detailed link information is displayed in a hierarchical tree (Figure 4b). The hierarchical tree can be expanded to show link information of nodes that are related to the selected node. Figure 4c 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 colored to differentiate their similarity value levels. The lighter the color the lower the similarity score a node has. When the node is hovered with the mouse, its similarity score is shown. In Figure 4b, the similarity value of "2.5.2 Resolving Through a Context" is 0.4. In Figure 4c, the similarity score of "InitialContext" at the second level is 0.8.

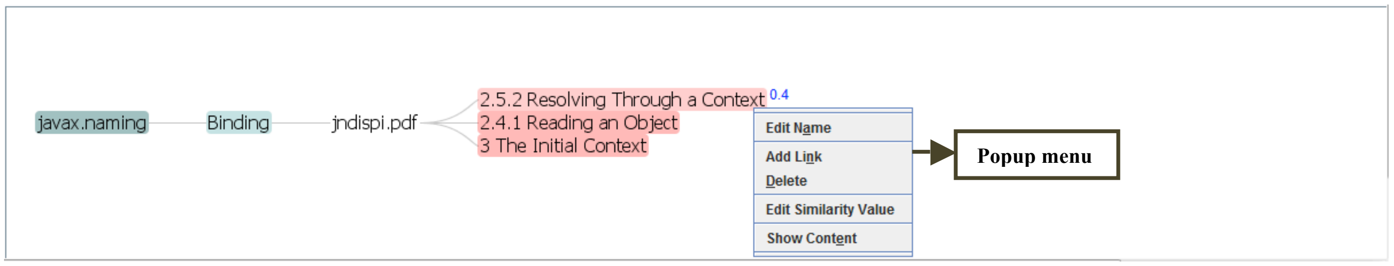
Once a node is clicked in the treemap, users can edit its links in both views. In the treemap, existing related nodes can be deleted and new nodes can be added (see the popup menu in Figure 4a). For example, a section called "Uses of JPS Attributes" (see the Description area in Figure 4a) in "JPS-PDF.pdf" can be added as a new link of the "Binding" class. To prevent unwanted structural changes, names of nodes related to the selected node cannot be edited in the treemap. However, they are editable in the hierarchical tree; the name of an existing related node can be changed to become a new node, and their similarity scores can be changed (see the popup menu in Figure 4b). In both 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.

V. Evaluation

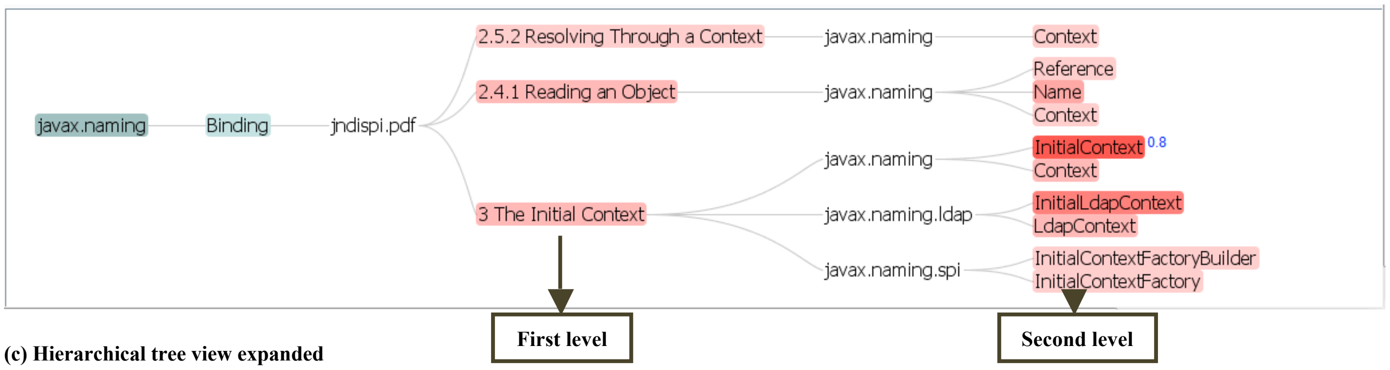
We undertook a usability study to answer the question: does our approach of combining treemap and hierarchical tree views help to support and improve the comprehension, browsing, and maintenance of traceability links in a system. The case used in this study is the JDK1.5 mentioned in Section 4. We recruited a group of 15 participants for the evaluation of our approach. Among the participants were 10 students, 1 academic, and 4 from industry (see Figure 5a). At the beginning, a brief introduction and a demonstration were provided to help participants to gain familiarity with our approach. The participants then performed three tasks. The first task was to understand the JDK1.5 system; the structure of the system and the overview of links between artifacts in the system. The second task was to understand how an artifact works; how a class works in order to fix a bug related to it, where the documentation of this class can be found, and what other classes are related to this class. The third task was to modify traceability links of an artifact; links of a class retrieved by IR recovery techniques may contain incorrect links or may miss correct links or may have low similarity score of correct links, these retrieved traceability links of the class need to be edited to contain only correct links: delete incorrect links or add missing links.


Traceability Perspective - jdk/javax/naming/Binding.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Traceability ...
TraceCodeDocuments View
jdk
bin
Documents
java
javax
.classpath
.project
dnd1.pdf
Appendix A : DropTargetPeer definition
Appendix B : DragSourceContextPeer definition
Appendix C : DropTargetContextPeer definition
Drag and Drop subsystem for the Java Foundation Classes
jndispi.pdf
1 Introduction
2 Building a Context Implementation
3 The Initial Context
4 Customizing A Context Implementation
Appendix A: Service Provider Example
Appendix B: Legend for Class Diagram
Appendix C: JNDI Change History
JPS_PDF.pdf
Attributes
Example: Print2DGraphics.java
Example: Print2DPrinterJob.java
Example: Print2DtoStream.java
Example: PrintGIF.java
Example: PrintGIFtoStream.java
Example: PrintPS.java
Introduction
JavaTM Print Service Glossary
Printing and Streaming 2D Graphics
Printing and Streaming Documents
Specifying Document Types
Selected node: "Binding"
Binding.java
* @(#)Binding.java 1.9 05/11/17
package javax.naming;
Packages
Documents
Documents Code Traceability
javax.print.attribute.standard
javax.print.attribute
java.awt.dnd
javax.naming.ldap
javax.naming
java.awt.print
javax.naming
javax.print
javax.print.event
java.awt
javax.naming.event
java.awt.dnd.peer
jndispi.pdf
dnd1.pdf
JPS_PDF
Add Link
Delete
Show Content
Exit
Nodes related to the selected one
Uses of JPS Attributes
Tree Map
Description area
Popup menu
Selected Node Information
javax.naming
Binding
jndispi.pdf
2.5.2 Resolving Through a Context
2.4.1 Reading an Object
3 The Initial Context

(a) The user interface of our visualization prototype



(b) Hierarchical tree view contracted

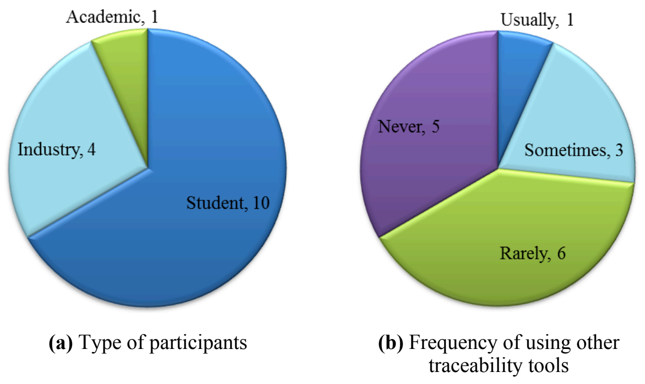


(c) Hierarchical tree view expanded

Figure 4. An example of treemap and hierarchical tree visualization of traceability links between classes and sections: (a) the user interface of our visualization prototype, (b) contracted hierarchical tree, (c) expanded hierarchical tree

After the completion of tasks, the participants answered a set of questions on our approach, as well as some general questions regarding their background. The former were aimed at finding out whether browsing, maintaining and understanding are improved as perceived by the participants. The latter were to gather information of participants' position, software development experience, and frequency of using other traceability tools. Finally, the participants were requested to provide open ended comments on our approach. During the evaluation, we observed and recorded how participants managed to complete tasks and their verbal responses and facial expression.

All of the participants had at least 1 year experience in software development. Among them, 2 had more than 10 years of development experience, 5 had less than 10 years but more than 5 years, 6 had less than 5 years, and 2 had more than 1 year but less than 5 years. Only one participant usually applied traceability tools to assist in comprehending or maintaining or programming software systems. 3 participants sometimes used traceability tools, 6 rarely used, and 5 never used such tools (See Figure 5b).



(a) Type of participants

(b) Frequency of using other traceability tools

Figure 5. Background of participants

All 15 participants completed the three tasks with times varying from 10 minutes to 15 minutes. The first interesting result was revealed when performing the first task. The majority of participants completed this in less than 1 minute and strongly agreed that the treemap view clearly illustrates the structure of the system and the overall overview of links in the system. All participants agreed that the detailed information provided in the hierarchical tree view is a good supplement to the treemap view while performing the second task. The majority of participants undertook the modification of traceability links of a class using the hierarchical tree view as they thought that this view was more intuitive and straight-forward for this task.

The main analysis of this evaluation was on the set of questions answered by participants based on their experiences of using our approach in comparison to other software tools they have used. The results can be seen in Figure 6. The diagram shows the four questions (easy to use, help comprehension, easy to browse, easy to maintain) 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 approach is easy to use, helps comprehension of traceability in the system, is easy to browse traceability links, and is easy to maintain links. No participants made a negative response to any of the four questions. All participants strongly agreed or agreed that it is easy to browse links. 14 of 15 participants (strongly) agreed that it is easy to maintain links. 13 of them agreed that our approach helps comprehension. 12 thought that it was easy to use. Several participants gave a neutral answer to questions for use, comprehension and maintenance. They responded this way because they could not undertake the comparison as they had never used other traceability tools. Overall, the results in Figure 6 clearly show that participants agreed that our approach can help them understand, browse, and maintain traceability in the system.

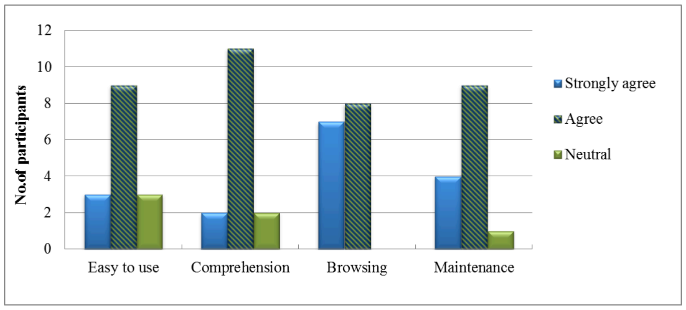


Figure 6. Results of the evaluation

Participants also reported many valuable comments on our approach. These include: (1) One participant pointed out that it is not feasible for color-blind users to discern nodes if we adopt inappropriate colors to represent the number of links that each node has in the treemap view and differentiate the similarity value level of each link in the hierarchical tree view. (2) Three participants suggested that it would be helpful to use different sizes of nodes in the treemap to represent the number of links that each node has or to reflect the sizes of classes/sections in the system. (3) Two commented that it is not easy to quickly notice the selected node and its related nodes. (4) Two suggested that words related to a selected node should be highlighted when showing the contents of the related node.

Based on our observations, we noticed that participants had difficulties in directly finding a specific node in the treemap. A key extension of our approach includes a navigator and search functions to help users quickly find an item that they are interested in. This extension also contains a filter to allow users to select different IR recovery techniques to retrieve traceability links and to filter out unwanted traceability links according to the similarity score level and the number of links. Our approach can be enhanced if the following methods are applied. (1) It would be more intuitive to employ different font sizes and/or colors of nodes in the hierarchical tree to display their similarity value levels. (2) It would be more noticeable to make the selected node and its related nodes stand out from other nodes in the treemap by enlarging these nodes. (3) In the contents window, words that are related to the selected node should be highlighted. (4) To apply or combine other methods to represent the relationship status of each node in the treemap would make the view more intuitive. These all represent potential future work in refining our approach.

There are three limitations of our approach. (1) The hierarchical structure of the system is not well communicated in the treemap. This may be ameliorated by including an additional hierarchical tree, which can be expanded and contracted, to represent the whole system and links in it. (2) The size of each node in the treemap becomes small in order to display a system with large numbers of artifacts in one screen. (3) The three color ranges used in the treemap may need to be extended to clearly distinguish nodes if the range of numbers of links that nodes have becomes large.

VI. Conclusions

It is well recognized that visualizing traceability links in a system assists in comprehension, browsing, and maintenance of traceability. However, it is a big challenge to visualize traceability links 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. The hierarchical tree view can be treated as the supplement of the treemap. When a node is selected in the treemap, the hierarchical tree 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). Their similarity scores also can be changed. Both views are interactive; changes made in one view can be reflected in the other view. Our usability study shows that our approach supports and improves comprehension, browsing, and maintenance of traceability links in a system.

Acknowledgment

The authors gratefully acknowledge the support of the Foundation for Research, Science and Technology and the University of Auckland for their financial support of this research.

References

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

[2] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentations", *TSE 28(10)*, Oct. 2002, pp. 970-983

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

[4] X. Chen and J. Grundy, "Improving automated documentation to code traceability by combining retrieval techniques", *26th ASE*, 2011, Lawrence, KS, pp. 223-232

[5] J. Cleland-Huang and R. Habrat, "Visual support in automated tracing", *REV 2007*, IEEE Computer Society, pp. 4-8

[6] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J.J. Van Wijk, and A. Van Deursen, "Understanding execution traces using massive sequence and circular bundle views", *15th ICPC '07*, Alberta, BC, pp. 49-58

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

[8] Egyed, "Trace Analyzer tool: a mini tutorial", 2006, http://www.alexander-egyed.com/tools/trace_analyzer_tool.html

[9] Egyed, S. Biffl, M. Heindl, land P. Grunbacher, "A value-based approach for understanding cost-benefit trade-offs during automated software traceability", *TEFSE 05*, 2005, California, USA, pp. 2-7

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

[11] M. Grechanik, K.S. McKinley, and D. E. Perry, "Recovery and using use-case-diagram-to-source-code traceability links", *ESEC/FSE'07*, Sep. 3-7, 2007, Croatia, pp. 95-104

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

[13] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey", *Journal of Software Maintenance and Evolution: Research and Practice*, 2003, 15, pp. 87-109.

[14] LDRA, Requirements traceability with TBreq, 2012, extracted from http://www.ldra.com/tbreq.asp

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

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

[17] Marcus, X. Xie, and D. Poshyvanyk, "When and how to visualize traceability links?", *TEFSE 2005*, Nov. 8, California, USA, pp. 56-61

[18] T. Merten, D. Juppner, and A. Delater, "Improved representation of traceability links in requirements engineering knowledge using Sunburst and Netmap visualization", *4th MARK*, 2011, Trento, pp. 17-21

[19] J. Pilgrim, B. Vanhooff, I. Schulz-Gerlach, and Y. Bervers, "Constructing and visualizing transformation chains", *4th ECMDA-FA '08*, 2008, Heidelberg, pp. 17-32

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

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

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

[23] W.J.P. Van Ravensteijn, "Visual traceability across dynamic ordered hierarchies", Master's thesis, Eindhoven University of Technology, August 2011

[24] J. B. Voytek and J. L. Nunez, "Visualizing non-functional traces in student projects in information systems and service design", *CHI 2011*, May, 2011, Vancouver, Canada

[25] J. J. van Wijk and H. van de Wetering, "Cushion treemaps: visualization of hierarchical information", *INFOVIS 99*, San Francisco, Oct. 25-26, 1999, pp. 1-6

[26] X. Zhou, Z. Huo, Y. Huang, and J. Xu, "Facilitating software traceability understanding with ENVISION", *COMPSAC 2008*, pp. 295-302