Interactive Traceability Links Visualization using Hierarchical Trace Map

Thazin Win Win Aung, Huan Huo, Yulei Sui
University of Technology Sydney
Sydney, Australia
thazinwinwin.aung@student.uts.edu.au
{huan.huo,yulei.sui}@uts.edu.au

***Abstract*—Traceability links between various software artifacts of a system aid software engineers in system comprehension, verification and change impact analysis. Establishing trace links between software artifacts manually is an error-prone and costly task. Recently, studies in automated traceability link recovery area have received broad attention in the software maintenance community aiming to overcome the challenges of manual trace links maintenance process. In these studies, the trace links results generated by an automated trace recovery approach are presented either in a bland textual matrix format (e.g., tabular format) or two-dimensional graphical formats (e.g. tree view, hierarchical leaf node). Therefore, it is challenging for software engineers to explore the inter-relationships between various artifacts at once (e.g., which test cases and source code files/methods are related to a particular requirement). In this position paper, we propose a hierarchical trace map visualization technique to explore inter-relationships between various artifacts at once naturally and intuitively.**

***Index Terms*—Traceability, Information Retrieval, Visualization**

I. Introduction

Traceability has been mentioned in many software development methodologies as part of software engineering practice. Maintaining the trace links between software artifacts is crucial to perform various software engineering activities including change impact analysis [1], requirements coverage analysis [2], requirements verification and validation [3], test case selection [4], compliance verification [5], issue assignment [6] and bug localization [7]. Over the last few decades, researchers have attempted to recover trace links automatically using Information Retrieval (IR) techniques. The idea behind such techniques is based on the assumption that most software artifacts such as requirements, use cases, interaction diagrams, source code, and test cases contain textual descriptions and meaningful source code identifiers [8]. IR techniques recover traceability links based on the similarity between the text contained in two related artifacts of different types [9], [10]. Traditionally, trace links are presented either in two or multi-dimensional matrix formats, which are commonly known as a Requirements Traceability Matrix (RTM) [11]. Likewise, tabular grid style formatting has been used in automatic trace recovery tools [12], [13] to present result sets for analyst verification. Even though the tabular format is applicable for small dataset verification; it is hard to explore inter-relationships between artifacts conveniently. Recent graphical-based approaches like hierarchical leaf node [14], tree view [15], [16], Sunburst [17], and Netmap [17] can only present the coarse-grain view of traceability between two artifacts at a time. The main drawback of these approaches is that they fail to explore relationships between multiple artifacts interactively to comprehend the overall structure of a system. Therefore, it is time-consuming and resource-intensive to explore and interpret the system. Traceability tools like Traceclipse [18] and TraceViz [19] are platform dependent and specific for JAVA development environment. In [20], researchers implement on manual trace links maintenance approach and cypher query language to retrieve the trace links between various requirement artifacts. However, tracing implementation artifacts such as source code is not included in their experiments. In this paper, we propose a hierarchical trace map visualization approach to support system comprehension and change impact analysis interactively. The main contributions of our approach are as follows:

- We propose a stand-alone interactive traceability links visualization approach which can consume data from external data storage and recover trace links automatically.
- Our hierarchical trace map visualization can assist in system comprehension and change impact analysis interactively by providing a visual trace links exploration space.

In the next section, we present how our raw hierarchical trace map visualization approach is used to present the artifacts relationships interactively. We designed our conceptual model based on the standard components of the IR-based traceability recovery process [21]. Due to space limitations, we presented our model in Github [1]. To explore our conceptual model, we developed the WebTrace [2] application and evaluated it with the EasyClinic dataset due to the availability of various software artifacts, including use cases, interaction diagram, test cases and source code class descriptions.

II. Visual Support for Traceability Links Visualization

The main goal of our Hierarchical Trace Map is to provide visual support in exploring the overall structure of the system

[1]https://github.com/thazin31086/WebTrace/blob/master/HTM.svg.
[2]http://www.webtrace.tech/Home/VisualizeTraceLinks.

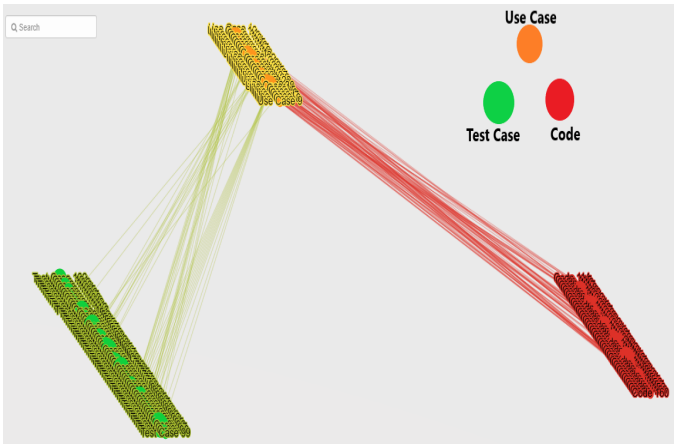

Fig. 1. Unfiltered view of Hierarchical Trace Map

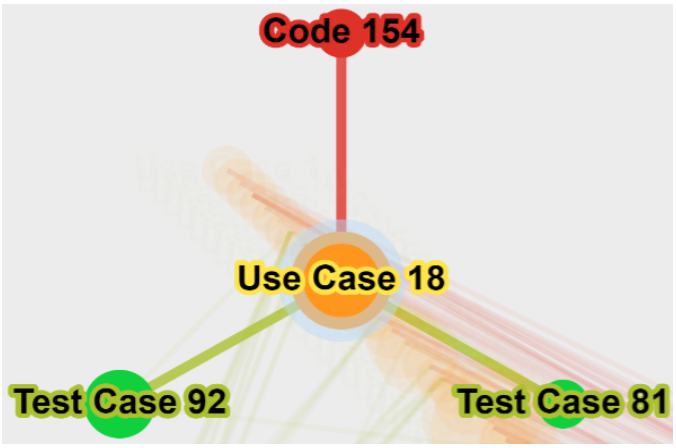

Fig. 2. Filtered view of a source artifact and two target artifacts relation

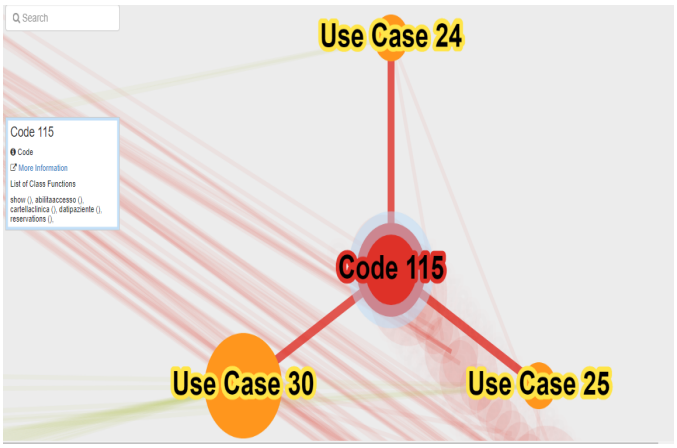

Fig. 3. Filtered view of a target artifact and source artifacts relation

in one area to reduce the tasks of cross-referencing and connecting multiple trace link results. It intends to provide an explicit dependency between artifacts of the system. Figure 1 presents the unfiltered view of use case, test case and source code artifact relations from EasyClinic dataset. Each artifact is formatted in different colours for easy differentiation between artifacts (e.g., Orange –Use Case, Red –Source Code, Green –Test Case). Each node represents an artifact and supports a browse-able filtered view of an artifact by either clicking on each node or using free-text search features. The line represents the relationships between artifacts. The high-level trace map view can assist an analyst in various software engineering tasks, including system comprehension and change impact analysis.

We positioned the source artifact on the top layer and its target artifacts on the lower layer to demonstrate hierarchical layer view of the system. In this view, the line represents one-to-many relations between the source and target artifact. Therefore, if there is no line between the source and target artifact, it means missing links. In Figure 1, some of the last section and middle section of use case artifacts are missing lines in test cases. By using this diagram, we can visually comprehend that some use cases do not have corresponding test cases. To switch from unfiltered view to filtered view, we included a node clicking feature and free-text search feature in our design to support the interactive visual exploration of the system. Free-text search features can search by file name or content.

Figure 2 presents the associations between a source and two target artifacts. We draw the size of the nodes based on either the textual volume of the artifact or the number of source code functions. However, we are still finalising on size measurement definition at this point. For change impact analysis, a developer can use this diagram to visually inspect the impact areas of source codes and test cases intuitively without cross-referencing multiple trace links results.

Figure 3 presents the filtered view between a target and source artifacts. For source code artifacts, we displayed the list of corresponding functions to assist in identifying change impact area effectively.

III. Conclusions

In this paper, we propose a hierarchical trace map visualization approach to assist system comprehension and change impact analysis tasks. Our overarching aim is to enhance the usability of trace link results with the support of interactive visualization techniques. In our future work, we intend to add a more detailed level of abstraction on the structure (e.g., filter feature for missing links artifacts, recover links feature for missing links, draw the size of the node based on an artifact content variation, display the number of associated artifacts) to assist in system comprehension and change impact analysis tasks. We plan to replace our current IR engine processing phase with deep learning techniques to leverage the accuracy of results. In terms of evaluation, we intend to explore our approach in an industrial dataset to explore effectiveness in a real-world software maintenance approach.

References

[1] N. Niu and A. Mahmoud, “Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited,” in *2012 20th*

IEEE International Requirements Engineering Conference (RE), Sep. 2012, pp. 81–90.

[2] G. Bavota, L. Colangelo, A. De Lucia, S. Fusco, R. Oliveto, and A. Panichella, "Traceme: Traceability management in eclipse," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2012, pp. 642–645.

[3] K. Mahmood, H. Takahashi, and M. Alobaidi, "A semantic approach for traceability link recovery in aerospace requirements management system," in *2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems*, March 2015, pp. 217–222.

[4] M. Rahimi, W. Goss, and J. Cleland-Huang, "Evolving requirements-to-code trace links across versions of a software system," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Oct 2016, pp. 99–109.

[5] J. Guo, M. Gibiec, and J. Cleland-Huang, "Tackling the term-mismatch problem in automated trace retrieval," *Empirical Software Engineering*, vol. 22, no. 3, pp. 1103–1142, 2017.

[6] K. Nishikawa, H. Washizaki, Y. Fukazawa, K. Oshima, and R. Mibe, "Recovering transitive traceability links among software artifacts," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 576–580.

[7] K. C. Youm, J. Ahn, J. Kim, and E. Lee, "Bug localization based on code change histories and bug reports," in *2015 Asia-Pacific Software Engineering Conference (APSEC)*, Dec 2015, pp. 190–197.

[8] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE transactions on software engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[9] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 16, no. 4, p. 13, 2007.

[10] C. Mills and S. Haiduc, "A machine learning approach for determining the validity of traceability links," in *Proceedings of the 39th International Conference on Software Engineering Companion*, ser. ICSE-C '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 121–123. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.86

[11] C. Ziftci and I. Krüger, "Getting more from requirements traceability: Requirements testing progress," in *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. IEEE, 2013, pp. 12–18.

[12] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, E. A. Holbrook, S. Vadlamudi, and A. April, "Requirements tracing on target (retro): improving software maintenance through traceability recovery," *Innovations in Systems and Software Engineering*, vol. 3, no. 3, pp. 193–202, 2007.

[13] A. De Lucia, R. Oliveto, and G. Tortora, "Adams re-trace: Traceability link recovery via latent semantic indexing," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 839–842. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368216

[14] J. Cleland-Huang and R. Habrat, "Visual support in automated tracing," in *Second International Workshop on Requirements Engineering Visualization (REV 2007)*, Oct 2007, pp. 4–4.

[15] X. Chen, J. Hosking, and J. Grundy, "Visualizing traceability links between source code and documentation," in *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Sep. 2012, pp. 119–126.

[16] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "When and how using structural information to improve ir-based traceability recovery," in *2013 17th European Conference on Software Maintenance and Reengineering*, March 2013, pp. 199–208.

[17] T. Merten, D. Jppner, and A. Delater, "Improved representation of traceability links in requirements engineering knowledge using sunburst and netmap visualizations," in *2011 4th International Workshop on Managing Requirements Knowledge*, Aug 2011, pp. 17–21.

[18] S. Klock, M. Gethers, B. Dit, and D. Poshyvanyk, "Traceclipse: an eclipse plug-in for traceability link recovery and management," in *Proceedings of the 6th international workshop on traceability in emerging forms of software engineering*. ACM, 2011, pp. 24–30.

[19] A. Marcus, X. Xie, and D. Poshyvanyk, "When and how to visualize traceability links?" in *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*. ACM, 2005, pp. 56–61.

[20] R. Elamin and R. Osman, "Implementing traceability repositories as graph databases for software quality improvement," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018, pp. 269–276.

[21] A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, "Information retrieval methods for automated traceability recovery," in *Software and systems traceability*. Springer, 2012, pp. 71–98.