StoryScreen: a Traceability Visualisation Modelling Tool for User Stories and Conceptual Models

Marcela Ruiz[1,*,†] and Carlos Olgiati[1,†]

[1] *Zurich University of Applied Sciences, Institute of Computer Science, School of Engineering, Winterthur, Switzerland*

Abstract

Requirements and conceptual model traceability is a core software engineering activity that supports decision-making during the entire software development life cycle. With traceability data sets growing, traceability visualisation techniques that properly communicate the relevant information are imperative to ensure quality, completeness, and transparency in enterprise models and information systems. We did a comparative analysis of various graphical visualisation techniques for conceptual model traceability such as graphical graphs, tabular forms, sunbursts, etc., The findings suggest that a one-size-fits-all solution does not exist because factors such as data structure, traceability purpose, and traceability-users' skills determine the level of detail, content, and type of traceability visualisation technique. As a proof of concept, this paper presents an approach towards a traceability visualization modelling tool named StoryScreen, which facilitates the visualisation of traces among user stories and conceptual models. StoryScreen allows users to identify missing traces and fix errors in conceptual models and user stories. StoryScreen provides a project management tool for user story-based projects, which includes various traceability visualisation techniques such as graphical graphs, tabular, and textual. We expect that StoryScreen will support traceability users in improving user stories and creating conceptual models employing traceability visualisations that are filterable and interactive.

Keywords

Requirements Traceability Visualisation, User Stories, Conceptual Models, Modelling Tool

1. Introduction

Requirements traceability and conceptual model traceability are important components of a well-designed and reliable software system. Software traceability allows software engineers and developers to track back and forward specific software artefacts [1]. The growing variety and quantity of traceability artefacts pose challenges for determining appropriate traceability visualisation techniques. This modelling tool paper presents our research efforts in examining the current state of the art for requirements traceability visualisation techniques and their purpose. Our conclusions led to the implementation of the proof-of-concept tool named StoryScreen. StoryScreen allows for the specification of software requirements in the form of user stories and the visualisation of the corresponding traceability links to related conceptual models. For our research, we have defined the following main research questions.

MRQ: Which traceability visualisation techniques have been proposed in the literature and for which purpose or task? We explored the literature regarding visualisation methods for requirements traceability and compiled a list of visualisation techniques. We examined which techniques are popular, how they fit specific traceability purposes and tasks, and if there are requirements that guide the matching of visualisation technique traceability tasks.

* Corresponding author.
† These authors contributed equally.

marcela.ruiz@zhaw.ch (M. Ruiz); olgiacar@students.zhaw.ch (C. Olgiati)
0000-0002-0592-1779 (M. Ruiz)

Companion Proceedings of the 17th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling Forum, M4S, FACETE, AEM, Tools and Demos co-located with PoEM 2024, Stockholm, Sweden, December 3-5, 2024.

This paper is structured as follows: In section 2 we present the reviewed literature summarising different traceability tools and their visualisation techniques. Section 3 presents the design and implementation of our proof-of-concept tool named StoryScreen, which allows for traceability visualisation of conceptual models and user stories, as well as error detection and correction. Finally, section 4 presents the main conclusions and suggests directions for future work.

2. Related work

This section explores software traceability tools and analyses their corresponding visualisation techniques.

2.1. General-purpose traceability tools

There are various traceability tools, each with its own way of visualising the trace data [2]. Some tools rely on rather simple methods. Program Explorer [3] uses directed graphs to convey the traceability links of object-oriented programs. It tries to avoid visual clutter by allowing users to "reduce the search space" by applying certain filters to the data. A different paper proposes enhancements to the conventional traceability matrix [4]. The authors suggest that overlaying different sets of trace links (for example, candidate links and confirmed links) could help recover and validate said links.

The main visualisation technique used in Extravis [5] is what the authors call a "hierarchical edge bundle" to give a global overview. It could be described as an (inverted) sunburst [6] graphic combined with a net map. Detailed data is visualised with "massive sequence views [7][8].

The ExplorViz [9] tool uses the visualisation concept called "3D city metaphor". This three-dimensional visualisation method resembles a city skyline to communicate data to the user more intuitively [10]. Another tool that makes use of a city metaphor for visualisation is DynaCity [11].

2.2. Requirements traceability tools

Regarding requirements traceability, there have also been approaches using sunburst and net map visualisations [12]. The authors were searching for a technique to let them visualise requirements traceability data at scale. Instead of combining the two methods into a single visualisation, the tool uses sunburst to give an overview and the net map technique to give a more detailed perspective for a more focused data set. Similarly, in[13], the authors also recognise the advantage of using two separate visualisations for overviews and detail views. Here, the researchers use a tree map for the overview while presenting a detailed view with a hierarchical tree. The research finds that the approach scales well, is easy to use and supports the user in understanding, browsing and maintaining the trace links.

The tool ReCVisu+ [14] gives a global view by representing all requirements as nodes and clustering them. The researchers focused on providing users with an interactive tool with zoom in and out and clustering capabilities.

Multi-Visio Trace [15] relies on four visualisations: sunburst, hierarchical tree, graph and matrix. In an experiment, the researchers observed which visualisation was checked by the users for what kind of task. The results showed that all visualisation types have been used to complete certain tasks; sunburst was the most common. The study's main finding is that it might be beneficial to offer different visualisations and let users decide how they want to approach a specific problem.

A series of studies evaluating existing visualisation techniques for (requirements) traceability have also been conducted. Many of these studies limit themselves to the most common visualisation types, namely matrices, graphs, lists and hyperlinks [16][17] have a broader scope

[18]. The findings of these studies coincide: each visualisation technique has its own benefits and drawbacks, a one-size-fits-all solution does not exist, and various visualisations need to be provided to support different traceability tasks [19].

2.3. Overview of traceability visualisation techniques

For detailed software traceability analysis, traceability visualisation techniques such as lists, matrices, graphs, trees and hyperlinks have been primarily used [16][17][19]. The most suitable visualisation depends on many factors [20]: What is the question that the user is trying to answer? What are the user's skills? Is there a specific level of detail or a need for a global overview? Depending on these factors, each visualisation type has its use case. Table 1 gives an overview of the presented techniques, the dimension, scalability, and level of detail in supporting a global

Table 1
Overview and comparison of the different traceability visualisation techniques

Visualisation Technique	Dimension	Scalability	Global overview?
Matrix	2-Dimension	Low	No[2]
Graph	Multidimensional	Medium	Yes
List	Multidimensional	Low	No
Hyperlink	Multidimensional	Low-Medium	No
Hierarchical Tree	Multidimensional	Medium	Yes
Sunburst	Multidimensional	Medium-High	Yes

Matrices can only represent two-dimensional data, do not scale well and fail to give a global overview. Nevertheless, requirement traceability matrices are widely used in practice as they are easy to use and implement. Similarly, with lists or hyperlinks, matrix scalability is limited to provide a traceability overview. Lists are suitable to help in the link recovery process and hyperlinks for browsing tasks.

Graph- or tree-based and sunburst visualisation techniques mitigate scalability issues. Their multidimensional nature is appropriate for communicating different traced artefacts in a compact manner. To the best of our knowledge, there is a lack of research and development solutions for specific traceability visualisation techniques for user stories and related conceptual models.

3. Design and architecture of the StoryScreen tool

This section presents the design and architecture of our proof-of-concept tool named StoryScreen. StoryScreen supports users in managing traces between user stories and related conceptual models. The tool offers various conventional visualisations, as well as a graph-based approach called the traceability graph,

3.1. Metamodel

We developed a traceability graph metamodel suitable for representing various software artefacts to be traced. The metamodel was created using the Eclipse Modelling Framework (EMF) and its ECORE metamodel (see Figure 1). Further, the metamodel serves as a basis for the database model used in StoryScreen.

[2] Can give a global overview if only two artefact types exist.

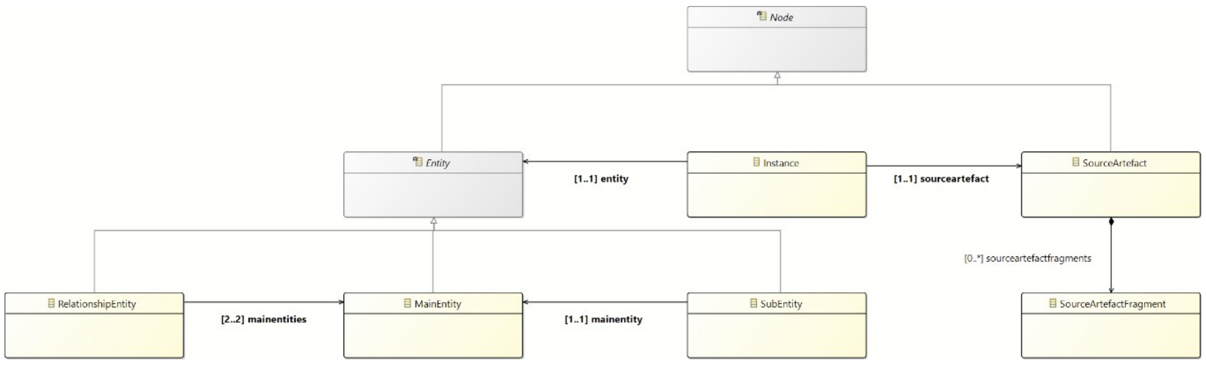


Figure 1: Traceability graph metamodel

In the metamodel, the main metaclasses are Node and Instance. The metaclass Node is specialised in Entity and SourceArtefact. The Entity metaclass represents an entity of a conceptual model or target artefacts with three specialisations:

- *MainEntity*: A MainEntity is a standalone entity, such as a class in a class diagram. It is not dependent on any other Entity.
- *SubEntity*: A SubEntity belongs to a MainEntity. Examples of sub-entities are class attributes or operations (a.k.a. methods).
- *RelationshipEntity*: A RelationshipEntity represents a relationship between two MainEntity classes.

The SourceArtefact metaclass can have multiple children or SourceArtefactFragment. A fragment represents a part of the entire SourceArtefact. For example, if a certain SourceArtefact is a user story, the string of words defining the role (a.k.a., actor) in the user story is considered as a SourceArtefactFragment.

3.2. Traceability Graph Visualisation

The main idea for the traceability graph (see Figure 2) is to present every artefact as a node. Such an artefact can be a user story, a part of a user story or part of a conceptual model (for example, a class or an attribute). Edges between nodes represent two types of relation: Trace link relation and conceptual model relation. Trace link relations represent the traces connecting a source artefact with a target artefact (see Figure 3).

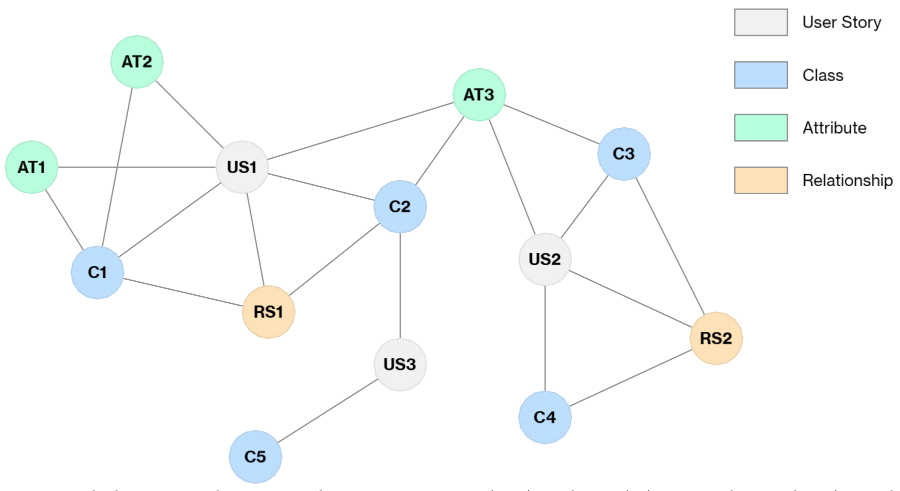


Figure 2: Traceability graph example. User Story (US), Class (C), Attribute (AT), Relationship (RS)

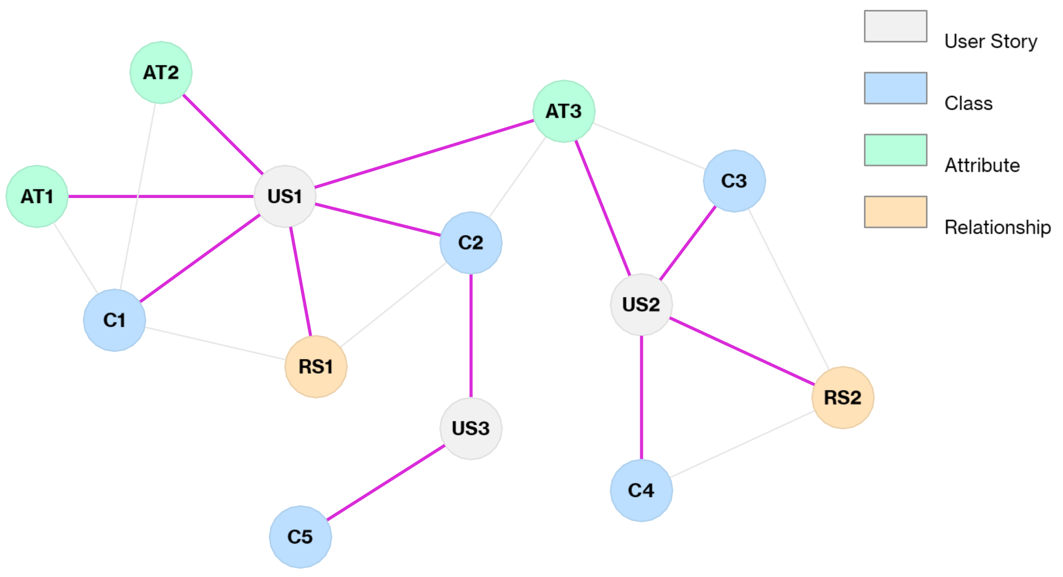


Figure 3: Traceability graph example with highlighted trace links

The conceptual model relations represent relationships between components within elements of a conceptual model (see Figure 4 (right)). An attribute belongs to a class if there is an edge linking the two (see Figure 4 (left)).

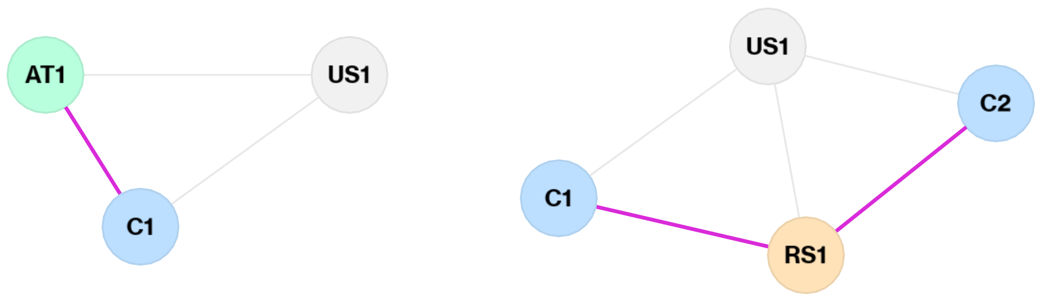
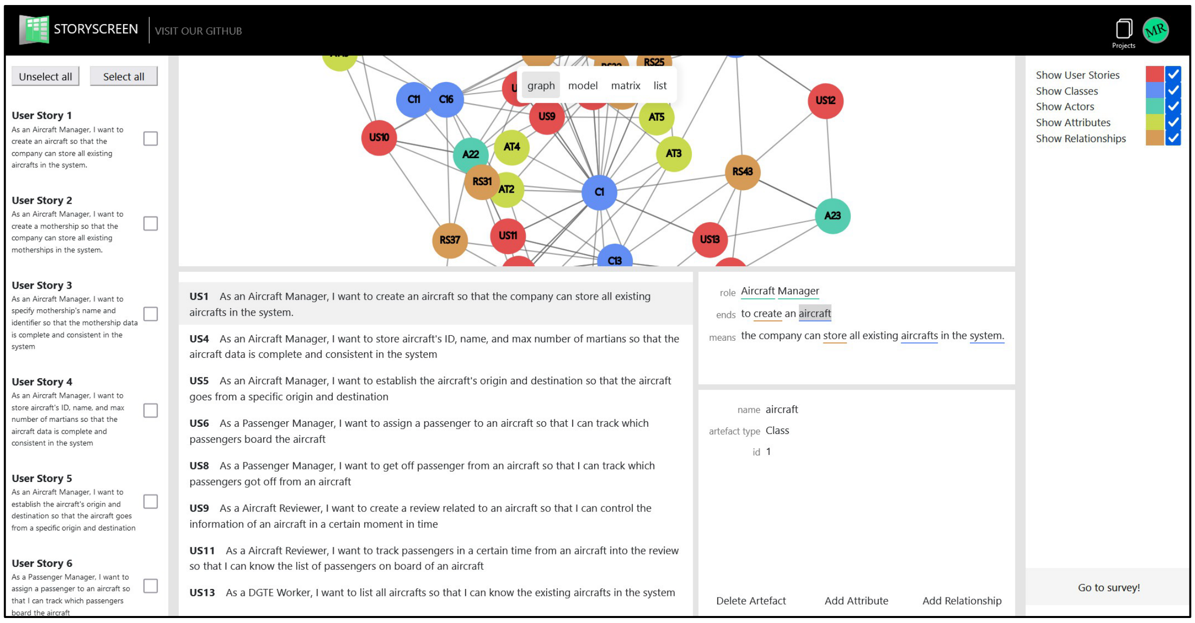


Figure 4: Conceptual model link between a class and an attribute (left) and between two classes and a relationship (right)

3.3. Traceability visualisation user interface

Figure 5 shows the user interface of StoryScreen's proof of concept for traceability visualisations. They are presented in the centre of the screen with a tab navigation at the top to select which visualisation should be displayed. The user can choose between the traceability graph, a conceptual model view, a requirements traceability matrix and a traceability list. The traceability graph and conceptual model allow users to modify the elements, select them, zoom in and out, and move elements around if required.



User Story 1
As an Aircraft Manager, I want to create an aircraft so that the company can store all existing aircrafts in the system.

User Story 2
As an Aircraft Manager, I want to create a mothership so that the company can store all existing motherships in the system.

User Story 3
As an Aircraft Manager, I want to specify mothership's name and identifier so that the mothership data is complete and consistent in the system

User Story 4
As an Aircraft Manager, I want to store aircraft's ID, name, and max number of martians so that the aircraft data is complete and consistent in the system

User Story 5
As an Aircraft Manager, I want to establish the aircraft's origin and destination so that the aircraft goes from a specific origin and destination

User Story 6
As a Passenger Manager, I want to assign a passenger to an aircraft so that I can track which passengers board the aircraft

US1 As an Aircraft Manager, I want to create an aircraft so that the company can store all existing aircrafts in the system.

US4 As an Aircraft Manager, I want to store aircraft's ID, name, and max number of martians so that the aircraft data is complete and consistent in the system

US5 As an Aircraft Manager, I want to establish the aircraft's origin and destination so that the aircraft goes from a specific origin and destination

US6 As a Passenger Manager, I want to assign a passenger to an aircraft so that I can track which passengers board the aircraft

US8 As a Passenger Manager, I want to get off passenger from an aircraft so that I can track which passengers got off from an aircraft

US9 As a Aircraft Reviewer, I want to create a review related to an aircraft so that I can control the information of an aircraft in a certain moment in time

US11 As a Aircraft Reviewer, I want to track passengers in a certain time from an aircraft into the review so that I can know the list of passengers on board of an aircraft

US13 As a DGTE Worker, I want to list all aircrafts so that I can know the existing aircrafts in the system

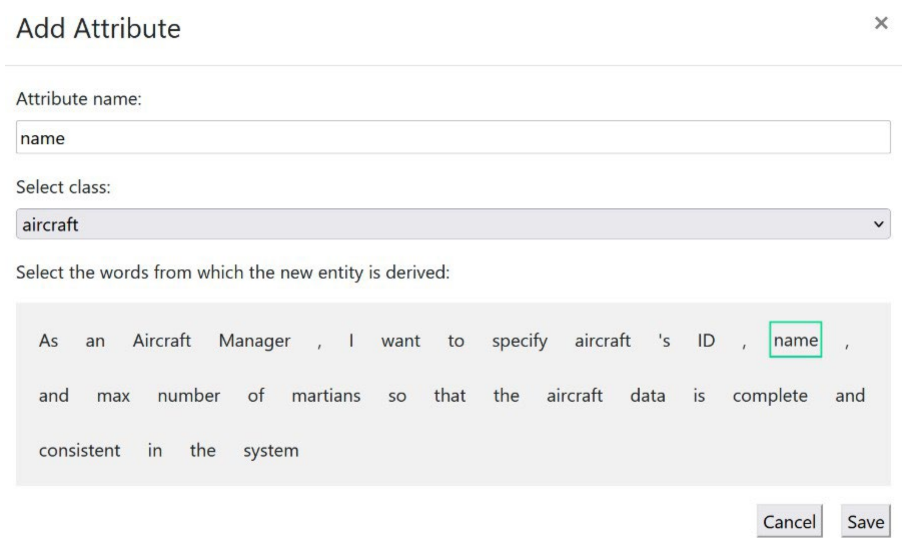
role	Aircraft Manager
ends	to create an aircraft
means	the company can store all existing aircrafts in the system.

name	aircraft
artefact type	Class
id	1

Figure 5: Screenshot of the traceability view

Filters. The goal of the filters is to allow users to fine-tune the presented traceability data to help them remove unnecessary and potentially distracting information. A list of all user stories is presented on the left-hand side of the user interface. These can be toggled on and off to select whether they should be displayed in the visualisations. At the top of the list, buttons to select and unselect all user stories are displayed. On the right side, filters for all node types are shown.

Detail View. When an artefact is selected, a detailed view is presented at the bottom of the screen. This view consists of three elements. A list of all user stories related to the node is shown on the left. The top right displays the current user story and its main fragments: "role", "ends", and "means". All fragments of the user story traced to a target artefact are underlined (colour- coded) and can be clicked to be selected. At the bottom right, a series of actions for the selected node are offered, such as: “delete artefact”, “add class”, “add attribute,” and “add relationship”. When an artefact is to be added, the user can name it and indicate which words (fragments) of the user story the artefact can be traced back (see Figure 7). If an attribute is added, the user also indicates to which class it belongs. Similarly, when adding a relationship, the two classes which are related should be selected



Add Attribute

Attribute name:

name

Select class:

aircraft

Select the words from which the new entity is derived:

As an Aircraft Manager , I want to specify aircraft 's ID , name , and max number of martians so that the aircraft data is complete and consistent in the system

Figure 6: Example form to add an attribute

3.4. Architecture

The architecture of the StoryScreen tool (see Figure 7) has three main components: the web application, the backend application, and the database. The web application that serves the StoryScreen website requests the backend. The backend makes the necessary queries to the database and some required computation on the data, before passing it on to the website.

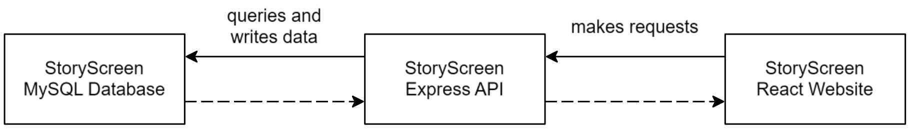


Figure 7: System architecture of the StoryScreen tool

4. Conclusions and future work

In this paper, existing approaches for visualising software traceability data with a focus on traceability among conceptual models and user stories have been introduced and analysed. Each approach is different, but some general visualisation patterns have been identified. A persistent challenge still to be solved is to find visualisation methods that scale and respond to the growing and complex interconnection of traces among software artefacts. We performed a literature exploration and found that various factors determine appropriate traceability visualisation techniques to suit the demands of traceability users and use cases. For example, Hierarchical and space-filing approaches like graphs are good for giving a global overview, while conventional approaches like matrices are helpful if more detailed data is needed. As a proof-of-concept, we have implemented the StoryScreen tool to analyse how different traceability visualization techniques can be implemented to visualise trace links among user stories and conceptual models. The implementation of StoryScreen gives us insights into further study of the influence that different traceability visualisation techniques can have over identifying missing or wrong traceability links and fixing software artefact errors. As part of future work, we plan to investigate how the StoryScreen tool can be used in educational settings to teach software traceability concepts in software engineering lectures. We also plan to investigate additional traceability visualization techniques that consider additional types of source and target artefacts.

Acknowledgements

We acknowledge the valuable research and development contribution of the graduate students Mr Joel Wichser and Mr Severin Schindler, who developed the first version of the StoryScreen tool to allow for the structured specification of User Stories. Their bachelor thesis has served as a foundation for the research in traceability visualisation techniques presented in this paper. The DIZH Fellowship program of ZHAW Digital has supported this research.

5. References

[1] O. C. Z. Gotel and A. C. W. Finkelstein, "Analysis of the requirements traceability problem," *Proceedings of the International Conference on Requirements Engineering*, pp. 94–101, 1994, doi: 10.1109/ICRE.1994.292398.

[2] A. Janes, X. Li, and V. Lenarduzzi, "Open tracing tools: Overview and critical comparison," *Journal of Systems and Software*, vol. 204, p. 111793, Oct. 2023, doi: 10.1016/J.JSS.2023.111793.

[3] D. B. Lange and Y. Nakamura, "Object-oriented program tracing and visualization," *Computer (Long Beach Calif)*, vol. 30, no. 5, pp. 63–70, May 1997, doi: 10.1109/2.589912.

[4] C. Duan and J. Cleland-Huang, "Visualization and analysis in automated trace retrieval," *First International Workshop on Visualization in Requirements Engineering, REV 2006*, p. 5, 2006, doi: 10.1109/REV.2006.6.

[5] D. Holten, B. Cornelissen, and J. J. Van Wijk, "Trace visualization using Hierarchical edge bundles and massive sequence views," *VISSOFT 2007 - Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pp. 47– 54, 2007, doi: 10.1109/VISSOF.2007.4290699.

[6] J. Stasko, R. Catrambone, M. Guzdial, and K. Mcdonald, "An evaluation of space-filling information visualizations for depicting hierarchical structures," *Int J Hum Comput Stud*, vol. 53, no. 5, pp. 663–694, Nov. 2000, doi: 10.1006/IJHC.2000.0420.

[7] B. Cornelissen, A. Zaidman, A. Van Deursen, and B. Van Rompaey, "Trace visualization for program comprehension: A controlled experiment," *IEEE International Conference on Program Comprehension*, pp. 100–109, 2009, doi: 10.1109/ICPC.2009.5090033.

[8] B. Cornelissen, A. Zaidman, and A. Van Deursen, "A controlled experiment for program comprehension through trace visualization," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 341–355, 2011, doi: 10.1109/TSE.2010.47.

[9] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring, "Live trace visualization for comprehending large software landscapes: The ExplorViz approach," *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013*, 2013, doi: 10.1109/VISSOFT.2013.6650536.

[10] F. Fittkau, S. Finke, W. Hasselbring, and J. Waller, "Comparing Trace Visualizations for Program Comprehension through Controlled Experiments," *IEEE International Conference on Program Comprehension*, vol. 2015-August, pp. 266–276, Aug. 2015, doi: 10.1109/ICPC.2015.37.

[11] V. Dashuber and M. Philippsen, "Trace visualization within the Software City metaphor: Controlled experiments on program comprehension," *Inf Softw Technol*, vol. 150, p. 106989, Oct. 2022, doi: 10.1016/J.INFSOF.2022.106989.

[12] T. Merten, D. Jüppner, and A. Delater, "Improved representation of traceability links in requirements engineering knowledge using Sunburst and Netmap visualizations," *2011 4th International Workshop on Managing Requirements Knowledge, MaRK'11 - Part of the 19th IEEE International Requirements Engineering Conference, RE'11*, pp. 17–21, 2011, doi: 10.1109/MARK.2011.6046557.

[13] X. Chen, J. Hosking, and J. Grundy, "Visualizing traceability links between source code and documentation," *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, pp. 119–126, 2012, doi: 10.1109/VLHCC.2012.6344496.

[14] S. Reddivari, S. Rad, T. Bhowmik, N. Cain, and N. Niu, "Visual requirements analytics: A framework and case study," *Requir Eng*, vol. 19, no. 3, pp. 257–279, Nov. 2014, doi: 10.1007/S00766-013-0194-3/FIGURES/18.

[15] A. Rodrigues, M. Lencastre, and G. A. A. De Cysneiros Filho, "Multi-VisioTrace: Traceability visualization tool," *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016*, pp. 61–66, Jan. 2017, doi: 10.1109/QUATIC.2016.019.

[16] S. Winkler, "On usability in requirements trace visualizations," *2008 3rd International Workshop on Requirements Engineering Visualization, REV'08*, pp. 56–60, 2008, doi: 10.1109/REV.2008.4.

[17] Y. Li and W. Maalej, "Which traceability visualization is suitable in this context? A comparative study," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7195 LNCS, pp. 194–210, 2012, doi: 10.1007/978-3-642-28714-5_17/COVER.

[18] A. A. Madaki and W. M. N. W. Zainon, “A visual framework for software requirements traceability,” *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 1, pp. 426–434, Feb. 2022, doi: 10.11591/EEI.V11I1.3269.

[19] A. A. Madaki and W. M. N. W. Zainon, “A Review on Tools and Techniques for Visualizing Software Requirement Traceability,” *Lecture Notes in Electrical Engineering*, vol. 829 LNEE, pp. 39–44, 2022, doi: 10.1007/978-981-16-8129-5_7/TABLES/2.

[20] D. Pfitzner, V. Hobbs, and D. Powers, “A Unified Taxonomic Framework for Information Visualization,” 2001, doi: 10.5555/857080.857087.

A. Video demonstration of the StoryScreen tool with traceability visualisation

https://youtu.be/8bEqyM1HbUw