ROOT: Requirements Organization and Optimization Tool

Katherine R. Dearstyne
Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
kdearsty@nd.edu

Alberto D. Rodriguez
Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
arodri39@nd.edu

Jane Cleland-Huang
Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
JaneClelandHuang@nd.edu

***Abstract*—Software engineering practices such as constructing requirements and establishing traceability help ensure systems are safe, reliable, and maintainable. However, they can be resource-intensive and are frequently underutilized. To alleviate the burden of these essential processes, we developed the Requirements Organization and Optimization Tool (ROOT). ROOT centralizes project information and offers project visualizations and AI-based tools designed to streamline engineering processes. With ROOT's assistance, engineers benefit from improved oversight and early error detection, leading to the successful development of software systems.**

Link to screen cast: https://youtu.be/3rtMYRnsu24

***Index Terms*—Requirements Management, Documentation, Software Engineering**

I. Introduction

Collaboration is critical to the success of building any complex software system, yet the coordination of individuals from different backgrounds and expertise presents challenges of its own [1], [2]. Practices such as constructing requirements, establishing traceability, and performing validation and testing are designed to mitigate potential issues earlier and facilitate better communication across individuals and groups [3], [4]. Despite the usefulness of these practices, they are often difficult to build into the process due to the additional time and resources required to properly implement them [5], [6], [7]. As a result, they are often ignored, delayed, or inadequately sustained [8], [9], especially in startups and small companies where speed is often prioritized over comprehensive requirements engineering processes [10], [11]. Furthermore, even when these practices are implemented, collaboration between groups can introduce inconsistencies or confusion that ultimately results in failures down the road [12]. Therefore, easing the burden of these processes on engineers is paramount to ensure systems are safe, reliable, and maintainable.

To ease the burden of requirements and software engineering processes, we developed **ROOT** (**R**equirements **O**rganization and **O**ptimization **T**ool). ROOT serves as the hub of all project information, organized into a hierarchical artifact tree (Figure 1) that allows users to easily navigate their projects. Additionally, ROOT offers a range of tools designed to expedite engineering processes and provide engineers with an additional layer of oversight to catch mistakes early, acting as an *assistant* rather than a replacement for human engineers. These tools encompass the following core features:

- **Integration Across Knowledge Sources:** ROOT integrates seamlessly with GitHub, Jira, and other project knowledge sources, centralizing all project information within the platform.
- **Project Summarization:** ROOT automatically generates a summary of the software project, highlighting subsystems and project features. It also offers summaries of individual source code files to accelerate project understanding.
- **Artifact Creation/Generation:** Engineers can load existing artifacts into ROOT or create new ones through a user-friendly interface. When documentation is lacking, ROOT automatically generates hierarchical layers of documentation from source code.
- **Project Visualization** Users can visualize their project in multiple ways, including an artifact tree or a table. To focus on a specific aspect at at time, they can select a subset of their project and create a *view* of it.
- **AI-based Project Chat:** ROOT provides a chat interface for project-specific queries, which references any pertinent artifacts in its responses, enabling users to verify information easily.
- **Trace-link Generation:** ROOT predicts trace links between related artifacts, providing explanations for each link. Reviewers can easily accept or reject predicted links and create new ones.
- **Project Vocabulary:** To encourage consistent terminology across the project, ROOT automatically extracts project concepts and vocabulary and links them to the relevant artifacts.
- **Requirements Assessment:** ROOT automatically checks requirements for inconsistencies or ambiguities. Reviewers can also manually flag requirements to alert the relevant parties.

The remainder of this paper is organized as follows: Section II presents previous work that ROOT builds upon. Section III outlines the technical details of the tool. Section IV performs a walk through of each features in more detail. Finally, Section V contains future work and Section VI concludes our paper.

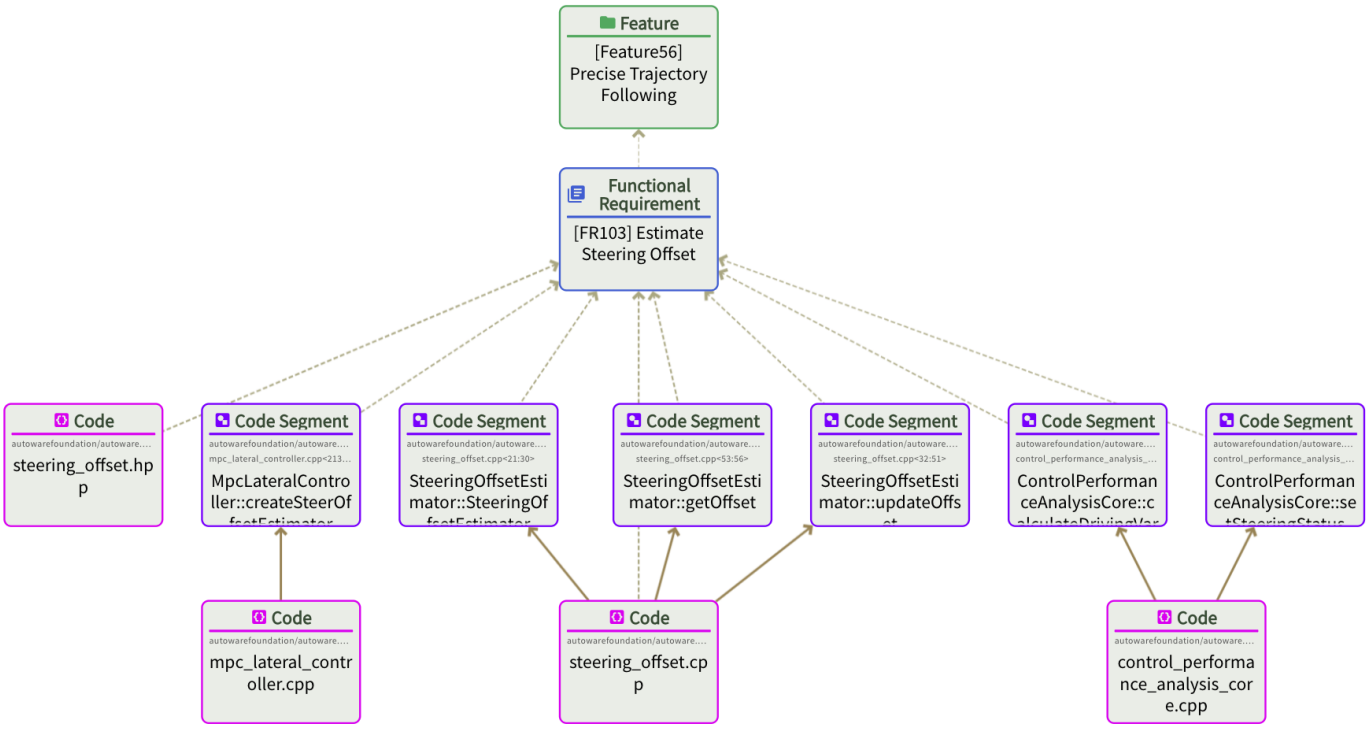

Fig. 1. Slice of documentation and links generated for Autoware [13] open source project.

II. Related Work

Early management systems like DOORs focused on artifact management and trace link organization, laying the groundwork for representing engineering processes in software. Over time, tools like JIRA and Siemens's Polarion expanded their roles, with JIRA evolving from issue tracking to managing the entire software development process, and Polarion addressing system requirements. While these tools were manual and labor-intensive, modern alternatives like JAMA, Visure Solutions, Osseno, and SpiraTeam have begun integrating AI for enhancements such as quality checks, onboarding assistance, and the generation of user stories and test cases, though they primarily augment existing processes rather than creating new ones. ROOT distinguishes itself by catering to startups and small companies that often lack rigorous software management in the early stages of product development. Its documentation generation pipeline produces an initial draft of system requirements linked to source code, significantly reducing the effort of documentation creation and allowing teams to focus on refinement. Furthermore, through collaborations with NASA's Goddard and Jet Propulsion Laboratories, ROOT has identified additional features that can help multi-team organizations detect conflicts early and stay aligned throughout the project life-cycle. These include identifying inconsistencies across an entire project's requirements and maintaining a centralized graph of project concepts. Collectively, these capabilities enable ROOT to support organizations at all stages of maturity.

III. Architecture and Processes

A. Architecture

The ROOT system consists of three key components: a front-end application (FEND), a back-end server (BEND), and a generation server (GEN). FEND, built with Vue.js and Cytoscape, visualizes projects, artifacts, and trace links as interactive graphs, with options to view these elements in tables or query them via a chat interface. BEND, implemented in Java with Spring Boot, manages user interactions, data retrieval, and job execution through a REST API, using a MySQL database to store all project-related data. GEN, a Python-based Django application with Celery for job management, handles AI-driven tasks like documentation generation and chat features, leveraging advanced language models and providing updates on job status and results. Together, these components create a scalable system for managing and generating project documentation and traceability.

B. Job-Execution

In ROOT, job execution begins when a user initiates a job through the front-end client, prompting the back-end server to create a job entry, start execution, and provide a job ID for tracking. The back end then asynchronously retrieves project data and sends it to the generation server (GEN), which processes the job and returns its ID for progress updates. Throughout the job, the back end polls GEN for updates, logs progress, and, upon completion, retrieves and processes results from an S3 bucket. The back end then saves new artifacts and trace links, marks the job as complete, and notifies the user

via email and WebSockets, ensuring efficient and automated requirements engineering tasks.

C. AI-based Features

ROOT's value lies in its comprehensive feature set, built upon existing research which we link in case more details are wanted. For tasks like summarization, document generation, chat, and explanations, ROOT leverages Anthropic's LLMs [14], known for their large context windows, and enhances their output with techniques such as Retrieval-Augmented Generation (RAG [15]), Chain-of-Thought Reasoning [16], and the ReAct approach [17]. For example, ROOT's chat and inconsistency detection methodologies are similar to those described in [18], while our trace-link explanations are based on methods used in [19]. Additionally, ROOT employs cross-encoder and embedding models, including BERT variants [20], [21] and Sentence-BERT [22], for tasks like trace-link prediction and clustering artifacts in the documentation generation pipeline.

IV. Walk Through

This section provides a graphical walkthrough of ROOT, illustrating the user experience and interactions with the system for each of the core features outlined in Section I. Throughout our walkthrough, we use a subset of the Autoware Foundation open-source project [23] as a running example.

A. On-boarding:

Upon entering the tool, users are encouraged to go through an on-boarding process to help them import their project in the platform. We have observed that many users possess only code and want to take advantage of ROOT's features designed to address gaps in their software engineering practices. Therefore, this initial on-boarding is specifically tailored for users whose primary data source is GitHub. Users with different needs can opt out of this workflow and start from the project import screen which allows for more flexibility.

During on-boarding, users are prompted to import their project codebase, generate a project summary, and create any missing documentation.

B. Integration Across Knowledge Sources

From the project import screen, users are given three ways to import a project: via GitHub, Jira, or flat files. If a user is beginning a new project, they may choose to begin with a blank project.

If a user opts to import an existing project, they are prompted to connect their GitHub or Jira account, or to upload the required files. Once this is done, an import job is initiated, allowing the user to monitor its progress.

For the remainder of the walk-through, we will assume that the user is connecting their Git-hub account and move through the remaining on-boarding steps.

C. Project Summarization

Once a user has imported their project, a summary of the project will be automatically generated, along with summaries for all code files. The project summary includes an overview of the system and sections detailing sub-systems, entities, features, and data flow.

D. Artifact Generation

The on-boarding process also allows users to generate two new layers of documentation for their project. If they choose to do so, the system initiates a multi-step pipeline that was developed and evaluated in previous work [24]. This pipeline first identifies groups of related code based on common functionality, then uses a LLM to generate Functional Requirements and then repeats the process to generate higher-level Features for them. While ROOT can also generate various document formats, such as user stories or design documents, we found that Functional Requirements and Features are particularly beneficial for users when they are first starting out. However, once the users enter the platform, they can generate additional types of documentation at any time.

E. Project Visualization

After onboarding, users are directed to the project home page (as shown in Figure 2), which displays the project's Traceability Information Model (TIM), a generated project summary, and graph controls. This overview helps users quickly grasp the project's structure and key features. ROOT offers two primary methods for viewing artifacts: a tree view (Figure 4) and a table view (Figure 5) . For large projects, the tree visualization is disabled by default to avoid overwhelming users, though they can target a single artifact to view in the graph. The tree view also allows for expanding and collapsing branches to manage complexity. The table view, suitable for handling numerous artifacts, enables filtering, sorting, and fuzzy searching, with an additional tab for viewing and managing trace links, including their review status.

When users click on an artifact in either view, a side panel opens with more detailed information (Figure 3), and double-clicking allows them to create a focused *view* centered on the selected artifact and its relationships. This setup is demonstrated using the Autoware project example, where users can efficiently locate and explore specific features, such as the braking functionality, and visualize its parent and children. A video walkthrough provides further details on these visualization capabilities, offering users a practical guide to navigating their projects within ROOT.

F. AI-based Project Chat:

Alongside the tree and table view buttons, users can access the chat interface. From this screen, users can ask questions about their project and receive responses generated by a LLM, as shown in Figure 6. To encourage accurate responses, the LLM is provided with related project artifacts. These artifacts are displayed to the user as clickable buttons beneath the LLM's response. Clicking on one of these artifacts opens a detailed view of that artifact.

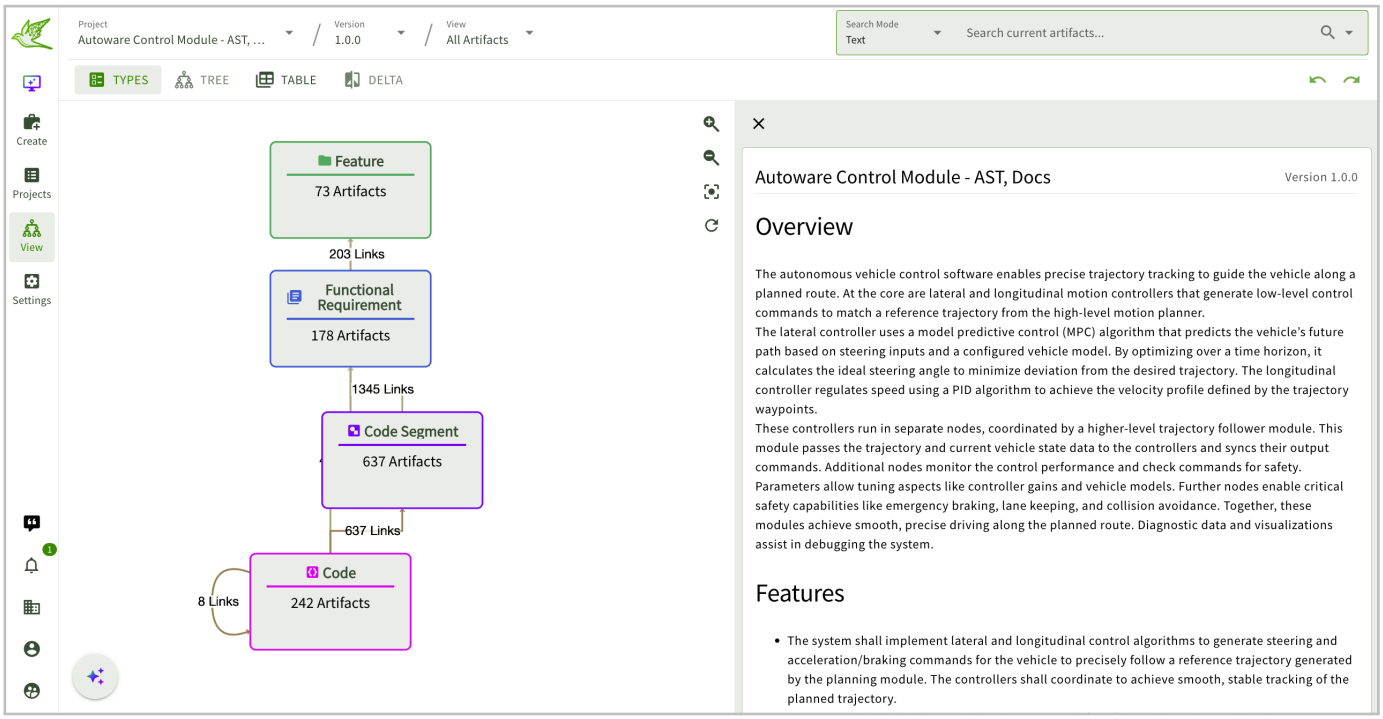

Fig. 2. Project overview screen containing Traceability Information Model (TIM) and graph controls. Project summary excluded for display purposes.

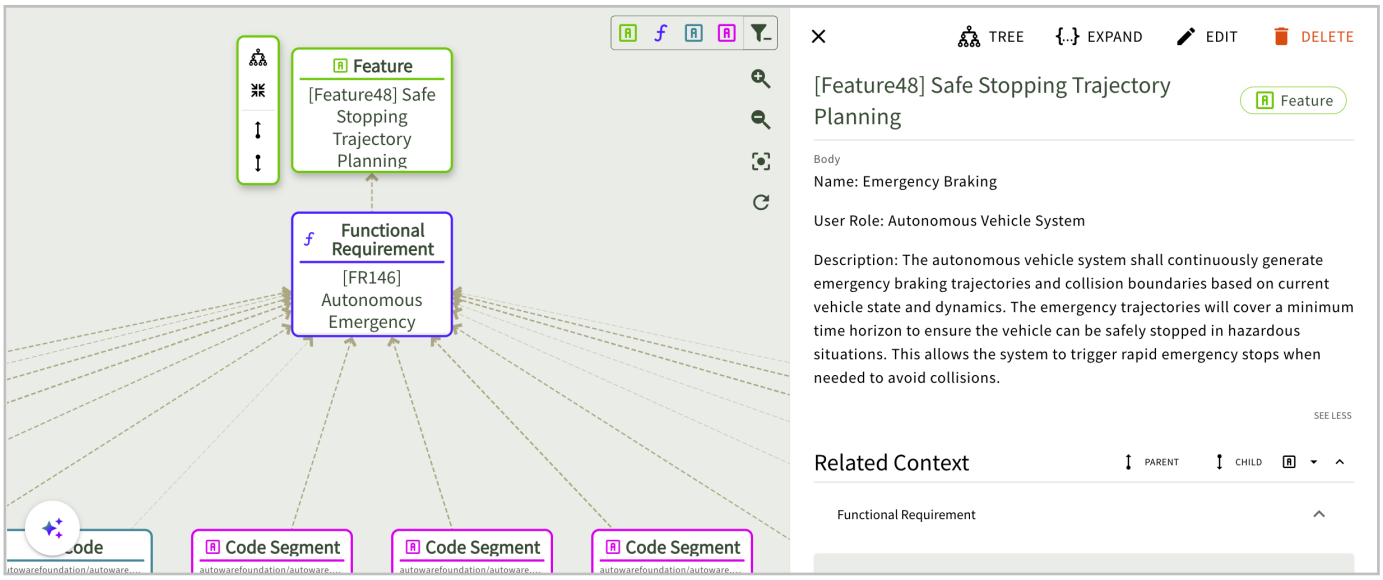

Fig. 3. Selected artifact view of generated feature containing description and further details.

G. Trace-link Generation:

To capture any potentially missed relationships, ROOT provides automatic trace-link generation. Users can select the child type(s) and parent type(s) for which trace-links will be predicted. Once the predictions are completed, all generated links will be added to the project as dotted lines for user review. Additionally, a confidence score and an LLM-generated explanation will be provided to elaborate on the potential relationship between the artifacts as can be seen in Figure 7.

H. Project Vocabulary:

ROOT also assists users in maintaining a project vocabulary to facilitate better alignment among different experts involved in a project. Terminology is automatically identified during health checks, as detailed in the following section. Additionally, users can manually add new concepts at any time, similar to adding a traditional artifact. These concepts are then utilized as context when viewing artifacts, querying in the chat, or evaluating other requirements.

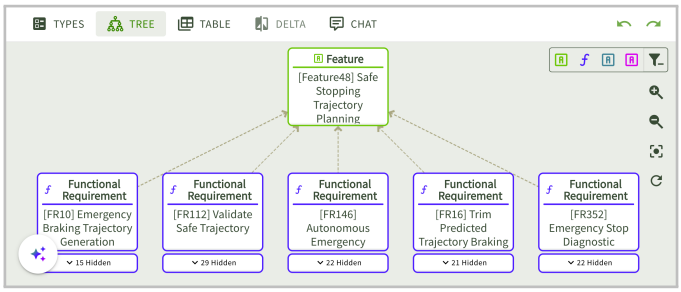

Fig. 4. Tree view of generated feature from Autoware Project.

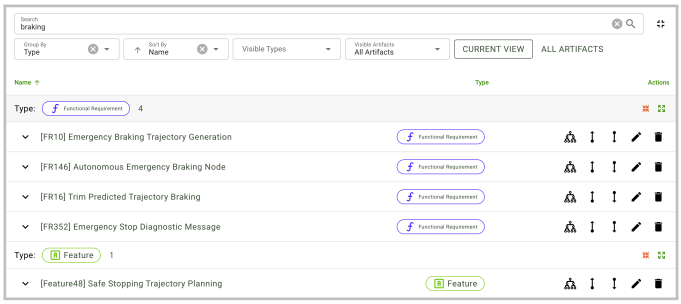

Fig. 5. Table view of search results for query 'braking'.

I. Requirements Assessment:

Users can perform a *health check* on any of their natural language documents to identify potential issues with requirements. During the *health check*, the system flags inconsistencies with other requirements and highlights undefined or ambiguous concepts within the current project vocabulary. If any terminology in the artifact is undefined, users can address the warning by adding the term to the project vocabulary. Furthermore, the system identifies any existing project terminology that is referenced in the artifact, with automatic links created between those concepts and the current artifact.

To illustrate these health checks, we have constructed a small project containing 4 requirements and 2 project concepts (*Job* and *Database Entity*) [25]. For illustration, two of the requirements are shown below:

R1: The system shall be able to save some entities to the database, perform a job, and return the result of the job to the user in under 1 minute.

R4: Saving entities to the database shall take between 0.5 seconds to 5 seconds to complete.

If we run a health check on Requirement *R1*, ROOT identifies several issues and display them for resolution (Figure 8). The term *system* has been flagged as a missing concept in the project vocabulary. An existing concept, *Job*, is directly cited in the artifact, so the system creates a link between them. Additionally, the concept *Database Entity*, although not directly cited, is predicted to be used in *R1*. Finally, a contradiction is detected between *R1* and *R4* (see below) and an explanation is provided.

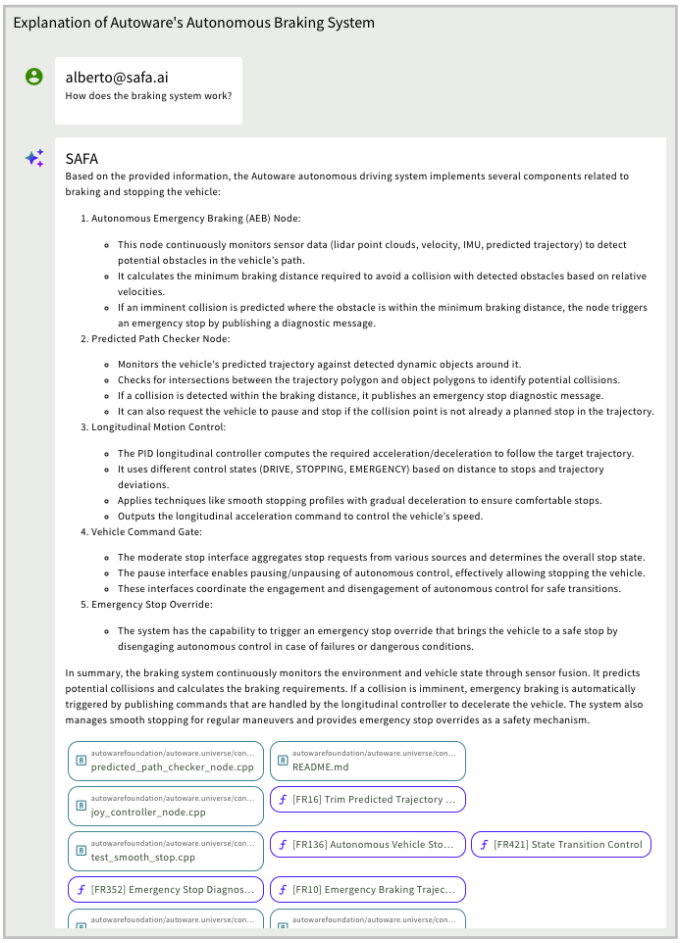

Fig. 6. Example chat response for Autoware project braking sub-system.

V. Future Work

Our ultimate vision for ROOT is to support all aspects of the software engineering process comprehensively. To achieve this, we anticipate several future enhancements to ROOT. Firstly, we plan to integrate additional data sources, such as DOORs and PDFs, allowing engineers to connect all elements of their projects seamlessly. Additionally, we aim to enable users to link back to other tools, such as opening a code artifact in their preferred IDE. We also seek to improve artifact maintenance by identifying outdated documentation and updating it automatically. Finally, we intend to implement more health checks, such as flagging incomplete or unverifiable requirements. With these enhancements, ROOT will be better positioned to streamline software workflows and enhance overall project quality.

VI. Conclusion

In this paper, we introduce ROOT, a tool designed to enhance collaboration and requirements management in software development. By automatically generating documentation and trace-links from the ground up, ROOT makes robust requirements engineering processes accessible even to startups and small companies. Through its visualizations and AI-based features, ROOT promotes more effective collaboration and

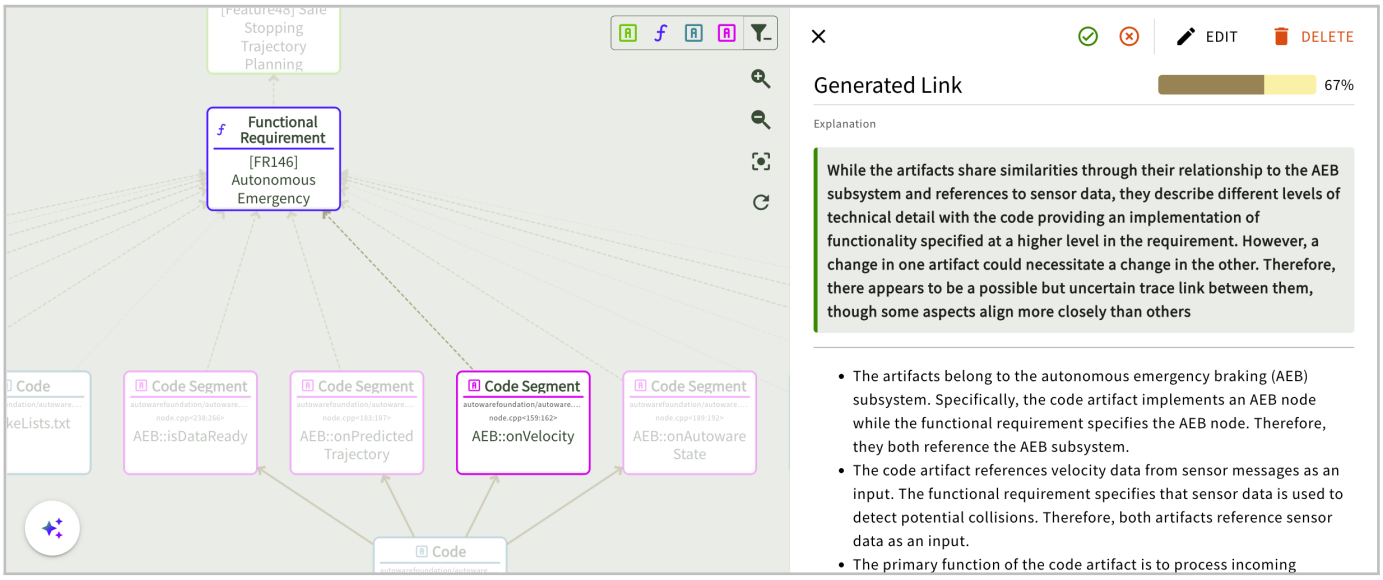

Fig. 7. Detailed view of generated link between generated documentation and a code segment.

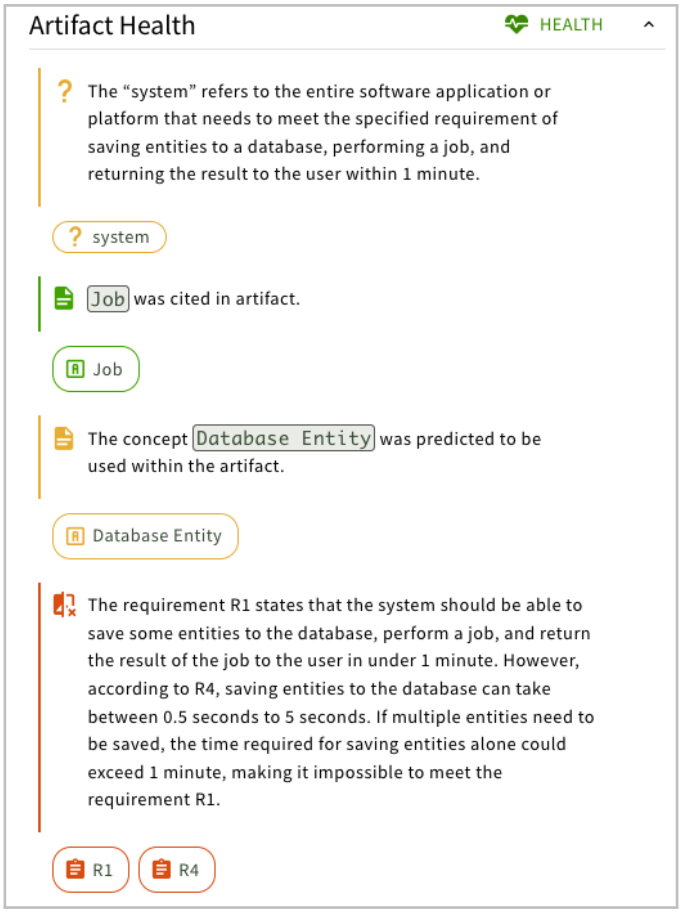

Fig. 8. Artifact health checks showing cited concept, undefined concept, predicted concept, and contradiction in artifact 'R1'.

early problem detection. Ultimately, ROOT aims to ease the burden of engineering processes and aid in developing safer, more reliable systems faster.

VII. Acknowledgements

The research described in this paper was partially supported by the USA National Science Foundation (NSF) under grant numbers 1909007 and 1901059.

References

[1] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams."

[2] R. Yasrab, J. Ferzund, and S. Razzaq, "Challenges and issues in collaborative software developments," Mar. 2019, arXiv:1904.00721 [cs]. [Online]. Available: http://arxiv.org/abs/1904.00721

[3] J. Whitehead, "Collaboration in Software Engineering: A Roadmap," *Future of Software Engineering (FOSE '07)*, pp. 214–225, May 2007, conference Name: Future of Software Engineering ISBN: 9780769528298 Place: Minneapolis, MN, USA Publisher: IEEE. [Online]. Available: http://ieeexplore.ieee.org/document/4221622/

[4] M. M. Lehman, "Software engineering, the software process and their support," *Software Engineering Journal*, vol. 6, no. 5, pp. 243–258, Sep. 1991, publisher: IET Digital Library. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/sej.1991.0028

[5] J. Bayer and D. Muthig, "A view-based approach for improving software documentation practices," Apr. 2006, p. 10 pp.

[6] J. Ahonen and T. Junttila, "A case study on quality-affecting problems in software engineering projects," Dec. 2003, pp. 145–153.

[7] E. Lim, N. Taksande, and C. Seaman, "A Balancing Act: What Software Practitioners Have to Say about Technical Debt," *Software, IEEE*, vol. 29, pp. 22–27, Nov. 2012.

[8] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang, "Mind the gap: assessing the conformance of software traceability to relevant guidelines," in *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, P. Jalote, L. C. Briand, and A. van der Hoek, Eds. ACM, 2014, pp. 943–954. [Online]. Available: https://doi.org/10.1145/2568225.2568290

[9] GhanbariHadi, VartiainenTero, and SiponenMikko, "Omission of Quality Software Development Practices," *ACM Computing Surveys (CSUR)*, Feb. 2018, publisher: ACMPUB27New York, NY, USA. [Online]. Available: https://dl.acm.org/doi/10.1145/3177746

[10] N. Paternoster, C. Giardino, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software development in startup companies: A systematic mapping study," *Information and Software Technology*, vol. 56, no. 10, pp. 1200–1218, Oct. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914000950

[11] C. Giardino, N. Paternoster, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software Development in Startup Companies: The Greenfield Startup Model," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 585–604, Jun. 2016, conference Name: IEEE Transactions on Software Engineering. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7360225

[12] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Commun. ACM*, vol. 31, no. 11, pp. 1268–1287, Nov. 1988. [Online]. Available: https://dl.acm.org/doi/10.1145/50087.50089

[13] Autowarefoundation, "Autowarefoundation/autoware.universe." [Online]. Available: https://github.com/autowarefoundation/autoware.universe

[14] A. Askell, Y. Bai, A. Chen, D. Drain, D. Ganguli, T. Henighan, A. Jones, N. Joseph, B. Mann, N. DasSarma, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, J. Kernion, K. Ndousse, C. Olsson, D. Amodei, T. Brown, J. Clark, S. McCandlish, C. Olah, and J. Kaplan, "A general language assistant as a laboratory for alignment," 2021.

[15] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Apr. 2021, arXiv:2005.11401 [cs]. [Online]. Available: http://arxiv.org/abs/2005.11401

[16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," Jan. 2023, arXiv:2201.11903 [cs]. [Online]. Available: http://arxiv.org/abs/2201.11903

[17] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," Mar. 2023, arXiv:2210.03629 [cs]. [Online]. Available: http://arxiv.org/abs/2210.03629

[18] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, "AI-based Question Answering Assistance for Analyzing Natural-language Requirements," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, May 2023, pp. 1277–1289, iSSN: 1558-1225. [Online]. Available: https://ieeexplore.ieee.org/document/10172663

[19] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability," Aug. 2023. [Online]. Available: https://arxiv.org/abs/2308.00229v1

[20] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained bert models," *arXiv:2102.04411 [cs]*, Feb 2021, arXiv: 2102.04411. [Online]. Available: http://arxiv.org/abs/2102.04411

[21] J. Lin, A. Poudel, W. Yu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Enhancing automated software traceability by transfer learning from open-world data," no. arXiv:2207.01084, Jul 2022, arXiv:2207.01084 [cs]. [Online]. Available: http://arxiv.org/abs/2207.01084

[22] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," Aug. 2019, arXiv:1908.10084 [cs]. [Online]. Available: http://arxiv.org/abs/1908.10084

[23] "GitHub - autowarefoundation/autoware.universe — github.com," https://github.com/autowarefoundation/autoware.universe, [Accessed 27-06-2024].

[24] K. R. Dearstyne, A. D. Rodriguez, and J. Cleland-Huang, "Supporting Software Maintenance with Dynamically Generated Document Hierarchies," Aug. 2024, arXiv:2408.05829 [cs]. [Online]. Available: http://arxiv.org/abs/2408.05829

[25] A. Rodriguez, "Requirement set with defects," Jun. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.12574870