AI-SQUARE: Knowledge Graphs for Requirements-driven Software Staging Management

David Mosquera[1,*], Marcela Ruiz[1], Olivier Mann[2] and Makram Hanin[2]

[1]Zürich University of Applied Sciences, Gertrudstrasse 15, Winterthur 8400, Switzerland
[2]Swiss Digital Network, Auf der Mauer 1, Zurich 8001, Switzerland

Abstract

C**ontext and motivation.** In today's fast-paced software development, stakeholders expect frequent deployments that incorporate the newest requirements and needed adaptations. Such fast pace poses significant pressure on site reliability engineers (SRE), who ensure software quality while managing transitions through stages (testing, development, or production stages). **Problem.** The primary challenge SRE face is the amount of ubiquitous data ranging from requirements to test results (often) interconnected from the different software development phases (requirements elicitation, development, testing, and deployment) that requires effective analysis to inform staging decisions. **Solution.** This challenge has motivated us to establish the AI-SQUARE innovation project; in which we envision the creation of an intelligent end-to-end software staging support platform for decision making and feedback. We propose to leverage the use of knowledge graphs to analyse data from various sources and facilitate quality analysis of aggregated data. **Results and conclusions.** In this poster & tool paper, we share our progress on designing the knowledge graph that supports the AI-SQUARE platform, enhancing intelligent agents and supporting site reliability engineers during software staging. We share our vision and open challenges in implementing and evaluating our knowledge graph.

Keywords

Knowledge Graphs, Software Staging and Quality, AI-SQUARE, Requirements Analysis, Continuous Deployment

1. Introduction and Motivation

Nowadays, software stakeholders are used to receiving new versions of their software with a fastening frequency. Behind the scenes, the trend towards more and faster releases stresses software quality assurance. Especially site reliability engineers—i.e., the role in charge of assuring software quality in continuous integration/continuous deployment pipelines (CICD)—face challenges to guarantee the software quality, the alignment with the software requirements, and its software staging through the development, testing, and production stages.

In CICD pipelines, staging represents the decision on how software goes from different environments as development, testing, or to production—i.e., stages. This decision is based mainly on software quality data analysed in quality gates (often in a cumulative way from one stage to another) and the experience of site reliability engineers (see Figure 1). OpenTelemetry initiatives [1] have used observability tools such as Dynatrace [2], Prometheus [3], and Jaeger [4], among others, serving as a way to extract software quality data and support site reliability engineers' decisions on when to transition from one stage to another. However, some challenges have arisen:

- **Software quality assurance data is ubiquitous (data is everywhere):** Software requirements, software test description, regulations, performance metrics, unit test results, application logs, test logs, and distributed traces, among others, are examples of potential software quality-related data—a.k.a. data sources—present for analysis on staging decision.

In: A. Hess, A. Susi, E. C. Groen, M. Ruiz, M. Abbas, F. B. Aydemir, M. Daneva, R. Guizzardi, J. Gulden, A. Herrmann, J. Horkoff, S. Kopczyńska, P. Mennig, M. Oriol Hilari, E. Paja, A. Perini, A. Rachmann, K. Schneider, L. Semini, P. Spoletini, A. Vogelsang. Joint Proceedings of REFSQ-2025 Workshops, Doctoral Symposium, Posters & Tools Track, and Education and Training Track. Co-located with REFSQ 2025. Barcelona, Spain, April 7, 2025.

*Corresponding author.
mosq@zhaw.ch (D. Mosquera); ruiz@zhaw.ch (M. Ruiz); olivier.mann@swiss-digital-network.ch (O. Mann); makram.hanin@swiss-digital-network.ch (M. Hanin)
0000-0002-0552-7878 (D. Mosquera); 0000- 0002-1320-8471 (M. Ruiz)

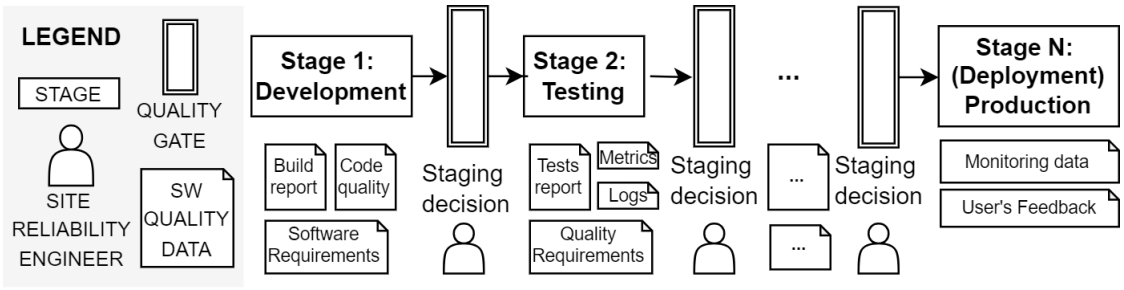

Figure 1: Example of Software Staging

- **Software quality assurance data is ubiquitous and (often) relates to each other but is often not well interconnection:** Isolated analysis of a metric, log, or requirement may neglect the effect on overall software quality. Thus, site reliability engineers must analyse software quality, considering often implicit relationships and connections among software quality data, often relying on their intuition.
- **Justify a software staging decision:** Time to market is critical in software development. If a staging decision for a software deployment should be aborted, site reliability engineers need to justify this decision based on observed data and, mainly because of the difficulties to analyse vast amounts of quality data, on their expertise.

Some authors have used taxonomies [5] and ontologies to produce toolkits for events and anomalies analysis. Others have used knowledge graphs [6, 7, 8] for root cause analysis and anomaly detection in cloud applications, focusing mainly on microservice applications. Other commercial tools have supported reliability engineers by providing data-specific analysis, such as correlations between performance metrics [9]. However, there is still a gap in the solution to providing end-to-end support for software staging decisions compiling different sources of data and quality attributes.

This gap has inspired the Swiss Digital Network (SDN) [10] and three research groups in Switzerland to develop the AI-SQUARE project: an *Intelligent Staging Management Platform*. The AI-SQUARE platform aims to support site reliability engineers in deciding where to or when not to transition stages using intelligent agents. In this poster & tools paper, we share our design (see Section 2) and vision (see Section 3) of the knowledge graph that supports the AI-SQUARE platform to address the aforementioned challenges and provide end-to-end staging support (see Figure 2).

2. AI-SQUARE Knowledge Graph Design and OpenTelemetry Example

A knowledge graph is a structured representation of real-world entities and their interrelationships organized within a directed graph, whose nodes represent **entities** and edges represent **relations between them** [11]. Knowledge graphs have been used in different domains to extract facts and assertions based on cross-domain data, such as Wikidata [12] and DBPedia [13]. This shows how knowledge graphs can ingest data from various sources and formats, demonstrating their flexibility for **storing interconnected ubiquitous and heterogeneous data from multiple sources**. Moreover, knowledge graphs have been used to support root-cause analysis (RCA) in anomaly detection [6, 7, 8]. This progress shows that knowledge graphs **allow to support decision-making**, providing facts and potential causes when anomalies are detected.

In this Section, we show the progress in designing the AI-SQUARE knowledge graph. To show our design, we follow the framework for designing and building enterprise knowledge graphs [14]: first identifying *the source of data*, then defining *the target knowledge graph*, and, finally, defining *how to map them*. In addition, we exemplify the design by means of the OpenTelemetry [15] running example.

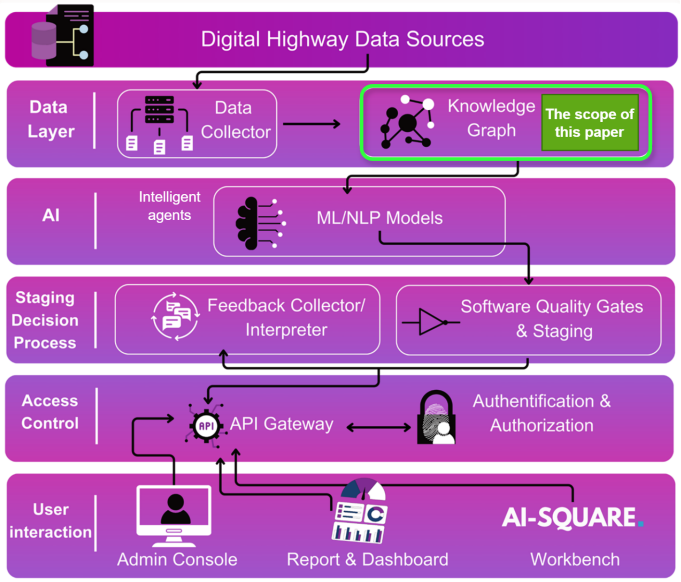

Figure 2: AI-SQUARE Architecture.

The OpenTelemetry application has different microservices related to an online Astronomy shop. In our running example, we assume two stages: i) a testing stage and ii) a production (deployment) stage. The site reliability engineers propose a quality gate to evaluate the transition between testing and production, evaluating the software performance—specifically, based on the desired response time. This quality gate data ranges from the expected performance requirement to observability-related data such as performance metrics and logs.

2.1. Data Sources

Software quality and staging-related data are distributed into structured and unstructured data. we name our data source as *Resources*. Resources represent files with quality data relevant for further analysis, such as metrics, logs, timestamps, containers, pots, tests, and requirements IDs. Using the OpenTelemetry example, we extract the following data: the performance requirements (in CSV format), CPU and memory usage metrics (from Dynatrace), and testing logs of a response time test (from TestKube). We show an excerpt of the data sources in our running example in Figure 3.

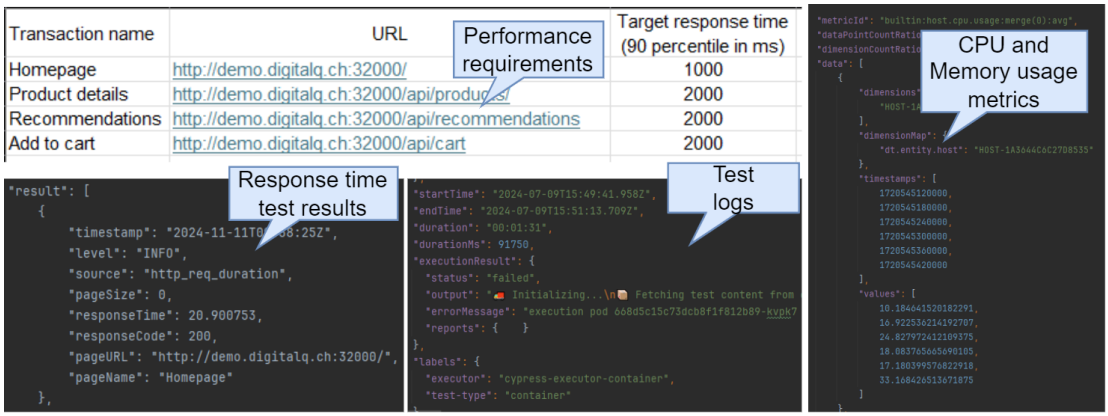

Figure 3: Example of Data Sources.

2.2. The Target Knowledge Graph

The target knowledge graph is a conceptual model (a.k.a. ontology/vocabulary/taxonomy/schema), which uses the *lingua franca* of data consumers [14]. As previously mentioned, software staging data is ubiquitous and (often) interconnected. Staging-related data can come from several phases of the software development lifecycle, from requirements and development to testing. Thus, our target knowledge graph for the AI-SQUARE platform must provide a vocabulary (i.e., *lingua franca*) that adapts, grows, and evolves over time, considering the diversity and context of software development projects. As an initial conceptual model, we consider *abstract* concepts as *Stage*, *Requirement*, *Test*, *Observable (Metrics, Logs)*, *Source*, *File*, and *Anomaly*. These *abstract* concepts are the core of the target AI-SQUARE knowledge graph, allowing the extension of the core vocabulary as needed for each software development project.

For instance, the OpenTelemetry example has a set of microservices in the context of specific stages, sources, requirements, and observable types. In Figure 4, we show the tailored target knowledge graph for the OpenTelemetry example, defining a project-specific vocabulary by extending the core concepts and instances.

These concepts are derived from existing schemas and based on observations from practice but are far from being complete. We acknowledge that further formal alignment (a.k.a. ontology alignment) is needed to generalize and provide soundness to the target knowledge graph core vocabulary—e.g., with existing ontologies such as SEPSES [16] or SEON [17]. We discuss this further in Section 3.2.

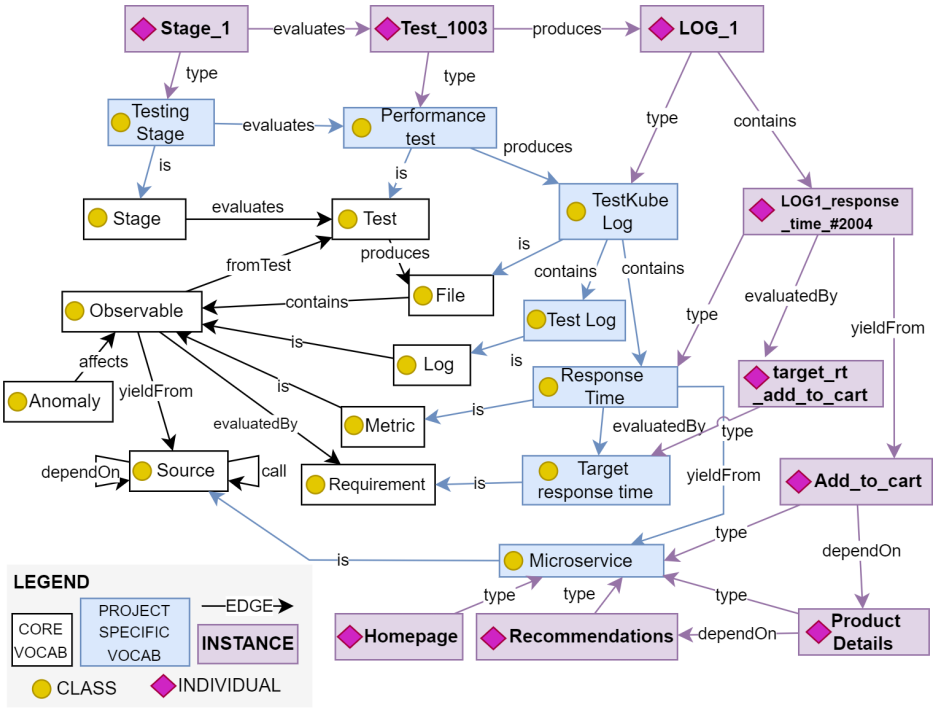

Figure 4: Target knowledge graph exemplified withing the context of our running example.

2.3. Mapping to the Target Knowledge Graph

Mappings are declarative associations between the data source and the target knowledge graph vocabulary. Having defined a vocabulary for our knowledge graph, we must provide a way to map the data sources to the target knowledge graph schema. Thus, we use the RDF Mapping language (RML)—specifically the YAML version, a.k.a. YARRRML [18]. RML is a mapping language that expresses customized mappings from heterogeneous data structures to RDF (Resource Description Framework). Figure 5 shows an example of a mapping following the YARRRML notation, extracting CPU metric triplets—i.e., subject + property (p) + object (o)—to build the target knowledge graph.

```
cpu_metric:
  sources:
    access: on-demand
    referenceFormulation: jsonpath
    iterator: $($.result[?(@.metricId =~ ".*cpu.*")].data[*])
  subjects: ai2apm:$($..resource.id)_CPU_$(@.dimensionMap["dt.entity.host"])
  po:
    - [rdf:type, ai2apm:ME-ISO-PE-000002]
    - [rdf:type, owl:NamedIndividual]
    - [rdf:type, ai2core:Observable]
    - [rdf:type, ai2core:Metric]
    - [ai2core:yieldFrom, 'ai2msa:$(@.dimensionMap["dt.entity.host"])']
```

Figure 5: YARRRML mapping example.

3. The AI-SQUARE Knowledge Graph: The Vision

The AI-SQUARE Knowledge Graph is a work in progress and in constant evolution. Two main data consumers (a.k.a. knowledge workers) benefit from getting answers to the knowledge questions that can be asked to the graph: i) intelligent agents and ii) site reliability engineers. In this Section, we describe how data consumers will benefit, observed challenges and vision for further steps and open research questions.

3.1. Supporting Intelligent Agents and Site Reliability Engineers

At the current stage of the AI-SQUARE Knowledge graph, we observe two potential benefits:

Enhancing intelligent agents with specific and augmented staging data. Some authors have used knowledge graphs in other domains to find vocabularies, automatically reasoning on formalized definitions and axioms, analyse multimodal data, agent collaboration, and improve explainability [19, 20]. Due to the nature of software staging data, intelligent agents at the AI-SQUARE platform can benefit from our knowledge graph by querying specific data, and augmenting it for collaboration among them.

For instance, in the OpenTelemetry example, if an intelligent agent specialises in detecting anomalies in response time metrics by correlating them with a specific target response time requirement, the intelligent agent can query the knowledge graph (by using SPARQL, for example) to gather this data and proceed with the analysis, including augmented data if needed—e.g., other requirements, sources, or anomalies—providing a context-aware data to the analysis (see Figure 6).

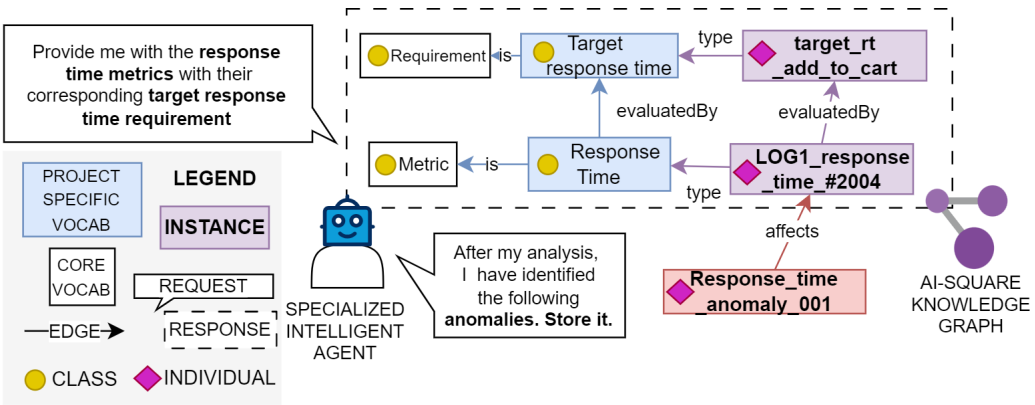

Figure 6: Example of how intelligent agents can benefit from the AI-SQUARE knowledge graph.

Supporting staging decision and explainability. The AI-SQUARE knowledge graph stores the dependencies among ubiquitous and interconnected staging data. When anomalies are detected by

humans or intelligent agents, the AI-SQUARE knowledge graph holds the potential to build a causality graph and explain the occurrence of anomalies to support a staging decision. Some authors [6, 7, 8] have used knowledge graphs for root cause analysis and identifying the root of bugs, errors, or exceptions. We foreseen root cause analysis to complement results from intelligent agents such as large-language model agents when explaining the staging decision after analysis. This extra graph-based data will provide site reliability engineers with a tool to verify intelligent agents' outcomes. For instance, in our OpenTelemetry example, after intelligent agents analyse the response time, they report anomalies in a microservice, suggesting site reliability engineers to not stage from the testing to the production stage. Then, site reliability engineers can benefit from the dependencies stored in our knowledge graph and root cause analysis to justify and explain this decision, finding, for example, that the anomalies in response time are due to a specific requirement, microservice, or observable type (see Figure 7).

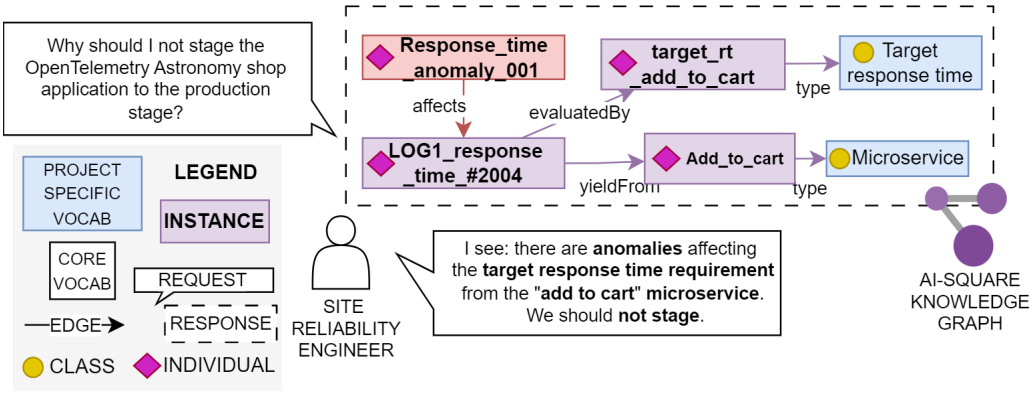

Figure 7: Example of how site reliability engineers can benefit from the AI-SQUARE knowledge graph.

3.2. Challenges, Open Research Questions and Next Steps

With the current design and implementation of the AI-SQUARE knowledge graph, we have identified the following challenges and open research questions:

Can we provide dependency/causality among *all* ubiquitous data involved in software staging using the AI-SQUARE knowledge graph? As mentioned in Section 2.2, our target knowledge graph must evolve, grow, and adapt to specific software development projects, specifically in their staging process. This implies that the AI-SQUARE knowledge graph requires flexibility to represent dependencies and causality between ubiquitous data. Therefore, research questions arise: how adaptable the AI-SQUARE knowledge graph is? Is it feasible to include existing vocabularies/ontologies from literature? What are the missing core concepts that allow the knowledge graph to be flexible enough to represent all ubiquitous data? can the AI-SQUARE knowledge graph support finding dependencies/-causalities from earlier software development lifecycle phases till the staging decision—e.g., finding dependencies from software requirements to staging quality gate results? These questions motivate further research on the AI-SQUARE Knowledge graph vocabulary.

Can we measure software staging maturity based on data stored in the AI-SQUARE knowledge graph? Some authors have proposed maturity models in DevOps environments to assess their CICD pipelines' maturity [21]. Maturity models are extensive bodies of knowledge that require certain data, stages, processes, and practices to be considered to guarantee a certain maturity level in the software development process. Considering that the AI-SQUARE knowledge graph stores dependencies from the software staging and lifecycle, research questions arise: can the dependencies and entities stored in the AI-SQUARE knowledge graph serve as input to assess the software staging maturity? What would require a maturity model to assess the staging process? What are the required observables, stages, requirements, and tests (among other concepts) to be included in software development staging to assess its maturity? These open research questions motivate further research on how the AI-SQUARE knowledge graph can be used in maturity assessment for software staging.

What are the effects for both intelligent agents and site reliability engineers when using the knowledge graph in the AI-SQUARE platform? The AI-SQUARE knowledge graph currently is under design and implementation, being actively developed in the context of the AI-SQUARE platform. Nevertheless, the effect of using or not using the AI-SQUARE knowledge graph to support staging decisions in the AI-SQUARE platform is still an unaddressed question. Thus, we plan to conduct empirical efforts—a.k.a., treatment validation cycles in Design Science [22]—following empirical methods such as quasi-experiments, focus groups, and case studies [23] to understand the effect of the AI-SQUARE knowledge graph when supporting a staging decision—including effectiveness, efficiency, and satisfaction [24]. Therefore, we can measure and conclude about how site reliability engineers perform staging decisions with and without the support of the AI-SQUARE knowledge graph.

4. Conclusions

In today's fast-paced software development market, the demand for rapid and frequent releases poses significant challenges for site reliability engineers. They are tasked with ensuring software quality throughout the continuous integration and continuous deployment (CI/CD) pipeline, encompassing development, testing, and production stages. The complexity of modern software staging pipelines introduces high volumes of ubiquitous data in all phases of the software development lifecycle, making it challenging to make a staging decision. This observed complexity from industry has inspired the Swiss Digital Network and three research institutions to propose the AI-SQUARE platform: an intelligent decision-support platform for end-to-end software staging management.

In this poster & tools paper, we shared our progress on designing a knowledge graph that supports the AI-SQUARE platform. We exemplified our design with the OpenTelemetry example in the context of staging from the testing environment to production. Moreover, we illustrated and provided our *vision* on how the AI-SQUARE knowledge graph can benefit intelligent agents and site reliability engineers by enhancing the software quality staging data and providing root cause analysis for explainability of staging decisions.

The AI-SQUARE knowledge graph is under constant development, which means there are research and implementation challenges to be addressed. We reflected on the vision steps to answer the research questions. We expect the AI-SQUARE platform will foster software's rapid evolution and requirements validation with continuous deployment while meeting required quality standards.

Acknowledgments

We sincerely appreciate the valuable contributions of the active members of the AI-SQUARE project. Their feedback, ideas, and implementations across other layers of the AI-SQUARE platform have been a valuable contribution to shaping our research. Moreover, we would like to extend our sincere gratitude to the anonymous reviewers for their valuable insights and constructive feedback. This project is fully funded by the Innosuisse AI-SQUARE project and the SDN.

References

[1] OpenTelemetry, Documentation, https://opentelemetry.io/docs/, 2024. [Accessed 07-02-2025].
[2] Dynatrace, Documentation, https://docs.dynatrace.com/docs, 2025. [Accessed 07-02-2025].
[3] Prometheus, Overview, https://prometheus.io/docs, 2025. [Accessed 07-02-2025].
[4] Jaeger, Documentation, https://www.jaegertracing.io/docs/1.18/, 2025. [Accessed 07-02-2025].
[5] I. Kumara, G. Quattrocchi, D. Tamburri, W.-J. Van Den Heuvel, Quality assurance of heterogeneous applications: The SODALITE approach, in: Advances in Service-Oriented and Cloud Computing, 2021, p. 173–178.
[6] J. Qiu, Q. Du, K. Yin, S.-L. Zhang, C. Qian, A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications, Applied Sciences 10 (2020).

[7] H. Wang, Z. Wu, H. Jiang, Y. Huang, J. Wang, S. Kopru, T. Xie, Groot: An event-graph-based approach for root cause analysis in industrial settings, in: 36th International Conference on Automated Software Engineering, 2021, p. 419–429.
[8] T. Wang, G. Qi, T. Wu, Kgroot: A knowledge graph-enhanced method for root cause analysis, Expert Systems with Applications 255 (2024) 124679.
[9] Dynatrace, Grail, https://www.dynatrace.com/platform/grail/, 2025. [Accessed 07-02-2025].
[10] SDN, Swiss Digital Network home page, https://swiss-digital-network.ch/, 2025. [Accessed 07-02-2025].
[11] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic Web 8 (2016) 489–508.
[12] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM. 57 (2014) 78–85.
[13] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, DBpedia: A nucleus for a web of open data, in: The Semantic Web. ISWC ASWC, 2007, pp. 722–735.
[14] J. Sequeda, O. Lassila, Designing Enterprise Knowledge Graphs, Springer International Publishing, 2021, p. 19–44.
[15] OpenTelemetry, Astronomy Shop Demo Application, https://github.com/open-telemetry/opentelemetry-demo, 2025. [Accessed 07-02-2025].
[16] E. Kiesling, A. Ekelhart, K. Kurniawan, F. Ekaputra, The sepses knowledge graph: An integrated resource for cybersecurity, in: The Semantic Web. ISWC, 2019, pp. 198–214.
[17] M. Würsch, G. Ghezzi, M. Hert, G. Reif, H. C. Gall, Seon: a pyramid of ontologies for software evolution and its applications, Computing 94 (2012) 857–885.
[18] P. Heyvaert, B. De Meester, A. Dimou, R. Verborgh, Declarative Rules for Linked Data Generation at your Fingertips!, in: Proceedings of the 15th ESWC: Posters and Demos, 2018, p. 213–217.
[19] R. Hoehndorf, P. N. Schofield, G. V. Gkoutos, The role of ontologies in biological and biomedical research: a functional perspective, Briefings in Bioinformatics 16 (2015) 1069–1080.
[20] I. Tiddi, S. Schlobach, Knowledge graphs as tools for explainable machine learning: A survey, Artificial Intelligence 302 (2022) 103627.
[21] M. Zarour*, N. Alhammad, M. Alenezi, K. Alsarayrah, A research on devops maturity models, International Journal of Recent Technology and Engineering (IJRTE) 8 (2019) 4854–4862.
[22] R. J. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer Berlin Heidelberg, 2014.
[23] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, SpringerLink, 2012.
[24] D. L. Moody, The method evaluation model: A theoretical model for validating information systems design methods, in: ECIS, 2003, pp. 79–96.