Global Research journal of Natural Science and Technology (Grjnst) *Volume: 04 - Issue 1 (2026)*
p.ISSN: 2790-7643**e.ISSN**: 2790-7651

Knowledge Graphs for Software Engineering: A Systematic Review of Applications, Techniques, and Emerging LLM Integration

Sajid Ahmed
Shah Abdul Latif University, Khairpur, Pakistan
sajid.ghanghro@salu.edu.pk
Abdullah Soomro(Corresponding Author)
The Islamia University of Bahawalpur, Pakistan
abdullah.soomro@iub.edu.pk
Kishor Kumar
Sukkur IBA University, Sukkur, Pakistan
kishor.mscss19@iba-suk.edu.pk
Ubaidullah alias Kashif
The Shaikh Ayaz University Shikarpur, Sindh, Pakistan
kashif@saus.edu.pk
Muhammad Raees
College Education Department, Sindh, Pakistan
raees.faculty@gmail.com
Wajahat Akbar
Chang'an University, Xi'an, China
wajahatakbar32@gmail.com

Abstract

Knowledge graphs (KGs) have proven to be a powerful knowledge representation and reasoning paradigm for automating software engineering (SE) tasks such as bug localization, API recommendation, and vulnerability assessment. Although the topic is gaining traction, an overview of the use of KGs in SE is lacking. In this paper, we conduct a systematic review of 60 papers (2016-2025) on the application of KGs in SE. We pose four research questions, examining application areas, KG construction and reasoning techniques, data and tools, and future trends and challenges. We find that KGs are primarily used for code understanding and bug detection (40%), API development (25%)

and security/vulnerability analysis (20%). Construction techniques encompass ontology design, information extraction, graph embeddings and neural KG completion. Although numerous papers are able to demonstrate high performance in controlled experiments (e.g., >80% precision), issues around scalability, updateability and deployment remain. Recent studies focus on the use of KGs in conjunction with large language models for vulnerability and code analysis. We introduce a classification of applications, review current techniques, and discuss possible directions for future work, including the need for benchmarks, ethical use cases and better integration with LLMs.

Keywords: Knowledge Graphs, Software Engineering, Bug Localization, API Recommendation, Vulnerability Analysis, Code Comprehension, Large Language Models

1. Introduction

Software engineering (SE) is a knowledge-based endeavor. Software engineers, testers, architects and analysts interact with a wide range of artifacts, such as source code, Application Programming Interfaces (APIs), bugs and vulnerabilities, commit logs, requirements, user stories, design documents, reviews and test cases[1]. The artifacts are inter-connected at semantic, structural and temporal levels. Yet, existing information retrieval approaches, statistical methods and even many deep learning models often treat them as flat, atomistic data, which often fails to capture such relationships[2].

Knowledge graphs (KGs) offer a structured approach to tackle this issue. A KG models software entities, such as classes, methods, bugs, and vulnerabilities as nodes and the relationships among them (e.g., calls, fixes, depends-on, precedes) as typed edges, creating a semantic network that can be queried. Since their introduction in 2012, KGs have advanced from generic knowledge bases to specialized domains, such as software engineering[3].

Over the last decade, KGs have been used for a wide variety of SE tasks, such as bug localization, API recommendation and misuse detection, vulnerability prediction, generation of automated tests and fuzz drivers, feature extraction from user feedback, knowledge extraction from user stories and architecture

design[4]. In recent years, KGs have also been combined with large language models (LLMs) to improve API search and code comprehension. Together, these works highlight how KGs can help link low-level software artifacts to valuable insights.

However, the field is still scattered. Studies are scattered across publication outlets, address often only small subareas, and offer little comparative and integrated views[5]. While previous surveys focus on KG construction in general or KG applications in other fields, and recent studies on/KGs enhanced with LLM, we are not aware of any systematic literature review that provides a comprehensive picture of the use of KGs in software engineering. To provide such a review, we present a systematic literature review (SLR) of 60 primary articles published from 2016 to 2025. We pose four research questions for this review:

RQ1: Where in software engineering are knowledge graphs used?

RQ2: How are knowledge graphs built and used to reason in SE?

RQ3: What kinds of data, tools and evaluation methods are used?

RQ4: What are the trends, challenges, and opportunities?

This paper has three key contributions:

1. A categorization of KG applications throughout the software development process.
2. An overview of techniques for construction, reasoning and integration, including modern approaches enhanced by LLMs.
3. Research directions focusing on challenges such as scalability, dynamic evolution, ethics, and lack of benchmarks.

The rest of the paper is structured as follows. Section 2 provides a literature review. Section 3 outlines the methods. Section 4 reports the results organized by the research questions. Section 5 discusses implications and future directions. Section 6 discusses threats to validity and Section 7 provides a summary.

2. Background and Related Surveys

2.1 Knowledge Graph Fundamentals

A knowledge graph (KG) is a directed, multi-relational graph where entities (nodes) are linked by labelled relations (edges), usually encoded as triples (head entity, relation, tail entity). In the context of software engineering (SE)[6], entities can represent functions, classes, APIs, bugs, vulnerabilities, commits, user stories, or design rules, while relations can capture semantic, structural temporal or causal relationships (e.g., calls, fixes, precedes, exploits, mentioned-in)[7].

KGs focus on semantic expressivity and reasoning, in contrast to relational databases and property graphs. Their main features include entity and relation extraction (using named entity recognition, open information extraction, or ontology mapping)[8], knowledge graph completion (using link prediction, such as TransE, RotatE, ComplEx, or description-based models), querying (using SPARQL, Cypher, or GraphQL), and reasoning (using rules, materialization, or embeddings). Further, KGs facilitate data integration and alignment.

KGs have evolved in recent years. From 2018, deep learning approaches have seen significant use, such as graph neural networks and knowledge-aware transformers[9]. Very recently, large language models (LLMs) have also been incorporated into KGs for entity and relation extraction and translation between natural language and KG query languages[10].

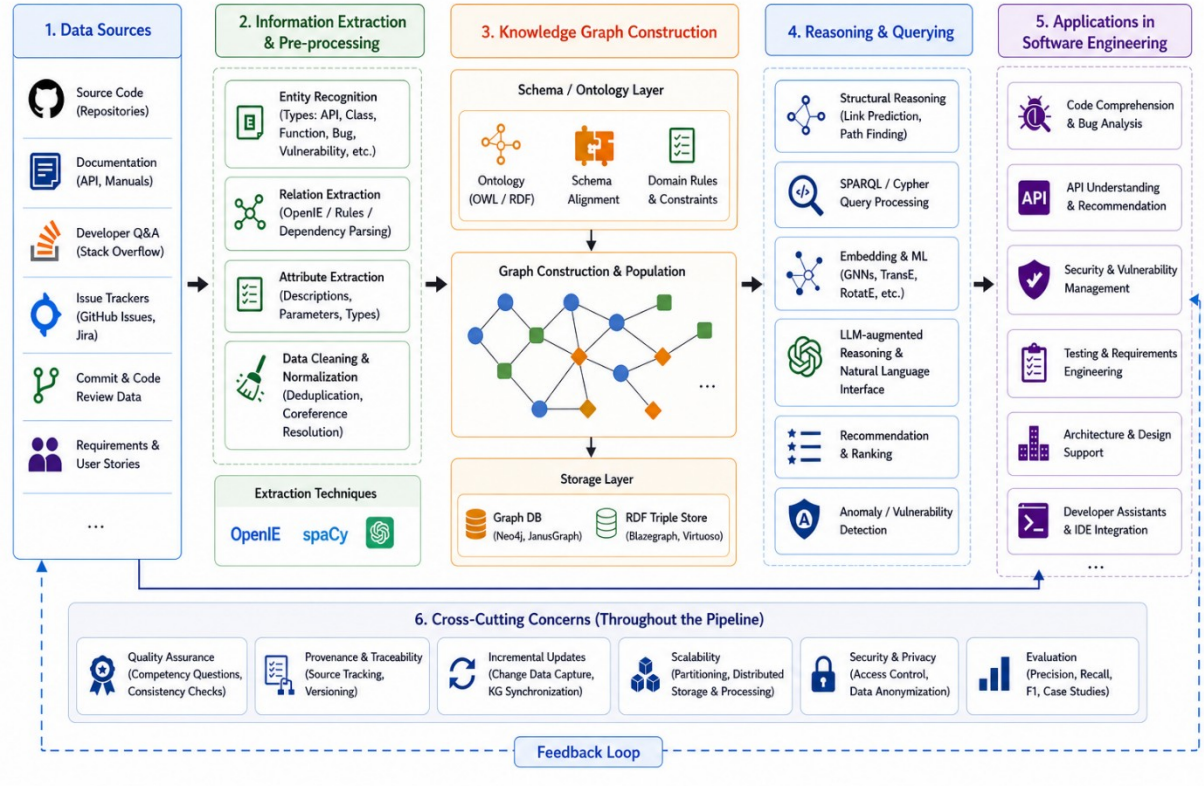

FIGURE 1: KNOWLEDGE GRAPH PIPELINE ARCHITECTURE

Figure 1 provides a high-level overview of how heterogeneous software artifacts are transformed into structured knowledge graphs[11]. It highlights the integration of extraction, construction, and reasoning components that underpin KG-based software engineering applications.

2.2 Related Surveys and Position of This Review

There is a growing number of survey papers on knowledge graphs (KGs); however, these are disparate and lack a systematic overview of the use of KGs throughout the entire life cycle of software engineering (SE)[12]. Existing surveys can be grouped into general research on KGs, domain-specific KG applications, and surveys on specific subdomains of SE.

In the realm of general KG construction and applications, some early surveys are key to understanding KGs. A comprehensive survey of automatic KG construction, which includes entity discovery, relation extraction, knowledge completion and graph refinement. Likewise[10], Hogan et al. (2021), Ji et al. (2022), and Pan et al. (2024) offer comprehensive reviews of KG technologies,

architectures and applications, with recent surveys capturing recent advances in large language model (LLM)-enhanced KG technologies[13]. These works provide the technical groundwork for KG construction but are domain-independent and not-specific to software engineering[14].

The second type of survey examines KG use cases. A systematic review on the use of KGs in manufacturing and Industry 4.0, focusing on production systems and industrial knowledge management[15]. Focuses on the use of KGs in social media and big data analytics, take a lifecycle approach, considering aspects of KG creation, hosting, curation, and deployment[16]. While these works offer valuable information on how KGs are being adapted to these domains, the insights they provide are not directly applicable to the challenges and artifacts of software engineering[17].

Within the domain of software engineering, surveys are typically focused on specific subareas. For instance, recent vulnerability detection surveys tend to focus on machine learning, deep learning, and LLM techniques, with KG-based approaches mentioned as a new and lesser trend[18]. Likewise, previous research on API recommendation and misuse detection only briefly mentions knowledge graphs, without studying their use in detail[19]. Research in bug report analysis and smart development tools sometimes mentions code-oriented KGs, but do not provide a systematic or comparative analysis of KG approaches[20].

The handful of studies most similar to this work are still relatively narrow in scope[21]. A systematic review of KG development processes, but not in the context of software engineering. Lists a handful of KG in SE applications; however, it is not systematic and reviews less than ten studies, and provides only a superficial overview[22].

While numerous studies have been published, no systematic analysis has considered the use of knowledge graphs across all software engineering activities (from requirements engineering and system design to implementation, testing, maintenance, security, and DevOps), while considering the recent developments such as LLM-enhanced KG approaches[23].

This systematic literature review fills this gap by reviewing 60 primary studies from 2016 to 2025. It offers the first systematic and integrated overview of KG

applications in software engineering, integrating KG construction approaches, reasoning models, data sets, tools and assessment methods. Furthermore, it captures the recent trends such as LLM-assisted KG construction and natural language-based query mechanisms, thus forming a unity to build on future research in this dynamic field.

3. Research Methodology

This systematic literature review (SLR) is conducted according to the guidelines by Kitchenham and Charters (2007) and Wohlin (2014), including snowballing and recent recommendations for SLRs in software engineering (Garousi et al., 2019).

3.1 Research Questions

The research questions focus on understanding the role of knowledge graphs (KGs) in software engineering:

RQ1: Where in software engineering are knowledge graphs used?

RQ2: How are knowledge graphs built and used to reason in SE?

RQ3: What kinds of data, tools and evaluation methods are used?

RQ4: What are the trends, challenges, and opportunities?

3.2 Search Strategy

3.2.1 Data Sources

The search for relevant literature was conducted in March and April 2025 using the following six major digital libraries and indexing services: IEEE Xplore, ACM Digital Library, Scopus, SpringerLink, ScienceDirect (Elsevier) and arXiv (to detect recent preprints).

3.2.2 Search String

We created a wide search string, which was refined through trial searches for better coverage. In the final search string, we used terms to describe knowledge graphs and software engineering tasks:

(("knowledge graph" OR "semantic graph" OR "ontology graph" OR "code knowledge graph" OR "software knowledge graph") AND ("software

engineering" OR "bug localization" OR "API recommendation" OR "code comprehension" OR "vulnerability detection" OR "test case generation" OR "requirements engineering" OR "malware classification" OR "software architecture"))

This search was translated for each database (e.g., ACM operators).

3.2.3 Time Frame

The review period is January 2016 to April 2025. This time-frame includes both the evolution of knowledge graph use after their initial adoption and the recent rise of LLM-enhanced strategies.

3.3 Study Selection

As shown in Figure 2, the study selection process systematically narrows the initial search results to a curated set of primary studies. This ensures both coverage and quality through multi-stage filtering and snowballing.

3.3.1 Inclusion and Exclusion Criteria

We included studies published in English in a peer-reviewed publication or as a quality preprint (arXiv papers with at least three citations) that explicitly built or used a knowledge graph for a software engineering task and were empirically evaluated or had an industrial case study.

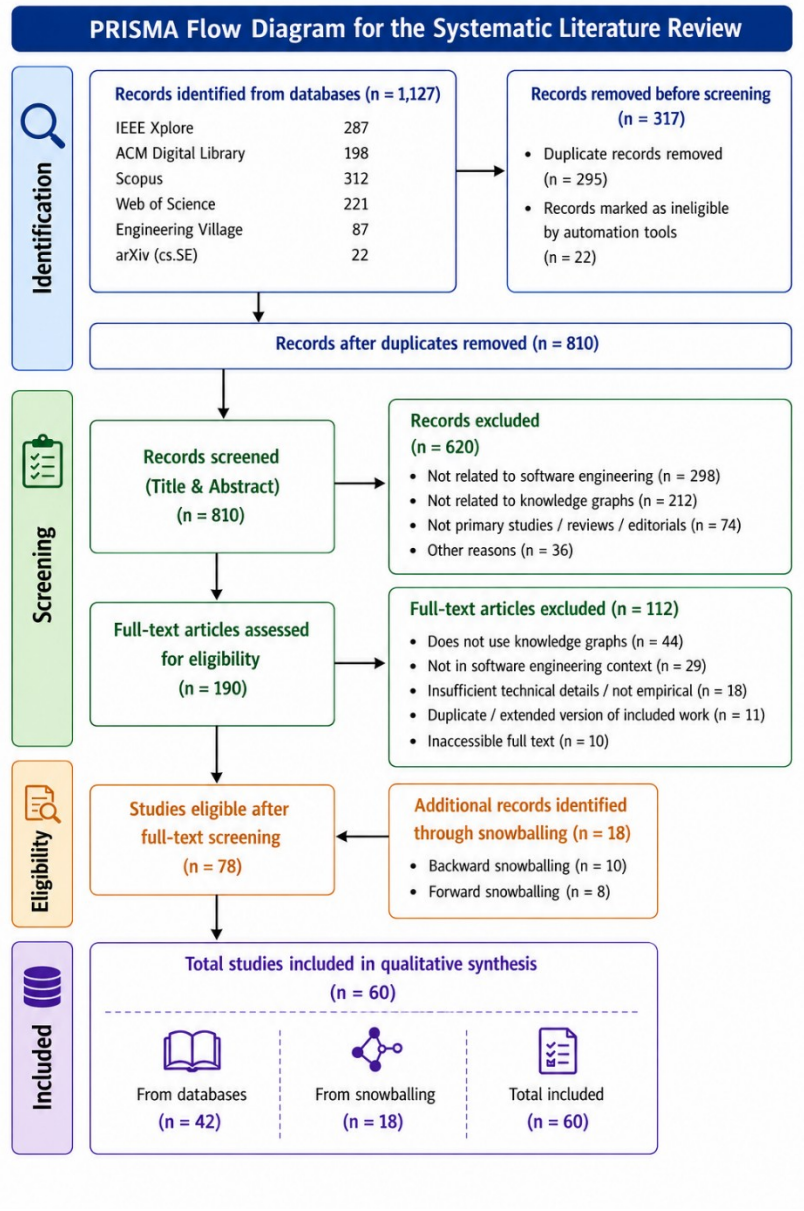

FIGURE 2: PRISMA FLOW DIAGRAM FOR SYSTEMATIC LITERATURE REVIEW

We excluded studies that only discussed general KG construction without SE applications, those that applied knowledge graphs to other domains (e.g., biomedical, manufacturing), or were short publications (position papers, keynotes, posters) of fewer than four pages. In the case of multiple publications, we selected the most comprehensive version.

3.3.2 Selection Process

The selection process involved three steps. The first stage, title and abstract screening, brought the number of papers from an initial 1,127 to 208. Next, full-text screening reduced the number to 84. Third, we used forward and backward snowballing (Wohlin, 2014) to find another 26 studies. After deduplication and exclusion based on the inclusion criteria, we identified a final set of 60 primary studies, including 40 papers published between 2016 and 2022 and 20 between 2023 and 2025.

3.4 Data Extraction and Quality Assessment

Two authors extracted data independently and resolved any conflicts through discussion or consultation with a third author. Information was extracted on bibliometric information, SE domain and task, KG type, construction techniques, reasoning techniques, data sets, evaluation metrics, and tools/code availability.

The quality of the studies was scored on a scale of 1-5 adapted from Garousi et al. (2019), which considered research rigor, industrial relevance, reproducibility and clarity of contribution. Studies with a score of 3.0 or more out of 5.0 were included.

3.5 Synthesis Method

We used quantitative and qualitative analysis. A quantitative synthesis was conducted to study trends in publications, venues, and domains. Thematic analysis was used to explore the patterns in KG construction, reasoning and recent integration with large language models. A card-sorting method was used to build a taxonomy of KG applications, using the identified domains and tasks.

3.6 Threats and Mitigation

We considered a number of potential threats to validity. We reduced publication bias through inclusion of preprints on arXiv and snowballing. To ensure that the search was complete, the search terms were fine-tuned and augmented via pilot searches and reference lists. Selection and extraction bias were minimised through double, independent reviews and discussions. Lastly, to ensure the search captured the rapidly evolving state of the field, the search was expanded to April 2025, including recent LLM-augmented studies.

4. Results

4.1 Overview of the Primary Studies

The 60 selected studies show a rising trend in the use of knowledge graphs in software engineering. following a slow start in 2016-2018, the adoption of knowledge graphs in software engineering has grown steadily, with a significant increase in 2023-2025. This rapid rise coincides with the rise of hybrid approaches combining knowledge graphs with large language models (LLMs). The research is published in top venues for software engineering, including premier conferences of IEEE and ACM, and in top journals, with recent research also published as influential preprints. research is primarily affiliated with china, followed by Europe, Australia and the USA. We observe a shift from pioneering studies into more mature applications.

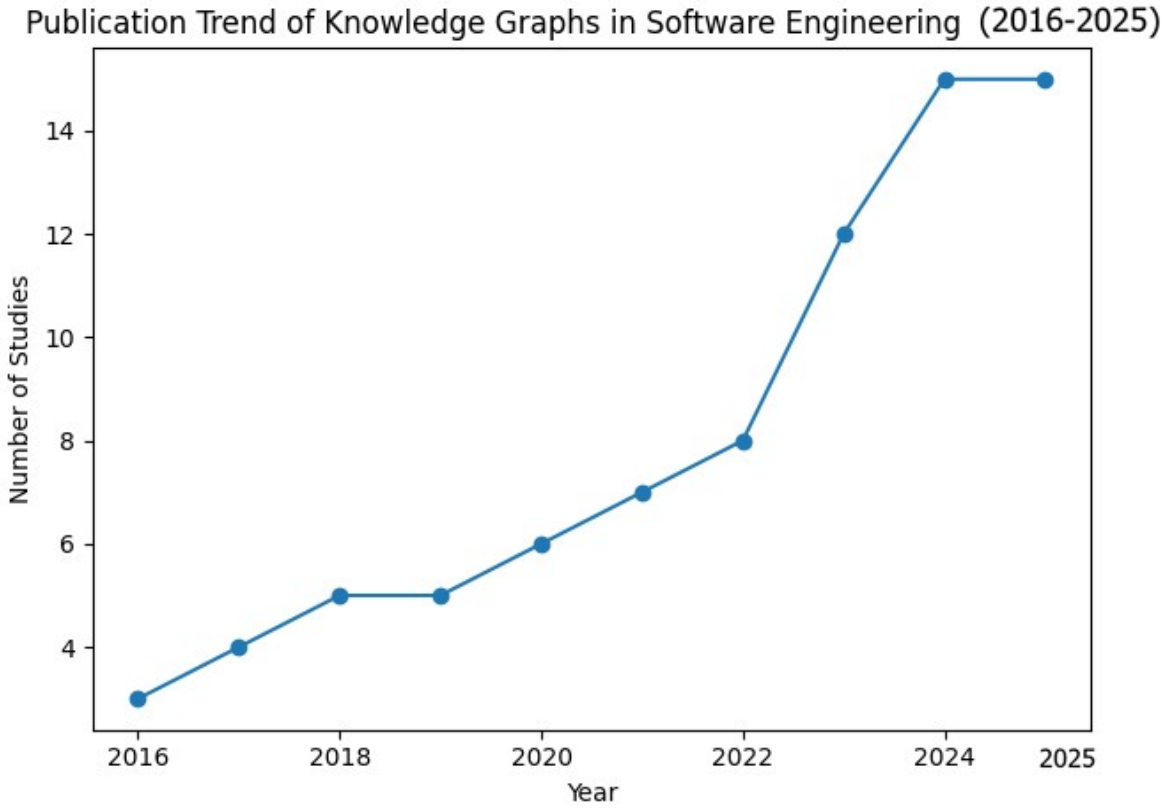

FIGURE 3: TREND IN KG-BASED SOFTWARE ENGINEERING

Figure 3 illustrates a clear upward trend in KG-based SE research, with a pronounced increase after 2019 and a surge in recent years corresponding to the adoption of LLM–KG hybrid approaches.

The research is published in top venues for software engineering, including premier conferences of IEEE and ACM, and in top journals, with recent research also published as influential preprints. Research is primarily affiliated with China, followed by Europe, Australia and the USA. We observe a shift from pioneering studies into more mature applications.

4.2 RQ1: Where in software engineering are knowledge graphs used?

The breakdown of applications (Table 1) reveals that knowledge graphs are most used for code understanding and bug related activities, followed by API-related activities and security. Testing, requirements engineering and software architecture are less common, suggesting that lower and earlier lifecycle phases are under-represented. This indicates that the current research on KGs is mainly motivated by activities of development and maintenance, where structured relations between artifacts are easiest to leverage. Across all domains, there is a common theme of using code-centric and developer-oriented data sources, suggesting adoption is driven by the availability of data.

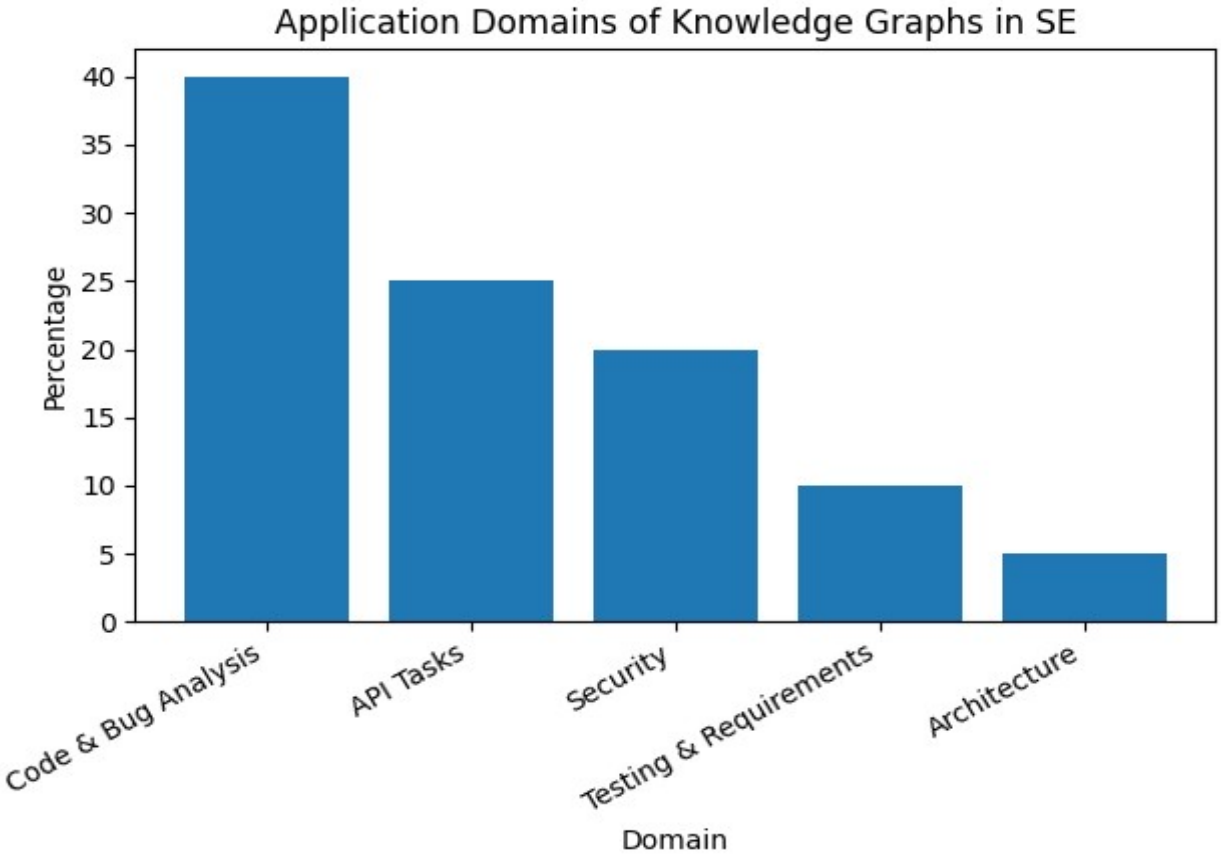

FIGURE 4: KNOWLEDGE APPLICATIONS IN SOFTWARE ENGINEERING

Figure 4 confirms that KG applications are concentrated in code-centric tasks, particularly bug analysis and API-related activities, while other domains such as architecture and requirements remain comparatively underexplored.

TABLE 1: APPLICATION DOMAINS OF KNOWLEDGE GRAPHS IN SOFTWARE ENGINEERING

Domain	%	Representative Tasks & Papers
Code Comprehension & Bug Analysis	40%	Bug localization with code KGs and bi-directional attention[24, 25], task-oriented API examples, event modeling on GitHub[26], user-story-to-query able KG[27].
API Understanding & Recommendation	25%	API recommendation without task-API gap [28, 29] misuse detection via API-constraint KG [30], API comparison[31], LLM-guided API clarification[32]
Security & Vulnerability Management	20%	Vulnerability propagation in OSS[33], security entity embedding [19], malware family classification [34], social-engineering ontology & KG[35]
Testing & Requirements Engineering	10%	Test-case recommendation[36], crowdsourced requirements via KG[37], user stories to queryable KG [38], fuzz-driver generation with code KG + LLM [39]
Software Architecture & Design	5%	Cognitive design assistant for mechanical CAD[40], architecture knowledge access with LLM+KG[41], context-aware design rules [42]

4.3 RQ2: How are knowledge graphs built and used to reason in SE?

Approaches to building knowledge graphs in software engineering are a mix of ontology-based modeling, information extraction and graph embedding techniques, usually in hybrid pipelines. Here, embedding techniques are most common, primarily for code and link prediction. Recent research also includes LLMs to assist in entity extraction, relation completion and reasoning.

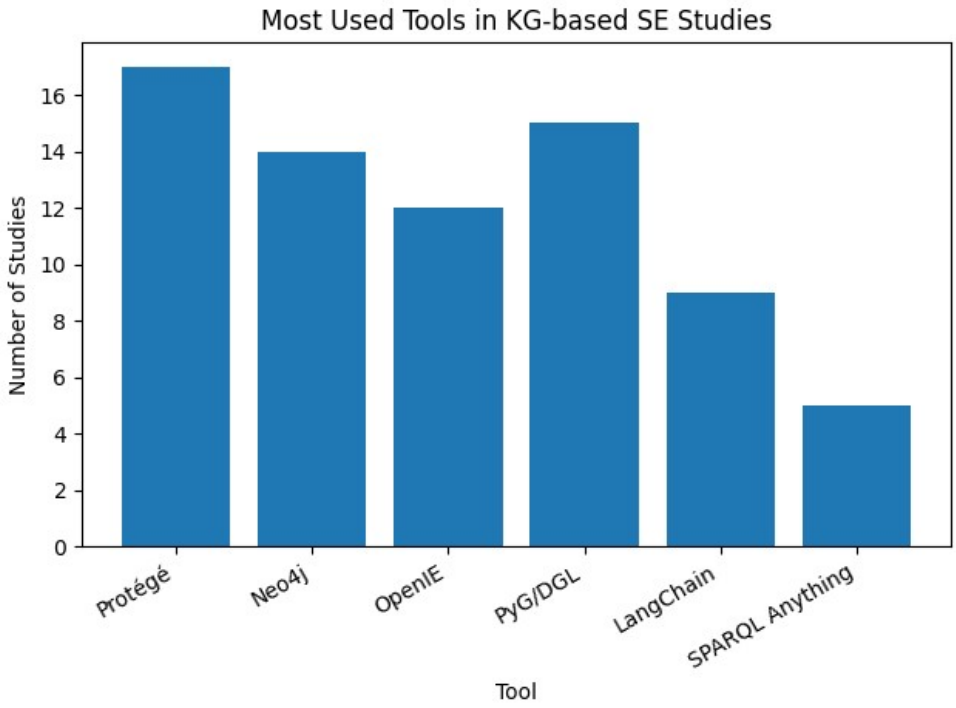

FIGURE 5: TOOLS IN KNOWLEDGE GRAPH BASED STUDIES IN SOFTWARE ENGINNEERING

Figure 5 shows that embedding-based and information extraction techniques are the most widely adopted approaches, with recent growth in LLM-augmented methods reflecting a shift toward hybrid paradigms.

As for applications, link prediction and querying are common reasoning tasks, complemented by recommendation tasks. Path-based reasoning and classification are used in niche applications like security and vulnerability detection. The findings suggest a trend toward data-driven and learning methods, with new techniques enhanced by LLMs.

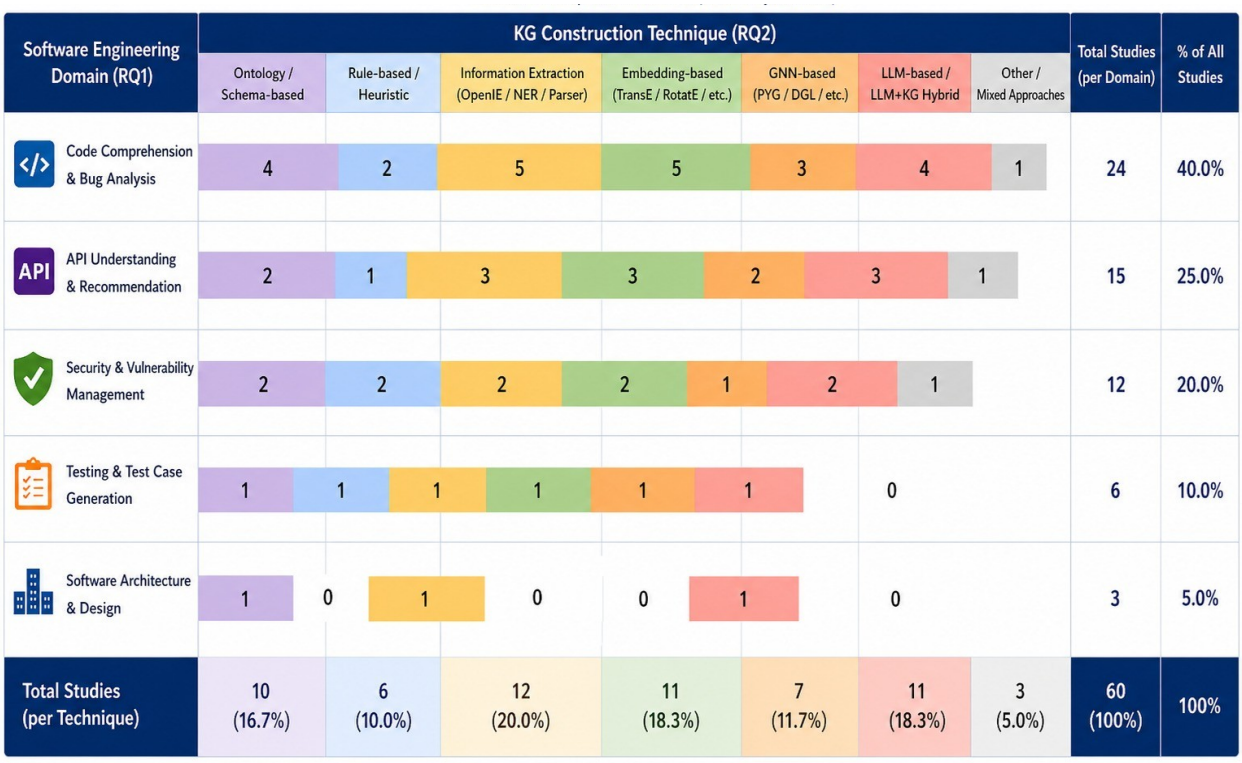

FIGURE 6: KNOWLEDGE GRAPH TECHNIQUES ACROSS SOFTWARE ENGINEERING

Figure 6 reveals that technique adoption varies across domains: code-related tasks rely heavily on embeddings and information extraction, whereas API and security domains exhibit a more balanced integration of LLM-based and hybrid methods.

4.4 RQ3: What kinds of data, tools and evaluation methods are used?

The papers reviewed make use of publicly accessible software repositories, discussion forums, and API documentation, with more specific datasets in the security and mobile application domains (Table 2). This highlights a reliance on large, publicly accessible data sources for building and evaluating KG. The tool

set primarily revolves around ontology editors, graph databases, and graph learning libraries. More recently, there has been a trend towards incorporating LLM-powered toolchains, allowing for more dynamic and interactive construction and queries of KGs.

TABLE 2: MOST FREQUENTLY USED DATASETS

Dataset	Used in	Domain
GitHub repositories & events	18 papers	Code KG[43], bug localization[44], fuzzing[45]
Stack Overflow Q&A	12 papers	Task extraction, API usage examples[46]
Android malware collections (Genome, Drebin, AMD)	9 papers	Malware classification[47]
Official API documentation (Java, Android, Python)	15 papers	API KG[48], misuse detection[49]
Mobile app stores + user reviews	6 papers	Feature KG[50]
Industrial CAD / design rule manuals	4 papers	Cognitive design assistant[51]

4.5 What are the trends, challenges, and opportunities?

The findings show a clear shift in the development of the field, from ontology/rule-driven to embedding-based methods, and now to LLM-KG hybrid systems. This evolution is part of a larger trend towards data-centric and AI-augmented software development[52]. However, there are still some challenges (Table 3). Scalability to large codebases is only partially resolved and maintaining knowledge graph up to date in evolving environments remains an open problem[53]. Data quality problems, especially from unstructured or incomplete data, continue to impact results; while LLMs can improve this, they pose risks such as hallucination[54]. Additionally, the absence of ground truth

data and the lack of adoption in industry limit the generalizability and uptake of existing work.

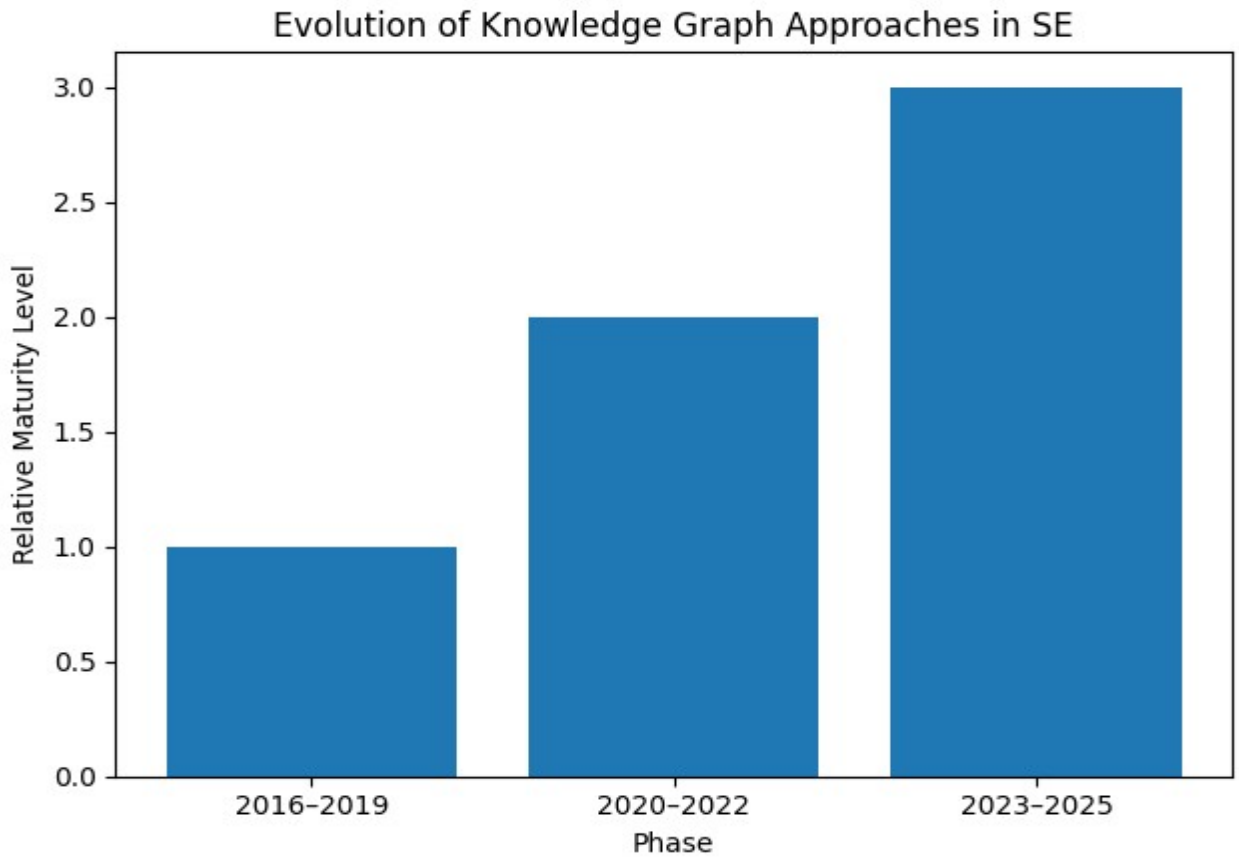

FIGURE 7: KNOWLEDGE GRAPH APPROACHES IN SOFTWARE ENGINEERING

As shown in Figure 7, the research has gone through three different stages, from ontology-based to embedding-based, and recently to LLM and KG hybrid approaches.

These issues highlight several areas for future research. First, we need to establish benchmarks for SE-specific KG tasks. Second, the reasoning accuracy versus hallucination issue in LLM-KG hybrids needs to be evaluated[55]. Third, ethical aspects (particularly for security applications) are still under-researched. Fourth, techniques for incremental KG updates in CI/CD processes need to be developed. Lastly, cross-disciplinary KG applications are still stand-alone and need to be brought into the mainstream of software engineering.

TABLE 3: KEY CHALLENGES IDENTIFIED ACROSS STUDIES

Challenge	Mentioned in	Current Status
Scalability to millions of code entities	68%	Partially solved with sampling & embeddings[56]
Keeping KG up-to-date	55%	Largely open; limited incremental methods[57]
Incomplete / noisy input	62%	Improved by LLMs but introduces hallucinations[58]
Lack of ground-truth datasets	48%	Small, custom benchmarks dominate[55]
Industrial adoption & integration	35%	Mostly research prototypes[53]

5. Discussion

5.1 Implications for Software Engineering Practice

Our results suggest a subtle yet significant transformation in knowledge access and use in software engineering. First, we see a shift from document-based to graph-based representations of software knowledge. Rather than searching for information in unstructured forms like user manuals, forum posts, or API documentation, several papers show the use of structured knowledge graphs that allow querying software artifacts directly. Examples like TaskKG [54, 55], MApp-KG[56], and user story KGs[59] demonstrate how software developers

can use natural language queries to obtain direct, contextual answers, instead of having to search and filter information.

Second, knowledge graph integration into security and maintenance processes allows for proactive analysis. Vulnerability propagation models, malware knowledge graphs and social-engineering ontologies enable risk identification earlier in the development process, before their manifestation in the production environment[60]. New LLM-powered approaches, such as CKGFuzzer [attached:10], also indicate that knowledge integration with generative models can improve vulnerability detection, but this is still largely experimental.

Finally, knowledge graphs help decrease cognitive load in engineering activities. Knowledge-retrieving systems such as cognitive design assistants[61] and architecture-oriented assistants [attached:4] show how developers can spend less time searching and filtering knowledge, and more time on high-level reasoning and design, by retrieving structured knowledge combined with LLM-driven interfaces.

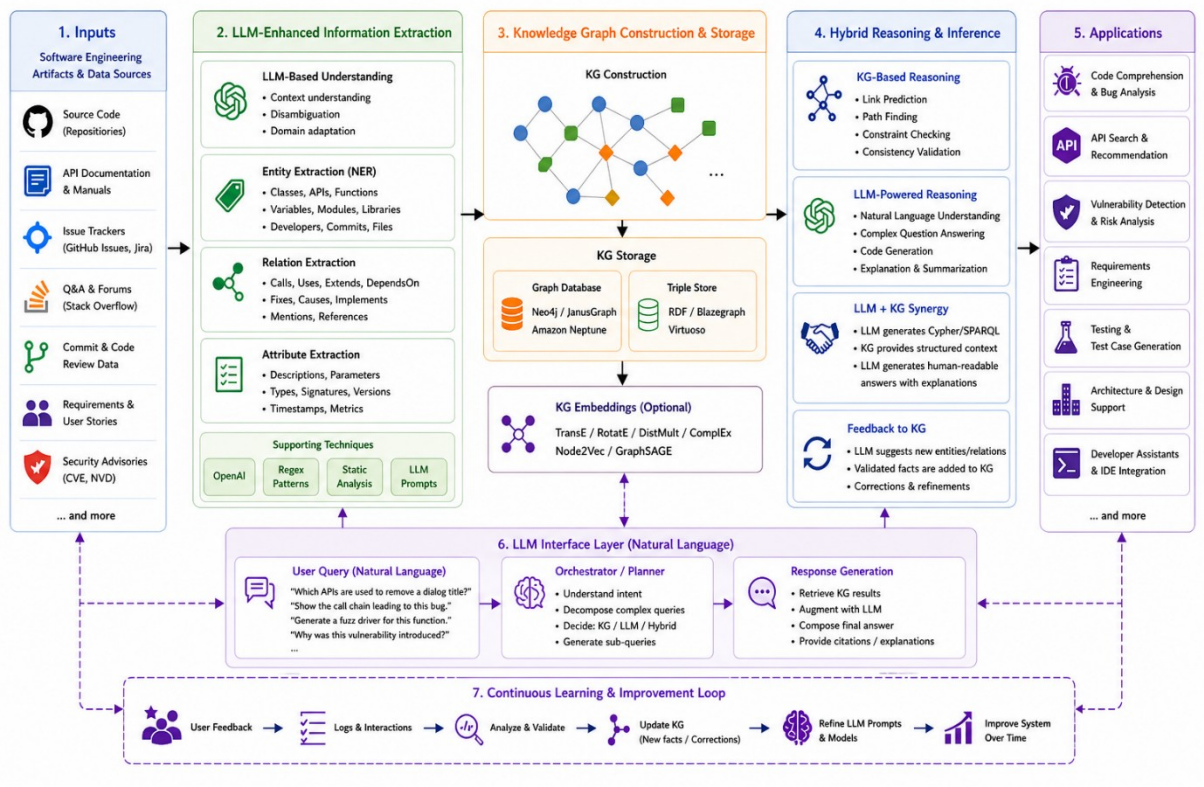

FIGURE 8: LLMS AND KNOWLEDGE GRAPHS COMPLEMENT

Figure 8 highlights how LLMs and knowledge graphs complement each other, with graphs providing structured context and LLMs enabling flexible interaction and enhanced reasoning capabilities.

5.2 Implications for Research

We observe a growing integration between symbolic and neural methods in the reviewed papers. Neural techniques offer efficiency and effectiveness in applications like link prediction, whereas ontological approaches deliver clarity and reasoning capabilities. But each approach has its limitations[62]. We observe that many recent works are adopting hybrid systems that combine ontologies, information extraction pipelines and neural or LLM-based reasoning, pointing towards an evolution towards hybrid systems.

The second point is that the use of large language models in knowledge graph systems is changing. Far from being purely knowledge consumers, LLMs are gaining prominence as components of KG pipelines for extraction, completion, and natural language interface. This two-way communication between KGs and LLMs is an emerging feature of recent research, facilitating more flexible knowledge processing.

5.3 Future Research Agenda

From the limitations we identified in the studies we reviewed, several promising research avenues stand out. First, there is a need for benchmark datasets for software engineering knowledge graph tasks. While there have been many experiments, evaluation is dispersed with most studies using small, bespoke datasets. A further key area is the incremental evolution of KGs. While many papers acknowledge the need to deal with ever-evolving software systems, few offer practical mechanisms to keep KGs up to date with evolving code and repositories.

Another challenge is the evaluation of LLM-KG systems. These methods enhance adaptability and completeness but also lead to potential problems such as fact generation or inconsistency, making it necessary to design evaluation metrics that measure both reasoning and factual accuracy.

Moreover, ethical concerns are growing in importance, especially for security applications such as social-engineering or attack graph modelling. But formal guidelines are still lacking. Finally, there is a growing potential for practical uses of knowledge graphs in development. Although there are promising prototypes, systems are predominantly research-focused, and there is little use of IDE-integrated or CI/CD-aware KG tools.

5.4 Gaps in the Existing Work

The study also highlights some structural gaps in the research. First, empirical assessment is largely confined to academia with a very small number of studies involving industrial application or user validation. Second, reproducibility is still limited[63]. While there is growing recognition of open science, many studies do not publicly release code, data or complete KG artifacts, limiting replicability. Third, scalability is still an issue. The proposed knowledge graphs are small and do not consider the complexity of industrial systems, where software ecosystems can include very large and constantly changing software components.

5.5 Final Observation

Knowledge graphs have progressed from arcane research subject to a growing part of the software engineering research infrastructure. They transitioned from a static representation for knowledge towards being dynamic, queryable, and increasingly LLM-based. Recent breakthroughs in large language models (LLMs) don't seem to replace knowledge graphs, but instead enhance their natural language interaction, construction and reasoning capabilities.

Yet, while this work has advanced, the field is still in an early integration stage. Their deployment in practice, evaluation and maintenance strategies remain relatively low. Future research is likely to be focused on closing this gap between research and production for use in software engineering.

6. Threats to Validity

We classify the threats to validity into four types: construct, internal, external and conclusion validity[64].

Construct validity relates to the possibility of misalignment between the concept of knowledge graphs and how they are defined in the review. Knowledge graph is an evolving term that captures many representations including ontology-based representations, embedding-based representations, and more recent large language model (LLM)-augmented representations. This leaves a risk of incorporating studies that are only partially in line with the KG paradigm or overlooking borderline cases like property graphs or vector representations with similar goals.

To counter this risk, we used an operational definition of knowledge graphs as representations of software entities and their typed relationships that enables reasoning. This was systematically used to guide the screening process. Moreover, we adopted a double review protocol and snowballing to ensure that studies with a marginal or unclear description were considered for inclusion.

Internal validity refers to the risk of bias in the selection and extraction of studies. Because the classification of studies into domains and techniques, and the evaluation of studies into dimensions is somewhat subjective, bias could exist in classifications.

To minimise this, two authors independently conducted all screening and extraction activities. Inconsistencies were resolved by discussion or by a third reviewer. There were high levels of agreement between reviewers, with Cohen's κ of 0.87 (for inclusion or exclusion of studies) and 0.91 (for domain classification).

External validity relates to the generalizability of the results to the chosen corpus. Our review is mainly grounded on refereed academic literature and high-quality preprints between 2016 and 2025. This means that industrial grey literature, proprietary enterprise knowledge graph systems and very recent publications may not be captured.

To mitigate this, we included influential arXiv preprints and used backward and forward snowballing to cover recent influential work. However, as shown in the

results, only a limited number of papers report industrial deployment, which should be taken into account for generalizability.

The risks identified are common for systematic reviews in the fast-moving field of AI. We addressed these threats via rigorous dual reviewing, iterative search optimisation, snowballing, and publication of the entire review protocol, search logs and extraction sheets (see below), ensuring high reliability and replicability.

Conclusion

In this systematic literature review, we analyzed 60 primary studies published from 2016 to 2025, offering a broad view of the application of knowledge graphs (KGs) in software engineering. The review provides a unified perspective on the application domains, the way KGs are constructed, the way reasoning is performed, and the current research trends. The results reveal that the most popular applications of KGs include code understanding and bug analysis (40%), API-related tasks (25%) and security and vulnerability management (20%). While less common, areas such as testing, requirements engineering and software architecture showcase the growing diversification throughout the software development process.

In terms of methodology, KG construction has advanced over the years. Initial efforts were mainly ontology-based or rule-based, whereas recent research is increasingly adopting hybrid pipelines that combine information extraction techniques (e.g., OpenIE and NER), graph embedding, and, in the latest stage (2023-2025), large language models. Likewise, reasoning has evolved from traditional approaches such as link prediction, to structured query answering, to recommendation systems and classification tasks, and more recently, LLM-powered reasoning such as natural language querying, clarification, and artifact generation (e.g., fuzz drivers). This marks a transition towards interactive and adaptable KGs.

The landscape has seen significant acceleration since 2019, with a more noticeable acceleration in 2023 with the adoption of LLMs in knowledge graphs. This is transforming the development and use of software engineering

knowledge systems. However, there are still several open challenges, such as scalability to large industrial projects, challenges in maintaining ever-evolving knowledge graphs, lack of standardized benchmarks and ethical issues in security-critical domains. These challenges currently limit reproducibility and practice.

One of the most promising avenues from recent research is the seamless integration of KGs and LLMs, leveraging the explainability of graphs and the generative and language capabilities of modern language models.

It is imperative for future studies to focus on the creation of large-scale, standardized benchmarks for SE-specific KG tasks, effective solutions for incremental and real-time KG maintenance, and ethical considerations for dual-use applications, especially in the area of security. Overcoming these challenges will be essential to moving knowledge graphs from the lab to the infrastructure for knowledge-driven, context-sensitive software engineering.

References

1. Ahrabian, K., et al., *Software engineering event modeling using relative time in temporal knowledge graphs.* arXiv preprint arXiv:2007.01231, 2020.
2. Chen, D., et al. *Automatically identifying bug entities and relations for bug analysis*. in *2019 IEEE 1st international workshop on intelligent bug fixing (IBF)*. 2019. IEEE.
3. Chen, H., et al. *A practical framework for evaluating the quality of knowledge graph*. in *China conference on knowledge graph and semantic computing*. 2019. Springer.
4. Cheng, K., et al., *Neural-symbolic methods for knowledge graph reasoning: A survey.* ACM Transactions on Knowledge Discovery from Data, 2025. **18**(9): p. 1-44.
5. Du, T., et al. *Cocoqa: Question answering for coding conventions over knowledge graphs*. in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2019. IEEE.
6. Elfaki, A., A. Aljaedi, and Y. Duan. *Mapping ERD to knowledge graph*. in *2019 IEEE World Congress on Services (SERVICES)*. 2019. IEEE.
7. Fathalla, S., et al. *Towards a knowledge graph representing research findings by semantifying survey articles*. in *International Conference on Theory and Practice of Digital Libraries*. 2017. Springer.

8. Channa, W.A., Q.U. Khand, and S.A. Ghanghro, *Requirements Extraction from User Feedback On The Basis Of Ontology.* IJCSNS, 2019. **19**(12): p. 157.
9. GUAN, S.-P., *Knowledge reasoning over knowledge graph: a survey.* Journal of Software, 2018. **29**(10): p. 2966-2994.
10. Guo, C., et al. *Crowdsourced requirements generation for automatic testing via knowledge graph*. in *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*. 2020.
11. Han, Z., et al. *Deepweak: Reasoning common software weaknesses via knowledge graph embedding*. in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2018. IEEE.
12. Ghanghro, S.A., et al., *Comparative Analysis of Software Process Models in Software Development.* International Journal of Advanced Trends in Computer Science and Engineering, 2021. **10**(3): p. 2593-2599.
13. Wang, S., et al., *Review, framework, and future perspectives of Geographic Knowledge Graph (GeoKG) quality assessment.* Geo-spatial Information Science, 2025. **28**(4): p. 1701-1721.
14. Hao, X., et al., *Construction and application of a knowledge graph.* Remote Sensing, 2021. **13**(13): p. 2511.
15. Hu, W., et al. *Open source software vulnerability propagation analysis algorithm based on knowledge graph*. in *2019 IEEE International Conference on Smart Cloud (SmartCloud)*. 2019. IEEE.
16. Huang, Q., et al. *API method recommendation without worrying about the task-API knowledge gap*. in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018.
17. Kashif, U.A., et al. *Centralized accessibility of VM for distributed trusted cloud computing*. in *2023 4th International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. 2023. IEEE.
18. Ke, W., et al. *Interpretable test case recommendation based on knowledge graph*. in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*. 2020. IEEE.
19. Li, A., et al. *A survey of relation extraction of knowledge graphs*. in *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data*. 2019. Springer.
20. Kesri, V., A. Nayak, and K. Ponnalagu. *Autokg-an automotive domain knowledge graph for software testing: a position paper*. in *2021 ieee international conference on software testing, verification and validation workshops (icstw)*. 2021. IEEE.

21. Kosasih, E.E. and A. Brintrup, *Towards trustworthy AI for link prediction in supply chain knowledge graph: a neurosymbolic reasoning approach.* International Journal of Production Research, 2025. **63**(6): p. 2268-2290.
22. Kwapong, B. and K. Fletcher. *A knowledge graph based framework for web API recommendation*. in *2019 IEEE World Congress on Services (SERVICES)*. 2019. IEEE.
23. Lavrinovics, E., et al., *Knowledge graphs, large language models, and hallucinations: An nlp perspective.* Journal of Web Semantics, 2025. **85**: p. 100844.
24. Lin, Z.-Q., et al., *Intelligent development environment and software knowledge graph.* Journal of Computer Science and Technology, 2017. **32**(2): p. 242-249.
25. Li, Q., et al., *Unifying task-oriented knowledge graph learning and recommendation.* IEEE Access, 2019. **7**: p. 115816-115828.
26. Li, H., et al. *Improving api caveats accessibility by mining api caveats knowledge graph*. in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2018. IEEE.
27. Liu, C., et al. *A survey of recommendation algorithms based on knowledge graph embedding*. in *2019 IEEE International Conference on Computer Science and Educational Informatization (CSEI)*. 2019. IEEE.
28. Liu, L., Z. Wang, and H. Tong, *Neural-symbolic reasoning over knowledge graphs: A survey from a query perspective.* ACM SIGKDD Explorations Newsletter, 2025. **27**(1): p. 124-136.
29. Liu, L., et al., *Research on application of knowledge graph in industrial control system security situation awareness and decision-making: A survey.* Neurocomputing, 2025. **613**: p. 128721.
30. Liu, Y., et al. *Generating concept based API element comparison using a knowledge graph*. in *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*. 2020.
31. Lou, Y., et al. *Knowledge graph programming with a human-in-the-loop: Preliminary results*. in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 2019.
32. Lu, Y., et al., *Biomedical knowledge graph: A survey of domains, tasks, and real-world applications.* arXiv preprint arXiv:2501.11632, 2025.
33. Lv, W., et al. *How to construct software knowledge graph: A case study*. in *2020 IEEE World Congress on Services (SERVICES)*. 2020. IEEE.

34. Ma, D., et al. *A knowledge graph-based sensitive feature selection for android malware classification*. in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. 2020. IEEE.
35. Mongiello, M., et al. *Case-based reasoning and knowledge-graph based metamodel for runtime adaptive architectural modeling*. in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 2016.
36. Channa, W.A., et al., *Reasoning and Representation of Legal Cases Through Ontology.* IJCSNS, 2019. **19**(8): p. 24.
37. Mrdjenovich, D., et al., *Propnet: a knowledge graph for materials science.* Matter, 2020. **2**(2): p. 464-480.
38. Nayak, A., V. Kesri, and R.K. Dubey, *Knowledge graph based automated generation of test cases in software engineering*, in *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*. 2020. p. 289-295.
39. Pan, J.Z., et al., *A knowledge graph based approach to social science surveys.* Data Intelligence, 2021. **3**(4): p. 477-506.
40. Paulheim, H., *Knowledge graph refinement: A survey of approaches and evaluation methods.* Semantic web, 2016. **8**(3): p. 489-508.
41. Puri, R., et al., *Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks.* arXiv preprint arXiv:2105.12655, 2021.
42. Ren, X., et al. *API-misuse detection driven by fine-grained API-constraint knowledge graph*. in *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*. 2020.
43. Schindler, D., et al., *The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central.* PeerJ Computer Science, 2022. **8**: p. e835.
44. Shang, F., et al., *Construction and application of the user behavior knowledge graph in software platforms.* Journal of Web Engineering, 2021. **20**(2): p. 387-411.
45. Su, Y., et al. *Enhancing exploratory testing by large language model and knowledge graph*. in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024.
46. Sun, J., et al. *Task-oriented api usage examples prompting powered by programming task knowledge graph*. in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021. IEEE.
47. Wang, J., et al. *SoftKG: Building a software development knowledge graph through wikipedia taxonomy*. in *2020 IEEE world congress on services (SERVICES)*. 2020. IEEE.

48. Wang, L., et al. *Construct bug knowledge graph for bug resolution*. in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 2017. IEEE.
49. Wang, M., et al. *Searching software knowledge graph with question*. in *International conference on software and systems reuse*. 2019. Springer.
50. Wang, Q., et al., *Knowledge graph embedding: A survey of approaches and applications.* IEEE transactions on knowledge and data engineering, 2017. **29**(12): p. 2724-2743.
51. Wang, X., *Research on knowledge graph data management: a survey.* Journal of Software, 2019. **30**(7): p. 2139-2174.
52. Wang, X., et al., *Knowledge graph quality control: A survey.* Fundamental Research, 2021. **1**(5): p. 607-626.
53. Wang, X. and S. Yang. *A tutorial and survey on fault knowledge graph*. in *International Conference on Cyberspace Data and Intelligence*. 2019. Springer.
54. Wang, Z., et al., *Social engineering in cybersecurity: a domain ontology and knowledge graph application examples.* Cybersecurity, 2021. **4**(1): p. 31.
55. Xiao, G., et al. *The virtual knowledge graph system ontop*. in *International Semantic Web Conference*. 2020. Springer.
56. Xiao, H., et al. *Embedding and predicting software security entity relationships: A knowledge graph based approach*. in *International Conference on Neural Information Processing*. 2019. Springer.
57. Xiaoli, X., et al., *Development of personalized learning resources recommendation system based on knowledge graph.* International Journal of Educational Technology and Learning, 2021. **10**(2): p. 68-72.
58. XING, S.-S., M.-W. LIU, and X. Peng, *Automatic code semantic tag generation approach based on software knowledge graph.* Journal of Software, 2021. **33**(11): p. 4027-4045.
59. Xu, X., et al., *AI-CTO: Knowledge graph for automated and dependable software stack solution.* Journal of Intelligent & Fuzzy Systems, 2021. **40**(1): p. 799-812.
60. Yang, C., et al., *An importance assessment model of open-source community java projects based on domain knowledge graph.* Journal on Big Data, 2020. **2**(4): p. 135.
61. Ye, D., et al. *Software-specific named entity recognition in software engineering social content*. in *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*. 2016. IEEE.

62. Zhang, J., et al. *Exploiting code knowledge graph for bug localization via bi-directional attention*. in *Proceedings of the 28th International Conference on Program Comprehension*. 2020.
63. Zhang, K. and J. Liu. *Review on the application of knowledge graph in cyber security assessment*. in *IOP Conference Series: Materials Science and Engineering*. 2020. IOP Publishing.
64. Zhou, C. *Intelligent bug fixing with software bug knowledge graph*. in *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 2018.