Toward model-driven automated code review: extracting quality intent knowledge from review comments and accepted code changes

Rikuto Sasaki[1,*], Tomoya Hirata[1] and Kazuhiro Takeuchi[1]

[1]Osaka Electro-Communication University, Neyagawa, Osaka 572-8530, Japan

Abstract

As generative AI increasingly produces source code from specifications that may not fully articulate quality-related intentions, model-driven automated code review requires reusable and traceable knowledge that connects such intentions to concrete implementations and review evidence. Rather than treating local open-weight large language models (LLMs) as predictors of ground-truth developer intent, this paper compares the candidate change purpose sets they generate from accepted code changes under controlled conditions. It then grounds these candidates in SEBoK-derived Quality Attributes and models the supported relations as Quality Intent Knowledge. Using 900 manually verified CodeReviewQA cases, we applied the same prompt and structured output schema to six local open-weight LLMs. Without access to the review comments, each model generated Claims and Implications from the Reference Change, defined as the difference between the pre-review and accepted post-review code. Each Implication was treated as an unvalidated candidate change purpose. All six models generated candidate change purposes, although the size of the resulting candidate sets varied by model. From `gpt-oss-120b`, which produced the largest candidate set, we retained 760 candidate change purposes from 208 Implication-dominant cases, in which review comments were more similar on average to Implications than to Claims. We grounded these 760 candidates in a SEBoK-derived Quality Attribute vocabulary and reassessed each proposed assignment against the review comment, Reference Change, and Claims. Candidates with at least one supported assignment were operationally defined as review-grounded purposes, yielding 579 review-grounded purposes from 205 cases and 606 supported Quality Attribute assignments. We represented the resulting Quality Intent Knowledge in an RDF knowledge graph based on a lightweight OWL/RDFS model linking review-grounded purposes and Quality Attributes to reassessment evidence, originating review cases, 255 mask regions, and reference completions. An executable SPARQL use case illustrates model-driven reuse by retrieving source cases for a specified Quality Attribute and constructing traceable contexts for review-comment and code-completion generation. We position this study as an initial knowledge-modeling architecture for constructing and traceably reusing Quality Intent Knowledge, rather than as a validated automated code-review method.

Keywords

automated code review, Quality Intent Knowledge, local open-weight LLMs, Quality Attribute grounding, knowledge graph

1. Introduction

Software quality assurance commonly relies on two complementary activities: testing and review. Testing executes a program to determine whether its observed behavior conforms to expected results, whereas review statically examines source code and related design artifacts to identify defects and inconsistencies with requirements or design intent.

As generative AI increasingly participates in software implementation, the objectives and constraints that developers are expected to consider are not always stated explicitly in specifications, and design intent traditionally shared among developers may not be adequately conveyed in prompts. For example, a prompt requesting implementation of connection-handling logic may not explicitly state that a potentially null connection object should be checked before dereferencing. Such a quality-related expectation may nevertheless appear in prior review comments and accepted changes even when it

Knowledge Graphs and Model-driven Systems Engineering (KGMDSE), in ISWC 2026 Workshops Joint Proceedings, October 25–26, 2026, Bari, Italy

*Corresponding author.

mi25a002@oecu.jp (R. Sasaki); takeuchi@osakac.ac.jp (K. Takeuchi)

0009-0006-4307-1767 (K. Takeuchi)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-05

CEUR-WS.org/Vol-4313/kgmdse-paper10.pdf

is absent from the target prompt. Related studies have also reported difficulties in satisfying multiple user-specified constraints, maintaining intention alignment among software-development agents, and applying fragmented domain knowledge in software development using large language models (LLMs) [1, 2, 3].

Consequently, review plays an increasingly important role in determining whether generated code conforms not only to explicit specifications but also to the requirements, constraints, and design intent governing its implementation. Moreover, as generative AI enables code to be produced rapidly and iteratively, it may become difficult to apply such judgments consistently through human review alone. Such review judgments should therefore be modeled as reusable knowledge that connects concrete code changes, the quality concerns they address, and supporting evidence. This knowledge provides an explicit and traceable basis for model-driven automated code review while remaining available to human reviewers.

In practice, code review examines not only whether an implementation provides the required functionality, but also whether it satisfies quality concerns such as reliability, maintainability, safety, and security. The Systems Engineering Body of Knowledge (SEBoK) treats Human Systems Integration, adaptability, reliability, availability, maintainability, resilience, safety, and security as system-level Quality Attributes [4].

In this study, higher-level quality purposes inferred from accepted code changes are treated as *candidate change purposes*. A candidate with at least one Quality Attribute assignment reassessed as `supported` is retained as a *review-grounded purpose*. We use *Quality Intent Knowledge* to denote the structured knowledge that connects review-grounded purposes to Quality Attributes, supporting evidence, originating review cases, and concrete edit regions. The SEBoK-derived categories in Table 1 provide a compact cross-language vocabulary for grounding and comparing change purposes derived from different review cases. These analytical assignments do not imply that a local code change alone establishes a system-level quality property.

Table 1
Quality Attribute vocabulary used in this study.

Quality Attribute	Operational description
Human Systems Integration	Quality of interactions between humans and systems.
Adaptability	Ability to accommodate changes in requirements, environments, or configurations.
Reliability	Ability to continue performing required behavior correctly.
Availability	Ability to provide a service when required.
Maintainability	Ease of understanding, diagnosing, modifying, and testing.
Resilience	Ability to withstand and recover from failures or unexpected conditions.
Safety	Prevention of states that may lead to hazards or harm.
Security	Prevention of unauthorized use, access, modification, or disclosure.
Other Quality Attribute	A supported quality concern outside the predefined vocabulary.

Existing code-related language resources commonly pair natural-language descriptions of functionality with source code that implements the described behavior [5, 6]. External programming knowledge, including API documentation and programming question–answer pairs, has also been incorporated into natural-language-to-code generation [7]. Prior work has further treated natural-language software requirements as a semantic-parsing problem and mapped them to formal or ontology-based representations [8, 9]. These studies primarily focus on functionality descriptions, software requirements, or programming knowledge rather than constructing Quality Intent Knowledge from review histories.

We therefore construct Quality Intent Knowledge by connecting review-grounded purposes to Quality Attributes, review and code evidence, originating review cases, and concrete edit regions. In this study, we use local open-weight LLMs to generate candidate change purposes from accepted code changes. We

ground the candidate change purposes in SEBoK-derived Quality Attributes and reassess each proposed Quality Attribute assignment against review comments and code evidence. The resulting Quality Intent Knowledge is instantiated as an RDF knowledge graph according to an OWL/RDFS model for Quality Intent Knowledge.

In this study, *model-driven* refers to a workflow in which the relations defined in the OWL/RDFS model are used to retrieve source-case knowledge grounded in a specified Quality Attribute, and the retrieved knowledge is used to condition LLM-based review and code generation. Generated review suggestions are candidates for human consideration, with links to the source cases and evidence used to condition the LLM. We present an initial knowledge-modeling architecture and executable proof of concept demonstrating the feasibility of constructing and traceably reusing Quality Intent Knowledge. The results do not establish its semantic correctness or an improvement in review effectiveness.

The contributions of this study are threefold. First, we define an OWL/RDFS model that reifies supported Quality Attribute assignments and explicitly records their reassessments. The model supports tracing each assignment to its review-grounded purpose, reassessment evidence, originating review case, edit region, and reference completion. Second, we implement a construction pipeline comprising candidate generation, Quality Attribute grounding, assignment reassessment, and RDF instantiation. A comparison of six local open-weight LLMs under a shared prompt and structured output schema characterizes variation in the size of the candidate pools supplied to this pipeline. Third, we demonstrate an executable workflow that retrieves source cases for a specified Quality Attribute and reuses them for review-comment and code-completion generation while preserving traceability to the source cases and their evidence.

2. Modeling review-grounded purposes as quality intent knowledge

CodeReviewQA (CRQA) [10] comprises 900 manually verified cases across nine programming languages, each containing pre-review code, a review comment, and an accepted revision. CRQA evaluates implicit review requests in terms of change type, location, and concrete solution, but does not represent higher-level purposes inferred from accepted changes as knowledge linked to Quality Attributes and supporting evidence. We therefore use the Claim–Implication distinction from CRScore [11] to connect candidate change purposes derived from CRQA cases to Quality Attributes, supporting evidence, and their originating review cases.

Figure 1 presents the overall construction of Quality Intent Knowledge from a CRQA case. The pipeline separates candidate generation from assignment reassessment so that intermediate results and supporting evidence remain traceable.

3. Quality intent knowledge extraction and graph construction

3.1. Local-LLM generation and selection of candidate change purposes

We define the *Reference Change* as the diff between the pre-review code and the accepted post-review code and use it to generate Claims and Implications. The accepted post-review code is treated as a benchmark reference reflecting the review request, but not as the only valid implementation or a complete specification. Following the distinction introduced in CRScore [11], a Claim describes a modification directly supported by the diff, whereas an Implication describes a possible higher-level effect or purpose inferred from that modification. Each generated Implication is treated as an unvalidated *candidate change purpose*. As illustrated in Figure 1, the Claim describes the directly observable field replacement, whereas the Implication expresses an inferred higher-level purpose candidate. Both are generated from the Reference Change alone; the review comment is used only for subsequent candidate selection.

For all 900 CRQA cases, we generate Claims and Implications using six local open-weight LLMs: `gpt-oss-20b`, `gpt-oss-120b`, `qwen2.5-14b-instruct`, `qwen2.5-32b-instruct`, `qwen2.5-`

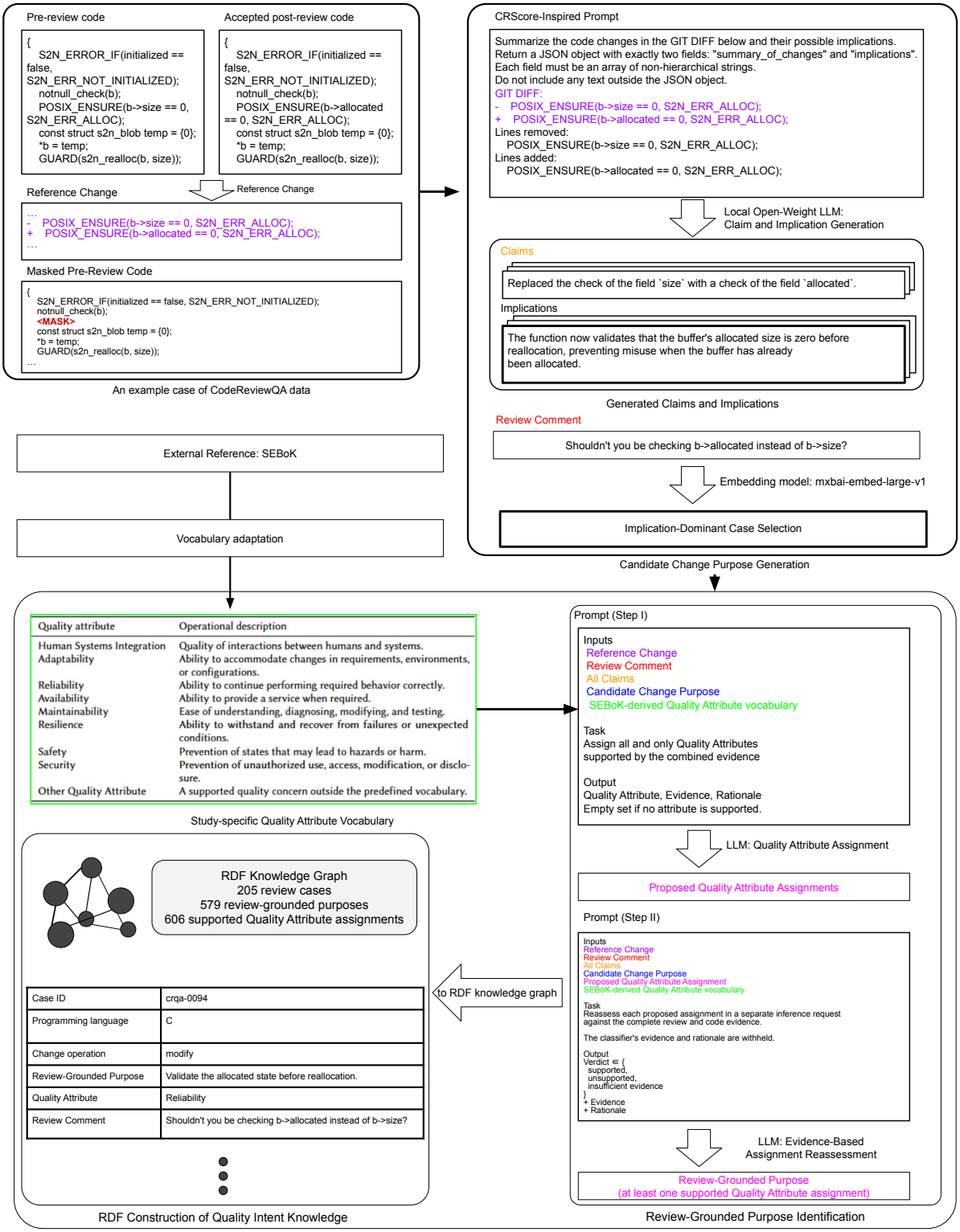


Figure 1: Running example of Claim and Implication generation, candidate change purpose selection, Quality Attribute grounding, assignment reassessment, and RDF construction of Quality Intent Knowledge.

`coder-14b-instruct`, and `qwen2.5-coder-32b-instruct`. All models receive the same zero-shot instruction without demonstrations and the same structured JSON output schema. The instruction asks each model to summarize the Reference Change and its possible implications as Claims and Implications. Withholding the review comment separates candidate generation from the review evidence used for subsequent case selection. The shared instruction and schema control the prompting condition across

models and allow Claims and Implications to be processed separately. The comparison examines candidate generation and variation in candidate-pool size, rather than accuracy in recovering developer intent.

To select cases for subsequent Quality Attribute grounding and assignment reassessment, we use `mixedbread-ai/mxbai-embed-large-v1`, also used in CRScore, to calculate cosine similarities between the review comment and the Claims and Implications generated for the same case. A case is defined as *Implication-dominant* when the mean review–Implication similarity exceeds the mean review–Claim similarity. Because Claims and Implications differ in abstraction level and lexical form, this difference may reflect representational variation as well as purpose-related correspondence. The criterion is therefore used only for case selection and does not establish the correctness of a generated Implication or demonstrate recovery of the developer's actual intention. Each Implication generated for an Implication-dominant case is retained as a *candidate change purpose* for subsequent Quality Attribute grounding and assignment reassessment.

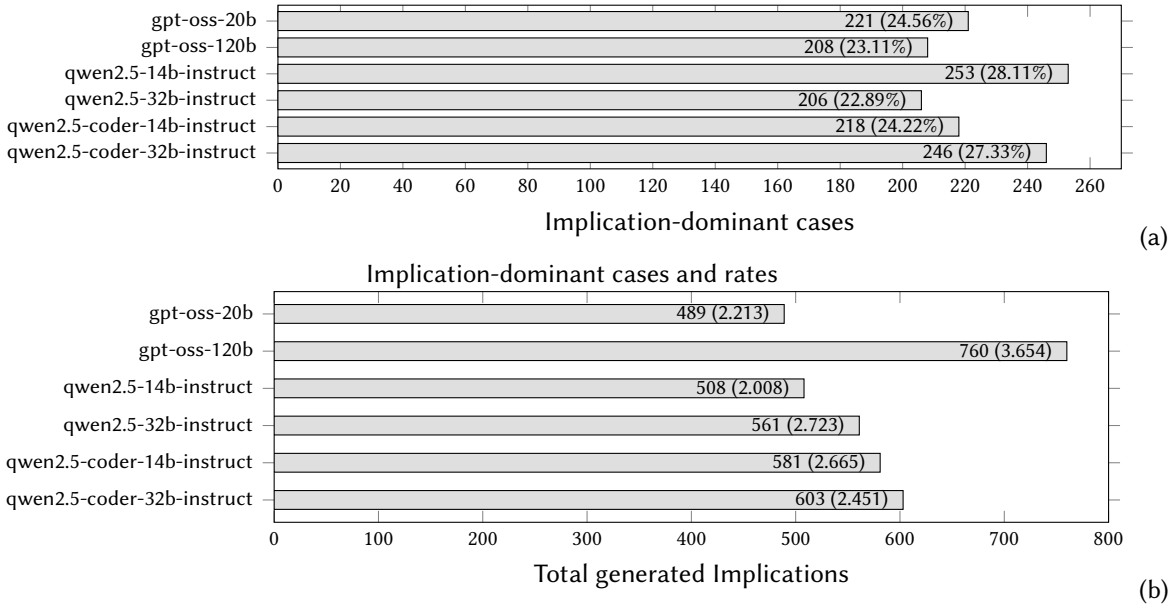


Figure 2: Candidate change purpose generation by six local open-weight LLMs under the same prompt and structured output schema. Panel (a) shows Implication-dominant cases and their rates among 900 CRQA cases; panel (b) shows the total and mean numbers of Implications for the selected cases.

As shown in Figure 2, all six models yielded Implication-dominant cases, but the size of the candidate pool available for subsequent grounding and reassessment varied by model. We used `gpt-oss-120b` because it yielded the largest pool: 760 candidates from 208 cases covering all nine CRQA languages. This selection maximises the number of candidates carried forward and does not imply higher generation accuracy.

3.2. LLM-based quality attribute grounding and assignment reassessment

To ground freely written candidate change purposes in a common vocabulary, we use an LLM-as-a-judge to propose Quality Attribute assignments from the vocabulary shown in Table 1. A candidate change purpose may receive multiple proposed assignments and is left unassigned when the first stage proposes no Quality Attribute. The grounding and reassessment prompts use task instructions that explicitly specify the task, input evidence, and structured output schema.

In the first stage, the candidate change purpose, programming language, review comment, Reference Change, Claims, and Quality Attribute definitions are provided to the LLM to propose all potentially applicable Quality Attribute assignments. The review comment and Reference Change are treated as

primary evidence, while the Claims provide supplementary natural-language descriptions of the change. For each proposed Quality Attribute assignment, the attribute, supporting evidence, and rationale are returned as structured output.

In the second stage, the Quality Attribute assignments proposed for each candidate change purpose are reassessed in a separate inference request using a different prompt and output schema from the first stage. The reassessment input includes the candidate change purpose, review comment, Reference Change, and Claims, but excludes the evidence description and rationale generated in the first stage. No new Quality Attribute assignments are added, and each assignment is classified as `supported`, `unsupported`, or `insufficient_evidence`. Both stages use `openai/gpt-oss-120b`. Separating the inference requests and withholding the first-stage evidence and rationale limits their direct reuse during reassessment, but does not remove model-specific bias. Independent-model or human assessment remains necessary for external validation of the supported assignments.

We operationally define a candidate change purpose with at least one Quality Attribute assignment reassessed as `supported` as a *review-grounded purpose*. A review-grounded purpose is not a ground-truth representation of the developer's internal intention; it is a candidate supported by review and code evidence under our assignment reassessment procedure.

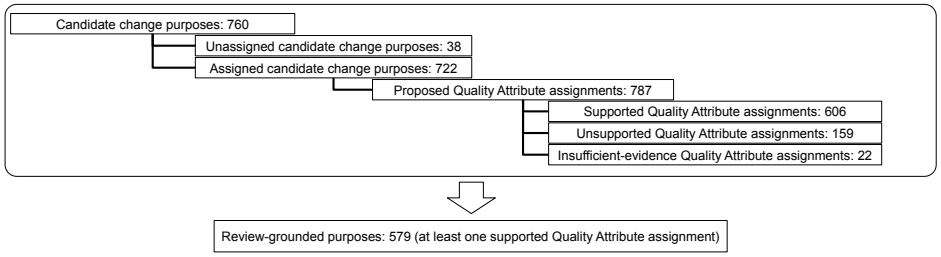


Figure 3: Quality Attribute grounding and assignment reassessment outcomes.

As shown in Figure 3, the procedure yielded 579 review-grounded purposes from 205 of the 208 selected cases. This operational result does not estimate developers' actual intentions; only supported assignments are retained in the RDF knowledge graph. In the first stage, multiple Quality Attribute assignments were proposed for 64 candidate change purposes. After reassessment, Reliability and Maintainability accounted for 259 and 180 of the 606 supported assignments, respectively, although this distribution is not representative of CRQA as a whole.

To represent review-grounded purposes and their supporting evidence as searchable and traceable Quality Intent Knowledge, we define a lightweight OWL/RDFS model for Quality Intent Knowledge. The model serves not only as a vocabulary for storing the results, but also specifies the elements and relations that constitute Quality Intent Knowledge. Figure 4 shows the core classes and relations of the OWL/RDFS model for Quality Intent Knowledge, connecting supported Quality Attribute assignments, reassessment evidence, originating review cases, and concrete change information. Each review case is instantiated in RDF according to this model, and the same classes and relations provide the structural basis for SPARQL retrieval and LLM generation-context construction. In the workflow presented here, these explicitly modeled relations are used for Quality-Attribute-conditioned source-case retrieval, evidence traceability, and LLM generation-context construction. The workflow does not use entailment-based reasoning to derive additional review-relevant relations.

Rather than representing the relation between a review-grounded purpose and a Quality Attribute as a direct edge, the model reifies each supported relation as a `SupportedQualityAttributeAssignment` instance. Each supported assignment is linked to an `AssignmentReassessment`, which in turn references the review comment, Reference Change, and Claims used in the reassessment. This representation makes not only the assigned attribute but also the evidence and decision path supporting the assignment traceable.

Each `ReviewCase` is linked to its review and code evidence and to a `MaskExample` containing one or more `MaskRegion` instances. Each region represents a contiguous edit and its reference completion; non-

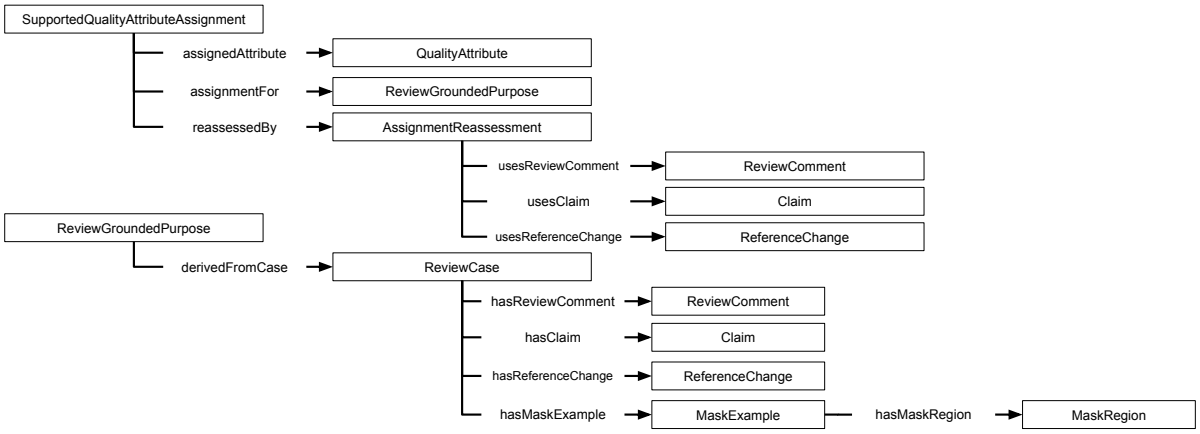


Figure 4: Core classes and relations of the OWL/RDFS model for Quality Intent Knowledge. Supported Quality Attribute assignments connect review-grounded purposes to their reassessment evidence, originating review cases, and concrete change information. Repeated boxes denote the same OWL classes and are duplicated only for readability.

contiguous edits are retained as separate regions. This structure enables tracing from a Quality Attribute to a review-grounded purpose, its reassessment evidence, originating review case, and concrete code change. The 205 review cases contain 255 mask regions, and their reference completions reconstruct the accepted post-review code.

4. Illustrative use case: attribute-conditioned review and code generation

This use case demonstrates how the OWL/RDFS model structures source-case retrieval and context construction for LLM generation. The workflow can serve as a review-support step in a software engineering process, with a specified Quality Attribute providing the review perspective. A predefined SPARQL query traverses the modeled relations among `QualityAttribute`, `ReviewGroundedPurpose`, `ReviewCase`, and `MaskExample`. Programming language, Quality Attribute, and mask structure provide the retrieval conditions; this example selects single-region C source cases grounded in Reliability. The query is not generated by the LLM.

The SPARQL `ORDER BY` clause determines the order of retrieved rows; relevance ranking with respect to the target code is performed separately after retrieval. After retrieval, the target case is excluded, and the remaining candidates are ranked by embedding similarity to the target pre-review code. The top three are selected as source-case knowledge.

The review-grounded purposes, Claims, source review comments, masked source code, and reference completions of the selected source cases are assembled with the target pre-review code to form the LLM generation context. Unlike candidate generation, the generation prompts in this use case follow a retrieval-conditioned, source-case-based prompting strategy. The retrieved source cases provide analogical review knowledge for the specified Quality Attribute, while the LLM assesses its applicability without a separate explicit representation of the problem-solving roles of target-code regions. The target review comment, Reference Change, and accepted post-review code are withheld from the generation context.

Using `openai/gpt-oss-120b`, the review-generation stage is conditioned on the specified Quality Attribute, target pre-review code, and retrieved source-case knowledge. It produces an applicability judgment, a review comment, a proposed review location, and source-case identifiers. The masked target code is then constructed from the resulting review location.

A separate code-generation prompt treats the generated review comment as a fixed review request and conditions on the masked target code and selected source-case knowledge to generate the code

completion replacing the mask as structured output. This separation treats review generation and code generation as distinct prompting stages.

Candidate generation uses zero-shot prompting without demonstrations. Review and code generation use retrieval-augmented generation, with retrieved source-case knowledge incorporated into the generation context. Quality Attribute grounding and reassessment use separate task instructions with explicit evidence inputs and structured outputs.

Figure 5 summarises this model-based retrieval, ranking, context construction, and review and code generation workflow.

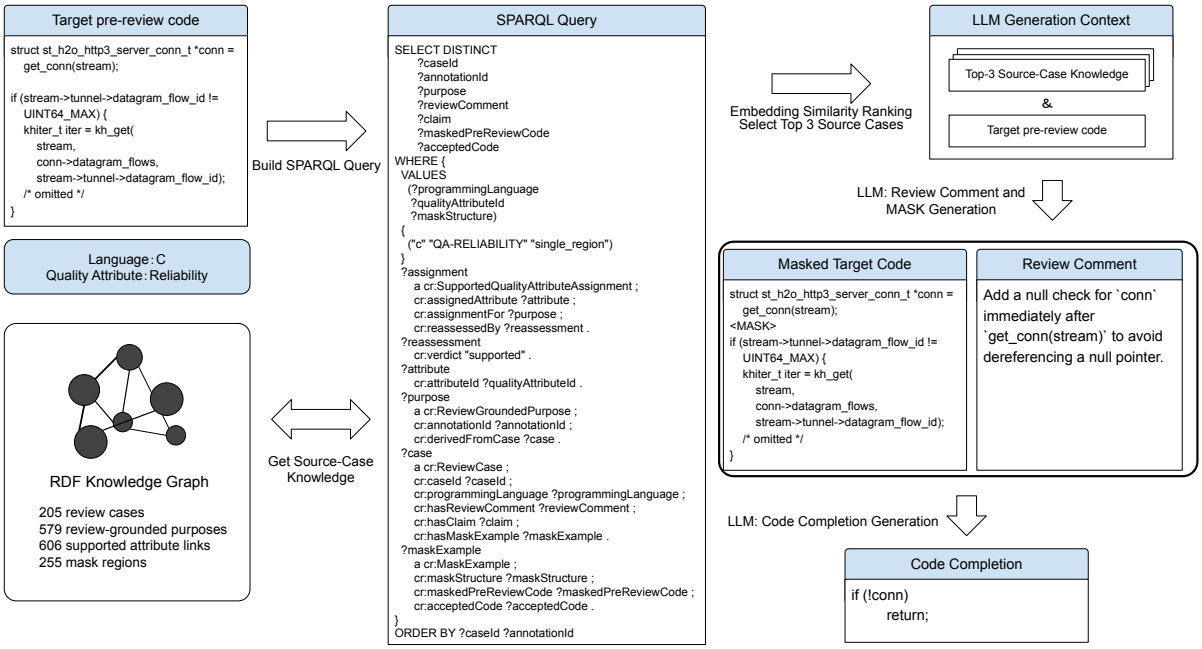


Figure 5: Attribute-conditioned source-case retrieval and case-based review comment and code completion generation with source-case traceability.

The model requested a null check before dereferencing `conn` and generated `if (!conn) return;` as the code completion at the masked insertion point. Retaining the identifiers of the selected source cases enables the Quality Intent Knowledge used for conditioning to be traced in the RDF knowledge graph. This illustrative use case demonstrates a workflow for retrieving and reusing Quality Intent Knowledge in LLM generation while preserving traceability; it does not evaluate whether such reuse improves review effectiveness.

5. Conclusion

We presented an OWL/RDFS model and an executable workflow for constructing and reusing Quality Intent Knowledge from review histories. Under a shared prompt and output schema, six local open-weight LLMs produced candidate pools of different sizes. Grounding and reassessing the 760 candidates from 208 cases generated by `gpt-oss-120b` yielded 579 review-grounded purposes from 205 cases and 606 supported Quality Attribute assignments.

The model connects purposes and Quality Attributes to reassessment evidence, source cases, and concrete code changes. The use case combines attribute-conditioned SPARQL retrieval with embedding-similarity ranking to construct contexts for review-comment and code-completion generation. This organization separates the quality perspective from code-level relevance while preserving traceability to the source cases and evidence used for generation.

These results demonstrate the operational feasibility of constructing and traceably reusing Quality Intent Knowledge, rather than validating its semantic correctness or review effectiveness. Future work

could complement this workflow with a non-LLM module that represents the problem-solving roles of target-code regions and their relations to functional sub-goals and problem requirements, providing explicit context for assessing the applicability of retrieved Quality Intent Knowledge.

Declaration on generative AI

ChatGPT was used to translate text originally drafted by the authors in their native language into English, and for linguistic polishing and proofreading. The authors reviewed the final manuscript and take full responsibility for its content.

References

[1] Z. Fang, Y. Dong, L. Mou, D. Jin, Z. Jin, G. Li, Intent↑Coding: Amplifying user intent in code generation, in: Findings of ACL 2026, 2026, pp. 33246–33261.

[2] J. Liu, G. Wang, R. Yang, J. Zeng, M. Zhao, Y. Cai, RTADev: Intention aligned multi-agent framework for software development, in: Findings of ACL 2025, 2025, pp. 1548–1581.

[3] X. Jiang, G. Li, J. Qian, X. Shi, C. Li, H. Zhu, Z. Wang, J. Zhang, Z. Zhao, K. Zhang, J. Li, W. Jiao, Z. Jin, Y. Dong, KoCo-Bench: Can large language models leverage domain knowledge in software development?, in: Proc. ACL 2026, 2026, pp. 28422–28441.

[4] SEBoK Editorial Board, The Guide to the Systems Engineering Body of Knowledge (SEBoK), 2026. URL: https://sebokwiki.org/, version 2.14, N. Hutchison, Editor in Chief. Accessed July 24, 2026.

[5] P. Yin, G. Neubig, A syntactic neural model for general-purpose code generation, in: Proc. ACL 2017, 2017, pp. 440–450.

[6] S. Iyer, I. Konstas, A. Cheung, L. Zettlemoyer, Mapping language to code in programmatic context, in: Proc. EMNLP 2018, 2018, pp. 1643–1652.

[7] F. F. Xu, Z. Jiang, P. Yin, B. Vasilescu, G. Neubig, Incorporating external knowledge through pre-training for natural language to code generation, in: Proc. ACL 2020, 2020, pp. 6045–6052.

[8] M. Roth, T. Diamantopoulos, E. Klein, A. Symeonidis, Software requirements: A new domain for semantic parsers, in: Proc. ACL 2014 Workshop on Semantic Parsing, 2014, pp. 50–54.

[9] M. Roth, E. Klein, Parsing software requirements with an ontology-based semantic role labeler, in: Proc. 1st Workshop on Language and Ontologies, 2015, pp. 15–21. URL: https://aclanthology.org/W15-0403/.

[10] H. Y. Lin, C. Liu, H. Gao, P. Thongtanunam, C. Treude, CodeReviewQA: The code review comprehension assessment for large language models, in: Findings of ACL 2025, 2025, pp. 9138–9166.

[11] A. Naik, M. Alenius, D. Fried, C. Rosé, CRScore: Grounding automated evaluation of code review comments in code claims and smells, in: Proc. NAACL 2025, 2025, pp. 9049–9076.