TU WIEN Informatics

Improving Software Engineering Student Grading Using Retrieval-Augmented Large Language Models

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Christoph Neubauer, BSc
Matrikelnummer 12023172

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig

Wien, 27. April 2026

Christoph Neubauer	Thomas Grechenig

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Improving Software Engineering Student Grading Using Retrieval-Augmented Large Language Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Christoph Neubauer, BSc
Registration Number 12023172

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig

Vienna, April 27, 2026

Christoph Neubauer	Thomas Grechenig

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Improving Software Engineering Student Grading Using Retrieval-Augmented Large Language Models

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Christoph Neubauer, BSc
Matrikelnummer 12023172

ausgeführt am
Institut für Information Systems Engineering
Forschungsbereich Business Informatics
Forschungsgruppe Industrielle Software
der Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig

Wien, 27. April 2026

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Christoph Neubauer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 27. April 2026

Christoph Neubauer

Danksagung

Ich möchte meiner Familie und meinen Freunden danken, die mich während meines Studiums unterstützt haben. Mein besonderer Dank gilt Daniel, Matthias und Philipp, die mich während meiner gesamten Zeit an der TU Wien begleitet haben.

Darüber hinaus danke ich allen, die diese Arbeit korrekturgelesen und wertvolle Anregungen zu ihrer Verbesserung gegeben haben.

Schließlich danke ich meinem Betreuer für seine Anleitung und seine Beiträge im Rahmen dieser Masterarbeit.

Acknowledgements

I would like to thank my family and friends for their support throughout my studies. Special thanks go to Daniel, Matthias, and Philipp, who accompanied me throughout my whole time at TU Wien.

In addition, I thank all those who proofread this thesis and provided valuable suggestions for its improvement.

Finally, I thank my supervisor for his guidance and input throughout this thesis.

Kurzfassung

Die Bewertung von Studierendenprojekten im Bereich des Software Engineering stellt für Lehrende eine anspruchsvolle Aufgabe dar. Verschiedene Aspekte wie Quellcode und Metriken aus agilen Entwicklungsprozessen sind für die Beurteilung relevant, liegen jedoch häufig in fragmentierten Datenquellen vor. Dadurch müssen Lehrende Zusammenhänge manuell rekonstruieren, was zu fehlendem Kontext führen und die Genauigkeit der Bewertung beeinträchtigen kann.

Diese Arbeit adressiert dieses Problem mithilfe eines Retrieval-Augmented-Generation (RAG)-Systems, das Informationen zu Studierendenprojekten aus den verfügbaren Datenquellen abruft und in Form eines umfassenden Berichts aufbereitet. Ziel ist es, Lehrende bei der Bewertung zu unterstützen, indem der Suchaufwand reduziert und zusätzliche Einblicke in die Arbeit der Studierenden ermöglicht werden. Darüber hinaus wird der Einsatz eines solchen KI-Systems im Bildungskontext untersucht, um den Nutzen den potenziellen Risiken gegenüberzustellen. Die Erkenntnisse aus der Literaturrecherche und den Experteninterviews wurden genutzt, um einen RAG-Prototypen zu entwerfen. Dieser berücksichtigt die Informationsbedarfe von Studienassistent:innen in Form von Softwarequalitätsmetriken sowie architektonische Entscheidungen für RAG-Pipelines.

Die Ergebnisse zeigen, dass die durch den RAG-Prototypen generierten Berichte verlässliche Zusammenfassungen der Studierendenprojekte liefern und sowohl Tutor:innen als auch Studienassistent:innen tiefere Einblicke ermöglichen. Zudem dienen die Berichte als geeigneter Ausgangspunkt und reduzieren den Zeitaufwand für die manuelle Informationssuche. Die Evaluation zeigt, dass Vertrauen und menschliche Überprüfung eine zentrale Rolle beim Einsatz von KI-Systemen im Bildungskontext spielen, um Risiken bewusst zu berücksichtigen. Insgesamt zeigt diese Arbeit die Eignung von RAG-generierten Berichten zur Unterstützung von Lehrenden bei der Bewertung von Studierendenprojekten

Schlüsselwörter: *Retrieval Augmented Generation, Large Language Models, Softwarequalitätsmetriken, Educational Data Mining, Softwareentwicklungs-Lehre*

Abstract

Grading Software Engineering (SE) student projects is a challenging task for educators. Many aspects, including source code and agile process metrics, are relevant for assessment, but data is often fragmented across multiple sources. Hence, educators must manually reconstruct contextual relationships, which often leads to missing context and affects grading accuracy.

This thesis addresses this issue by proposing a Retrieval Augmented Generation (RAG) system that retrieves information about student projects from available data sources and generates a comprehensive report to provide an overview of students' work. This aims to support educators during assessment by reducing the effort required to search for information and providing more insights. Furthermore, the use of such an Artificial Intelligence (AI) system in education is examined to assess the benefits against potential risks.
The insights from literature search and expert interviews are used to design a RAG prototype. This includes the assessment information needs of study assistants in the form of Software Quality Metrics (SQMs) and architectural choices for RAG pipelines.

The findings indicate that reports generated by the RAG prototype provide faithful summaries of students' group work, enabling both tutors and study assistants to gain greater insight. Furthermore, the reports serve as a good starting point, reducing the time required to manually search for information. The evaluation indicated that trust and human verification play a significant role in the use of AI systems in an educational context, ensuring awareness of introduced risks and appropriate handling. Overall, this thesis demonstrates the viability of supporting educators during assessment with RAG-generated reports.

Keywords: *Retrieval Augmented Generation, Large Language Models, Software Quality Metrics, Educational Data Mining, Software Engineering Education*

Contents

CHAPTER 1

Introduction

1.1 Problem Description

Grading semester-long student software projects for groups using an agile process poses a challenging task for study assistants and tutors. Student groups follow an agile development process, hence produce numerous artefacts, including source code, documentation, time logs, Merge Requests (MRs), and more. The relationships between all these elements are often difficult for outsiders to reconstruct. Therefore, tutors and study assistants often lack complete contextual information during assessment and final grading. In practice, several tools are used to tackle this demand. Visualisation tools are used to gain insights into specific aspects of the development process, but data from different sources must still be compared and analysed manually. Consequently, these tools provide only fragmented insights. Prior research highlights that missing context negatively affects both grading accuracy and consistency in code review and assessment tasks [1], [2]. As Parizi et al. note, "*Most commonly, in order to estimate student contributions, instructors use arbitrary and subjective judgment derived from observations and evaluations. The currently used process is not a complete picture and is time-consuming since it requires numerous observations and extensive paperwork review.*" [2].

At the same time, Large Language Models (LLMs) show strong potential for contextual reasoning, summarisation, and code analysis when provided with the relevant information. RAG [3] systems incorporate external data sources to provide LLMs with the necessary information, reducing hallucinations [4] and improving reliability in tasks that require factual grounding.

This thesis proposes a RAG-based system that retrieves and consolidates data relevant to student projects, including several SQM, and uses this information to generate comprehensive assessment reports. The goal is not to replace human educators, but to support

tutors and study assistants by providing structured insights into project activity, reducing manual inspection workload, and compensating for missing or otherwise overlooked information. Because the EU AI Act classifies AI systems used in educational decision-making as *high-risk* systems [5], the approach emphasises transparency, verifiability and human oversight. Therefore, the generated reports include intermediate information breakdowns and traceable data-source references to enable verification by educators.

1.2 Aim of the Thesis

Achieving consistent assessment across heterogeneous student projects is difficult due to the high volume and fragmented nature of the available data. With current tools, obtaining a comprehensive summary of all data sources is time-consuming. This required effort increases the risk that relevant aspects are overlooked or weighted inconsistently during grading.

This thesis aims to leverage current advances in Generative Artificial Intelligence (GenAI), specifically RAG [3], to improve the current assessment workflow in SE education. The goal is to design, implement, and evaluate a RAG system that automatically retrieves, integrates, and summarises project-related data into structured analytical reports to support the assessment of student software projects. The proposed system combines heterogeneous data sources, including GitLab project management artefacts, Git repository data and computed SQM, to provide a verifiable summary of various aspects of student project data in a single report.

As a preparatory step, the information needs of educators are identified by reviewing commonly used SQM and this set of metrics is refined through expert interviews with study assistants from the TU Wien undergraduate SE course, where the proposed RAG system is applied. These SQM form part of the RAG system's information basis and ensure that the generated reports align with both established industry practises and course-specific assessment needs.

The central objective of this thesis is to evaluate whether RAG-generated assessment reports are faithful to the underlying data, consistent in retrieval behaviour, and beneficial for educators in terms of time savings and information completeness of their assessment. Furthermore, this thesis examines how tutors and study assistants perceive the trustworthiness, transparency, and the extent of verification required and performed during assessment. The availability of these AI-generated reports may introduce automation bias [6], [7], leading to insufficient verification and over-reliance on the reports.

Formalising these objectives results in the following research questions:

RQ1: What is the faithfulness and retrieval consistency of a RAG-generated assessment report operating on student project data?

RQ2: How does the integration of RAG-generated reports impact the time efficiency and information discovery of tutors during the assessment of student software projects?

RQ3: How do tutors and study assistants perceive the trustworthiness, reasoning transparency, and completeness of RAG-generated reports?

RQ4: How does reliance on AI-generated reports change the verification workflow of tutors and study assistants?

1.3 Contributions

This thesis establishes a data-driven RAG system for supporting the assessment of SE student projects. By integrating heterogeneous data sources, such as GitLab analytics, Git repository data, and SQM, this approach provides more comprehensive insight into student projects, which in turn reduces the effort required to search for information manually. In addition to the concrete implementation, the role of transparency and human oversight in AI systems is explored, whether the immediate risk of over-reliance on such tools is present, or verification of information is maintained. By evaluating both the potential benefits and risks of employing a RAG-based AI tool in an educational context, this thesis provides a foundation for further research.

1.4 Structure

The rest of this thesis is structured as follows.

- Chapter 2 introduces the concepts used in this thesis. This includes domain concepts, SQM, LLMs, RAG, and evaluation methods for RAG.
- Chapter 3 presents the current state of the art. Recent research regarding Educational Data Mining (EDM) and RAG with a focus on education is highlighted. Additionally, this chapter describes the distinction of this thesis from previous research.
- Chapter 4 describes the methodological approach of this thesis. The chapter gives an overview, followed by the details of all stages of this thesis in Sections 4.1 to 4.4. Finally, the approach for each research question is summarised in Section 4.5.
- Chapter 5 describes commonly used SQM in industry and education, and the resulting information needs of the concrete SE course. The findings form the basis for the subsequent ETL and RAG implementations.

- Chapters 6 and 7 describe the prototype implementation in detail. Chapter 6 focuses on the initial Extract, Transform, Load (ETL) process, describing the structure of the available data and all required stages to collect it in a structured way for later retrieval by the RAG system. Chapter 7 presents the implementation of the RAG prototype. There, corresponding design decisions and the overall setup and workflow of the RAG system are presented.
- Chapter 8 covers all aspects of the final evaluation of the RAG prototype, including a discussion regarding the results of the different evaluation perspectives.
- Chapter 9 lists the findings of this thesis with regard to the research questions.
- Chapter 10 presents the findings and conclusion of this thesis. Additionally, a look ahead to future work is provided on how the concept of RAG may further support SE education.

CHAPTER 2

Foundation

In this chapter, the fundamental concepts of the thesis are described. First, relevant domain concepts from SE are presented in Sections 2.1 and 2.2. Then, the technological foundations of the proposed approach are outlined. This includes an overview of LLMs and the underlying transformer architecture in Section 2.3, followed by the concept of RAG, which enables LLMs to incorporate external knowledge during response generation (Section 2.4). In addition, the principles of LLM agents and tool use are introduced in Section 2.5, as these mechanisms form the core of the system architecture implemented in this thesis. Finally, commonly used evaluation methods for RAG systems are presented in Section 2.6, which provide the basis for assessing the quality of generated outputs in later chapters.

2.1 Domain Concepts

2.1.1 Distributed Version Control Systems

Version Control Systems (VCS) are widely used in SE to manage source code and track changes accross projects projects [8]. They are especially helpful in collaborative environments where multiple developers work on a single project. VCS record all source code changes made during the development process, allowing tracking of the project's history and individual collaborators' work. Distributed Version Control Systems (DVCS) are a type of VCS, where in addition to a central repository on a server, each collaborator holds a local repository [8]. This way, developers can work offline and synchronise their changes with a shared repository.

Repository Repositories constitute the central storage location of VCS [9]. The resources of such repositories can be created, updated, or deleted via a VCS. In the context of this thesis, two types of repositories are relevant: The source code repository

is the location where the main development process happens. This process includes proper version control management, separate branches for features, and code review before features from different branches are merged into a single state. The wiki repository corresponds to the GitLab wiki, where documentation is stored rather than source code. While this wiki repository supports version control, it is typically not used that way, and only the latest version is relevant.

Git Git is one of the most used DVCS, as it elevates each software version to first-class-citizens [10]. With Git, developers have extensive control over version management in their local copies of a repository, such as amending changes, reordering or reverting commits. In addition, Git supports branching and merging specific versions, enabling parallel development of independent features.

GitLab GitLab [11] is a platform for hosting Git repositories that provides many features that enable teams to manage the entire software project lifecycle in one place. For the group phase of the SE course which is described later, GitLab serves as the main platform, where each group has its own GitLab repository, including, but not limited to, the following features:

- Software repository: The source code is hosted on the GitLab instance.
- Wiki repository: A separate repository where all documentation is uploaded. In the SE course at TU Wien, changes to this repository are made primarily via the web interface, where Markdown files are created and edited.
- Issue tracking system: Issues are used to manage collaboration among team members by organising tasks and assigning responsibilities. Source code commits may reference issues for better traceability of implementation effort.
- Merge Requests: MRs are created when merging branches. GitLab provides an interface for team members to review MRs and either approve an MR or start a discussion with the author of the MR, to improve code quality.
- Time logs: GitLab allows time tracking on both issues and MRs. This way, the time commitment of each team member is recorded and can be used during project planning for task assignment or, in the context of this thesis, for student assessment.

Mining Software Repositories Software repositories not only hold the current state of a software system but also contain its entire evolution history via a VCS. This includes all prior and intermediate versions, as well as metadata such as the contributing authors and timestamps. Mining Software Repositories (MSR) aims to analyse available data to uncover useful information about software systems, thereby enabling and justifying follow-up actions [12]. Applications range from industry, where MSR is used to improve software processes, to the field of EDM, where student data are collected for educational purposes [2], [13]–[16].

2.1.2 Course Concepts

The SE course in which these RAG-generated reports are used is structured in two phases. The first phase, referred to as the individual phase, covers the first month of the semester. During this phase, each student implements a small, largely CRUD-based web application, with a Java[1] and Spring Boot[2] backend layer and Angular[3] frontend layer.

After successfully completing the individual phase, students self-organise into groups of up to six members for the group phase, in which they develop a larger software project until the end of the semester. During this group project, students follow an agile Scrum process [17]. This process includes planning sprints with well-defined features, conducting peer code reviews, documenting retrospectives, and continuously adapting the application throughout development.

The course schedule defines several fixed milestones throughout the semester, most importantly, three Management Reviews (MRx). These consist of meetings with a study assistant: the first review (MR1) defines the final project scope, the second review (MR2) serves as an intermediate progress check, and the third review (MR3) represents the final submission and grading milestone. In addition, student groups meet weekly with a tutor (i.e., a student teaching assistant) for regular progress updates and organisational guidance. As a result, tutors typically have closer insight into the project development process, whereas study assistants are primarily responsible for the final assessment and grading decisions.

The nature of the group project depends on the study division to which each group is assigned:

Div. A: Each student group develops a similar web application, with seven mandatory User Stories and a pool of three extended User Stories, of which two must be implemented.

Div. B: Each student group defines their own project. The only constraints are a time budget of 110 hours per person and the technology stack: Java and Spring Boot for the backend, and Angular for the frontend. During the planning phase, students must submit a preliminary project proposal and later a project contract that specifies their final deliverables at the end of the semester.

This structural division leads to a wide range of project outcomes, from largely comparable solutions in Div. A, to highly heterogeneous projects in Div. B. This variation presents a challenge for consistent grading decisions, particularly for groups in Div. B, where deliverables differ substantially due to the project scope agreed upon.

[1] `https://www.java.com/de/`, Accessed: 02.03.2026
[2] `https://spring.io/projects/spring-boot`, Accessed: 02.03.2026
[3] `https://angular.dev/`, Accessed: 02.03.2026

For this thesis, only data from the group phase is considered. Although each student maintains a GitLab repository during the individual phase, it is primarily used for linear progress updates on a single branch. Consequently, only the source code and commit history are available for analysis in this phase. In contrast, group projects yield more diverse and fragmented data, making them better suited to demonstrate the advantages of the proposed RAG system.

Assessment Several aspects are relevant to students' final grades, which can be divided into two categories: project and process. The project category focuses on the implementation and quality of deliverables. For example, whether all promised features were delivered, their quality, the occurrence of bugs, and adherence to coding guidelines. This project assessment is conducted at the end of the semester at MR3, where the study assistant assigns final grades to each student. The process category evaluates how the work was accomplished. Students are expected to contribute evenly throughout the semester, plan development in sprints, maintain proper documentation, and collaborate as a team. Thus, the quality of both the final project and the management of the development process throughout the semester are important for assessment.

The tutors play an important role in this process. By meeting weekly with the student group, they stay informed about the project's status. Therefore, they can effectively guide students, providing feedback on both process management and application implementation. Tutors report their findings to the study assistant, and all insights are collected before MR3, when the study assistant grades the students.

In its final proof-of-concept implementation, the RAG-generated reports are intended for use by tutors and study assistants prior to MR3. The reports contain data on both the group's project implementation and process management. Thus, they can be used to obtain well-founded insights into multiple aspects of the project before performing the final grading during MR3.

2.2 Software Quality Metrics

Software Quality Metrics make up a core part of the data used during retrieval of the RAG prototype (Chapter 7). The IEEE 1061 standard [18] defined a SQM as

"A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality."

Not all of the data aspects of this thesis, which are labelled as metrics, meet this strict definition of a metric. Chapter 5 includes several metrics that are not a single numerical value. This can be caused by several properties:

- Way of measurement: The work distribution between team members can be measured in several different ways, depending on the usecase or preference. For example, the Gini coefficient [19] of the time commitment can be computed to quantify the disparity in effort in a single value. The same can be done by reporting the total hours spent by each member, yielding multiple values rather than a single value as required by the definition of a metric.
- Nature of the data: Other aspects, such as the quality of commit messages, cannot be reliably quantified in a numerical value. While it is a subjective matter, there are guidelines and standards to adhere to, and commit messages may be labelled as high or low quality.
- Overarching concept: Some aspects describe a broader concept that may encompass several metrics. For instance, the *documentation written* is referenced in several research articles as a metric (see Chapter 5). This can mean several things: the number of written project documents, the percentage of issues that include a description, or the quality of protocol content.

For simpler terminology, all of these are referenced as SQM in the scope of this thesis, as they all contribute insight into different aspects of software quality, even when they do not strictly meet the definition of a software quality metric. The remainder of this section introduces the metrics referenced in Chapter 5, whose names are not self-explanatory.

Cyclomatic Complexity The cyclomatic complexity was introduced by McCabe in 1976 [20]. It is based on a program's control-flow graph and measures the number of linearly independent paths. This metric was created as an indicator to split overly complex procedures or modules into several smaller ones.

Cognitive Complexity The cognitive complexity approximately measures the understandability of source code [21]. It is measured through the control flow of a program; breaks in the linear flow, such as conditional structures and loops, and nested structures, increase the cognitive complexity. A high value suggests that passages should be refactored to improve source code readability, which can be an important factor in collaborative projects, where developers need to understand code written by others.

Halstead Metrics The Halstead metrics are a collection of metrics based on the numbers of total and distinct operators and operands of a program [22]. With these values, metrics such as the *program vocabulary*, *program length*, the *difficulty* to write or understand, or the time *effort* needed for implementing said program, can be computed.

Chidamber & Kemerer (CK) Metrics The CK metrics suite [23] is a collection of six SQM which measure different aspects of object oriented programming: the Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Response For a Class (RFC) and Lack of Cohesion in Methods (LCOM).

Bus Factor The bus factor is known in SE as *"The number of key developers who would need to be incapacitated, i.e. hit by a bus, to make a project unable to proceed"* [24]. This definition is easy to understand but vague and thus needs to be adapted to the specific context of this thesis. In the context of the SE group project, the bus factor is therefore defined as:

Definition 2.2.1 *"How many students would need to leave the project such that 50% of codebase component knowledge (backend/frontend) is lost?"*

This way, the bus factor can be measured by the students' code contributions. This definition is not fully error-proof, as additional aspects may be considered when describing project knowledge, such as general domain knowledge and code review during the agile process. However, it is accepted by study assistants during interviews concerning SQM (Section 5.2) for the proof-of-concept implementation of this thesis.

Mean Time to Repair In businesses, the Mean Time To Repair (MTTR) is measured as the average time it takes to repair a system after it has failed [25]. This time ranges from the moment the failure occurs until the problem is resolved. A more precise definition is needed in the context of a student project, in particular, the definition of the moment of occurrence. This may be defined by the time of the bug-introducing commit, the time the commit is deployed to production, or the time the bug is discovered in the system. Depending on this specification, the metric's value varies widely.

Static Code Analysis Metrics Several metrics mentioned in this thesis are computed by static code analysis, in particular with the SonarQube tool [26]. The measurement of metrics such as bugs, code smells, and security vulnerabilities are based on fixed rules that determine whether specific lines of source code present an issue. While bugs are faulty sections of code, code smells on the other hand denote structural weaknesses that make code harder to maintain. All of these measures capture different aspects of code quality and indicate where maintenance is needed.

Tech Debt Technical debt describes the fact that code with poor quality needs to be refactored again later on in the process, leading to additional cost [27]. In this thesis, tech debt, as computed by static code analysis, is defined as the number of hours required to fix all maintainability issues. Each issue is assigned a multiplicative factor based on its severity.

2.3 Large Language Models

The use of LLMs has become widespread in everyday applications, as they are able to generate natural-language text of various forms, such as essays or source code, given only unstructured textual input [28]. The foundation of modern LLMs is the transformer architecture, originally introduced with an encoder-decoder structure by Vaswani et al. [29]. The two components, encoder and decoder, are shown on the left and right sides of Figure 2.1, respectively. Before being passed to the encoder, the input text is segmented into individual tokens, each of which represents a subword or a single symbol from the input. The encoder maps this sequence of tokens to an internal numerical vector representation that incorporates positional embeddings and captures the input's contextual meaning. The decoder generates output tokens one at a time, taking both the encoder output and the previously generated tokens into account. Both the encoder and the decoder consist of multiple stacked transformer blocks, each comprising a self-attention mechanism and a feed-forward neural network (Figure 2.1). Most modern LLMs, such as GPT [30], extend this original design and employ decoder-only architectures.

The attention mechanism is the core of this architecture, as it models the contextual relationships among all tokens in a sequence [29]. For this purpose, a query vector (Q) and key-value vector pairs (K, V) are constructed. For a given computation step, the query represents the current token seeking context, the keys represent all available tokens in the sequence being attended to, and the values hold the actual semantic content of the keys to be aggregated. The attention score is calculated as a scaled dot-product, yielding a weighted sum of the values:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Because each token must attend to every other token, this computation scales quadratically ($O(N^2)$) with the sequence length N, making it computationally expensive for long inputs. In addition, LLMs must be explicitly trained to handle extended context lengths [28]. These two factors impose practical limits on the maximum context window size of an LLM, which represents a central design consideration for the RAG implementation of this thesis (Chapter 7).

Once the attention mechanism establishes a context-aware representation of the input, the feed-forward neural network processes and refines this information. Previous work by Geva et al. [31] has shown that these networks can serve as a distributed knowledge base by storing information in their learned parameters. Through training on large-scale datasets, the model parameters are optimised to encode linguistic and factual patterns.

Finally, probability distributions over possible next tokens are computed. Depending on the decoding strategy, one token is selected at each step [28]. For example, *top-k* sampling randomly selects the next token from the k most probable candidates based on

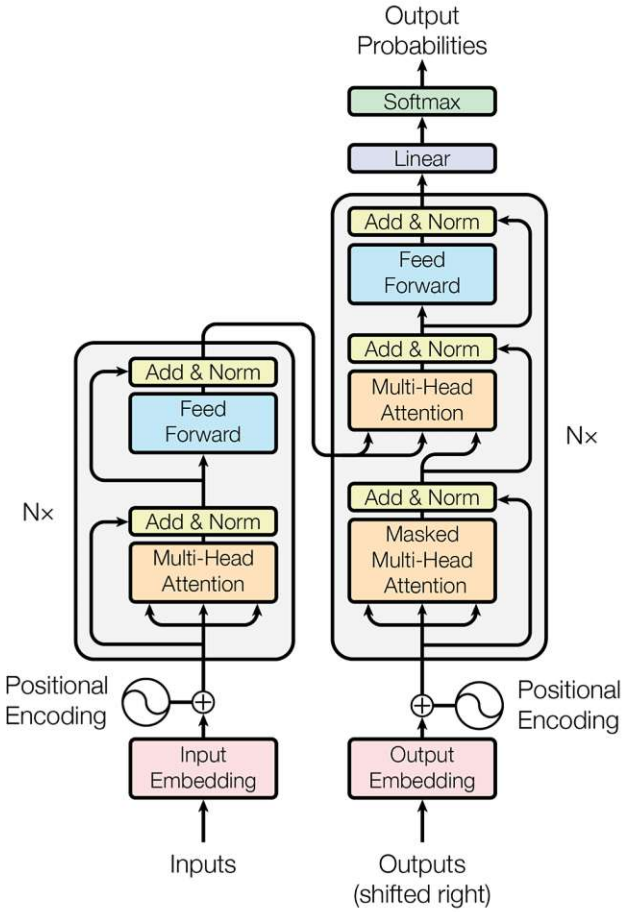

Figure 2.1: The Transformer LLM Architecture. [29].

the previously generated tokens. This is used in combination with a *temperature* value, where a lower temperature increases the likelihood that more probable candidates are selected. Because this form of natural language generation is based on likelihoods rather than pure facts, identical inputs may produce different outputs. Consequently, LLMs can generate factually incorrect outputs, especially when the input of the LLM concerns topics that were not included in the training data [32]. Although training and fine-tuning strategies can reduce such errors, they cannot fully eliminate them.

2.4 Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) has gained significant attention since its introduction by Lewis et al. in 2020 [3] and has since evolved into a highly active research area within AI [33]–[38]. While LLMs are trained on vast amounts of data and can generate text based on the knowledge encoded in their parameters, they remain limited by a fixed training cutoff. Information that was not included in the training data or emerged only after training cannot be reliably incorporated into their responses. This limitation is referred to as "out-of-date" internal knowledge [35].

In such cases, LLMs frequently produce hallucinations [32], confidently generating factually incorrect answers. This occurs because token generation is driven by statistical likelihood rather than by access to verified external information. Hallucinations therefore present a substantial challenge, particularly in domains where factual correctness is critical, such as in medical and legal settings [39].

RAG addresses the problem of out-of-date internal knowledge by incorporating external knowledge sources retrieved during generation. These external sources can take several forms: for example, documents stored in a vector database or files directly attached to the prompt to supplement the LLM's internal knowledge. Some systems, such as NotebookLM[4], rely exclusively on external knowledge to strictly minimise hallucinations. Research by Shuster et al. [4] demonstrates that RAG significantly reduces hallucinations since the LLM grounds its generated answers based on verifiable, external knowledge.

2.5 Large Language Model Agents and Tool Use

A recent development in AI closely related to RAG is the concept of LLM agents. Russell et al. defined an agent in 1995 as "*anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.*" [40]. This broad definition includes humans, animals, and machines. In the context of AI, an LLM agent refers to a computational entity (typically an LLM in a RAG system or similar environment) that can perceive external information or a software system's state and act upon it according to its task [41]. Such tasks may include summarising data, retrieving facts or performing more complex operations. These perceptions and actions are often carried out by the agent using specific tools, which are described later.

Xi et al. present LLM agents as a conceptual framework comprising three components: brain, perception, and action (Figure 2.2) [41]. In this framework, the brain of the LLM agent is the LLM itself. It contains internal knowledge acquired during training and maintains a form of memory, such as the history of received tasks, planned steps, or previously executed actions. The brain performs reasoning and planning, determining what information to perceive (e.g., which data to retrieve) and which actions to take based on the retrieved data.

[4] `https://notebooklm.google/` Accessed: 01.03.2026.

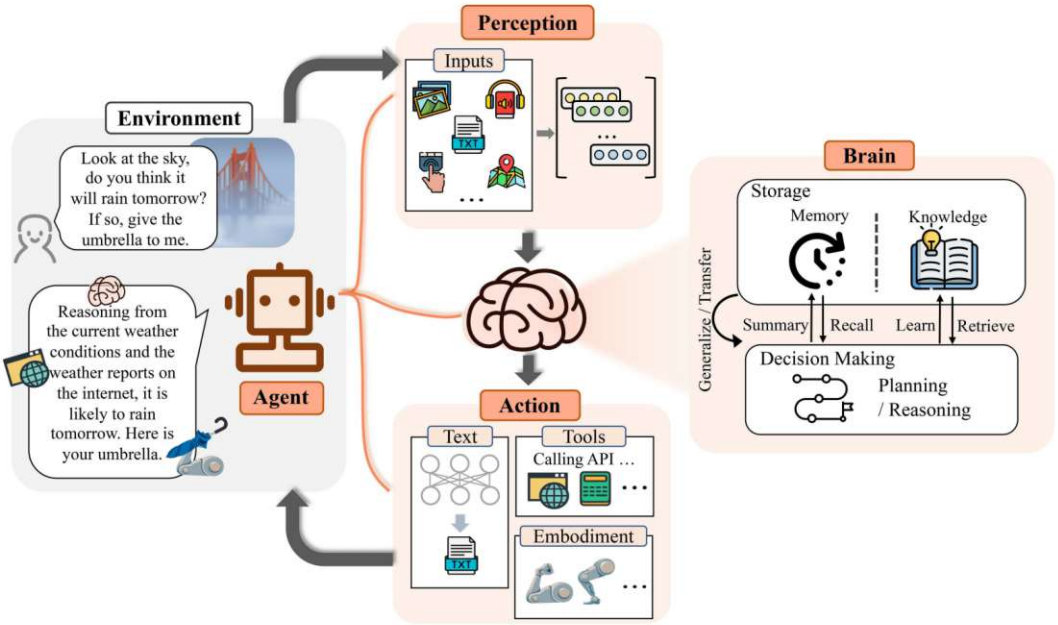

Figure 2.2: Conceptual framework of an LLM-based agent with three components: brain, perception, and action [41].

The perception component comprises the process of receiving information via an input. While this is often textual, it may also be visual or auditory data, depending on the specific use case.
The action component refers to the outputs that the LLM agent can produce. This includes generating textual responses and invoking tools. In the context of actions, tools enable the LLM agent to modify a current system state or interact with other external resources.

Yao et al. [42] propose an approach that improves upon the limitations of immediate, full-reasoning and static planning in typical Chain-of-Thought prompting [43]. They introduce a more dynamic solution through their `ReAct` (Reasoning and Acting) framework, which interleaves verbal reasoning with task-specific perceptions and actions, enabling the LLM to react and adjust its strategy in response to the current context. This happens in decision cycles: "*Thought* $\rightarrow$ *Action* $\rightarrow$ *Observation* $\rightarrow$ *Thought* $\rightarrow$ ..." until a final answer is produced. This methodology significantly improves performance over Action-Only (`Act`) baselines, which do not allow any updates to action plans between steps. This approach is used in parts of the RAG prototype implementation, specifically the Replanner agent, which is described in detail in Section 7.3.2.

Tools In the context of AI and LLM agents, Wang et al. define a tool as: "*a function interface to a computer program that runs externally to the LM, where the LM generates the function calls and input arguments in order to use the tool*" [44]. Such tools enable an LLM to extend its internal knowledge and interact with its environment. In addition to the previously mentioned categories of perception and action tools, Wang et al. introduce a third category: computational tools.

- **Perception tools:** These tools gather information from the external environment, such as calling a weather-API or retrieving text from a document.
- **Action tools:** These tools modify the environment or execute concrete actions, for example, by sending an email.
- **Computation tools:** Tools in this category do not directly perceive or modify the environment but perform complex computational tasks such as doing mathematical calculations or translating text into another language. They are often used in combination with other tools; for instance, a perception tool may first be needed to retrieve the data on which a computational tool then performs calculations.

Given the central role of tools in enabling agentic AI, a key challenge is to teach LLMs to use them reliably. Hsieh et al. [45] demonstrate a method for enabling LLMs to use tools effectively in a zero-shot setting (i.e., without providing the model any prior examples of a task), eliminating the need for the few-shot examples traditionally required, as proposed by Brown et al. [46]. Although LLMs possess a broad knowledge base, they often lack task-specific knowledge on how to approach or solve a user's request. As a result, their performance typically improves when a few examples of how the task is correctly performed are provided [46]. However, in a LLM-agent-with-tool-calling setting, these few-shot examples can be replaced by proper tool documentation. A textual description of what a tool does and how it is invoked is sufficient for LLMs to achieve performance similar to that of a few-shot setting [45].

2.6 Evaluation Methods of Retrieval Augmented Generation

The systematic evaluation of RAG systems presents a significant challenge [47]. Manual human evaluation is infeasible for assessing the correctness of long, unstructured responses. Additionally, the modular nature of RAG systems must be accounted for, and both the retrieval of relevant context and the generation of outputs from the retrieved context must be considered.

Recent research tackles this gap by employing *LLM-as-a-Judge* techniques [48]. Frameworks such as RAGAs [49], ARES [50], DeepEval [51], TruLens [52], RAGChecker [53], or RefChecker [54] utilise *LLM-as-a-Judge* in different granularities, but the concept is similar: the original query, reference context and generated output of a RAG system are fed to an LLM, which helps to evaluate the correctness of the RAG system. This can be either direct, where the LLM judge returns an evaluation score, or indirect, where the LLM serves only as a preprocessor and relevant metrics are computed from the preprocessed data.

The following metrics are commonly considered for evaluating RAG systems:

- **Factuality:** This metric describes whether a generated response is factually correct [55]. Considering a range between 0 and 1, higher values denote a higher degree of correctness [49].
- **Faithfulness:** The faithfulness measures how factually consistent a response is with the retrieved context [49]. In contrast to factuality, generated outputs may be faithful within the given context, but factually wrong.
- **Groundedness** The groundedness is a similar concept to the faithfulness. It captures how well a response is grounded in the retrieved context [56].
- **Answer Relevancy:** This measure considers the fact that the generated answer should address the input query that was provided. A higher value indicates greater focus on the actual question [47].
- **Context Relevancy:** This metric measures how well the retrieved context corresponds to the given query. The retrieval should not only return relevant information, but also be focused and contain only as much information as needed; retrieving all documents is equally unhelpful as retrieving no documents [47].
- **Summarisation:** The summarisation metric measures how well a summarised text captures the important aspects of the original retrieved context [49].

In this thesis, the metric concepts of the RAGAs framework [49] are applied, as they do not require human-annotated reference answers. Only input queries, retrieved contexts, and generated outputs are needed. Furthermore, only the **faithfulness**, **groundedness**, **answer relevancy**, and **summarisation** are relevant and used in the context of this thesis, and their computation is described below. Faithfulness is used in place of factuality because the evaluation focuses more on the RAG system than the correctness of the ETL pipeline. Context relevancy is not computed because the RAG implementation does not use a common vector-storage similarity-based retrieval [57]. Rather, a fixed set of retrieval tools is implemented, returning highly relevant data for different information aspects. For further information, see Section 7.4.

Faithfulness RAGAs computes the faithfulness based on the original user query, retrieved contexts and generated response. In the first step, based on user input, an LLM breaks the generated answer into atomic statements (S). Afterwards, the LLM judges whether each of these atomic statements is supported by the retrieved context. The set of supported statements according to the LLM is denoted as V. With this, the faithfulness score f is calculated as $f = \frac{|V|}{|S|}$.

Groundedness The *ResponseGroundedness*, as RAGAs calls it, is taken as a complementary metric to the faithfulness, since it measures the same aspect, but in a different way. To do this, an LLM evaluates two aspects using both the RAG-generated response and the corresponding retrieved context. First, it judges whether the retrieved context fully (=2) or partially (=1) supports the generated response, or not at all (=0). The second aspect is evaluated on the same scale but refers to the strength of the connections among the claims in the response. These two scores are then averaged and normalised to the 0-1 range, yielding more coarse-grained values than the faithfulness score.

Summarisation The summarisation is a relevant factor for the concrete RAG design of this thesis, as described in Chapter 7. First, a set of keyphrases is extracted from the retrieved context, and a set of questions Q is generated based on these keyphrases, such that the retrieved context always answers the questions. Afterwards, it is assessed whether the context summary answers the generated questions. The summarisation score s is then calculated as $s = \frac{|\hat{A}|}{|Q|}$, where A is the set of correctly answered questions by the summary. In addition, a conciseness score c is computed to confirm that the retrieved content is effectively summarised. This conciseness score is calculated as:

$$c = 1 - \frac{min(l_S, l_C)}{l_C + \epsilon} \tag{2.1}$$

where l_S is the character length of the summary and l_C is the length of the retrieved context. These two scores, s and c, are equally weighted, and the final *qa_score* is returned as:

$$qa\_score = s * 0.5 + c * 0.5 \tag{2.2}$$

An additional correction is used to set the final *qa_score* to zero if s is exactly zero. The conciseness of a summary should not be attributed to the score if the summary does not actually contain any relevant information from the retrieved context.

Answer Relevancy To compute the answer relevancy, an LLM takes the original query and the generated answer of a RAG system as input and generates several questions based on the answer, including a boolean value indicating whether the answer addresses the question. For example, the two answers "*I do not know what the capital of France is.*" and "*The capital of France is Paris.*" both lead to the generated question "*What is the capital of France?*". If none of the generated questions is answered, the relevance score is 0. Otherwise, a vector embedding of both the original query and the generated questions is created, and a cosine similarity score in the range 0-1 is calculated to get the degree to which the RAG-generated answer addresses the original query.

Using these metrics, the RAG implementation in this thesis is automatically evaluated, together with manual inspection of RAG-generated outputs. More details can be found in Chapter 8.

It must be noted that, while *LLM-as-a-Judge* approaches are frequently used, they are not perfect. Zheng et al. [48] present in their original work regarding *LLM-as-a-Judge* that LLM judges achieve over 80% agreement with human annotators. This corresponds to the level of agreement between humans. However, more recent research shows that this level of agreement and especially the judgment consistency depend on the experimental setup [58]–[60]. Wang et al. [59] and Shi et al. [60] uncover positional bias when LLMs judge two or more choices to pick the best option. Their results show that the ordering of the options in the input sequence significantly affects the outcome of the judgment. Moreover, Wang et al. [58] describe a *Score-Comparison Inconsistency*, where independently lower-rated responses outperform responses with a higher rating in pairwise comparisons. While these specific biases do not affect the approach of this thesis, as results are evaluated independently, more research is needed to uncover other potential limitations of *LLM-as-a-Judge*. Imtiyaz et al. [47] note in their review of currently available RAG evaluation frameworks that none of them provides a comprehensive solution and that further research is needed to establish reliable assessment techniques to ensure dependability and security of RAG systems.

CHAPTER 3

State of the Art

In this chapter, related work relevant to the approach of this thesis is reviewed. First, existing research in the field of EDM is discussed in Section 3.1, focusing on methods that analyse student software repositories to gain insight into learning processes and project quality. This includes approaches that apply SQM and visualisation tools which are intended to support educators in interpreting SE student data. Subsequently, advances in the use of LLMs for data analysis are presented in Section 3.2, highlighting their ability to process heterogeneous and unstructured data. Building on this, their application to MSR is discussed. Next, research on the use of RAG and LLMs in educational contexts is presented in Section 3.3. In addition, applications of RAG and LLM agents in software development processes are outlined in Section 3.4, with a focus on supporting code review and improving code quality.
Finally, the chapter summarises the main differences between the approaches found during the literature review and the RAG system proposed in this thesis (Section 3.5). This comparison highlights the research gap addressed by the RAG prototype.

3.1 Educational Data Mining

The field of EDM has evolved since 1995 to enable the effective tracking of student learning experiences [61]. Over time, this has shifted from face-to-face interactions with students to submissions of electronic artefacts. Therefore, EDM aims to gain insight into students' learning progress across various environments. EDM is an active area of research in SE education, as software repositories managed with a VCS offer many opportunities for automated analysis. This section introduces several approaches where EDM is applied.

Hamer et al. [62] examine the source code quality of undergraduate student projects by analysing selected commits using SonarQube for their impact on code complexity, security vulnerabilities, and maintainability. Their approach allows educators to address

declining project quality and students' knowledge gaps promptly. Hamer et al.'s results show that both complexity and maintainability increase over the project timeline, as expected given the added functionality. Still, aspects such as code duplication suggest that students are not properly maintaining their code. Through this commit-impact analysis, educators can address such issues already during the project.

Koetter et al. [63] analyse student software repositories, primarily to assess maintainability. Subcategories include reusability, analysability, modifiability, and testability. They, for the most part, use the object-oriented CK metrics to assess their quality. Their approach provides informed insight into the maintainability of student projects, and also identifies important factors for SE student projects in general: first, student teams with an even workload distribution deliver better results. This includes regular communication among team members to share project status updates. Also, the number of team members is a relevant factor: larger teams delivered software of lower quality.

Available Visualisation Tools Some EDM approaches combine data mining of student artefacts with visualisation techniques, to enable educators to gain better insight into various aspects of the students' work. Two tools are highly relevant to the context of this thesis, as both are used by tutors and study assistants for the same SE course in which the proposed RAG system is evaluated.

The visualisation tool `Binocular`[1] by Grabner et al. [64] has been used at TU Wien for several years to analyse different aspects of student projects. With this tool, both project data, such as source code ownership, and process data, such as contribution patterns, can be visualised. This way, the progress of the entire project and of individual students is tracked visually, rather than requiring educators to interpret raw data.

`Gizual`[2] was developed by Schintler and Steinkellner at TU Graz [65], [66]. It specifically tracks the authorship and age of code. With these insights, educators can analyse software repositories to identify implementation hotspots among student team members and to determine which project modules are continuously revised.

3.2 Large Language Models for Data Analysis

Historically, extracting information from heterogeneous sources has required domain-expert knowledge to manually design complex data pipelines [67]. This has shifted as the capabilities of LLMs have increased, as these models now natively possess complex data-comprehension capabilities. This advancement allows them to bridge semantic knowledge gaps, thereby reducing dependence on human experts to interpret complex data structures. Recent research demonstrates implementations of such LLM data science agents in practise [67]. Building on these developments, recent work has applied LLMs

[1] `https://github.com/INSO-World/Binocular`, Accessed: 11.03.2026
[2] `https://github.com/gizual/gizual`, Accessed: 11.03.2026

to the domain of SE, particularly MSR, where heterogeneous repository artefacts must be analysed and interpreted.

Romero-Arjona et al. [68] explore the use of LLMs in the field of MSR, identifying 85 studies in total. 70% of these approaches use LLMs for classification or generation, leveraging heterogeneous, unstructured artefacts from software repositories. For instance, Colavito et al. [69] investigate the use of GPT-like LLMs to categorise GitHub issue reports based on their textual content. Their results demonstrate that a zero-shot approach using GPT-3.5 achieves classification performance similar to state-of-the-art BERT models, eliminating the need for manually annotated training data. Zhang et al. [70] employ LLMs to generate pull request titles by formulating the task as a one-sentence summarisation. By combining pull request descriptions, commit messages and linked issues, they evaluated several state-of-the-art models. They found that their best-performing model generated titles that evaluators preferred over the original human-written ones. Other categories identified by Romero-Arjona et al. include the use of LLMs data extraction and retrieval tasks, as well as the prediction of future developments in the SE process [68].

These approaches demonstrate that LLMs are increasingly being applied to analyse heterogeneous artefacts from software repositories. However, they often focus on isolated tasks within a repository [69], [70], rather than integrating multiple data sources. As a result, they provide only a partial view of the development process and lack a holistic understanding of project dynamics.

3.3 Retrieval Augmented Generation in Education

The following section describes several different approaches to utilising RAG for educational purposes. While some of the referenced works do not explicitly state their methods as RAG, they employ closely related techniques by incorporating external resources into the generation process, enabling an LLM to draw on information beyond its internal knowledge.

Doughty et al. [71] propose a RAG-like system for generating Multiple-Choice Questions (MCQs) for Python programming classes. GPT-4 is used to generate questions based on a high-level course context (i.e., course name and description) and on more specific medium-level learning objectives. Using this input and the system's own MCQ design resources (MCQ principles, example questions, and required output format), the model generates questions that are then evaluated on several criteria: their alignment with the currently given learning objective, whether they contain exactly one correct answer alongside high-quality distractors, and, when applicable, whether code fragments are both syntactically and logically correct. The authors compare 651 LLM-generated MCQs with 449 human-crafted questions across 246 learning objectives to assess whether the generated questions achieve similar quality as the human-crafted ones. Their results show that the LLM-generated MCQs match the quality of those created by humans and exhibit

strong alignment with the learning objectives. Consequently, the approach significantly reduces the time needed for assessment preparation.

Phung et al. [72] focus on the other end of student assessment, by generating feedback on student-written Python programs. Their system, PyFiXV, uses Codex (a GPT-3-based LLM fine-tuned on millions of GitHub repositories [73]) to analyse students' solutions, generate a correct version of the submitted code and provide comprehensive textual feedback, similar to how tutors would give feedback. An additional component of PyFiXVis a run-time validation step that checks whether the produced feedback is suitable for sharing with students. If the feedback does not clearly explain what is incorrect and where an error is located in the submitted program, the validator rejects it, forcing the LLM to repeat the process. This helps ensure high-precision feedback, which is crucial before deploying such systems in a classroom environment.

Xie et al. [74] propose a LLM-based grading system `Grade-Like-a-Human` for the domain of Automated Short Answer Grading (ASAG). Short-answer responses are typically graded using predefined grading rubrics, which introduces several challenges: First, crafting high-quality rubrics for all questions is time-consuming, and each rubric must accurately capture the degree of correctness of given answers. Second, rubrics must be consistent and fair. An answer should receive similar scores across several grading sessions. This is particularly difficult when using LLMs, as the outputs may vary for identical inputs. `Grade-Like-a-Human` addresses these challenges with a multi-agent system consisting of three stages. First, grading rubrics are generated based on an initial human-created template and the set of student answers from the exam. The finalised rubric is then used in the second stage, where the students' answers are graded. In the third stage, a post-grading review detects and revises unreasonable results that may occur due to the LLM's inherent randomness or hallucinations. Their experiments demonstrate improved performance compared to existing ASAG systems, though the approach still has limitations, for instance, the time overhead introduced by the multi-agent design.

Zhou et al. [75] also focus on generating feedback using GPT-4, in this case to support and encourage primary school students during their learning process. Because the system must produce cumulative feedback over an extended learning period, the authors address the challenge of handling large, heterogeneous input data by preprocessing it and deriving a set of 34 individual tags. Each tag is assigned a description to one of three categories: *Basic Analysis*, *Knowledge Category Analysis*, and *Ability Analysis*. For each student, the LLM identifies which of these tags are present or absent and then uses this information to generate a comprehensive feedback report. This report covers the three analysis categories, additional recommendations, and an overall summary. An evaluation of the method by 32 primary-level mathematics teachers indicated that the generated reports were effective, provided clear reasoning and reduced the teachers' workload.

`SteLLA` is a system proposed by Qiu et al. [76] in the domain of ASAG of college-level students. They present a so-called R-RAG approach specifically designed for grading, in which the LLM (GPT-4 in their experiments) receives not only the exam question and the student's answer but also a grading rubric and an instructor-provided reference answer.

This setup is inspired by human assessment of responses: it evaluates whether the student addresses the asked knowledge point at the semantic level, rather than comparing the answer at the syntactic level using textual similarity. The approach shows high agreement with human graders and also generates feedback. One limitation they observe is GPT-4's tendency to over-infer from students' answers, especially when the answers are factually correct but do not actually address the question.

The increasing relevance of such autonomous, agentic systems is evident not only in education but also in industry forecasts. In 2024, Deloitte predicted that approximately 25% of companies utilising GenAI will have pilot or proof-of-concept projects based on LLM agents by the end of 2025, with adoption into existing workflows in the second half of the year [77]. Based on this rapid acceleration, Deloitte further anticipates that the percentage of companies testing agentic applications will increase to 50% by 2027.

3.4 Retrieval Augmented Generation in Software Development Processes

The following section describes several different approaches to utilising RAG for code review workflows.

Aðalsteinsson et al. [1] examine the effectiveness of using LLMs in the code review process at WirelessCar Sweden AB. Through interviews, they identify two major challenges in the existing workflow: first, high cognitive load due to frequent context switching; second, a lack of complete contextual information. To address these issues, they introduce a RAG-based workflow. Their study does not primarily evaluate performance metrics but instead focuses on the perceived usefulness of an LLM-assisted code review system. Two modes of assistance are explored: the first is a direct code-review mode, in which the LLM generates a summary of the reviewed code that highlights major changes and potential points of interest for the human reviewer to utilise. The second is a passive, on-demand mode in which the reviewers can query the LLM for additional information during the manual review. The results indicate that both modes add value, with the direct review mode being preferred, as it "*confirms the developer's own thoughts or surfaces something they might not have otherwise caught*" [1]. Despite these benefits, there are still limitations, including developers' lack of trust in the system and the risk of over-reliance on LLMs suggestions. The authors emphasise that AI assistance should complement, rather than replace, human reviewers to ensure accurate and reliable outcomes.

Rasheed et al. [78] address a closely related problem in the same domain by examining how an LLM-based AI agent system can enhance the code review process. Their approach focuses on two main aspects: First, the system performs code reviews and documents development best practises, and finds bugs and vulnerabilities. Second, it produces recommendations and concrete fixes for the detected issues. In addition, the system performs code optimisation to enhance quality and educate developers on best practises. This distinguishes their approach from traditional static code analysis tools, such as those

discussed by Balachandran et al.[79] and Singh et al. [80], which typically only report the detected issues and do not provide actionable feedback. To achieve their results, Rasheed et al. employ four specialised LLM agents dedicated to *Code Review*, *Bug Report*, *Code Smell* and *Code Optimisation*. Their findings indicate that their system valuably supports the code review process by both improving code quality and contributing to developer education.

3.5 Distinction from Current Research

While several studies employ related strategies and techniques, the literature review conducted for this thesis indicates that several limitations remain when applying these methods to the assessment of student SE projects. Many state-of-the-art approaches typically focus on specific dimensions, leaving notable gaps:

- Approaches grading SE students based primarily on statically retrieved metrics [62]–[64] lack a broader contextual understanding of the students' development process.
- Existing applications of LLMs in software repository mining often address isolated tasks, such as classification or summarisation of isolated artefacts [68]–[70], and do not reconstruct the broader development context.
- Existing RAG approaches in educational contexts typically only target highly structured and narrow assessment formats such as MCQ or ASAG [71], [72], [74]–[76].
- LLM agents used to support code review and improve code quality [1], [78] are primarily designed for industry workflows and do not account for academic grading criteria.

As mentioned in earlier sections, both grading students and comprehensive code review often lack sufficient contextual information, leading to incomplete results [1], [2]. This thesis addresses this limitation by incorporating multiple complementary data sources, including statically retrieved SQMs and information extracted from the Git repository. Additionally, it integrates student groups' GitLab data, capturing issue tracking, time records, and documentation for both the application and the agile development process. Together, these sources constitute the external knowledge base for the RAG prototype, providing the context required to generate comprehensive reports. Finally, this approach is intended to support study assistants during grading by reducing manual effort and providing a comprehensive, more consistent assessment basis.

CHAPTER 4

Methodology

The following chapter presents the methodological approach used to address the research questions of this thesis. The research process is organised into five main phases, following the Design Science methodology proposed by Wieringa [81].

Following the Design Science framework, this thesis's research process includes problem investigation, treatment design and evaluation. The problem investigation addresses missing contextual information and fragmented data when assessing SE student projects. The designed artefact is a RAG system that integrates heterogeneous data and generates assessment reports. The evaluation corresponds to treatment validation, in which the usefulness and impact of the designed artefact are assessed using archived data and expert feedback before deployment in the real grading process.

First, to investigate the shortcomings of the status quo, a literature review is conducted to establish the current state of the art regarding both commonly used SQM and the applications and advantages of RAG. In parallel, a preliminary feasibility analysis of several open-source and proprietary LLMs is performed to identify suitable candidates for the prototype implementation. Building on the insights from the literature review, semi-structured expert interviews are conducted to identify information needs specific to project assessment within the undergraduate SE course at TU Wien. Based on this foundation, a proof-of-concept RAG system is designed and implemented. This prototype extracts and integrates heterogeneous data from SE student projects and generates comprehensive reports, along with intermediate data outputs, to support the transparency and verifiability of its reasoning process. The evaluation phase addresses the research questions through three complementary evaluation dimensions, which are described in detail in Section 4.4.

4.1 Literature Review

The research conducted in this thesis spans four, often overlapping domains: EDM, SQM, automated assessment and grading (in computer science), and RAG (in education). To gain a comprehensive overview of the state of the art, a semi-structured literature review is conducted based on a model proposed by Snyder [82].

To provide an overview of commonly used SQMs in industry and education, the work by Begel et al. [83] and Buse et al. [84] serves as a starting point for further research. Google Scholar[1] serves as the primary search engine, using search terms such as "software quality", "software quality metrics", and related synonyms. To further refine the literature base regarding EDM, RAG and grading SE students, theses by Weiß [85] and Jansen [86] serve as initial references, from which relevant works are identified through citation chaining and additional keyword searches. The platforms Connected Papers[2] and Litmaps[3] also prove useful for identifying thematically related research papers.

4.2 Semi-Structured Expert Interviews

Semi-structured interviews with study assistants are conducted to collect SQM-related information needs in the SE course context. The interviews are guided by an online questionnaire[4]. The majority of questions are closed-ended on a Likert scale [87] to assess the relevance of various SQM. Several open-ended questions are included to clarify and identify missing quality metrics. The results of these interviews guide the implementation process by determining which SQM are considered relevant and therefore computed and made available to the RAG system for retrieval, and which are excluded from the prototype.

4.3 Proof of Concept

Based on insights from the literature review and expert interviews, a prototype is implemented. This prototype comprises two main components: an ETL pipeline that collects all relevant information in a structured format for later retrieval, and the core component of the RAG system. The latter takes user questions as input and generates comprehensive reports grounded in the retrieved information about SE student projects. The ETL implementation incorporates the results of expert interviews regarding relevant SQM. The RAG implementation is based on a preliminary feasibility analysis and a technology selection process for available LLMs, in combination with recent research in RAG [36]–[38], [88]–[90].

[1] https://scholar.google.com/, Accessed: 25.02.2026.
[2] https://www.connectedpapers.com/ , Accessed: 25.02.2026.
[3] https://app.litmaps.com/, Accessed: 25.02.2026.
[4] https://workspace.google.com/intl/de_at/products/forms/, Accessed: 25.02.2026.

4.4 Evaluation

The evaluation is divided into three main parts: first, an empirical analysis is conducted, in which 150 reports generated by the RAG prototype are analysed for factual correctness and consistency in reporting information. Second, a quantitative evaluation is conducted with tutors performing a defined assessment task, once without RAG-generated reports and once with them as aid. This comparison provides insights into how the reports affect the assessment time. Finally, both tutors and study assistants participate in semi-structured interviews to qualitatively evaluate the generated reports. The specific setup of each evaluation stage is further described in Sections 8.1 to 8.3. This observer triangulation [91] improves evaluation precision by incorporating multiple perspectives and comparing results across different aspects.

The evaluation is carried out at the end of the university semester using archived data. This third-degree collected data [91] is available via a regular export process at the end of each semester. The reports are generated retrospectively for student groups that have already been graded. This setup reflects a treatment-validation approach using expert opinion, as described by Wieringa et al. [92], in which a treatment (i.e., the RAG system and generated reports) is validated before full implementation (i.e., use for assessment during an ongoing semester). This approach also aligns with the definition of offline validation by Wohlin et al. [93], minimising the risk when introducing a new technology relative to current practice. Using this evaluation setup avoids interfering with real student grading processes during the semester and, therefore, experimenting with students' grades. The participating tutors do not evaluate random student groups, but the same groups they supervised during the semester. This reflects the realistic scenario in which the reports would be applied.

4.5 Research Questions

This section outlines how each research question defined in 1.2 is addressed within the overall research design.

RQ1. The empirical analysis examines to what extent the RAG-generated reports are faithful to the underlying data and how consistent the results are. A lack of either faithfulness or consistency would make the reports less valuable. First, incorrect data and conclusions may, naturally, not be used to assess students. Second, a RAG system that produces widely varying results across generated reports can also not be used as a support tool, as inconsistent assessments risk introducing unfairness.

RQ2. RQ2 is answered through the quantitative evaluation, which measures the time required for tutors to perform defined assessment tasks once without and once with RAG-generated reports. Subsequent interviews with the tutors provide insight into their

information discovery and into whether using the generated reports can help them gain a more complete view of the student project.

RQ3. RQ3 is addressed through semi-structured interviews with both tutors and study assistants. Participants rate the trustworthiness and the transparency of reasoning in the generated reports (including the built-in intermediate data outputs), as well as the completeness of the information in the reports.

RQ4. Similar to RQ2, this research question is primarily addressed through the quantitative evaluation with tutors, complemented by subsequent interviews. While performing the defined assessment tasks, supported by a RAG report, their verification behaviour is monitored alongside time measurement. Furthermore, study assistants are asked during interviews how much verification is needed before these AI-generated reports may be used in the regular course context. From this, it is possible to assess whether the tutors inherently meet the verification requirements of study assistants, and where adjustments are needed.

CHAPTER 5

Quality Metric Needs in Software Engineering Education

Since the RAG-generated reports incorporate software quality metrics to a substantial extent, the relevant metrics for this SE course were first identified and evaluated. This chapter first presents the results of a literature search on commonly used SQM in industry and education (Section 5.1) and subsequently reports the findings from expert interviews (Section 5.2). In addition, a set of pre-defined evaluation questions was ranked by the experts (Section 5.2.3), and the top subset was later used in the final evaluation of the RAG system (Chapter 8).

5.1 Literature Search of Software Quality Metrics

To obtain an overview of commonly used SQM in industry and education, a literature search was conducted as described in Section 4.1. This resulted in a set of quality metrics derived from 24 research works across education ([2], [14]–[16], [62], [86], [94]–[97]), as well as industry and general research ([78], [98]–[110]). Table 5.1 shows which SQM were identified, and how often they appeared in the literature. Apart from the two most frequently mentioned metrics, the results are sparse because most studies focus on a specific subset of metrics tailored to their particular research context. Many metrics that occur only once are primarily relevant in an educational setting, such as those used for grading students rather than for general SE practice. Therefore, all metrics from Table 5.1 were included in the expert interviews, as each of them may still be relevant in the context of this thesis.

Table 5.1: Software Quality Metrics from Industry and Education.

Metric	Occurrences
Lines of Code	12
Cyclomatic Complexity	11
Written Documentation	6
Halstead Metrics	4
CK Metrics	4
Number of Commits	4
Number of Merge Requests [1]	3
Bugs (SonarQube)	3
Code Smells (SonarQube)	3
Files/Lines Added	2
Time Spent [1]	2
Contribution Frequency [1]	2
Commit Message Quality	2
MTTR	2
Security Vulnerabilities (SonarQube)	2
Cognitive Complexity [1]	1
Work Distribution [1]	1
Git Blame [1]	1
Code Duplication (SonarQube)	1
Tech Debt [1]	1

[1] Only found in research papers regarding education.

5.2 Interview Results

The expert interviews were guided by a questionnaire (see Appendix A) comprising primarily closed-ended Likert-scale questions about the relevance of specific SQM to the SE course. A small number of open questions were included to clarify the answers and identify missing metrics or additional requirements. The interviews were structured into four parts:

1. **Demographics**, collecting information on teaching experience and semester workload.
2. **Relevance of SQM** identified in the literature search.
3. **Relevance of other SQM** not covered in the literature.
4. **Rating of pre-defined questions** used later in the prototype evaluation.

5.2.1 Demographic

In total, six interviews with study assistants were conducted. All participants have substantial teaching experience of at least ten years, as shown in Figure 5.1. Most supervised either two or three student groups during the 2025 winter semester, while one study assistant supervised five groups. Given their extensive experience as educators, the interviewees have a well-established understanding of how grading in this course should be conducted and which aspects of student work are relevant for assessment.

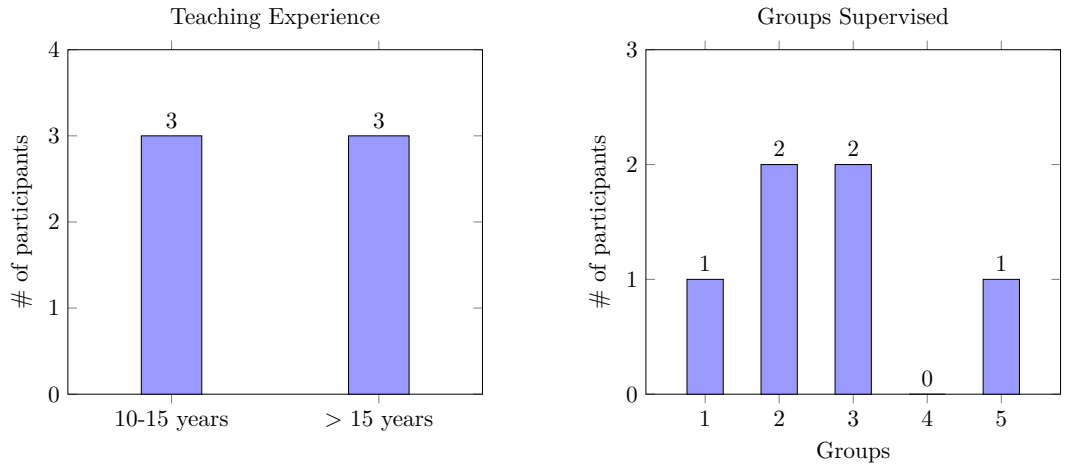

Figure 5.1: Experience and Supervision Workload of the Interview Participants

5.2.2 Relevance of Software Quality Metrics

The second and third parts of the interview aimed to identify which SQM address the study assistants' relevant information needs. Metrics rated as relevant by the participants were included in the prototype implementation, while irrelevant metrics were excluded. This avoids misleading the RAG system by suggesting that a metric is important simply because the data is available.

Lines of Code Figure 5.2 shows that lines of code were rated neither irrelevant nor highly relevant. This likely explains why the metric is frequently cited in the literature (see Table 5.1): it is easy to calculate and provides a rough estimate of project size. However, especially in an educational context, it has limited interpretive value when used alone. A student who writes many lines of code may mainly produce boilerplate or inefficient code, while another may implement a complex algorithm with relatively few lines.

Cyclomatic Complexity Opinions on this metric are divided (Figure 5.3). Three participants rated it as irrelevant, two as relevant, and the remaining two as neutral. With an average score of 2.5, cyclomatic complexity is considered to be of limited relevance for

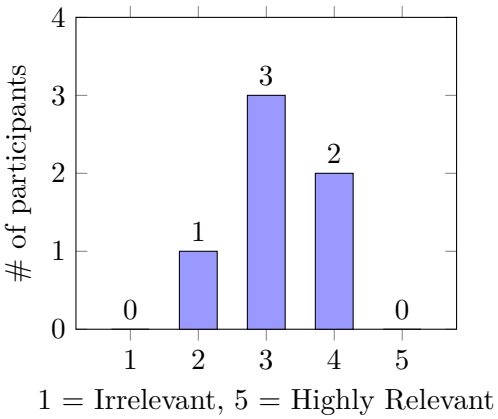

Figure 5.2: Are the Lines of Code (per Student) relevant?

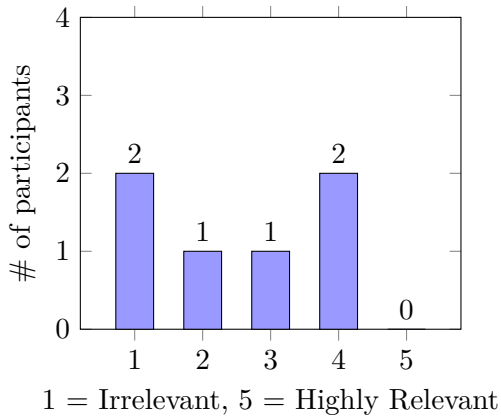

Figure 5.3: Is the Cyclomatic Complexity relevant?

this course. One participant noted that it is unsuitable because it does not reflect the overall development process or the contributions of individual students in any way.

Cognitive Complexity Cognitive complexity was rated as relevant by nearly all participants (Figure 5.4). This is attributed to the project's collaborative nature: students must read and understand each other's code during reviews and feature development. Thus, readability plays an important role alongside functional correctness.
One participant rated this metric as irrelevant, arguing that readability is subjective and depends on personal preferences, such as whether recursive functions or loop-based constructs are considered easier to understand.

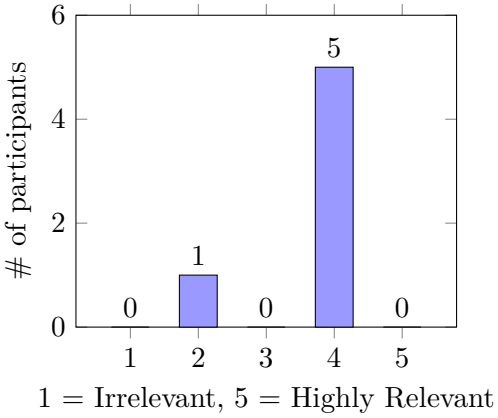

Figure 5.4: Is the Cognitive Complexity relevant?

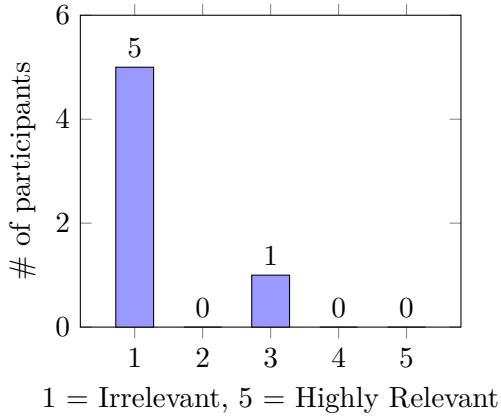

Figure 5.5: Are the Halstead Metrics relevant?

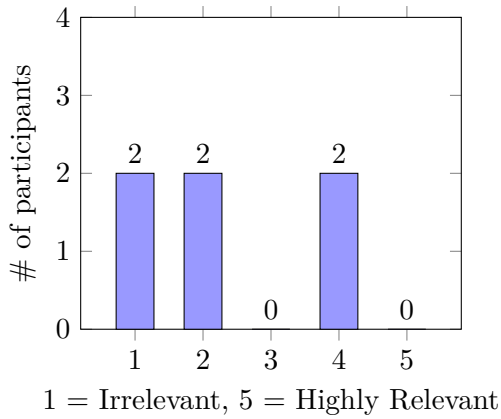

Figure 5.6: Are CK (Chidamber and Kemerer) Metrics relevant?

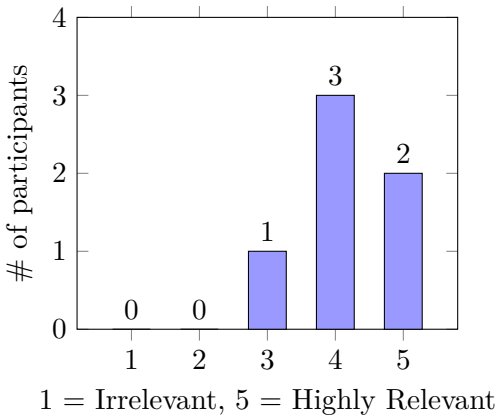

Figure 5.7: Is the Number of Commits relevant?

Halstead Metrics All but one participant rated Halstead metrics as irrelevant (Figure 5.5), stating that they provide little useful insight for the assessment process.

CK Metrics Opinions on CK metrics were mixed (Figure 5.6). While most participants considered them irrelevant, two rated them relevant due to their focus on object-oriented design. Since Java is used for the backend layer of the student projects, these metrics were regarded as conceptually important. However, they were not included in the prototype as explicit metrics.

Number of Commits This metric was rated as relevant by all participants (Figure 5.7). It provides a quick overview of student contributions and version-control practices, for example whether work is committed incrementally or in large blocks.

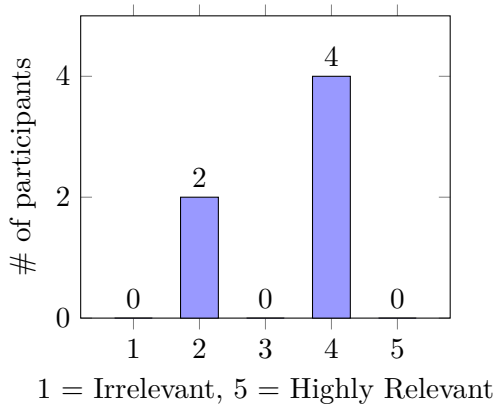

Figure 5.8: Are Files/Lines Added/ Deleted (per Commit) relevant?

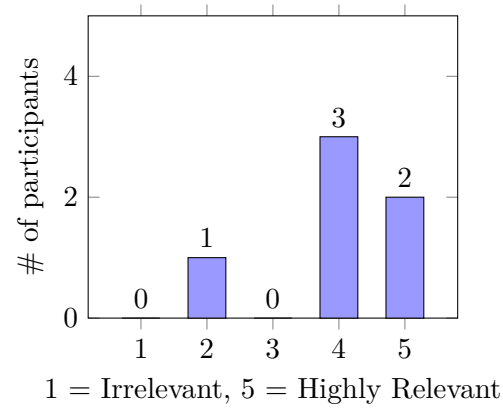

Figure 5.9: Is the Time Spent (per Student) relevant?

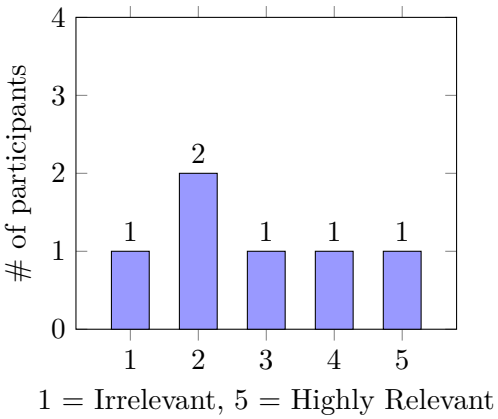

Figure 5.10: Is the Number of Bugs relevant?

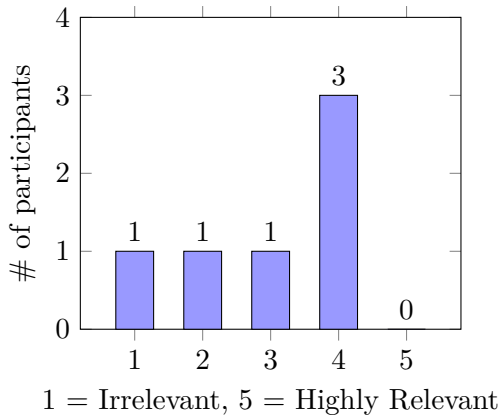

Figure 5.11: Is the Number of Code Smells relevant?

Files and Lines Added/Deleted Four participants rated this metric as relevant, and two as irrelevant (Figure 5.8). It complements the number of commits: students' commit logs with fine-grained commits typically exhibit fewer changes per commit, while others commit large chunks of code at once.

Time Spent Time spent by each student on the project was rated as (highly) relevant by most participants(Figure 5.9). One participant considered it less important than other criteria used in their grading process.

Number of Bugs The number of bugs detected by static code analysis is the most diversive metric (Figure 5.10). Ratings ranged from totally irrelevant to highly relevant. Supporters argued that many bugs indicate sloppy coding or insufficient code reviews. Critics on the other side argued that static code analysis is not fully transparent to students and is highly dependent on configuration settings. Due to these concerns and its overall slightly negative average rating of 2.83, this metric was not included in the implementation.

Number of Code Smells Code smells were rated more positively than bugs, with three participants rating it as relevant and only one participant rating it as totally irrelevant (Figure 5.11). Participants viewed them as indicators of adherence to coding guidelines and software design quality.

Work Distribution Work distribution between the students of the group was rated as highly relevant by three participants and neutral or relevant for two more (Figure 5.12), as the workload should be distributed equally among the team members. One participant considered it irrelevant, as they rely more on other assessment criteria.

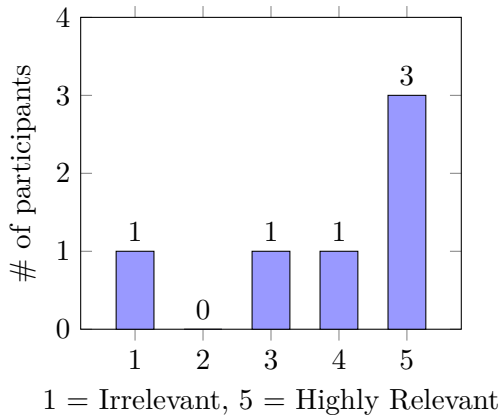

Figure 5.12: Is the Work Distribution (Time spent or Commits) relevant?

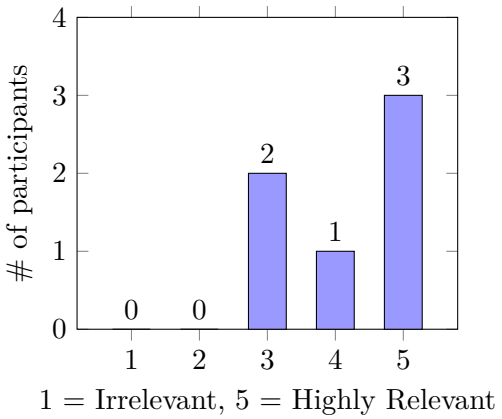

Figure 5.13: Is the Contribution Frequency (per Student) relevant?

Contribution Frequency The contribution frequency was rated as neutral or relevant by all participants (Figure 5.13). Since the course follows an agile Scrum process, work is expected to be distributed evenly throughout the semester rather than concentrated near deadlines.

Commit Message Quality Commit message quality was rated as relevant by four participants and irrelevant by two (Figure 5.14). While all participants agreed that commit message quality is an important aspect of SE, two participants rated it as irrelevant for assessing students, due to its relative unimportance compared to other grading criteria.

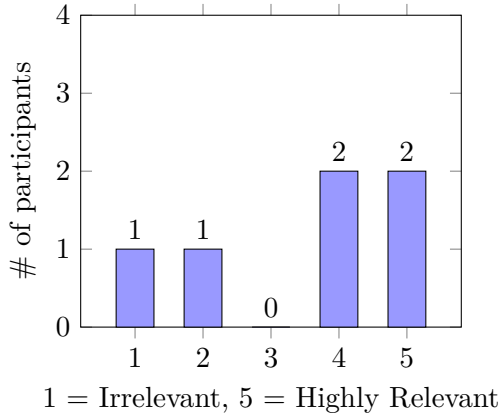

Figure 5.14: Is the Commit Message Quality relevant?

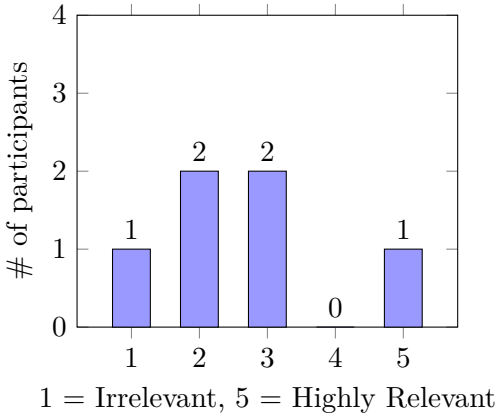

Figure 5.15: Is the Mean Time to Repair (MTTR) relevant?

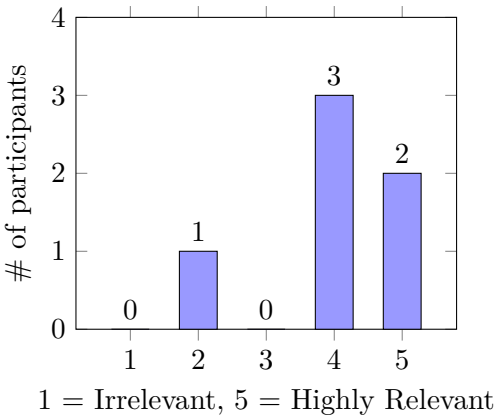

Figure 5.16: Is the Documentation Written relevant?

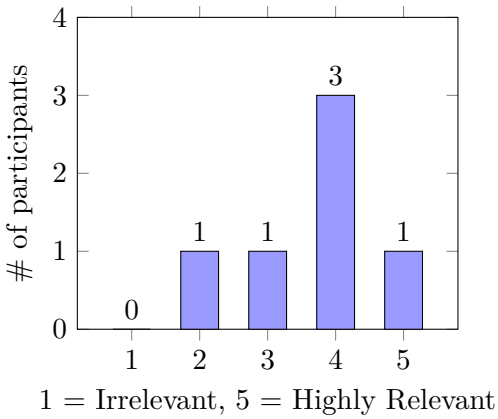

Figure 5.17: Is the Git Blame data relevant?

Mean Time to Repair The MTTR is another metrics with diverse opinions (Figure 5.15). The average rating of 2.67 and the fact that most participants rated it on the lower half of the scale indicate that it is not considered relevant in the specific context of this course. One participant rated the MTTR as highly relevant, and others agreed that it is an important metric *in general*, but not well suited for a student project.
Measuring MTTR is challenging as it requires a clear definition of when a bug is introduced. One possible definition is the time at which the bug is detected, which is easy to measure but may lead to large variations depending on the review and testing process. Another definition measures from the bug-introducing commit, which is more precise but requires significant effort to trace bugs back to their origin. Due to these challenges and the slightly negative overall rating, the MTTR is not included in the implementation.

Documentation Written While this metric may include code-level documentation, it mainly targets documentation outside the codebase. In the context of the course, this refers to the documentation written in the GitLab wiki. This metric was rated (highly) relevant by all participants except one (Figure 5.16). That participant regarded documentation primarily as support for the students themselves, rather than as an assessment-relevant criterion.

Git Blame Data The code-ownership data was rated as neutral or relevant by five of six participants (Figure 5.17), but all expressed reservations. This metric can be quickly distorted by refactoring operations, as such changes overwrite the ownership of code originally written by another contributor. Therefore, this metric should be used with caution and not interpreted in isolation. For this reason, some participants ranked the Git blame metric as less relevant.

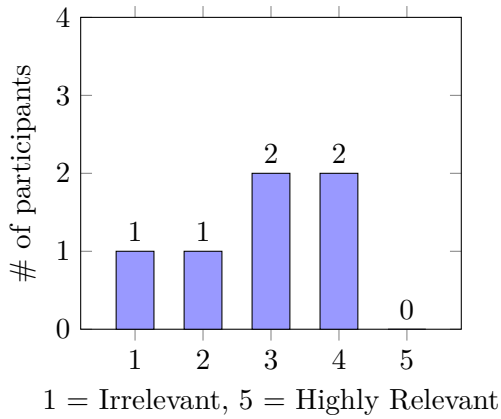

Figure 5.18: Is Code Duplication relevant?

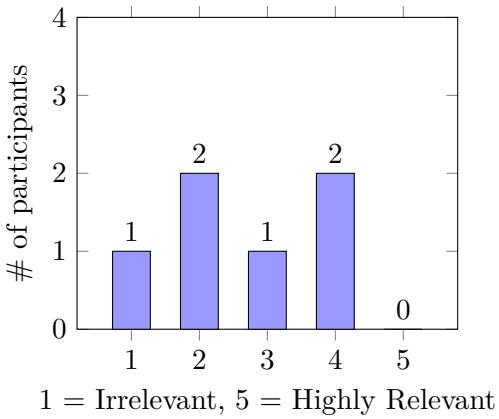

Figure 5.19: Is Tech Debt relevant?

Code Duplication Code duplication was rated as irrelevant, neutral, and relevant by two interviewees each (Figure 5.18), with no strong arguments expressed either in favor of or against it. Since this metric is derived from static code analysis, it requires no complex computation and is unlikely to be misinterpreted. Consequently, it is included in the implementation.

Tech Debt Two participants rated tech debt as relevant (Figure 5.19), for reasons similar to code smells: adherence to coding guidelines and sound software design help keep technical debt within acceptable limits. The remaining four participants, who rated it from neutral to totally irrelevant, argued that this metric largely duplicates the code-smells metric and introduces a time-based dimension, making it more difficult to interpret. Based on these arguments and an average rating of 2.67, the tech debt as a metric is not included in the implementation.

Security Vulnerabilities Figure 5.20 shows that this metric was rated as neutral or relevant by five participants. One participant rated it as irrelevant, stating that security-related aspects should rather be assessed through direct code reviews with students and do not require an additional metric.

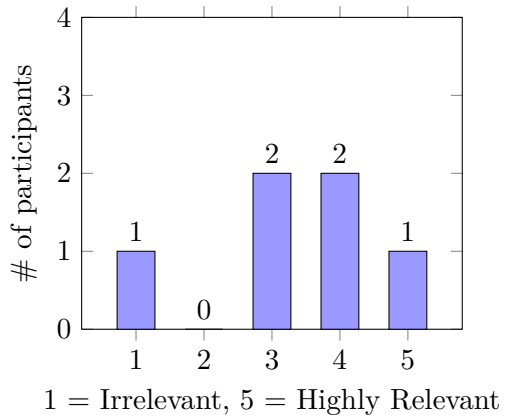

Figure 5.20: Are Security Vulnerabilities relevant?

In addition to the 20 metrics identified in the literature search, two further metrics were added to the questionnaire due to their expected relevance for the student project: **Test Coverage** and the **Bus Factor**. For the context of the course, the bus factor was defined in Definition 2.2.1 as *"How many team members have to suddenly drop out of the project before 50% of component knowledge is lost"*. Component knowledge was defined as the amount of code written in either the frontend or the backend of the project. Consequently, a project group may have two different bus factor values. Both metrics were rated exclusively as neutral or relevant (Figures 5.21 and 5.22), with the bus factor being the highest-rated metric overall.

Other Metrics The participants were asked whether the presented set of SQM covered all important aspects of the student project and whether any additional metrics were missing. Several participants mentioned metrics related to the agile workflow and its

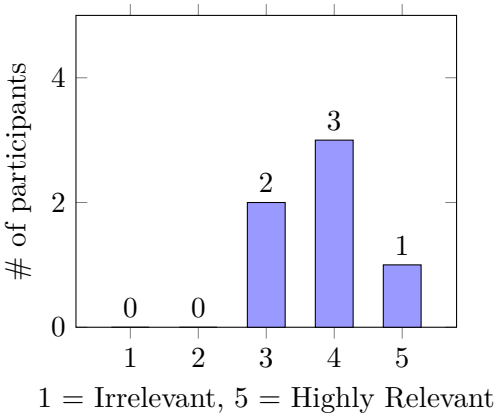

Figure 5.21: Is Test Coverage relevant?

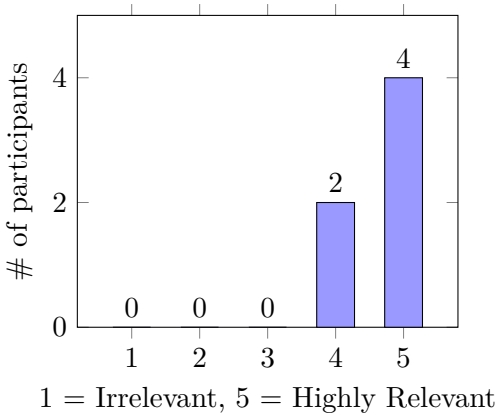

Figure 5.22: Is the Bus Factor relevant?

documentation. These included questions such as: Who reviews whose code? How many merge requests are reviewed? How well are GitLab issues documented? How consistently are branching and merging strategies applied? These aspects were incorporated into the implementation of the RAG system.
Additional suggestions concerned Non-Functional Requirements (NFRs), including the specification of realistic, project-specific NFR, as well as their implementation and compliance. Explicit quantitative calculations of NFR were not integrated into the RAG system. However, they can still be assessed by querying the RAG prototype to analyse relevant wiki documentation and cross-reference it with the commit histories and time logs.

Finally, the study assistants were asked how fine-grained static code analysis metrics should be reported. A value of 1 on the scale in Figure 5.23 denotes project-level aggregation, while a value of 5 denotes file-level granularity. Opinions varied across the full scale. Following the suggestion of one participant and early results on the RAG system's retrieval capabilities, metrics are stored at the component level, with one value for the backend and one for the frontend.

Table 5.2 summarises all rated SQM ordered by their average relevance score. Metrics with an average rating above 3 were included in the implementation of the RAG system. In addition, code smells were included due to their median rating of 3.5 and their perceived practical relevance. Code duplication was also included despite a slightly negative average rating, as it was already available through static code analysis and had a median rating of 3.0. The number of bugs with the same average rating was excluded due to the aforementioned concerns. By incorporating a wide range of different SQM, the greater context can be considered. Aspects where a single metric is insufficient to determine quality may be addressed by considering multiple metrics.

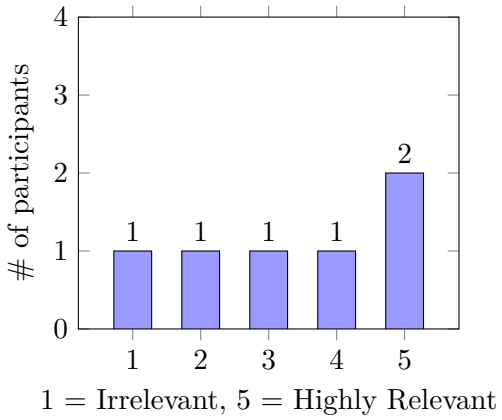

Figure 5.23: Should the Static Code Analysis Metrics be on a Per-File Basis?

Table 5.2: Relevance of Software Quality Metrics

Metric	**Average Rating**
Bus Factor	4.67
Contribution Frequency	4.17
Number of Commits	4.17
Time Spent	4.00
Documentation Written	4.00
Work Distribution	3.83
Test Coverage	3.83
Cognitive Complexity	3.67
Git Blame data	3.67
Commit Message Quality	3.50
Files/Lines Added/Deleted	3.33
Security Vulnerabilities	3.33
Lines of Code	3.17
Number of Code Smells	3.00
Code Duplication	2.83
Number of Bugs	2.83
MTTR	2.67
Tech Debt	2.67
Cyclomatic Complexity	2.50
CK Metrics	2.33
Halstead Metrics	1.33

5.2.3 Ranking of Evaluation Questions

Beyond identifying relevant SQM, the study assistants were asked which questions about student groups should be provided to the RAG system to generate reports that are later used during the evaluation of the RAG prototype (Table 5.3).

These questions were not included in prior research but were designed to cover different project aspects and levels of granularity. The top-rated question in Table 5.3 was intentionally formulated openly, allowing multiple valid interpretations. Other questions, such as those related to testing documentation in the GitLab wiki, were more specific and aim for only one clear answer.
The study assistants also proposed additional questions, which are shown in the lower part of Table 5.3. Overall, the rating reflected typical assessment concerns: What did each student contribute? Are there over- or underperforming team members? Details on the further evaluation procedure are provided in Chapter 8.

Table 5.3: Average Relevance of Questions to use during Evaluation (top) and Question Suggestions (bottom).

Question	Relevance
What did each student in the group work on?	4.83
Did someone in the group only work on minor fixes/documentation?	4.67
Is a student of the group carrying the team based on code volume?	4.50
Is there proper testing documentation in the group's wiki?	4.33
Are there Knowledge Silos in the project of the group?	3.67
Which top 5 issues remained open the longest in the group?	3.33
Which issues have the most time spent?	3.33
How often did the group merge code in the last week?	3.33
Where is the highest collaboration in the group's codebase?	2.83
Which file or module of the group was reworked the most?	2.50
Where are Security checks in the Backend made?	
Who broke the master branch pipeline most often?	
Did some merge conflicts provoke overly complex follow-up merge requests?	

5.3 Threats to Validity

Number of participants In total, six study assistants participated in the interviews regarding SQM information needs. This sample size is too small to allow for statistical generalisation. However, excluding the study assistants directly involved in supervising this thesis, the participants represent approximately half of all study assistants responsible for supervising student groups in the SE course at the time of the interviews. Therefore, although the sample size is limited, it still reflects a substantial portion of the target population in this specific course context.

Participant selection All participants were members of the core teaching staff of the SE course at TU Wien at the time of the interviews. Hence, the results regarding SQM information needs can not be directly transferred to other SE courses. This limitation is inherent to the goal of the expert interviews, which was to identify relevant quality metrics for one specific course rather than to obtain generally applicable conclusions. The participant selection did not follow a random sampling strategy. Instead, it combined purposive and convenience sampling [111]. While this approach ensured that experienced study assistants with domain knowledge were included, it may have influenced the results.

Influence of the interviewer The interviewer may have influenced participants' ratings of the relevance of the SQM. This risk was mitigated as much as possible by minimising the interaction between the interviewer and the interviewee during the rating of quality metrics. Nevertheless, this influence cannot be fully eliminated, as interaction was necessary to clarify the reasoning behind certain ratings. Such interactions may have

subtly shaped participants' responses and therefore poses a potential threat to internal validity.

CHAPTER 6

Extract, Transform, Load Implementation

This chapter describes the implementation of the ETL pipeline that transforms heterogeneous project data into a unified dataset suitable for analysis by the RAG system. The pipeline integrates data from multiple sources, including GitLab exports, source code repositories, wiki documentation repositories, and static code analysis results obtained by SonarQube.

The chapter is structured as follows: First, the organisation and structure of the raw input data are introduced (Section 6.1). Next, the overall ETL execution flow and configuration are described (Section 6.2.2), followed by a discussion of the database schema (Section 6.2.3). The subsequent sections explain the individual processing stages, including parsing GitLab data (Section 6.2.4), Git repository data mining (Section 6.2.5), automated plugin integration (Section 6.2.6) and static code analysis with SonarQube (Section 6.2.7).

6.1 Data Organisation

To enable testing of the prototype, archived data from past semesters were used. Only for the final evaluation were exports from the most recent 2025 winter semester processed. Consequently, in this initial proof-of-concept implementation, data is read from and transformed using GitLab exports rather than querying the GitLab Application Programming Interface (API) of live student projects. The course administrator provided the relevant GitLab exports in conjunction with the standard archiving procedure at the end of each semester. Therefore, no additional export cycle was required, as the data was already archived as part of the regular course workflow.

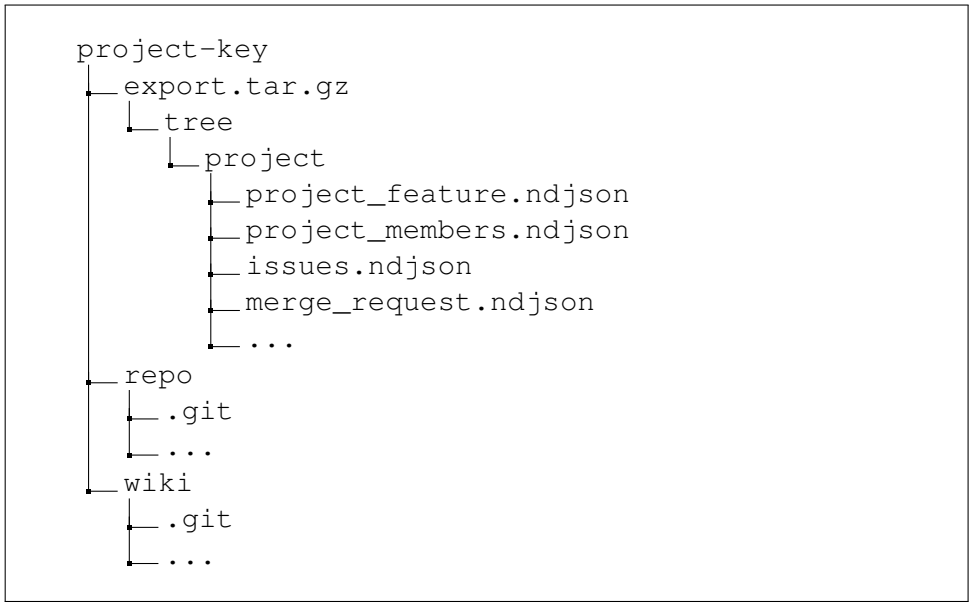

Figure 6.1: GitLab's Export Structure

Figure 6.1 shows the structure of the exported data of each student group's project. An `export.tar.gz` file is created during the export step, containing all relevant information from GitLab. In addition, the `repo` and `wiki` directories contain the respective source code repositories and the students' wiki documentation.

The GitLab export consists of multiple `ndjson` files. This format is a variant of JSON in which each object is stored on a single line, separated by newline characters, rather than contained within a single global root object. Each ndjson file contains information about a specific aspect of the GitLab project. Four of these files, shown in Figure 6.1, contain data relevant for the prototype implementation:

project_feature.ndjson This file contains a single entry with high-level project information. It serves mainly as a connection between the other files, as it stores the GitLab-internal project ID. Internally, during the ETL process, projects are primarily referenced by their `project-key` (see Figure 6.1), as the static code analysis setup in SonarQube also uses the same identifier.

project_members.ndjson This file lists all members associated with the project, including students, tutors, and study assistants, but only student data is relevant for analysis. The `access_level` (Listing 6.1) is therefore used as a filtering criterion, as an access level of 30 denotes the developer role, which is assigned exclusively to students. The username contains the students' matriculation numbers, which are required to identify contributors when matching commits to the corresponding GitLab member.

```
{
  "access_level": 30,
  ...
  "user": {
    "id": 1234,
    "username": "12345678",
    "public_email": null
  }
}
```

Listing 6.1: Example of an Object in `project_members.ndjson`

As a consequence of the `public_email` field being empty for all students, only email addresses found during data mining of the code repository are used to identify contributors.

issues.ndjson and merge_requests.ndjson These files contain all information related to GitLab issues and merge requests respectively (see Listings 6.2 and 6.3). Their structure is largely similar and includes titles, descriptions, creation and closing timestamps, and associated notes and timelogs. The notes object serves two purposes: it is used to extract discussion content between students and to identify students' full names, which are not stored in `project_members.ndjson`. Time logs are also extracted from these files. The `time_spent` values are given in seconds, and the `created_at` timestamp is used as the primary temporal reference. Although students may optionally specify a `spent_at` timestamp, this field is ignored. Students are encouraged to log time immediately after completing a task, making the `created_at` timestamp a more consistent reference point.
Issue-specific information includes the `label_links` field, which stores assigned labels (e.g., *test*, *feature*, *organisational*). Merge-request-specific fields include the `source_branch` and `target_branch` of each merge operation.

6.2 Data Transformation Implementation

This section describes the overall process of transforming the extracted data and providing an interface that enables the RAG system to retrieve it. Beyond collecting and storing data in a suitable structure, several mechanisms are required to ensure data completeness and the correctness of references. One necessary aspect is matching code contributors to the correct students, particularly when contributors do not use their standard student email addresses. This matching is performed automatically during the ETL process based on predefined rules. In cases where automated matching fails, a separate script enables manual verification and correction. Another important aspect is ensuring the correct Java build environment configuration, enabling SonarQube to reliably extract test coverage metrics.

```
{
  "iid": 49,
  "author_id": 1234,
  "description": "",
  "title": "bugfix: Input fields in profile data",
  "state": "merged",
  "source_branch": "fix/profile-data-input-ui",
  "target_branch": "dev",
  "created_at": "2026-01-29T19:13:15.965Z",
  "merged_at": "2026-01-29T19:25:25.187Z",
  "closed_at": null,
  "notes": [
    ...
  ],
  "timelogs": [
  ...
  ],
  ...
}
```

Listing 6.2: Example of an Object in `merge_requests.ndjson`

6.2.1 Choice of Technology

Before starting with the implementation, design decisions about the programming language and available frameworks had to be made. Python was chosen due to its widespread use in data science, machine learning, and AI [112], [113], as well as the author's prior experience with it. This choice corresponds well with the availability of frameworks and libraries for both ETL and RAG pipelines:

- The tarfile[1] library was used to read data directly from the GitLab export archives. This approach avoids manually extracting all export files before running the ETL process.
- GitPython[2] was used to iterate over code and wiki repositories and to access repository-level information.
- SQLAlchemy[3] was used as an object-relational mapping library for creating, populating, and querying the database.
- The pydantic[4] library was used for configuration management and data validation.

[1] `https://docs.python.org/3/library/tarfile.html`, Accessed: 18.02.2026.
[2] `https://gitpython.readthedocs.io/en/stable/`, Accessed: 18.02.2026.
[3] `https://www.sqlalchemy.org/`, Accessed: 18.02.2026.
[4] `https://docs.pydantic.dev/latest/`, Accessed: 18.02.2026.

```
{
  "title": "UI-MockUps",
  "iid": 52,
  "description": "",
  "author_id": 1234,
  "state": "closed",
  "created_at": "2025-11-30T06:47:05.753Z",
  "closed_at": "2025-12-20T04:50:53.804Z",
  "notes": [
    {
      "author_id": 1234,
      "note": "added 2h 30m of time spent at 2025-11-30 06:47:05 UTC
          ",
      "author": { "name": "Max Mustermann" },
      "system": true,
      "created_at": "2025-11-30T06:47:05.948Z",
      "updated_at": "2025-11-30T06:47:05.958Z",
    },
    ...
  ],
  "label_links": [],
  "timelogs": [
    {
      "user_id": 1234,
      "time_spent": 9000,
      "created_at": "2025-11-30T06:47:05.793Z"
    },
    ...
  ],
  ...
}
```

Listing 6.3: Example of an Object in `issues.ndjson`

- thefuzz[5] was used to provide fuzzy matching suggestions when manually assigning code contributors to students.
- Although not part of the ETL pipeline, LangGraph[6] was used for implementing the RAG system.

Building upon the choice of SQLAlchemy, the system's data layer ensures compatibility with a wide range of relational databases. For this prototype, PostgreSQL[7] was selected, primarily because of the author's familiarity with its SQL dialect. However, any database

[5] https://github.com/seatgeek/thefuzz, Accessed: 18.02.2026.
[6] https://www.langchain.com/langgraph , Accessed: 18.02.2026.
[7] https://www.postgresql.org/, Accessed: 18.02.2026.

supported by SQLAlchemy could have been used instead.
Finally, the database management tool DBeaver[8] was used to primarily explore the stored data, verify the correctness of the ETL process, and cross-check the results produced by the RAG prototype.

6.2.2 ETL Configuration and Execution Flow

A single Python script serves as the entry point for managing the entire ETL process, providing several command-line arguments to control individual steps. This allows the data mining process to be executed either as a whole or in isolated stages. For example, only the ingestion of GitLab data into the database can be performed. Table 6.1 summarises the available options. For the pipeline to function correctly, the GitLab parsing step must be executed first, as it creates the main project entities in the database, which are referenced in all other stages. Thus, while the individual steps can be run independently, a fixed execution order must be followed.

Table 6.1: Arguments of the ETL script.

Option	Description
-j, -jacoco	Update the backend configuration with the required plugins.
-s, -sonar-scan	Instructs the script to run the SonarQube scan.
-m, -sonar-metrics	Instructs the script to retrieve the SonarQube metrics.
-r, -mine-repos	Instructs the script to mine the code and wiki repository.
-g, -fetch-gitlab	Instructs the script to parse the GitLab data.
-a, -all	Perform all steps of the ETL process.

In addition, a Pydantic-based configuration file manages the global settings required throughout the ETL and RAG pipeline. The ETL-relevant parameters are shown in Listing 6.4. The `REPO_DIR` field specifies the root directory containing the exported project data files. Each project is stored in a separate subdirectory, following the structure defined in Figure 6.1. Furthermore, the database URL and an access token with the appropriate permissions for the later-described SonarQube server are required to execute the complete pipeline.

A schematic overview of the ETL execution flow is shown in Listing 6.5. Each processing step is run sequentially, and all projects are processed one after another within each stage. Performance optimisation was not a primary focus of this thesis, as the data extraction is performed only once, and the majority of development effort was dedicated to the RAG prototype. The only exception is the SonarQube scanning module, which is parallelised across four CPU cores. Without parallelisation, compiling and scanning all projects sequentially proved prohibitively slow. The four main stages of the ETL pipeline are described in more detail in Sections 6.2.4 to 6.2.7.

[8] `https://dbeaver.io/`, Accessed: 18.02.2026.

```python
from pydantic_settings import BaseSettings, SettingsConfigDict
from pydantic import Field

class Settings(BaseSettings):
    DATABASE_URL: str = Field(..., description='Write Access DB URL
        for ETL')
    DATABASE_URL_READ_ONLY: str = Field(..., description='Read-Only
        DB URL for LLM for Analysis Tools')
    SONAR_GLOBAL_TOKEN: str
    SONAR_HOST_URL: str = 'http://localhost:9000'
    REPO_DIR: str = './data/repos'

    AQUEDUCT_API_KEY: str    # RAG config
    LLM_URL: str             # RAG config

    model_config = SettingsConfigDict(
        extra='ignore',
        env_file='.env',
        env_file_encoding='utf-8',
    )

settings = Settings()
```

Listing 6.4: Configuration Settings for the ETL pipeline.

6.2.3 Database Schema

Data from four different sources are integrated into the database: GitLab, the code repository, the wiki documentation repository, and SonarQube metrics. As a result, the schema exhibits moderate complexity (Figure 6.2) by preserving the necessary relationships among entities to support later retrieval by the RAG system.

A central design goal was to explicitly model the relationships among artefacts to facilitate cross-referencing within the RAG system. For example, the links from commits to issues and merge requests are preserved, enabling traceability between development activities and project management artefacts. Timelogs and notes are associated with either issues or merge requests, allowing the reconstruction of both technical and organisational aspects of the development process.
Furthermore, students are represented as first-class entities and linked to commits, issues, merge requests, time logs, and code ownership summaries. This design enables the computation of contribution-based metrics and supports queries such as individual workload distribution or review participation.
Branches are modelled to preserve contextual information about the development workflows and to support blame-based ownership calculations. However, the level of information supplied by branch data is limited by the development process practised by students,

```
args = parse_arguments()
run_db_init()

repo_dir = os.path.normpath(settings.REPO_DIR)

if args.fetch_gitlab or args.all:
    gitlab_fetcher.process_all_projects()
if args.mine_repos or args.all:
    git_miner.mine_git_repositories()
if args.jacoco or args.all:
    integrator = JacocoIntegrator()
    integrator.process_repos()
if args.sonar_scan or args.all:
    scanner = sonar_scanner.SonarScanner()
    scanner.run_batch_scan()
if args.sonar_metrics or args.all:
    metrics_fetcher = SonarMetricsFetcher()
    metrics_fetcher.run()
```

Listing 6.5: Schematic ETL Execution Flow.

since many groups delete feature branches after merging them into the development or main branch, thereby removing this contextual information. In such cases, branch context can still be partially reconstructed using merge request data from GitLab, although commit- and file-change-level information is lost.
The static code analysis results from SonarQube are stored separately as aggregated metrics and individual issues, categorised by component (frontend or backend). This separation allows both high-level summaries and detailed inspection of detected problems.

Overall, the schema was designed to support efficient retrieval of both quantitative metrics and qualitative contextual information, which is essential for generating comprehensive RAG reports.

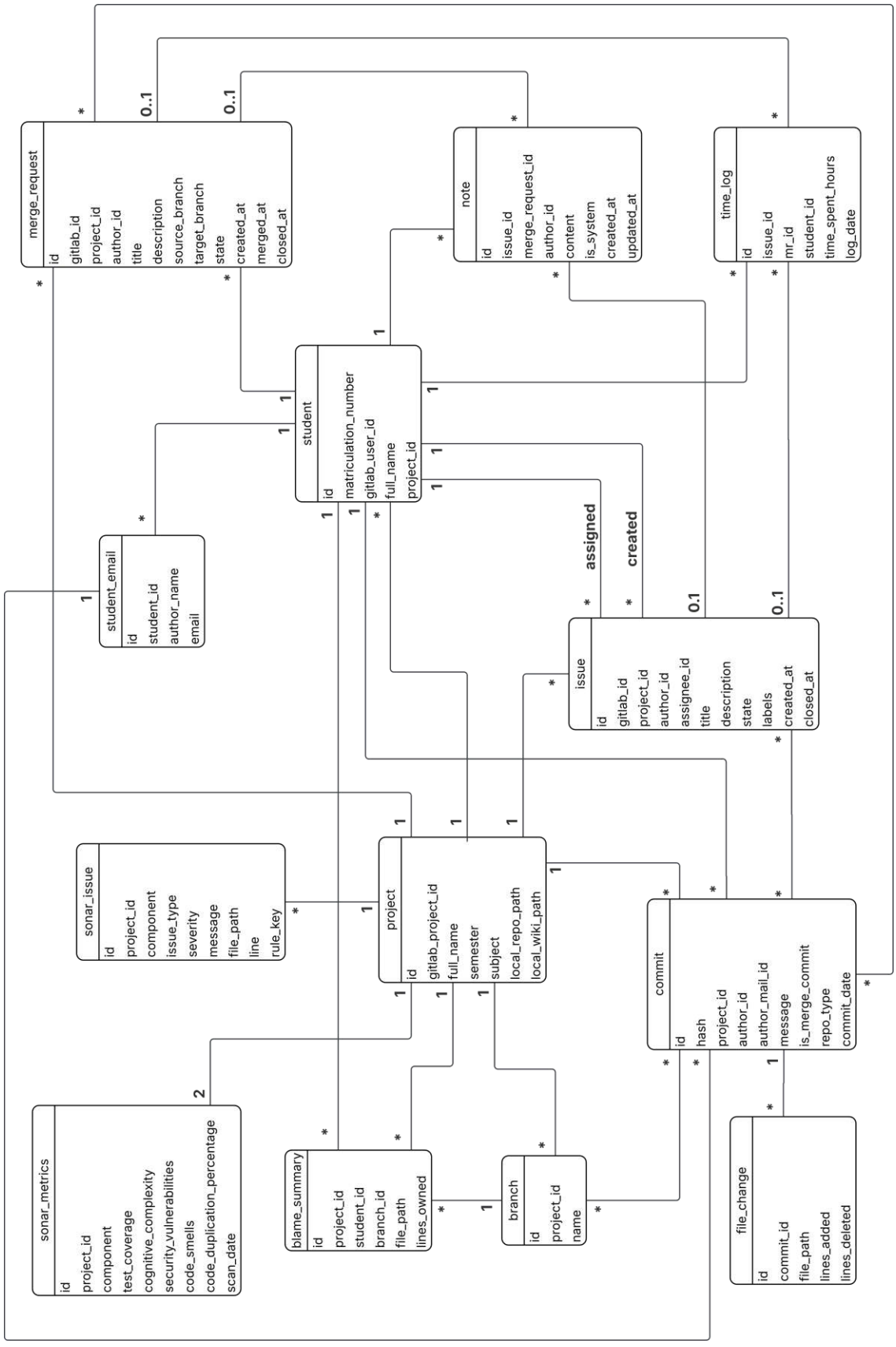

Figure 6.2: Database Schema

6.2.4 Gitlab Data Parsing

Parsing the available GitLab data is the first step in the ETL pipeline. Using the four relevant ndjson files described in Section 6.1, the main project and student entities are populated for each student group. In addition, the project key of the export structure follows a strict schema that contains relevant information: `SEM-SUBJ-DIV-ID`.
The first part denotes the semester. The format is either, for example, 'WS20' or '25ws', and it is the same within each semester. Both formats are preserved as they convey the required information about the referenced semester. The `SUBJ` component refers to the course type, since a parallel SE course is offered in the master's program. Only the bachelor's course is relevant for this thesis. The 'DIV' denotes the study division to which the student group was assigned, and the 'ID' is a running number to uniquely identify each group. This naming convention is identical to the one used by tutors and study assistants during the semester. Thus, there are no ambiguities in the context of this thesis. Using this project key, groups are uniquely identifiable, which is why it is primarily used as the reference identifier throughout the implementation.

The GitLab data are processed in a straightforward manner: relevant data fields are extracted, transformed, validated and finally persisted in the database. The notes provided with each issue and merge request JSON object are used for multiple purposes. They contain both system messages (e.g., time-spent notifications or issue assignments) and student messages discussing the content. Student messages are parsed in the same way as all other textual data. However, the system messages contain additional important information, such as the full names of students and changes of assignments to issues. The author of each note is matched to the corresponding student in `project_members.ndjson` using their GitLab ID until all students' names are resolved and stored in the database. Furthermore, system messages are evaluated to determine who was *last* assigned to an issue when issues were reassigned during development.

6.2.5 Git Repository Data Mining

Each student group maintains two git repositories: the source code repository and the wiki repository. The data mining process is identical for both repositories, even though the wiki repository contains less VCS information, and is designed to satisfy the identified information needs with respect to the code repository:

1. The existing branches are gathered.
2. The commits of each branch are analysed. This includes checking whether a commit is linked to a GitLab issue or merge request via the commit message and storing the file changes of each commit.
3. Code ownership information is computed via the `git blame` command line option. This is performed only on the master branch, as it represents the project's

assessment-relevant state. Any work not merged into the master branch by the end of the semester is typically considered nonexistent for evaluation. Additionally, certain blame information is excluded because some changes originate from the course administrator or automated tools (e.g., renovate bots for dependency updates), which should not be attributed to students.

This process applies to both repositories, but is not strictly necessary for the wiki repository. Most students update the wiki documentation only via the GitLab web interface. As a result, there is usually only one branch, and many commits are minor updates or corrections. Nevertheless, the complete version-control history and ownership of the wiki can be analysed to provide valuable contextual information for retrieval during RAG inference.

A major challenge in repository mining is matching code contributors to the correct student. This task is straightforward when students use their default university email address (e12345678@student.tuwien.ac.at) or their full name as the author. However, deviations from this convention lead to ambiguities, ranging from wrongful identification to cases that are entirely unresolvable. To illustrate the severity of this issue, one student committed all their work using their brother's email address, whose first name matched that of another uniquely identifiable group member. Such cases must not result in incorrect matches, as this would invalidate the statistical value of commit, file-change, and code-ownership data.

Tables 6.2 and 6.3 illustrate examples of the implemented automatic contributor matching rules, ranked from highest to lowest confidence:

1. Direct match based on the standard matriculation email format (R8 in Table 6.3).
2. Check whether the author name is a (proper) subset of a student's name (R1, R2), to account for middle names.
3. Check whether a student's name is a subset or equals the email username (R4), as this email is often used instead of the standard TU Wien email address by students who also act as tutors.
4. Check whether a student's last name occurs in the author name (R3).
5. Check whether the student's last name occurs in the email username (R5).
6. If no last name of the author is present, check whether the first name matches a student's name (R6).

If more than one student of one group satisfies rules 2-6, the match is considered ambiguous, and no automatic matching is performed. In such cases, the contributor's identity is temporarily assigned to a new 'unknown' student, who is then associated with the

Table 6.2: Example List of Students for Contributor Matching.

ID	Full Name	Matriculation Number
S1	John A. Smith	11111111
S2	Alice Wonderland	87654321
S3	Bob Builder	33333333
S4	Eve Apple	44444444
S5	Chris Neubauer	55555555
S6	John Doe	66666666

project. After completion of the ETL process, all unmatched contributor identities are re-evaluated using a separate Python procedure that applies a fuzzy matching algorithm based on the Levenshtein distance [114]. This script computes a similarity score between each student's full name and both the author name and the email username. Then, it selects the maximum of the two values and proposes a match with the student who achieves the highest score (R7, R9 in Table 6.3).

Upon user confirmation, the contributor is reassigned to the matched student in the database, and the corresponding commit, file changes and ownership data are updated. If all previously unmatched contributors of a project are resolved, the temporary 'unknown' student entity is removed. In some cases, no match can be made with sufficient confidence, such as R10 in Table 6.3. These cases often involve isolated commits at the beginning of a project, before students correct their Git client configuration or are informed by the tutor to do so. In such cases, the result remains unmatched, and the corresponding data is excluded from RAG analysis, accepting this small deviation. If a large number of commits were made by an unidentified contributor, the students or their assigned tutor must be contacted to resolve the discrepancy.

Table 6.3: Results of the Automated and Manual Matching Algorithm Scenarios.

ID	Author Name	Email	Match	Rule Triggered
R1	John A. Smith	john@work.com	S1	Full Name
R2	John Smith	john@work.com	S1	Subset Name
R3	Bobby Builder	b@builder.com	S3	Last Name
R4	cneubauer	chris.neubauer@tuwien.ac.at	S5	Tutor Email
R5	Chris	neubauerchris@mail.com	S5	Email Lastname
R6	Eve	e@apple.com	S4	Name Substring
R7	John	jdo@gmail.com	S6	Fuzzy (Email)
R8	Unknown	e87654321@student.tuwien.ac.at	S2	Email Regex
R9	Al Wonder	aw@mail.com	S2	Fuzzy (Name)
R10	gitii	gitii@users.noreply.github.com	*None*	Unidentifiable

6.2.6 Automated Plugin Integration

To enable code coverage analysis, the required plugins must be included in the Maven configuration of the Java backend component for each student group. A common template is used by all groups. However, the students may modify the configuration to suit their specific project needs. This results in several cases that must be handled automatically by adapting the backend's `pom.xml`:

No existing Jacoco configuration. Student groups are not required to measure the code coverage of their unit tests. Coverage analysis is only performed if it is defined in the group's project contract or explicitly requested by the assigned study assistant. Most groups use Jacoco[9] for basic report generation, which is also required for the SonarQube static code analysis scan. If the plugin is missing, it is automatically inserted with a predefined configuration.

Existing Jacoco configuration. Groups that already include coverage reporting often employ individual, heterogeneous configurations. To ensure consistency of the collected metrics in the ETL pipeline, any existing Jacoco plugin configuration is removed and replaced with the standardised configuration used in the first case.

Surefire configuration. For correct code coverage report generation, the Maven Surefire[10] plugin must be properly configured. The Surefire plugin executes the unit tests during the test phase of the Maven build lifecycle and produces the corresponding reports. Although this plugin is preconfigured in the course template, some groups modify or remove the configuration during development. Thus, the plugin integration phase verifies whether the required configuration is present and restores it if necessary.

6.2.7 Static Code Analysis Scan and Metric Retrieval

A running SonarQube server instance is required to perform static code analysis. For this thesis, SonarQube is hosted locally via a Docker container[11] [115]. In addition, an access token with sufficient permissions to create projects and execute analyses on the server must be generated and provided in the ETL configuration (Listing 6.4). The measurement and storage of SQM derived from static code analysis are performed in the following five stages:

Creation of SonarQube projects A dedicated SonarQube project is created for each student group using its project key (Figure 6.1). This is achieved by calling the REST API endpoint '`/api/projects/create`', using the project key as both the project name and identifier.

[9] `https://mvnrepository.com/artifact/org.jacoco/jacoco-maven-plugin`, Accessed: 19.02.2026

[10] `https://maven.apache.org/surefire/maven-surefire-plugin/`, Accessed: 19.02.2026

[11] `https://hub.docker.com/_/sonarqube`, Accessed: 19.02.2026

```
sonar.projectKey=project-key
sonar.projectName=project-key
sonar.host.url=http://localhost:9000
sonar.sourceEncoding=UTF-8
sonar.modules=backend,frontend

backend.sonar.projectBaseDir=backend
backend.sonar.projectName=Backend
backend.sonar.sources=src/main/java
backend.sonar.tests=src/test/java
backend.sonar.java.source=25
backend.sonar.java.binaries=target/classes
backend.sonar.junit.reportPaths=target/surefire-reports
backend.sonar.java.libraries=target/dependency/*.jar
backend.sonar.coverage.jacoco.xmlReportPaths=target/site/jacoco/
    jacoco.xml

frontend.sonar.projectBaseDir=frontend
frontend.sonar.projectName=Frontend
frontend.sonar.sources=src
frontend.sonar.exclusions=node_modules/**,dist/**,coverage/**,**/*.
    spec.ts,**/*.test.ts
```

Listing 6.6: SonarQube sonar-project.properties File.

SonarQube properties A `sonar-project.properties` file is added to the root directory of each exported student group project. This file contains the configurations required for the SonarQube scan (Listing 6.6). The analysis is structured separately for the backend and frontend components, while still performed within a single scan. This distinction is necessary because SonarQube relies on compiled bytecode for the Maven-based backend module, whereas the frontend Angular/TypeScript module is analysed directly from source code.

Compiling the backend via Maven. To prepare a project for static analysis, the backend classes must be compiled, and all tests must be executed. The ETL script therefore runs the following command:

```
mvn clean verify dependency:copy-dependencies
    "-Dmaven.test.failure.ignore=true" "-DskipTests=false"
    "-Djacoco.skip=false"
```

The '`copy-dependencies`' argument ensures that SonarQube has access to the bytecode of external libraries. The remaining arguments guarantee that tests and coverage reports are executed even if individual test cases fail.

```
self.metric_keys = [
    'coverage',
    'cognitive_complexity',
    'vulnerabilities',
    'code_smells',
    'duplicated_lines_density',      # in percent
]
...
keys_string = ','.join(self.metric_keys)
api_url = f"{self.sonar_host}/api/measures/component_tree"
params = {
    'component': project.full_name,
    'metricKeys': keys_string,
    'qualifiers': 'DIR,TRK', # project and sub-projects
}
...
response = requests.get(api_url, params=params, auth=self.auth)
```

Listing 6.7: Retrieval of Aggregated SonarQube Metrics.

Executing the SonarQube scan After these preparations, the ETL script triggers the SonarQube scan using the command "`sonar-scanner`".

This command locates the `sonar-project.properties` file and executes the static code analysis using the default rule set of the SonarQube installation [115]. After completion, all metrics can be inspected via the SonarQube web interface, or retrieved programmatically through the API using the configured access token.

Retrieving relevant SQM As described in Section 6.2.3, the SQM computed by SonarQube are retrieved in two ways. First, aggregated values for code coverage, cognitive complexity, security vulnerabilities, code smells, and code duplication are obtained for both backend and frontend components via the '`component_tree`' API endpoint (Listing 6.7).

Second, detailed issue information for security vulnerabilities and code smells is retrieved (Listing 6.8). Bugs are excluded, as they were deemed not relevant for the SE course (Section 5.2). Since the API allows retrieval of only 500 issues per request, and some projects exceed this limit, issues are sorted by severity before fetching. All issues classified as *Blocker*, *Critical*, and *Major* are stored. All Issues of minor severity or informational level exceeding the limit are excluded. This design decision was made intentionally to focus on significant issues rather than on a large number of minor issue details.

Parallelisation This stage of the ETL pipeline was parallelised to reduce the runtime of the Maven compilation process and the subsequent SonarQube scan. A `ThreadPoolExecutor` was used to execute the five stages described above concurrently with four

```
api_url = f"{self.sonar_host}/api/issues/search"
params = {
    'componentKeys': project.full_name,
    's': 'SEVERITY',
    'asc': 'false',
    'types': 'VULNERABILITY, CODE_SMELL',
    'ps': 500
}
...
response = requests.get(api_url, params=params, auth=self.auth)
```

Listing 6.8: Retrieval of Detailed SonarQube Issues.

worker threads. For the conducted experiments, this level of parallelisation proved sufficient. However, the number of workers can be increased in future deployments to further reduce the overall processing time, depending on the available hardware resources.

6.3 Data Loading

Two different mechanisms are used to access the data stored in the database. For most parts of the ETL and RAG implementation, an SQLAlchemy session is utilised to persist and retrieve data. This session is the same one used during the data mining stage and therefore provides full read/write access to the database. This database access layer is used by the RAG tools described in Section 7.4. These tools retrieve relevant information and transform it into representations suitable for both human interpretation and processing by the LLM.
In addition to this structured access, the RAG prototype provides a dedicated tool that enables the LLM to write its own SQL queries to retrieve information not directly exposed by the predefined tools. This data access path is restricted to a read-only database connection (see Listing 6.4), ensuring that the LLM cannot modify or delete any stored data.

CHAPTER 7

Retrieval Augmented Generation Implementation

This chapter describes the implemented RAG system in detail. It begins with a feasibility analysis and the selection of suitable LLMs (Section 7.1), followed by the design of the multi-agent architecture (Section 7.3), the tools accessible to the LLM (Section 7.4), and the processing pipeline from user queries to the generation of final feedback reports (Section 7.3.3). In addition, the system produces log files that store all intermediate tool outputs and their respective summaries to support human verification and traceability.

7.1 Feasibility Analysis and Technology Selection

The selection of an appropriate LLM is an important design decision, since the quality of the generated reports proved to strongly depend on the LLM's capabilities. Several factors were considered before choosing a final model. Initial experiments showed that both the number of parameters and the size of the context window are essential for processing large amounts of repository data and maintaining contextual coherence across different data sources. In addition, tool-calling support and latency were evaluated. Tool-calling is mandatory because the RAG system relies on external tools to retrieve and process relevant student information. Latency, while relevant for usability, is not a strict constraint. The system is intended to be executed once per user queries, and the resulting report is subsequently used as assessment support rather than for real-time interaction.

Table 7.1: LLM Selection in Preliminary Experiments.

LLM	Hosted	Parameters	Context Window	Tool-calling
llama3.1 [4]	locally	8b	128K	yes
mistral [5]	locally	7b	32K	yes
codellama 7b [6]	locally	7b	16K	yes
codellama 13b [7]	locally	13b	16K	yes
gemma3 [8]	locally	12b	128K	no
Gemini 2.5 Flash [9]	Google Cloud	undisclosed	1M	yes
Gemini 2.5 Pro [10]	Google Cloud	undisclosed	1M	yes
qwen-coder [11]	TU Wien	30b	16.384	yes
GLM 4.6 [12]	TU Wien	355b	200,000	yes
GLM 4.7 [13]	TU Wien	355b	200,000	yes

Three hosting strategies were considered:

1. running the LLM locally via Ollama [1],
2. using Google's proprietary Gemini models via API [2], and
3. using on-premise hosted LLMs provided by the TU Wien Datalab[3].

An overview of all evaluated models is provided in Table 7.1. All preliminary experiments were conducted using data from the author's private software project and did not involve student data.

In the first iteration, several LLMs were executed locally using Ollama to assess their computational viability. The model *gemma3* was eliminated early due to the lack of support for tool calling. All tests were performed on a workstation equipped with an AMD Ryzen 3800X CPU, 32GB RAM, and an NVIDIA GeForce RTX 3070 Ti GPU

[1] https://ollama.com/, Accessed: 12.02.2026
[2] https://gemini.google.com/, Accessed: 12.02.2026
[3] https://datalab.tuwien.ac.at/aiml/aqueduct/, Accessed: 12.02.2026, As of now, only accessible via TUvpn
[4] https://ollama.com/library/llama3.1, Accessed: 12.02.2026
[5] https://ollama.com/library/mistral, Accessed: 12.02.2026
[6] https://ollama.com/library/codellama:7b, Accessed: 12.02.2026
[7] https://ollama.com/library/codellama:13b, Accessed: 12.02.2026
[8] https://ollama.com/library/gemma3, Accessed: 12.02.2026
[9] https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash?hl=en, Accessed: 12.02.2026
[10] https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-pro?hl=en, Accessed: 12.02.2026
[11] https://huggingface.co/QuantTrio/Qwen3-Coder-30B-A3B-Instruct-GPTQ-Int8, Accessed: 12.02.2026
[12] https://huggingface.co/QuantTrio/GLM-4.6-AWQ, Accessed: 12.02.2026
[13] https://huggingface.co/QuantTrio/GLM-4.7-AWQ, Accessed: 12.02.2026

with 8GB of VRAM. The experiments revealed that the size of the maintained context window substantially affects performance. Increasing the context window caused data to be offloaded from GPU memory to system RAM, resulting in prohibitively high latency. Individual queries required up to 15 minutes to complete, and the final prototype issues up to 100 queries to generate a single report. In addition, the locally hosted models had comparatively small parameter counts and produced low-quality outputs. For these reasons, running an LLM locally was deemed impractical.

An alternative option was using Google's LLM Gemini, particularly the, at the time, newest 2.5 Flash and Pro versions. These models provide a context window of up to one million tokens and strong reasoning capabilities. Moreover, an educational grant made this option financially viable for the scope of this thesis. However, this approach was rejected due to data protection constraints. Student repositories contain personally identifiable information [116], such as matriculation numbers, names, and email addresses, embedded in commit logs and issue-tracking information. The terms of service of the consumer tier of the Gemini Apps explicitly state that Google may use inputs and outputs to improve its services and train its models [117]. Under the General Data Protection Regulation (GDPR), particularly the principles of purpose limitation and confidentiality, such processing of personal data by a third party is non-compliant without consent from each affected individual [116]. Furthermore, the EU AI Act classifies AI systems used to assess learning outcomes or evaluate students as *high-risk* systems [5]. This classification requires proper data governance and human oversight. These requirements are addressed in the system designed as described in Section 7.2.

Anonymising all data was considered, but the risk of re-identification through unique coding styles or metadata combinations would remain [118]. Additionally, because the system's purpose is to assess specific individuals, complete anonymisation would diminish the purpose of the generated reports.

Consequently, to ensure full compliance with GDPR and the TU Wien data protection guidelines for academic AI usage [119], the final decision was to use on-premise-hosted LLMs, ensuring that all sensitive student data remains within the university's controlled infrastructure. The TU Wien DataLab team maintains regular contact with the university's data protection officer to ensure compliance with applicable data protection regulations. Among these on-premise models, *qwen-coder* could not be used because its context window of 16.384 tokens was insufficient for the required data volume. The model GLM 4.6 was initially employed during early testing and was later replaced by GLM 4.7 following an infrastructure update. The model GLM 4.7 was ultimately used for the final experiments. Overall, GLM 4.7 provides a sufficiently large context window to handle the required data volume, has proven in experiments to deliver high-quality reasoning for complex analytical tasks, and, although its latency is higher than that of proprietary cloud models such as Gemini, it remains acceptable for the scope of this thesis.

7.2 Compliance with the EU AI Act

AI systems used in educational assessment contexts are classified as *high-risk* under the EU AI Act [5]. While a full legal conformity assessment is beyond the scope of this thesis, several requirements are considered during the design of the RAG prototype.

Transparency (Article 13) High-risk AI systems must provide sufficient transparency to enable users to understand and interpret their outputs [120]. In this thesis, transparency is addressed through the design of the RAG pipeline and the structure and content of the prototype's generated output. Sections 7.3.2 and 7.3.3 describe how the system exposes intermediate data retrieval and processing steps, ensuring that relevant information is not hidden within the generation process. This design reduces the risk of obscure and non-interpretable outputs. Furthermore, the generated reports include references to the underlying data sources. This allows tutors and study assistants to trace statements back to their source and verify the correctness of the information provided.

Human Oversight (Article 14) According to Article 14 of the EU AI Act [121], high-risk AI systems must be designed to allow effective human oversight. This requirement is inherently reflected in the intended use case of the RAG prototype. The system is designed as a support tool to gain insight into students' work, and does not perform any automated assessment or grading. The generated reports may be used to gather information and complement the tutors' or study assistants' own observations. Still, users must review and validate the generated information before directly using it in assessment contexts. This requirement directly relates to RQ4, which investigates the verification workflow of tutors and study assistants when interacting with AI-generated reports. The students' final assessment is then conducted in person when students present their project submissions. This design ensures that humans remain fully responsible for assessment decisions and can intervene when inaccuracies are identified.

Accuracy (Article 15) The EU AI Act requires that high-risk AI systems achieve an appropriate level of accuracy and robustness, and that they perform consistently throughout their lifecycle [122]. Due to the probabilistic nature of LLMs, absolute correctness of generated outputs cannot be guaranteed. Therefore, accuracy is addressed in this thesis through both system design and evaluation methodology. Using RAG reduces hallucinations by grounding generated outputs in retrieved data from external sources. Additionally, as later described in detail in Section 8.1, the prototype's performance is evaluated using *LLM-as-a-Judge* techniques, which represent the current state of the art for assessing RAG systems. Metrics such as faithfulness, groundedness, and summarisation quantify how accurately the generated reports reflect the underlying data.

However, achieving full compliance with the EU AI Act requires additional measures beyond the scope of this prototype. These limitations and possible future steps are outlined in Chapter 10.

7.3 Retrieval Augmented Generation Graph Setup

The RAG implementation is designed with a specific goal in mind: How can all relevant data be retrieved and analysed in a way that prevents the LLM from being overwhelmed by the raw amount of data while preserving contextual integrity? Providing a large amount of data to the LLM at once and directly asking it to answer user queries would lead to truncation and information loss, even with the relatively large context window of 200,000 tokens supported by the GLM 4.7 model. Even if all data fits in the context window, LLMs struggle to process very large inputs effectively, which can lead to the '*lost in the middle*' phenomenon. In this case, contextually important information is overlooked solely due to its position within the input [123].

Following recent advancements in agentic architectures [38], [88], [124], this problem is addressed by decomposing the analysis process into smaller steps. In this setup, multiple LLM agents each handle a dedicated subtask with a smaller amount of data. Each agent receives specific instructions for its task, enabling clearer assignment of responsibility and allowing more controlled, targeted reasoning. In this way, the overall analysis becomes manageable, and the risk of losing relevant information due to context window limitations is significantly reduced. Applying this form of agent-based task decomposition has been shown to be effective for analysing large-scale data from software repositories, improving reasoning performance and accuracy compared to monolithic retrieval-based approaches [89].

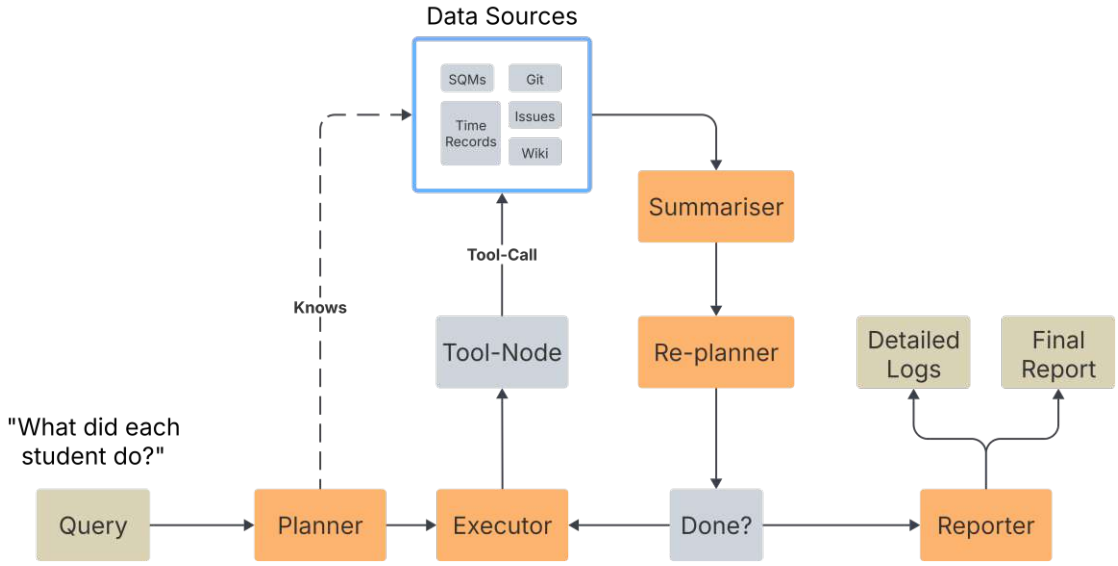

Figure 7.1: Retrieval Augmented Generation Graph Setup.

This multi-agent RAG setup is implemented using the LangGraph[14] framework [125], together with tools provided by its parent framework LangChain [126]. These frameworks provide the necessary building blocks for implementing a RAG system that can analyse user queries, retrieve relevant information, and generate structured reports based on the gathered data. LangChain provides an interface to the OpenAI-compatible API of the TU Wien Datalab to prompt the GLM 4.7 LLM. LangGraph provides components for constructing a graph-based workflow, including node definitions, state handling, and control-flow mechanisms for passing data among LLM agents.

Figure 7.1 shows the final setup of the RAG graph. The graph comprises five LLM agents developed for this thesis: Planner, Executor, Summariser, Replanner, and Reporter. In addition, a prebuilt LangGraph `ToolNode` is used. This node receives the JSON-formatted tool-call information produced by the Executor agent and performs the corresponding Python function call with the given parameters. By retrieving each data source independently through the Executor agent and summarising it immediately to its most important findings via the Summariser agent, the amount of information processed at each stage remains within the context window limits of the LLM [36], [90]. Afterwards, the Reporter agent uses only these summaries with compressed information to stay within its context limits. This design is well supported by LangGraph, as the graph can be constructed by defining nodes and connecting them via directed edges. The detailed implementation of each agent node is described in Section 7.3.2.

7.3.1 Internal Knowledge State

All nodes in the RAG graph share a global `AnalysisState`, which is passed to the graph at creation time and is shown in Listing 7.1. The original user question contains the single or multiple queries provided to the script that executes the RAG pipeline (Section 7.3.3). This information is required by several agents in order to reference the original user intent throughout the analysis. The plan contains the task sequence, generated by the Planner agent and may be modified by the Re-planner agent. Together with the current step, it determines which part of the analysis is currently being executed. The analysis summary stores the summary of each individual tool output. This field serves as the long-term memory and is later used by the Reporter agent to generate the final report. The messages field plays a special role, as it enables communication with the prebuilt `ToolNode`. The Executor agent generates a tool call in JSON format that specifies which tool to invoke and which parameters to use. The `ToolNode` reads this description and performs the corresponding Python function call. The output of this call is returned as a `ToolMessage`, which can then be processed by the Summariser agent. Finally, after all data has been gathered and summarised, the Reporter agent analyses it and stores the full report in the field `final_answer`.

[14] `https://www.langchain.com/langgraph`, Accessed: 20.02.2026

```python
class AnalysisState(TypedDict):
  original_user_question: str
  plan: List[str]
  current_step: int
  analysis_summary: Annotated[str, operator.add]

  # needed so that the graph ToolNode finds
  # the corresponding toolcall
  # the Summariser agent reads the output from here
  messages: Annotated[List[BaseMessage], operator.add]

  final_answer: str
```

Listing 7.1: RAG Graph Internal State.

7.3.2 Agents

Each agent has its own dedicated role in the RAG pipeline. While task decomposition primarily serves to manage large amounts of data and preserve contextual integrity, three additional objectives are achieved simultaneously. First, task distribution enables more precise agent profiling [127]. Through their prompts, the agents receive explicitly defined roles and traits that specify their reasoning responsibilities and guide their behaviour during their task execution. Second, this iterative pipeline of retrieving data from different sources one step at a time and passing the results to a subsequent agent for summarisation allows the interception and storage of raw intermediate outputs. This supports later interpretability, transparency and human verification. As Ribeiro et al. state for machine learning models, which equally applies to LLMs: "*If the users do not trust a model or a prediction, they will not use it.*" [128]. Consequently, the final generated report alone is significantly less valuable without the verification counterpart of the intermediate data outputs. Third and finally, although each agent has a dedicated responsibility, all agents operate in a single-turn interaction paradigm. For each step, a new interaction is initiated, with only the information required for the current task. This mitigates the risk of reduced reasoning quality due to context pollution [129], which may occur in long, multi-turn conversations with LLMs. The prompts for all agents described below are located in Appendix B.

Planner Agent. The Planner agent serves as the entry point of the RAG pipeline. It takes as input the available tools, including their names, parameter signatures, and textual descriptions of the data they return. The user query is provided to the LLM agent through a separate input message. Based on this input, the Planner agent creates a sequential plan of atomic tasks for the remaining agents to follow. The plan must be atomic because the Executor agent can only access one data source tool at a time. To support this behaviour, few-shot examples are included to demonstrate correct and incorrect planning behaviour, as well as an example plan [46]. Additionally, the agent is

instructed to create implicit iteration steps. At this stage, neither the internal state nor the Planner agent has concrete information, such as student names or wiki filenames of a specific student group. These are retrieved in early execution steps, and the Replanner agent later expands implicit iteration steps based on the information gathered so far. During testing, selective analysis was introduced as a guideline for the agent. When tasks involved processing large collections (e.g., wiki documents), the Planner agent often generated plans to process *all* available items, which proved computationally expensive but rarely provided proportional benefits. This selective analysis rule instructs the agent to first identify only the items relevant to the user query and to ignore irrelevant entries. Finally, output format constraints are specified to ensure that the plan can be processed correctly to update the graph's internal state.

In the final version, the Planner agent is informed that a tool for executing custom SQL queries is available. However, it does not receive the database schema as part of its input. Across hundreds of experiments, the Planner *never* included instructions to write a custom SQL query, as the other predefined tools (see Section 7.4) were sufficient to address the user queries. Thus, the schema was excluded from the agent's prompt to reduce context usage and focus on task-related instructions.

Executor Agent This agent's task is to retrieve information from a data source by invoking exactly one tool for a given step. The tool selection is based on the current step in the plan and the accumulated analysis summary. The agent must return only a tool call in a special JSON-formatted message, which the LangGraph `ToolNode` (Figure 7.1) processes to invoke the corresponding Python function. The GLM 4.7 LLM is well-trained to generate such tool calls, but LLMs may still produce errors. In approximately 2,000 early test executions, the Executor agent failed to generate a valid tool call in fewer than ten cases. This occurred when calling a tool without parameters, resulting in the textual output '`<tool_call>tool_name</tool_call>`'. This issue was resolved by introducing additional error handling that parses such outputs and programmatically constructs the correct tool call. After this adjustment, all tool calls were executed successfully.
Notably, the Executor's prompt does not include the list of tools in textual form as the Planner agent's prompt does. Instead, the tools are bound programmatically to the agent LLM via the LangChain framework. Consequently, the Executor agent is the only agent in the implemented RAG graph capable of generating executable tool calls.

Summariser Agent The Summariser agent represents a bottleneck in the RAG pipeline. If the data are compressed too aggressively, relevant details may be lost, distorting the final reasoning of the Reporter agent. Conversely, insufficient summarisation risks overwhelming the Reporter agent LLM with excessive data, leading again to the '*lost in the middle*' [123] phenomenon. This balance is addressed both programmatically and through prompt design. First, raw tool outputs resulting from the Executor's tool call are evaluated by length. If the output contains fewer than 300 words, it is directly reused as a summary to avoid unnecessary compression. Second, through prompt engineering

[130], the Summariser's prompt (see Listing B.3) is refined to guide aggregation strategies. In particular, the agent receives explicit instructions for handling large collections of commits and time logs. It is prompted to aggregate findings by issues or functional features and to highlight significant code changes. Furthermore, the original user goal and the current plan step are provided as context so that the summarisation remains aligned with the intended analysis objective.

Replanner Agent The Replanner agent performs an important role in the RAG pipeline. The Planner agent can only generate an initial plan based on the user query and available data sources, without knowledge of the concrete data contents, such as student names or file paths. This knowledge gap is addressed by the Replanner agent, whose primary responsibility is to expand steps. For example, the original plan step
"*2. For each student of group 25ws-SUBJ-DIV-01 found in step 1, retrieve the commit history.*"
is replaced by a sequence of atomic steps such as
"*2. Retrieve the commit history of Student A of group 25ws-SUBJ-DIV-01.*
3. Retrieve the commit history of Student B of group 25ws-SUBJ-DIV-01.
4. ...", and so forth.
This ensures that each step remains atomic and can be executed by the Executor agent. During step expansion, only candidates relevant to the original user query are expanded, thereby avoiding irrelevant processing. Additionally, the Replanner agent handles error recovery. If a tool call fails due to incorrect parameters, it generates a new step with corrected arguments.
With these instructions, the Replanner edits the initially static plan by incorporating concrete entities discovered during execution, while leaving the overall plan structure unchanged. This form of iterative refinement builds upon recent work on replanning and feedback-driven retrieval, which shows that intermediate results are required to ground abstract plans in concrete data and improve robustness in multi-step reasoning tasks [37], [131]–[133].

After the Replanner agent, a conditional edge in the RAG graph is evaluated. If further plan steps remain, the workflow returns to the Executor agent. Once all steps have been completed, control is passed to the Reporter agent.

Reporter Agent The Reporter agent generates the final report based on the accumulated analysis summaries. Its prompt (Listing B.5) contains both structural and content-related instructions. The report is organised according to the original user questions and grouped by functional areas rather than by data sources. Additionally, the report must include an explicit analysis methodology section that outlines the steps performed during retrieval. All claims must reference the corresponding step numbers, thereby improving transparency and enabling human verification. Finally, the agent is specifically instructed to avoid overly positive language, which was frequently observed during the early experimentation phase.

Algorithm 7.1: RAG Analysis Pipeline

Input: User Queries (`user_queries.yaml`), Config (Model, Temperature)
Output: Markdown Analysis Report, Tool Logs, Summaries

Initialisation: Load Environment and Parse CLI Arguments
Directory Setup: Create `run_dir`, `tool_outputs/`, and `tool_summaries/`
Logging: Initialise Logger for full tracking

Load `user_questions` from YAML
Combine questions into a single `combined_query`
Initialise LangGraph workflow using specified LLM

Input State: `{original_user_question: combined_query}`
while *RAG Graph is streaming updates* **do**
  foreach *node_update in updates* **do**
    switch *node_name* **do**
      case *executor* **do**
        Capture messages containing ToolCalls
        Log Tool metadata and save raw outputs to `tool_outputs/`
      end
      case *summariser* **do**
        Capture `analysis_summary`
        Save formatted summary to `tool_summaries/`
      end
    end
  end
end

Extract `final_answer` from the final graph state
Save Markdown Report with Model Metadata and Final Answer

7.3.3 RAG Configuration and Execution Flow

After describing the internal structure of the RAG graph, the following section explains how this graph is instantiated and executed by the surrounding script. The process flow of the RAG system is illustrated in Algorithm 7.1.

Table 7.2: Arguments of the RAG analysis script.

Option	Description
-o, --output_name	Name for the output folder and report.
-m, --model	LLM Model to use (Options: glm-4.7-355b, qwen-coder-30b).
-t, --temperature	LLM temperature (0.0 to 1.0).

```
output_name
├── tool_outputs
│   ├── 01_students_of_project_...
│   ├── 02_get_project_metadata_...
│   └── ...
├── tool_summaries
│   ├── 01_summary
│   ├── 02_summary
│   └── ...
├── output_name_logs
└── output_name_report.md
```

Figure 7.2: RAG Output File Structure

First, the required configuration and command-line arguments (Table 7.2) are parsed. By default, the script uses the GLM 4.7 model with a temperature of 0.0 to minimise variability in the generated text. The only mandatory input argument is the name of the output directory, which is created to store all generated artefacts, including raw tool outputs, summaries, a log file and the final report. In addition, both the access token for the Aqueduct AI gateway and the corresponding API endpoint of the TU Wien Datalab, for communication with the LLM, are loaded from a local environment file and passed to the same configuration file used by the ETL pipeline (Listing 6.4). This configuration also includes database connection settings and the directory containing the student group exports, both of which are required for the ETL and RAG pipelines.
Afterwards, the output directory structure is created as shown in Figure 7.2. It consists of a base output directory with subdirectories for intermediate tool outputs and summaries, a log file and the final LLM-generated report.

User queries are read from a YAML file, where the questions are organised as shown in Listing 7.2. The user may specify one or multiple questions. These questions can address different aspects of a single student group or compare characteristics across multiple student groups. All questions are combined into a single string (Lines 4-5 in Algorithm 7.1) and passed as input to the RAG graph. This avoids redundant tool calls and reduces the number of API requests, compared to letting the LLM agents process each query one at a time. Moreover, the agents receive the full query context at once and can generate more nuanced, coherent reasoning across different questions.
The RAG system attempts to answer queries using the available data sources. If a question refers to information not present in the database, the Reporter agent states that insufficient data are available to draw conclusions.

```yaml
questions:
- id: 1
  text: "What did each student of project group 25ws-SUBJ-DIV-01 do?"
- id: 2
  text: "Is one student of group 25ws-SUBJ-DIV-01 carrying the team
     based on code volume and complexity?"
- id: 3
  text: "Did someone in group 25ws-SUBJ-DIV-01 only work on minor
     fixes and documentation, rather than critical features?"
- id: 4
  text: "Is there proper testing documentation in the wiki of group
     25ws-SUBJ-DIV-01?"
```

Listing 7.2: User Query Configuration in user_queries.yaml

During execution (Lines 7-21 in Algorithm 7.1), the RAG graph is executed step-by-step using LangGraph's streaming functionality, allowing the system to capture and process internal updates as they occur:

- The Executor agent's output is intercepted to log all tool calls and their parameters.
- The `ToolNode` output is intercepted to store raw tool outputs in the corresponding output files.
- The Summariser agent's output is intercepted to store the generated summaries for each tool call.

After the graph execution completes, the final Markdown report is extracted from the internal state and stored in the output directory.

7.4 Available Tools

Commonly, vector storages are used in RAG systems to efficiently store and search for relevant information in a large collection of documents [57]. Retrieval is addressed differently in this thesis because user queries are typically directed to specific entities, such as projects or students. The design of the underlying database enables direct retrieval of different aspects of the available data, rather than retrieving document chunks via a textual similarity search. By providing tools with well-defined inputs and outputs, information can be retrieved in a specific manner and, in turn, enable greater transparency of the retrieval process, since the RAG system tracks all invoked tools, their input parameters, and the returned content.

The design of the tools follows several principles essential for reliable use by LLM agents. On the one hand, the tools must present the available data in a well-organised, semantically coherent manner. Each tool should retrieve information from a clearly defined

thematic domain. For example, a single tool should not simultaneously retrieve code commits contributed by a student, issue metadata from GitLab, and discussion statistics for merge requests. On the other hand, the number of available tools to an LLM must be limited [134]. Binding tools to an agent permanently reserves space in the LLM's context window, since the agent must continuously retain knowledge of which tools are available, which parameters they require, and what functionality they provide. Furthermore, an excessive number of tools can lead to a decision-fatigue-like effect, in which the agent struggles to select the most appropriate tool for a given task. This negatively affects both effectiveness and latency of the overall RAG system [134].

Table 7.3: Available Tools to the Executor Agent.

Tool	**Information Granularity**
list_available_projects	Overview
get_students_of_project	Overview
get_project_metadata	Overview
get_project_code_contribution_statistics	Overview
get_project_wiki_contribution_statistics	Overview
get_project_workload_distribution_statistics	Overview
get_project_sonar_metrics	Overview
get_project_sonar_issue_details	Details
get_student_code_commit_history	Details
get_student_wiki_commit_history	Details
get_student_time_logs	Details
get_contribution_frequency	Overview and Details
get_issue_statistics	Overview
get_merge_request_statistics	Overview
get_issue_details	Details
get_merge_request_details	Details
get_collaboration_interactions	Details
get_wiki_document_list	Overview
read_wiki_file	Details
get_project_branches	Overview
get_file_change_details	Details
get_repository_structure	Details
get_file_ownership	Details
get_discussion_thread	Details
read_source_file	Details
run_custom_sql_query	Custom

Therefore, a balance must be achieved between functional coverage and tool set size. A sufficient variety of tools is required to capture all thematic aspects of the project data, while some tools intentionally bundle closely related information to keep the total number of tools manageable. In total, 26 tools are available to the Executor agent. An overview is given in Table 7.3. Several of these tools provide high-level insights into the project data. Others enable fine-grained inspection of specific artefacts and interactions within the student projects. These detailed tools are typically invoked only when explicitly required by the user query. A detailed description of each tool is provided in Appendix C.

In general, all tools can be classified as perception tools, as they are used to retrieve data from the external environment, namely, the database populated with information from the student projects. The only exception is the `run_custom_sql_query` tool, for which the input parameter directly contains an SQL query. This tool can therefore also be classified as a computation tool. Nevertheless, the RAG agents *never* selected this specific tool during the experiments. The formatting of the results into structured textual output is handled within the tool implementations themselves, making the data readable both for the LLM and for manual human verification.

The type definitions and docstrings of the implemented tools are automatically translated by the LangChain framework into structured tool descriptions that are provided to the Executor agent. These descriptions specify what each tool does and how a tool call must be formatted. This form of documentation enables highly effective tool selection and reduces the need for few-shot examples to guide the LLM agent [45]. An example of such tool documentation is shown in Listing 7.3.

```
1  @tool
2  def get_discussion_thread(group_name: str, enitity_type: Literal['
       issue', 'mr'], gitlab_id: int) -> str:
3      """
4      Retrieves the full context and conversation history of a specific
           Issue or Merge Request.
5      Includes the title and all user comments chronologically.
6
7      Args:
8          group_name: The project name (e.g., "25ss-subj_div-01").
9          entity_type: Type of entity to retrieve. Must be either '
               issue' or 'mr'.
10         gitlab_id: The ID of the issue (e.g. 15 for #15) or MR (e.g.
               3 for !3) as seen in GitLab.
11     """
```

Listing 7.3: Example of Tool Documentation available through Type Definitions and Docstring.

7.5 Design Discussion

This chapter has so far described important aspects considered in the design of the RAG system. Data retrieval and text generation are split into many smaller tasks to avoid overloading the context window of an LLM, ensuring a proper workflow and faithful results, which in turn enables transparency into all performed steps. Furthermore, each agent has a well-defined purpose, enabling better control over the different components of the RAG pipeline. These design decisions have a major downside: *resource usage*. Each agent makes their separate API call to the TU Wien Datalab infrastructure to communicate with the hosted LLM. The use of the Datalab infrastructure was agreed upon for the scope of this thesis and remained within the allowed LLM usage limits (except for extensive use during evaluation; see Chapter 8 for further information). Still, the resource usage is not optimised because this thesis focuses on different priorities instead. This leaves room for improvement, making resource consumption an important consideration for future work.

CHAPTER 8

Evaluation

This chapter covers the multidimensional evaluation of the RAG prototype of Chapter 7. First, Section 8.1 presents automated and manual analyses to address RQ1 and provide insight into the expected results of further evaluation. Second, a quantitative study with 10 tutors addresses RQ2, evaluating how RAG-generated reports support assessment (Section 8.2). Finally, semi-structured interviews with the same tutors and eight study assistants address RQ2, RQ3, and RQ4 (Section 8.3).

8.1 Experiment Setup and Analysis

To address the variability of LLM-generated responses, 150 RAG reports were generated. For all experiments, the LLM generation temperature was set to its lowest value: *0.0*. Still, the reports vary in tone, utilised data sources, and drawn conclusions. This occurs due to the setup of the RAG system, in which multiple agents handle many small tasks with varying wording or degree of summarisation, which are later analysed and merged into a final report by the Reporter agent. This section examines the extent to which the results vary, the degree to which the reports faithfully reflect the underlying raw data, and the types of errors that occur.

Data from ten student groups were used for the experiments. Five groups came from the study division Div. A, where all groups implement the same project idea. Five groups came from Div. B, where each group may implement their own project idea. For each of these groups, 15 reports were generated with the same four input queries Q1-Q4:

Q1 What did each student of project group *25ws-subj-div-XX* do?

Q2 Is there proper testing documentation in the wiki of group *25ws-subj-div-XX*?

Q3 Did someone in group *25ws-subj-div-XX* only work on minor fixes and documentation, rather than critical features?

Q4 Is one student of group *25ws-subj-div-XX* carrying the team based on code volume and complexity?

These four questions were selected based on high ratings in early interviews with study assistants (see Chapter 5). The same questions were used for the tutor evaluation (Section 8.2) to: (1) provide queries that target relevant yet distinct aspects of student projects, and (2) keep the assessment tasks in Section 8.2 within a manageable time frame for the tutors.

The resulting dataset includes 150 reports. Each report contains all raw tool outputs and intermediate summaries, totalling 4338 items. The dataset is used for two purposes: (1) manual analysis to determine the consistency of which data sources are consulted by the RAG system and the factual errors in the generated reports, and (2) automated analysis, which computes the faithfulness, groundedness, summarisation, and answer relevancy score. An example report can be found in Appendix D.
Resource usage is not a direct focus of this thesis. However, generating 150 reports within a short time period revealed performance limitations of the RAG system on the TU Wien Datalab infrastructure. Each report contains data from 20 to 35 invoked tools, resulting in 70 to 100 API calls to the LLM. In tests on single reports, the RAG system completed data retrieval, summarisation, and report generation in approximately 20 minutes, including the LLM's latency as it processed the data. However, generating 10 reports in parallel over two days triggered automatic rate limiting of the author's student access token, resulting in an average generation time of 58 minutes per report.

For faithfulness, groundedness, and summarisation, the average for all intermediate steps in a report is computed. Each raw tool output is used as input, and the Summariser agent's summary is the output. This score is required for the overall calculation of faithfulness and groundedness as described below. Shortcut cases, where short tool outputs are directly reused as summaries as described in Section 7.3.2, are included in the calculation. These cases receive a perfect score of *1.0* for faithfulness and groundedness, as all data from the tool outputs are present in the summary and thus meet the requirements for both metrics. Report-level faithfulness and groundedness are computed using all summaries as input and the generated report as the output. The overall scores of one report R are then computed as:

$$faithfulness_R = (\frac{\sum f_i}{N_R}) * f_R \tag{8.1}$$

$$groundedness_R = (\frac{\sum g_i}{N_R}) * g_R \tag{8.2}$$

Table 8.1: Results of the Automated Evaluation of the Test Dataset.

Metric	Average	Median
Faithfulness	0.912	0.933
Groundedness	0.939	0.971
Summarisation	0.713	0.745
Answer Relevancy	0.813	0.815

where f_i and g_i are the faithfulness and groundedness of each intermediate step of report R, N_R is the total number of intermediate steps of report R, and f_R and g_R are the faithfulness and groundedness of the final generation step from all summaries to the final report, performed by the Reporter agent.

8.1.1 Results

Table 8.1 shows the average and median result of the automated RAG metrics of the 150 generated reports. The faithfulness and groundedness are both high, which aligns with the insights from the manual analysis of the results: the RAG system generated text based on consistent with the raw retrieved facts (e.g., number of commits or total time spent by students), but on occasion made mistakes in the reasoning around these facts. Such errors occurred between 0 and 2 times per report, but consistent patterns across the test dataset were detected:

- The Reporter agent often misreported the classification of the highest/lowest number of commits or the time spent by students (see student E and F in the example report in Appendix D). It reported "*Student E workload: Logged 99.25 hours, the highest in the group.*", even though another student spent 102 hours. The numbers for each student are correct, but the classification as highest/lowest is wrong.

- When students tracked multiple time logs for a single issue, the LLM often wrongfully aggregated the total time spent on that issue. The report would state "*Student A tracked a lot of time on meetings (37.13h).*", but the time logs on meetings actually sum up to 34 hours. This error was also discussed during subsequent expert interviews and is described in more detail in Section 8.4.

The summarisation metric is lower than the previous metrics, with an average value of 0.713. This is a result of the RAG systems' underlying design to avoid overloading the LLM with excessive data. Summarising each step inevitably loses some information. Specifically, when summarising time logs, the Summariser agent sometimes became overstrained, causing a timeout and resulting in an empty summary for one student's time logs. Similar summarisation losses were also observed during the tutor evaluation. These occurrences were discussed with tutors and study assistants and are further described in Section 8.3.

Table 8.2: Number of reports in which each tool was invoked for by the Executor agent.

Total Reports	**150**
Tool	**Times Invoked**
get_students_of_project	150
get_project_code_contribution_statistics	150
get_project_workload_distribution_statistics	150
get_student_code_commit_history	150
get_wiki_document_list	150
get_student_time_logs	149
read_wiki_file	148
get_issue_details	142
get_project_sonar_metrics	138
get_student_wiki_commit_history	101
get_issue_statistics	100
get_project_wiki_contribution_statistics	88
get_merge_request_details	35
get_merge_request_statistics	33
get_contribution_frequency	7
get_project_sonar_issue_details	5
get_repository_structure	2
get_file_ownership	1
get_collaboration_interactions	1
list_available_projects	0
get_project_metadata	0
get_project_branches	0
get_file_change_details	0
get_discussion_thread	0
read_source_file	0
run_custom_sql_query	0

Finally, the average answer relevancy is 0.813. This also aligns with the results of the consistency analysis and subsequent tutor and study assistant interviews: the RAG-generated reports answer the input queries in a particular way, with some aspects described in more detail than others. Therefore, the input queries are answered, but in a manner that may differ from the expected one.

Consistency Analysis To analyse the consistency of the 150 test reports, the invoked tools for each report were recorded. For tools retrieving wiki documentation, it was additionally recorded whether too many or too few wiki documents were retrieved. Table 8.2 shows, in descending order, for how many reports each available tool was invoked by the Executor agent throughout the experiments.

The tool calls in the resulting log can be categorised into three groups: (1) tool calls that always retrieved data for answering **Q1**-**Q4**, (2) tools that were sometimes invoked for a greater context, and (3) tool calls that produced data not relevant for answering the input queries. The first group shows that the RAG system consistently retrieves highly relevant data, including statistics and details of each student's work and the team's work distribution. Also, the wiki documentation is retrieved whenever applicable. Twice, the RAG system decided not to retrieve any wiki documents because the student group had not uploaded any testing-related documentation. In the 13 other reports generated for that group, the project specification document was retrieved to assess whether the test process is specified at all. Regarding the retrieval of testing documentation, the RAG system retrieved *more* documents than needed in 7 of 150 cases and *fewer* documents than available in 12 cases. When groups had thorough testing documentation comprising up to 25 test protocols, the RAG system sometimes retrieved only a subset of them rather than all related documents. This shows that, while the selective expansion of the Replanner agent is effective, further fine-tuning is needed.

The second group provides insight into the variance in retrieval by the RAG system. In 101 cases, the wiki commit history for each student was retrieved and considered relevant to answering the given queries. The remaining 49 reports do not contain information about the concrete wiki commit history. This aligns with the computed answer relevancy: the bulk of relevant information is retrieved as an information base for the reports (group one data), but the depth of some aspects, in this case, wiki commit histories, varies across reports.

Finally, the third group provides insight into the Planner and Replanner agents' correct decisions. Tools that do not retrieve relevant data for the given user queries are not blindly invoked, ensuring targeted retrieval.

8.2 Quantitative Tutor Evaluation

To answer RQ2, 10 tutors were asked to perform defined assessment tasks to evaluate whether tutors benefit from having RAG-generated reports. Each of these tutors performed the assessment task for the groups they had been supervising over the past semester. This resembles a real setup in which the RAG reports complement the usual environment. Figure 8.1 shows how long each participant has already been a tutor in the SE course. Most participants have little experience, with only one or two semesters, while one participant has already reached the maximum of nine semesters that they are allowed to be a tutor at TU Wien.

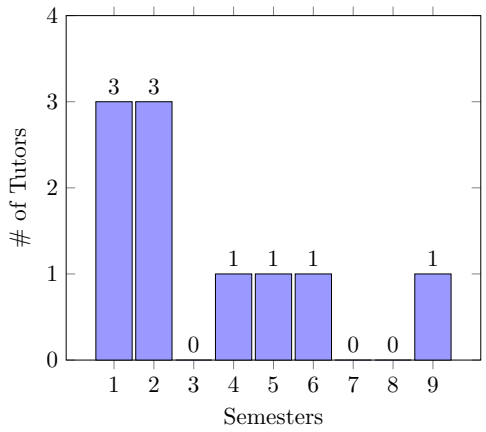

Figure 8.1: Teaching Experience of Tutors.

The quantitative tutor evaluation was split into two phases:

- **Phase One:** Each tutor assessed their student group by answering the questions **Q1-Q4** (see Section 8.1) without the support of a RAG-generated report.
- **Phase Two:** The tutors performed the same assessment again, but supported by a RAG-generated report.

The setup was the same for both phases: the tutors answered the given questions one by one, where the time required per question was measured. The order of questions was always the same. Thus, information gained from earlier questions may have influenced the time required for later questions. Phase one was carried out at the end of the 2025 winter semester, when the final assessment with similar objectives is typically conducted. That also means that all information was still fresh for the tutors at that time. The second phase was conducted after a washout period of 4 to 5 weeks. This setup was designed to avoid carryover effects from phase one by allowing time to pass. This was reinforced by the fact that the semester had ended and the tutors no longer met with the group regularly. Still, the threat to internal validity due to carryover effects remains and is further described in Section 8.5.
To evaluate whether differences in assessment time between phase one and two are statistically significant, a paired t-test [135] was conducted. As each tutor performed the same task in both phases, the total assessment times can be compared for each participant, using a significance level of $\alpha = 0.05$. Furthermore, Cohen's d [136] was computed to quantify the effect size of the observed time difference.

Table 8.3: Time elapsed for all assessment tasks without RAG-generated reports [mm:ss].

Tutor	Q1	Q2	Q3	Q4	Total
T1	21:18	06:35	**15:42**	09:42	**53:17**
T2	**27:34**	04:24	06:04	09:34	47:36
T3	20:53	02:25	09:17	12:32	45:07
T4	16:05	02:34	05:32	05:30	29:41
T5	15:37	06:04	05:27	**12:41**	39:49
T6	19:56	01:56	05:15	11:36	38:43
T7	20:13	07:15	03:06	03:02	33:36
T8	23:00	**07:17**	02:07	04:57	37:21
T9	03:43	02:01	02:53	02:18	10:55
T10	08:35	02:29	01:25	02:14	14:43
Average	17:41	04:18	05:40	07:24	35:04
SD	07:02	02:16	04:12	04:15	13:34
Median	20:04	03:29	05:21	07:32	38:02

8.2.1 Phase One

For phase one of the quantitative evaluation, the tutors were prompted to perform the assessment tasks **Q1**-**Q4** "traditionally". This means that all sources of information they normally use were accessible, including the GitLab repository, their own meeting notes, and various visualisation tools. The answers to the four assessment tasks were recorded for later comparison after phase two. Table 8.3 shows how much time was required for each tutor to complete the assessment tasks. The assessment behaviour varied greatly among the participants.

Table 8.4: Time elapsed for all assessment tasks with RAG-generated reports [mm:ss].

Tutor	Q1	Q2	Q3	Q4	Total
T1	**30:12**	**10:33**	02:49	**10:58**	**54:32**
T2	27:08	04:37	**05:51**	06:38	44:14
T3	21:13	04:24	01:51	04:02	31:30
T4	16:54	01:47	02:10	05:31	26:22
T5	13:27	01:57	03:38	03:41	22:43
T6	08:37	03:56	01:33	03:19	17:25
T7	11:45	02:01	02:23	04:01	20:10
T8	10:45	04:19	01:28	06:50	23:22
T9	06:25	02:13	03:54	01:05	13:37
T10	13:49	02:21	01:06	01:18	18:34
Average	16:01	03:48	02:40	04:44	27:14
SD	07:51	02:37	1:26	02:55	12:53
Median	13:38	03:08	02:16	04:01	23:02

While some tutors cross-checked all information from their regular meetings against the single-source-of-truth GitLab repository (participants T1, T2), others relied more on their memory and verified it only superficially. Tutors T9 and T10 were identified as special cases because, in both their groups, one student dropped out of the project during the semester due to poor performance. Therefore, both T9 and T10 had already performed this assessment before, enabling a rapid evaluation.

8.2.2 Phase Two

After a washout period of at least 4 weeks, the tutors were tasked with performing the same assessment as in phase one, only this time with the support of a RAG-generated report. For this, the reports from Section 8.1 were reused.

For each student group, 15 reports were generated, some of which contain errors and others do not. For this reason, a report was selected at random from the 15 available reports to avoid biasing the results. Therefore, some tutors received a report with a few isolated errors as described in Section 8.1.1, while others received a report without errors. Before starting the evaluation, each tutor received an introduction to the generated report, including how information is referenced and how they can access and interpret all intermediate raw tool outputs and summaries. Additionally, they were informed which parts were AI-generated and which resulted from Python function calls. Therefore, they could choose which information to take at face value and which to verify.

Table 8.4 shows the time needed per task when tutors were supported by RAG-generated reports. Some tasks required more time than during phase one, especially answering Q1. This is attributed to the time between the end of the semester and the start of phase two of the evaluation, as well as to the initial learning curve of working with the RAG report. Still, the average and median times required for all questions were lower than during phase one. The average assessment time decreased from 35:04 (SD = 13:34) in phase one to 27:14 (SD = 12:52) in phase two. This translates to a reduction of approximately 7 minutes (-22,3%) in average assessment time, with the median time reduced by 15 minutes. The paired t-test returned a p-value of 0.023 (<0.05), indicating a statistically significant reduction in assessment time between the two phases. This corresponds to a medium effect size (Cohen's d = 0.59), meaning that the average assessment time was reduced by approximately 0.59 standard deviations when tutors were supported by the RAG reports. This improvement is also reflected in the tutors' perception, which is further described in the next section. Figure 8.2 shows the comparison distribution of elapsed times across all tasks. The greatest relative speed-up is recorded for Q3 ("*Did someone work only on minor fixes ...*") and Q4 ("*Is one student carrying the team ...*"). For these, the reports helped quickly gain an overview that confirmed or refuted the statements in question, resulting in much faster outcomes.

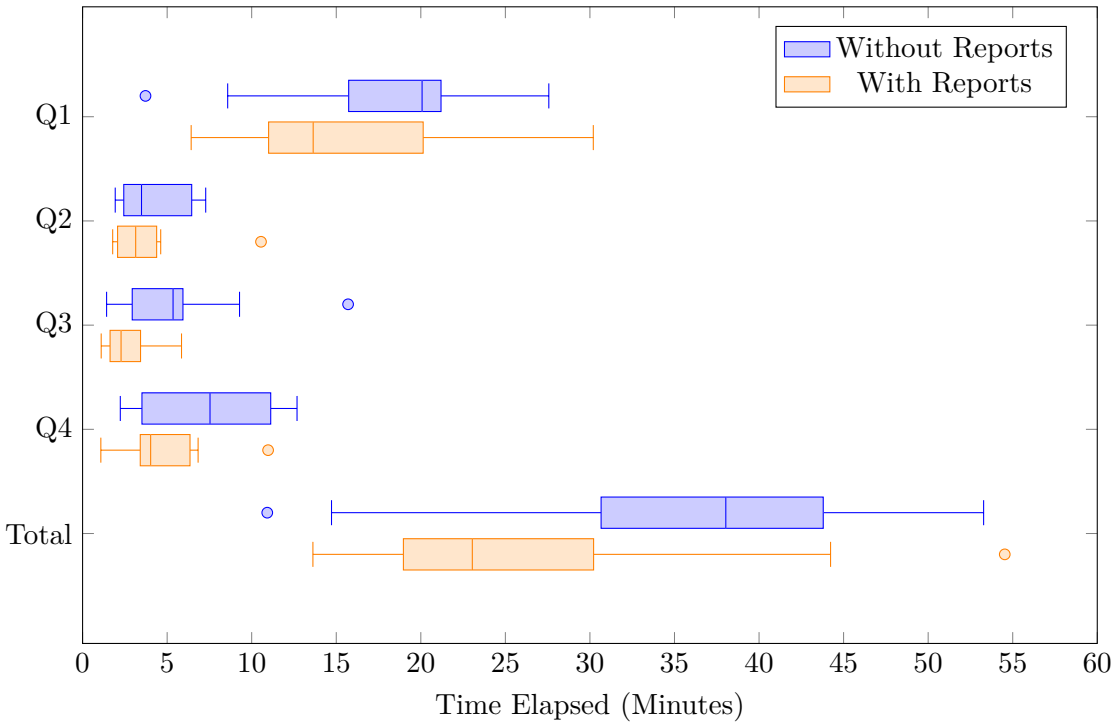

Figure 8.2: Distribution of assessment times (without vs. with RAG-generated reports).

8.3 Semi-Structured Interviews

This section presents the results of interviews with tutors and study assistants who evaluated the RAG-generated reports. The guiding questionnaire is available in Appendix E. The interviews aimed to determine whether the reports support both tutors and study assistants, identify concrete use cases, and assess participants' perceptions of AI-generated results. The interviews used closed-ended Likert-scale questions to target specific aspects, while the discussion with the participants provided further insight.

8.3.1 Tutor Interviews

Immediately after phase two of the quantitative evaluation, the tutors answered several questions about the usefulness of the RAG reports, their trust in the reports, and their verification workflow.

The first aspect discussed was the extent to which the RAG reports supported tutors for the given assessment tasks. The results are shown in Figures 8.3 to 8.7.

Notably, all tutors rated the reports as useful for the defined assessment tasks (Figure 8.3), with specific emphasis on the reduced manual search workload and the related cognitive effort (Figures 8.4 and 8.5). The tutors described it as a shift in their approach to the assessment process. Rather than searching for each piece of information individually, they use the RAG report as a solid starting point, with concrete references to the data sources, thereby enabling them to look up specific information directly.

After comparing the results of the assessment performed by tutors with those from phase one, the tutors agreed that the report helped them find either the same amount of information or more (Figure 8.6). This is attributed to cross-references in the report that connect data sources. In comparison, tutors would often look up only one data

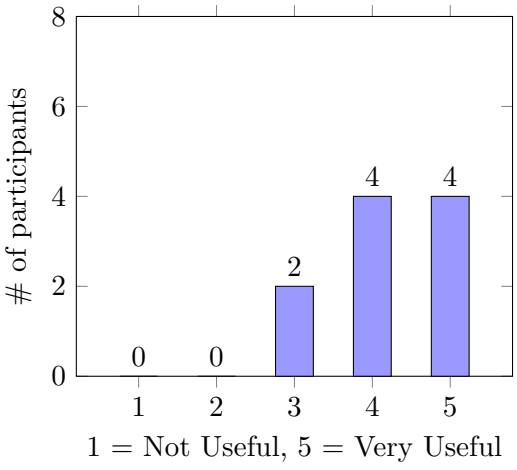

Figure 8.3: How useful was the generated report for answering the given questions?

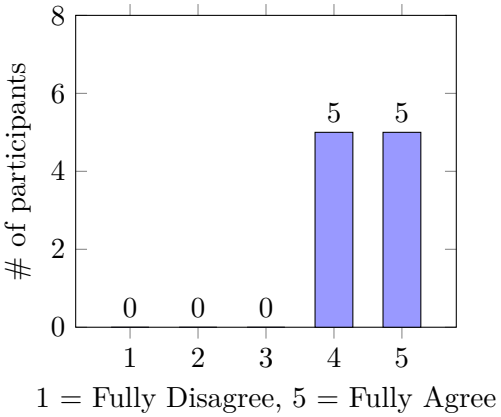

Figure 8.4: The report reduced my manual search effort.

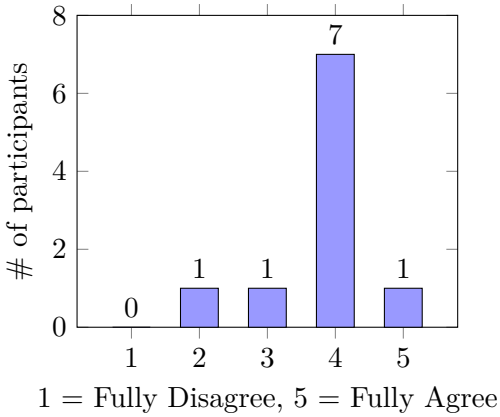

Figure 8.5: Using the report reduced my cognitive effort during the assessment task.

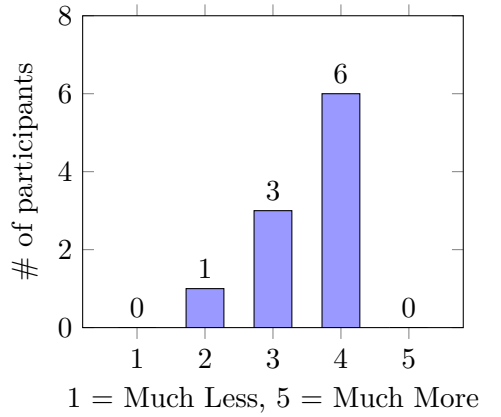

Figure 8.6: Did the report provide more/less relevant information than you found without the report?

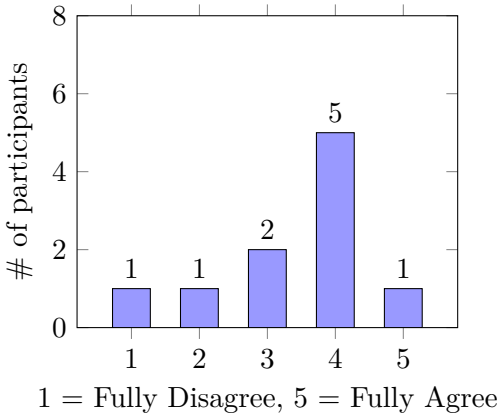

Figure 8.7: The report helped me detect issues or insights I previously overlooked or would not have found easily without the tool.

source (e.g., the GitLab issues students spent time on for task Q1) during the assessment of phase one. Only one tutor reported finding more information when performing the assessment without the RAG report. These results are consistent with Figure 8.7, where the overall consensus was that the report either helped tutors gain more insight or at least helped find information more easily.

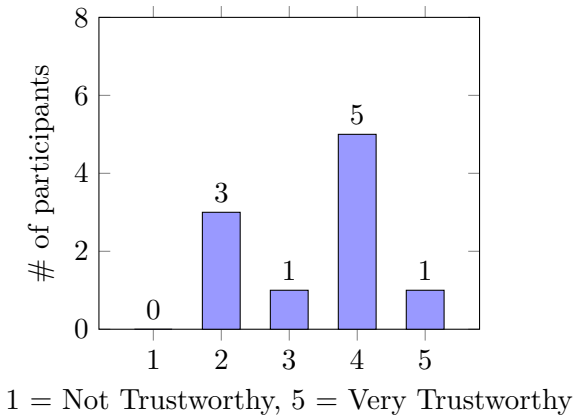

Figure 8.8: How trustworthy did you find the report?

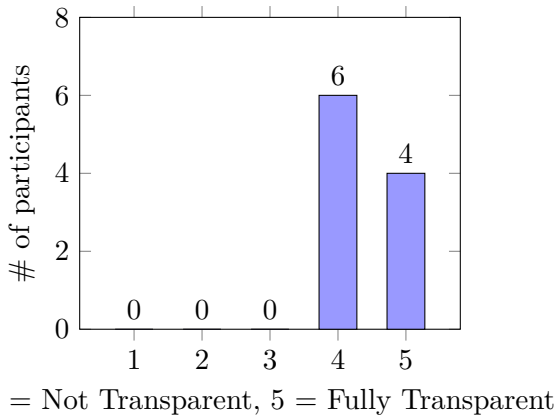

Figure 8.9: How transparent was the reasoning from raw data to the final report?

The second major aspect of the tutor evaluation was the tutors' trust in the AI-generated reports and the resulting verification workflow for the presented data. Tutors expressed mixed feelings about the trustworthiness of the RAG reports (Figure 8.8). Still, trustworthiness was rated above average. A major reason for this perception was the transparency regarding the original source of the data reported (Figure 8.9). This greatly helped tutors by enabling them to verify information, thereby improving the trustworthiness aspect for many of them.

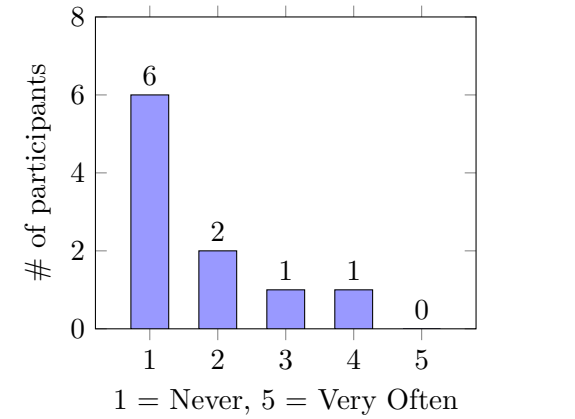

Figure 8.10: How often did you consult the intermediate tool outputs or summaries to verify the report?

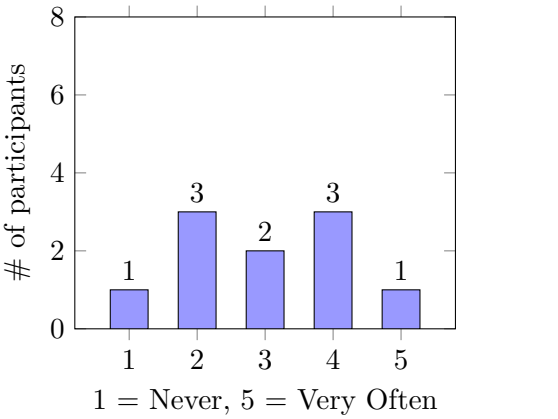

Figure 8.11: How often did you consult other data sources (e.g., Gitlab, Meeting Notes, Binocular,...) to verify the report?

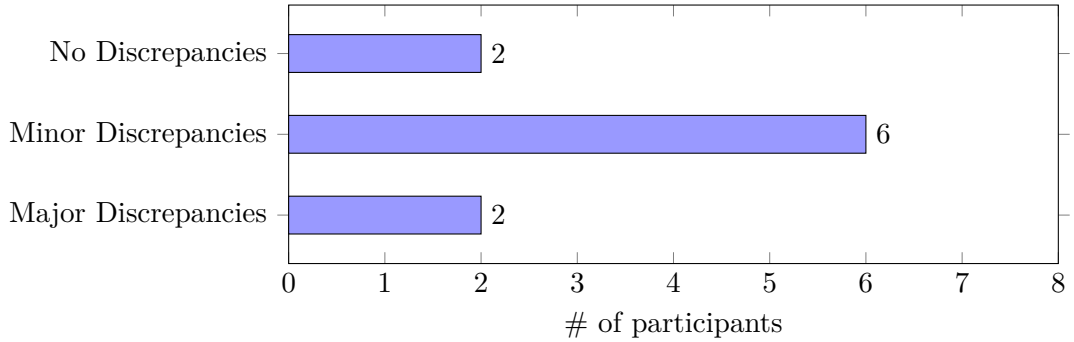

Figure 8.12: Did you notice any discrepancies between the generated report and the actual repository data?

Figures 8.10 and 8.11 show how often tutors consulted either the built-in intermediate tool outputs and summaries, or other data sources for verification. These two aspects are complementary: tutors who never consulted intermediate summaries of the report, rather relied on other data sources, and vice versa. The four tutors who used the intermediate tool outputs for verification rated their helpfulness as above average, with an average value of 3.75. The results show a common theme: when verifying the RAG report, most tutors checked information directly in the GitLab repository, as it is the original data source. Still, the majority of tutors emphasised that the availability of all intermediate data and summaries is important, even if they are not used for verification. As mentioned earlier, their availability improves transparency and trust in the reports and may serve as starting points for locating information on GitLab.

Notably, all tutors verified the RAG report to varying degrees. Tutors who performed less verification argued that the report reflected the same impression they had of their student group during and at the end of the semester, reducing the perceived amount of verification required. Nevertheless, all tutors identified aspects of the report that did not reflect their impressions (Figure 8.12). One of the major discrepancies arose from the available data for one of the previously mentioned student groups, in which a student dropped out early. This student was removed from the group's GitLab repository and was therefore not included in the available data at the time of export. This also demonstrates how the RAG system's performance is limited by the underlying available data. The other major discrepancy was a legitimate oversight of the RAG system: one student had overwritten the master branch of their project just before the deadline, resulting in highly skewed estimates of code ownership and effort in the generated report.

The remaining discrepancies were attributed to errors already mentioned in Section 8.1. Four tutors reported missing relevant information, which was lost during summarisation. Also, several tutors noted that the RAG reporting is written in an overly optimistic and praising manner, despite the Reporter agent's system prompt to avoid this behaviour. This, in turn, reduced the trust two tutors had in the report.

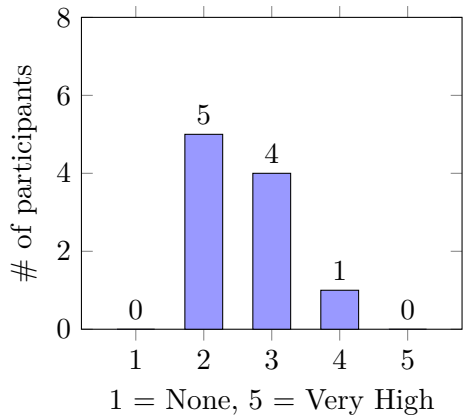

Figure 8.13: How much additional effort was required to verify or correct the report's statements?

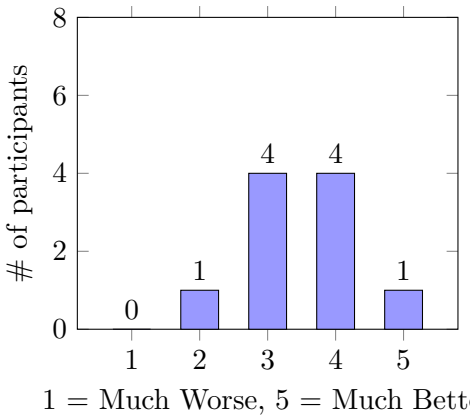

Figure 8.14: Overall, how reliable did you find the report compared to your manual assessment?

To sum up, the effort required for manual verification of the RAG-reports was manageable for the tutors (Figure 8.13). In fact, as already highlighted before (Figures 8.2 and 8.4), the total effort was even lower compared to phase one without the RAG reports. All but one tutor rated working with the RAG-generated report as at least as reliable as without it. Overall, they favour the assessment with reports, with an average rating of 3.5 (Figure 8.14).

The tutor evaluation showed a consistent pattern: using these kinds of RAG reports reduces the time required to manually search for all information, as the report summarises information from various sources in a single place. Still, the AI-generated reports were not blindly trusted, and verification was performed as needed. Also, while the reports did not contain all available information or were partially misleading, the tutors were able to offset this factor thanks to their good insight into the groups they supervise.

8.3.2 Study Assistant Interviews

The interviews with eight study assistants were structured differently from the tutor evaluation to focus more on qualitative, high-level evaluation of the RAG reports, rather than on specific assessment tasks. Figure 8.15 shows the teaching experience of the study assistants: all participants have substantial experience of at least 10 years and therefore have the necessary knowledge to properly evaluate generated reports in the context of SE education. The interviews were structured as follows: First, the aim of this thesis and the setup of the RAG prototype, as described in Section 7.3, were introduced. Then, the participants read one report generated by the RAG prototype. Finally, the study assistants expressed their opinions on trustworthiness, verification requirements, the completeness of information, and the general use case of these reports in the SE course context, guided by closed-ended questions. During this discussion, the study assistants

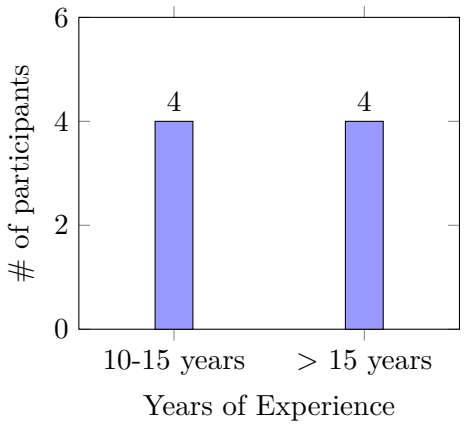

Figure 8.15: How long have you been working in software engineering education?

were also informed of the results of the automated and manual analyses (i.e., *faithfulness*, *summarisation score*, etc., see Section 8.1) and the types of errors that occur.

Since the study assistants have a different role than tutors in the SE course, the generated reports were tailored to their needs. Study assistants are responsible for the final assessment and grading of student projects at the end of the semester. Therefore, the report should provide a comprehensive view of the aspects visible in the available data. Where possible, a group supervised by the respective study assistant was analysed. The following questions were used as input to the RAG system to generate reports.

- Do a full assessment of the project and the agile process of group 25ws-subj-div-XX. The following questions may be used as guidelines.
- What did each student do? Organisation and code implementation-wise, and what they spent their time on.
- Did the students fulfil their assigned role (in the project contract or other wiki document)?
- Did someone only work on minor fixes or documentation rather than features?
- Are one/several student(s) carrying the team based on code volume or complexity?
- Were all the features that were promised in the project contract implemented?
- Are there Knowledge Silos in the project? Are these critical features?
- Did the group specify realistic and project-specific Non-functional Requirements? If yes, did they implement them?
- How was their agile process? Sprint planning, Retrospectives, Review process/Merge Request discussions (who, how much,..), Contribution Frequency (for each student separate), etc..
- How often did they merge code in the last week of the project? Did some Merge Conflicts provoke overly complex follow-up Merge Requests?
- Is there proper testing documentation in the wiki?
- What can you tell about gathered quality metrics?

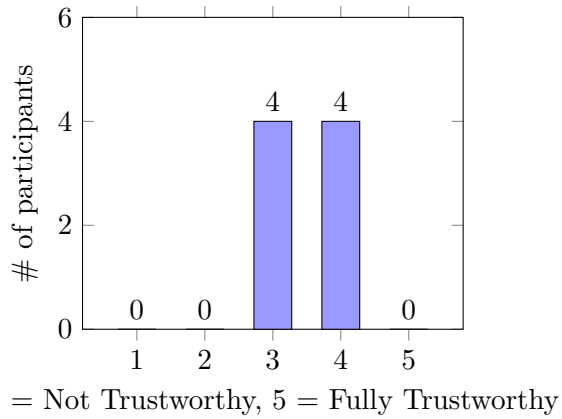

Figure 8.16: How trustworthy did you find the report?

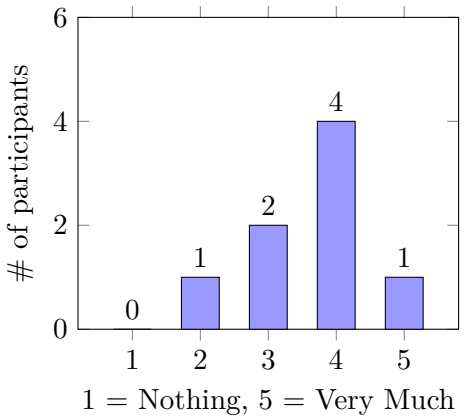

Figure 8.17: How much verification (e.g., checking GitLab or intermediate tool outputs) would be necessary before you would feel comfortable using this report for assessment?

These questions originated from the initial interviews on information needs (Section 5.2) and from the addition of individual queries to achieve a comprehensive picture. Most participants approved these input queries during the interviews as fitting for a final assessment. Two study assistants noted that not all of these aspects should be processed by the RAG system, as they are better discussed in person with students. The subject of what aspects the RAG system should and should not address is covered in the subsequent discussion in Section 8.4.

First, participants were asked to rate the trustworthiness of the RAG-generated reports (Figure 8.16) and the required verification effort that follows from it(Figure 8.17). The participants, for the most part, agreed that the AI-generated reports must not be trusted blindly. However, several study assistants argued that this is not purely due to the reports being AI-generated, but rather to a general scepticism about the information they receive. For example, even the tutor supervising a student group may hold false impressions that require independent verification. Furthermore, most participants require that the report is verified before using such RAG reports for assessment. The level of verification also depends strongly on the use case of such reports; if the report is used as aid in discussions with student groups, verification may be superficial, as students themselves may defend their own work and resolve misconceptions. Two participants argued that the amount of verification further depends on the prior performance of the student group. If they delivered flawless results throughout the semester, and the RAG report reflects that at the end of the semester, less verification is needed than for groups where the performance of individual students varied more. Finally, several study assistants also noted that the amount of verification required is experience-based: initially, more verification is required,

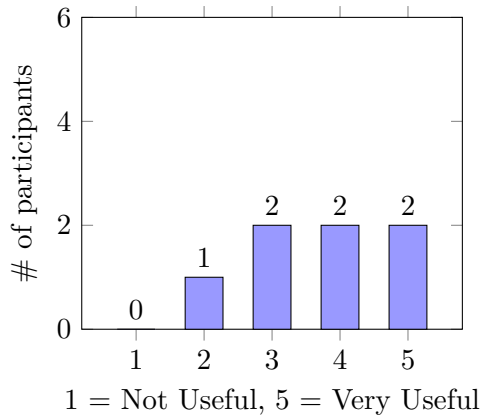

Figure 8.18: How useful were the intermediate tool outputs and summaries for understanding or verifying the report?

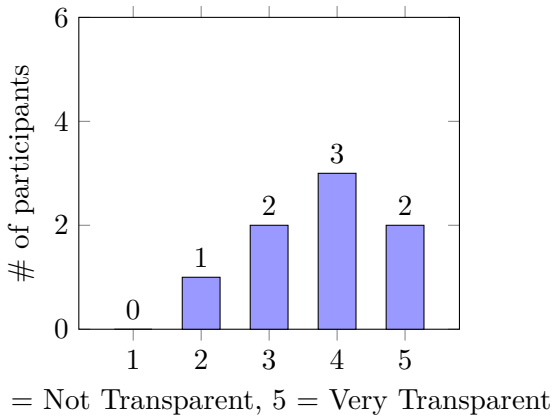

Figure 8.19: How transparent was the reasoning from raw data to conclusions?

but after several semesters of regular use and consistent results, verification of such RAG reports may become less crucial.

This experience-based approach is also reflected in participants' ratings of the intermediate raw tool outputs and summaries (Figure 8.18) as well as data transparency (Figure 8.19). As the participants read the RAG-generated reports for the first time during the evaluation, some were sceptical about how exactly the report's reasoning can be traced back to the source of information. One participant abstained from answering the question about intermediate data logs because they had insufficient information at the time. One study assistant explicitly mentioned that transparency needs to be improved. Comparing this with the result from the tutor evaluation (Figure 8.9) shows that experience with the RAG reports is helpful to reduce scepticism about the tool. The tutors thoroughly examined the reports while completing their assigned assessment tasks, and all tutors rated transparency at least 4 out of 5.

Apart from trustworthiness and transparency, all study assistants confirmed that the RAG system reports achieve their original aim to a good extent: they capture heterogeneous data from various sources in a single place, and summarise the SE student group's work (Figures 8.20 to 8.22). Besides that, study assistants addressed existing limitations of the RAG system, as well as concrete use cases. This is described in the following section.

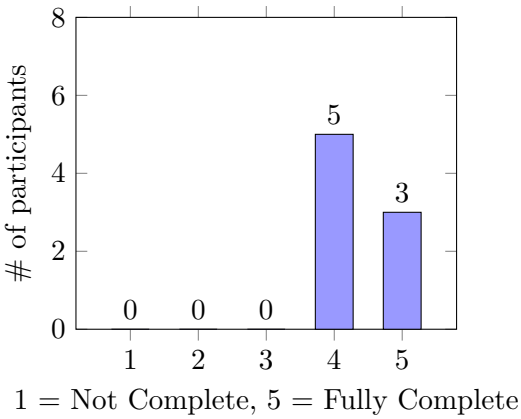

Figure 8.20: How complete was the information provided? (Regarding the used questions)

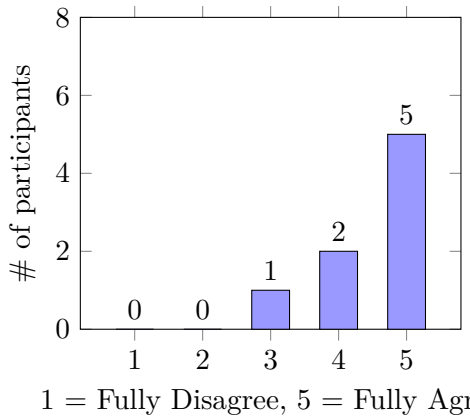

Figure 8.21: The report provided relevant context for understanding the group's work.

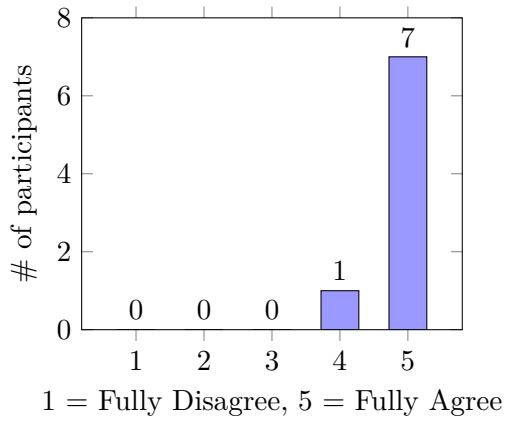

Figure 8.22: The report effectively cross-referenced multiple data sources (code commits, time logs, documentation, and quality metrics) to establish a comprehensive context.

8.4 Discussion

Overall, both tutors and study assistants rated the RAG reports as supportive for assessing SE students, as they provide a solid, well-structured summary (Figure 8.23) that serves as a baseline for gaining insight into student groups. For tutors, the effort shifted away from the entirely manual search required to obtain this baseline information. From there, further actions can be taken: either to verify the report against the group's GitLab repository or to communicate directly with the group to implement measures or

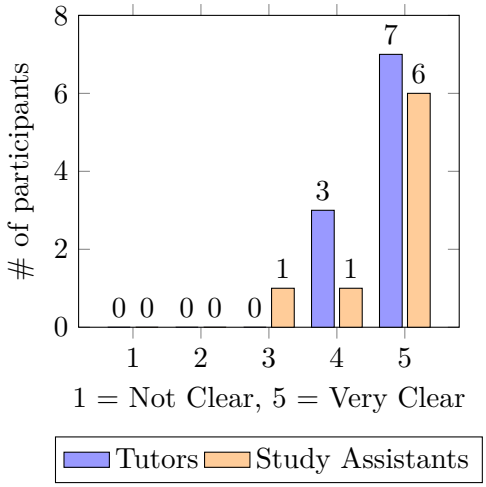

Figure 8.23: How clear was the structure of the report?

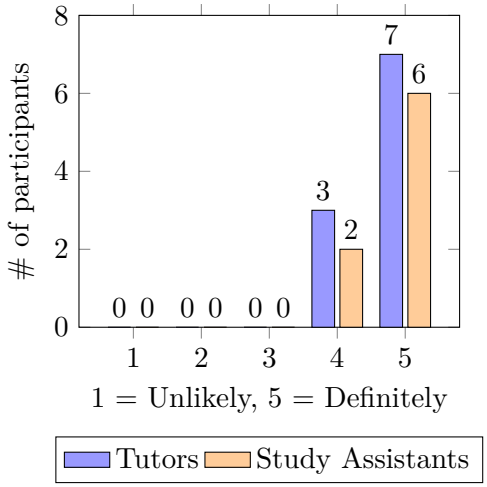

Figure 8.24: Would you use such a report (including intermediate outputs) in a real assessment scenario?

assess them.

Nevertheless, both tutors and study assistants expressed concerns about the RAG system and the reports it generated. First, several tutors noted that not all available information is included in the reports. However, as the reports are otherwise very comprehensive, this may lead to over-reliance and overlooked information. Concrete examples during the evaluation proved the opposite: after reading the reports, the tutors quickly identified when information they expected to find was missing (Figure 8.12). Two study assistants, on the other hand, argued that the report contained information that should not be included and should instead be discussed with students in person. This includes aspects that cannot be captured from the available data sources, such as verbal knowledge sharing between students. They suggested that removing these aspects from the reports would make other aspects the reports cover more trustworthy.
This is partly attributable to the input queries to the RAG system, but it also confirms a point made by three other study assistants: the RAG system needs a detailed specification of what is expected of students and what exactly the reports should contain. They provided concrete suggestions to improve the report structure and the preprocessing and presentation of intermediate data, to enhance both trustworthiness and data traceability. These aspects are closely related to the transparency requirements defined in Article 13 of the EU AI Act [120], where the traceability of data sources is essential for informed use. This thesis explores the application of RAG with a highly LLM-focused LangGraph setup, but certain aspects, such as the Summarisers agent's task on how to summarise different data, may again be handled in a more structured way instead of an LLM.

Finally, the evaluation showed that both tutors and study assistants regarded the RAG-generated reports with some scepticism. When asked to utilise the reports, tutors verified

the AI-generated results as needed. This behaviour directly reflects the requirement for human oversight of high-risk AI systems in education, as specified in Article 14 of the AI Act [121]. None of the participants disregarded the risk posed by AI-generated content, and when information was obviously incorrect, they ignored it and used other data sources instead. To quote one study assistant: "*Software engineers are inherently sceptical, because they know how LLMs work, and that they may as well hallucinate their response.*"

Despite reservations, all 18 participants confirmed they would use these RAG reports in the regular SE course context, as shown by Figure 8.24. Tutors would use them to save time on information gathering. Study assistants would not use the reports as a substitute for their final assessment, but rather use them beforehand as an additional data-based opinion to minimise the risk of overlooking information. This way, study assistants can base their assessments of good performance or shortcomings on facts and communicate more directly with students during the final phase of the project.

8.5 Threats to Validity

Carryover and Order Effects All tutors performed the quantitative evaluation in the same order, performing the manual assessment before using the RAG reports. This fixed ordering was necessary because the RAG prototype and its corresponding evaluation setup were only finalised after phase one of data collection at the end of the 2025 winter semester, making a counterbalanced design infeasible. This design introduces potential threats to internal validity. In particular, learning effects may have occurred: during phase two of the quantitative evaluation, tutors were already familiar with the evaluation procedure, had seen the questions before, and may have remembered relevant answers. As a result, the observed time saving cannot be attributed to the use of RAG reports alone. To mitigate carryover effects, a four-week washout period was introduced between phases, coinciding with the end of the semester, during which tutors no longer met with student groups. This was intended to prevent tutors from recalling specific answers and forcing them to re-engage with the material.
Although the threat to the validity of the measured time savings remains, complementary findings from the qualitative evaluation provide additional context. Tutors consistently reported a reduction in perceived cognitive and manual search effort when using the RAG reports. These qualitative findings still indicate a beneficial effect of the system on the assessment process.

Participant selection All study assistants were members of the core teaching staff of the SE course at TU Wien at the time of the interviews. The participant selection did not follow a random sampling strategy. Instead, it combined purposive and convenience sampling [111]. While this approach ensured that experienced study assistants with domain knowledge were included, it may have influenced the results. The tutors were also selected by convenience sampling. As a result, most participants had little experience as

tutors in this SE course. While this reflects the normal distribution of the tutor teaching staff, it may have influenced the evaluation results.

Influence due to the Interviewer The interviewer may have influenced participants' perceptions of the RAG prototype and the generated reports. This may have occurred during the initial introduction and presentation of the results, as well as in suggestive phrasing during the interviews. During the quantitative evaluation with tutors, all interaction was minimised, except when participants asked questions for clarification.

Bias due to Phrasing The phrasing of the questions used in the interviews could be ambiguous, potentially confusing participants about what aspect they aim at. This effect was minimised by conducting a pilot interview to identify such questions and refine the wording. However, this effect cannot be fully eliminated, and the resulting confusion was clarified in the interviews.

CHAPTER 9

Results

Through the multidimensional evaluation, the following conclusions can be drawn regarding the original research questions.

RQ1: What is the faithfulness and retrieval consistency of a RAG-generated assessment report operating on students data. The initial analysis and tutor evaluation indicate that the generated reports are highly faithful. The vast majority of statements in the generated reports are faithful to the underlying data (*faithfulness = 0.912*, *groundedness = 0.939*), and the major discrepancies that occurred were quickly discovered by tutors and study assistants. This may further be improved by better data preparation within the presented data retrieval tools. For instance, by aggregating time logs directly, the LLM can use the preprocessed data rather than performing calculations itself. The RAG system exhibits a consistent retrieval behaviour in the data that the input queries require (Table 8.2). Beyond that, the generated output exhibits some variation, with uncertain effects on the overall results. This, in combination with other concerns expressed by study assistants, may be mitigated by providing a more detailed specification of the data that the reports must contain and which they may not.

RQ2: How does the integration of RAG-generated reports impact the time efficiency and information discovery of tutors during the assessment of student software projects? The quantitative evaluation showed that the assessment time can be sufficiently reduced by integrating RAG-generated reports for support. Individual tutors required more time when using the reports (T1: +1:15, T9: +2:42, T10: +3:51), which may be attributed to their exceptionally fast assessment without the reports and the initial learning curve of using the RAG reports. However, most tutors experienced a significant average speedup of 22.3%, saving approximately 7 minutes when using the generated reports. This reduction was found to be statistically significant (paired t-test, $p = 0.023 < 0.05$, Cohen's d = 0.59). Furthermore, all participants expressed a reduction in manual search effort. Finally, six of ten tutors reported finding more information when using the reports than when not using them, and three reported a similar amount

of information. Only one participant reported that their manual assessment without RAG-generated report was more comprehensive.
Nevertheless, this reduction in human effort must be contrasted with the RAG system's high generation latency. As described in Section 8.1, the average generation time of 10 reports in parallel was 58 minutes per report. These infrastructural and design considerations must be addressed and are open for future research.

RQ3: How do tutors and study assistants perceive the trustworthiness, reasoning transparency, and completeness of RAG-generated reports? On average, participants rated trustworthiness positively, with a mean of 3.44. The evaluation showed that, especially, the perceived level of transparency and the resulting trustworthiness depend on the user's familiarity with the tool. Study assistants did not use the generated reports in depth, with several participants noting that the reports were not sufficiently transparent. The tutors, on the other hand, all worked with the report in detail, including tracking down information to its source and verifying it, and all rated the transparency at least 4 of 5. Still, since study assistants are responsible for student grading, greater transparency is required to ensure that the reports meet their standards.
The perceived completeness of information appears to depend on the purpose for which the RAG-generated reports are used. The study assistants rated the completeness of the information positively, as they would use these reports as summaries to gain more insights into students' work. Tutors, who already have greater insight into the groups they supervise, require a higher level of completeness than they currently have. Using their existing knowledge of the student groups, the tutors identified missing information in the generated reports and shortcomings in the RAG system.

RQ4: How does reliance on AI-generated reports change the verification workflow of tutors and study assistants? Both study assistants and tutors regarded the RAG reports as an additional tool for gaining insight into student groups, rather than as an immediate solution for grading. The study assistants expressed the need for a certain level of verification, which could be performed in various forms. Some saw this as the tutors' obligation, given their greater insight into the student groups and their reporting to the study assistants. Others would combine the verification process with direct communication with students to ensure full transparency for all parties.
The evaluation showed that tutors met this verification requirement, as they confirmed or corrected the RAG reports as needed, drawing on their existing insights and using GitLab directly as the source of truth. Additionally, the evaluation showed no signs of over-reliance on the generated reports, as they were treated as an additional source of information rather than as facts taken at face value. Given tutors' prior insights into their assigned student groups, discrepancies in the generated reports were quickly identified and corrected. This observed verification behaviour aligns with the requirements for human oversight as defined in Article 14 of the EU AI Act [121]. It further confirms that the prototype is used as intended, namely as a decision-support tool rather than automated decision-making.

CHAPTER 10

Conclusion

This thesis explores a novel approach to support educators in an undergraduate SE course using Retrieval Augmented Generation. Traditionally, in the SE course at TU Wien, insights into students' software projects are gathered manually, with visualisation tools used to highlight fragmented aspects of students' work. This work improves the process by integrating data from various sources into comprehensive reports. As a result, less time and effort are needed to locate and contextualise fragmented data.

To achieve this, a literature search was conducted, revealing a research gap in the use of RAG in education and reviewing commonly used SQM in education and industry to inform the information needs for assessment in the SE course at TU Wien. This was refined through expert interviews with study assistants to determine the relevant metrics for student assessment. The results of the interviews were presented in Chapter 5. In parallel, various LLMs were reviewed for their technological feasibility for the proposed RAG system, as described in Section 7.1.

Based on these insights, a RAG prototype was implemented, with a focus on task separation and traceability of information back to its sources. Using the prototype, over 150 reports were generated and a sample was evaluated by tutors and study assistants to address the individual dimensions of the research questions in Chapter 8.
The generated reports demonstrated a high level of faithfulness (RQ1), as initially indicated by an empirical analysis and subsequently confirmed by tutors' evaluations. Retrieval behaviour (RQ1) is consistent for essential data, but variation remains for further, non-essential data. Although generated responses are similar across multiple runs, they are not deterministic. The evaluation further indicated that the availability of RAG-generated reports reduced assessment time and improved tutors' information discovery rate (RQ2). Both tutors and study assistants rated trustworthiness and reasoning transparency (RQ3) as moderately positive, with transparency and verifiability as major factors for positive ratings. By ensuring traceability, users can verify the RAG reports' reasoning more easily, leading to higher trust. The evaluation showed that not all

aspects were viewed positively. The completeness of information (RQ3) and the selection of information in the reports leave room for improvement: some information was missing, and other data should be excluded due to concerns about misrepresenting aspects not well covered by the available data sources.

Nevertheless, the RAG-generated reports were perceived positively, with all 18 participants seeing use cases for them. Additionally, the evaluation indicated no over-reliance on the generated reports (RQ4), indicating responsible use of AI-generated content in an educational context. However, because the RAG system was implemented as a proof-of-concept prototype, further work is required before deployment in a real course context.

10.1 Future Work

Several aspects of either the initial design of the RAG prototype or of suggested improvements remain for future work. The following section highlights potential next steps.

First, the resource usage and feasibility of employing the RAG prototype need to be further investigated. Resource consumption did not pose an issue when generating single reports, but intensive use of the RAG system across many student groups resulted in insufficient available resources. This depends largely on the LLM used, the infrastructure on which it is hosted, and the number of concurrent users. At TU Wien Datalab, user numbers are increasing continuously, potentially leading to insufficient resource capacity in the future.

The overall structure and content of the generated reports revealed several opportunities for improvement. As suggested during the evaluation, processing individual data sources (i.e., the RAG tool outputs) in a more structured format, rather than natural-language summaries generated by an LLM agent, could improve the consistency of the generated reports. Furthermore, establishing clear guidelines in the RAG system for which contents in the reports to include and exclude may enhance their overall quality. Finally, definitions of various SQM should be included to avoid user confusion.

Since the prototype's features were selected to allow rapid development and testing during implementation, its ETL component used archived data. For use in the regular course context, this needs to be adapted to directly extract data from GitLab, thereby making the RAG prototype more accessible to educators.

From a regulatory perspective, achieving full compliance with the EU AI Act requires additional measures beyond the scope of this prototype [137]. In particular, aspects such as formal risk management procedures, continuous system monitoring, and record-keeping should be addressed. While this thesis demonstrates how selected requirements can be incorporated in the design, a complete compliance strategy would require integrating these aspects into the greater institutional workflow.

Overview of Generative AI Tools Used

Apart from the various LLMs mentioned in Section 7.1 that were used for the concrete approach of this thesis, Grammarly AI[1] was used during the writing of this thesis. The browser plugin was used to directly correct grammatical errors during writing in the Overleaf LaTeX editor, and the web interface of Grammarly was used with the following pre-defined prompts:

- *Fix critical mistakes*
- *Improve accuracy and clarity*
- *Shorten long sentences*

In addition, the *Open WebUI*[2] of the TU Wien Datalab with the LLM GLM 4.7 was used to unify the formatting of the bar-chart figures of Chapter 5 and Chapter 8.

[1] https://www.grammarly.com/about, Accessed: 16.03.2026
[2] https://chat.ai.datalab.tuwien.ac.at/, Accessed: 16.03.2026

List of Figures

List of Tables

List of Algorithms

Listings

Glossary

GitLab GitLab is a source code hosting platform [138].. 2, 3, 6, 8, 24, 36, 39, 40, 43–46, 48–50, 52, 53, 71, 81, 82, 85, 87, 92, 98, 100, 155, 156, 158

hallucination Phenomenon where a language model confidently generates factually incorrect or nonsensical information.. 1, 13, 22

Management Review Regular assessment milestones during the software engineering course group phase.. 7

Ollama An open-source framework designed for the local deployment and management of Large Language Models.. 60

prompt The input text provided by the user to a Large Language Model to guide it in generating a specific text response.. 13

SonarQube A static code analysis tool [26].. 10, 19, 43–45, 48–50, 55–57, 153

token A sub-word unit (typically a word fragment, word, or piece of punctuation) that a Large Language Model uses to break down, process, and generate text.. 11

User Stories Short, simple descriptions of a software feature written from the end-user's perspective to drive agile development.. 7

Acronyms

AI Artificial Intelligence. xv, 2, 3, 13, 15, 23, 28, 46, 62, 69, 82, 84, 86, 88, 90, 94, 98, 100

API Application Programming Interface. 15, 43, 55, 57, 60, 64, 69, 73, 76

ASAG Automated Short Answer Grading. 22, 24

CRUD Create, Read, Update, Delete. 7

DVCS Distributed Version Control Systems. 5, 6

EDM Educational Data Mining. 3, 6, 19, 20, 26

ETL Extract, Transform, Load. 3, 4, 17, 26, 43–48, 52, 54–58, 69, 100

GDPR General Data Protection Regulation. 61

GenAI Generative Artificial Intelligence. 2, 23

GPT Generative Pre-trained Transformer. 11

JSON JavaScript Object Notation. 44, 52, 64, 66

LLM Large Language Model. 1, 3, 5, 11–26, 58–66, 69–73, 75–77, 93, 94, 97, 99–101, 105, 151

MCQ Multiple-Choice Question. 21, 24

MR Merge Request. 1, 6

MSR Mining Software Repositories. 6, 19, 21

MTTR Mean Time To Repair. 10, 30, 36, 40

NFR Non-Functional Requirement. 39

RAG Retrieval Augmented Generation. xiii, xv, 1–5, 7, 8, 11, 13, 14, 16–29, 31, 39, 40, 43, 45–50, 53, 54, 58, 59, 62–73, 75–95, 97–100, 104, 105

SD Standard Deviation. 81, 82

SE Software Engineering. xv, 2–7, 10, 19–21, 24–26, 29, 30, 35, 41, 52, 57, 79, 88, 89, 91, 92, 94, 95, 99

SQM Software Quality Metric. xv, 1–3, 8–10, 19, 24–26, 29–31, 38–41, 55, 57, 99, 100, 152

VCS Version Control Systems. 5, 6, 19, 52

YAML YAML Ain't Markup Language. 69

Bibliography

[1] F. S. Aðalsteinsson, B. B. Magnússon, M. Milicevic, A. N. Davidsson, and C.-H. Cheng, *Rethinking code review workflows with LLM assistance: An empirical study*, May 23, 2025. DOI: 10.48550/arXiv.2505.16339. arXiv: 2505.16339[cs]. [Online]. Available: http://arxiv.org/abs/2505.16339 (visited on 11/14/2025).

[2] R. M. Parizi, P. Spoletini, and A. Singh, "Measuring team members' contributions in software engineering projects using git-driven technology", in *2018 IEEE Frontiers in Education Conference (FIE)*, Oct. 2018, pp. 1–5. DOI: 10.1109/FIE.2018.8658983. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8658983 (visited on 11/11/2025).

[3] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks", in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 9459–9474. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html (visited on 11/11/2025).

[4] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, *Retrieval augmentation reduces hallucination in conversation*, Apr. 15, 2021. DOI: 10.48550/arXiv.2104.07567. arXiv: 2104.07567[cs]. [Online]. Available: http://arxiv.org/abs/2104.07567 (visited on 11/11/2025).

[6] K. Goddard, A. Roudsari, and J. C. Wyatt, "Automation bias: A systematic review of frequency, effect mediators, and mitigators", *Journal of the American Medical Informatics Association*, vol. 19, no. 1, pp. 121–127, Jan. 1, 2012, ISSN: 1067-5027. DOI: 10.1136/amiajnl-2011-000089. [Online]. Available: https://doi.org/10.1136/amiajnl-2011-000089 (visited on 03/03/2026).

[7] J. Laux and H. Ruschemeier, "Automation bias in the AI act: On the legal implications of attempting to de-bias human oversight of AI", *European Journal of Risk Regulation*, vol. 16, no. 4, pp. 1519–1534, Dec. 2025, ISSN: 1867-299X, 2190-8249. DOI: 10.1017/err.2025.10033. [Online]. Available: https://www.cambridge.org/core/journals/european-journal-of-risk-regulation/article/automation-bias-in-the-ai-act-on-the-legal-implications-of-attempting-to-debias-human-

oversight-of-ai/C97C85015056C09326944DE55CBC4D2C#fn6 (visited on 03/03/2026).

[8] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version control system: A review", *Procedia Computer Science*, The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life, vol. 135, pp. 408–415, Jan. 1, 2018, ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.08.191. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918314819 (visited on 03/03/2026).

[9] M. Blaha, D. LaPlant, and E. Marvak, "Requirements for repository software", in *Proceedings Fifth Working Conference on Reverse Engineering (Cat. No.98TB100261)*, Oct. 1998, pp. 164–173. DOI: 10.1109/WCRE.1998.723186. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/723186 (visited on 03/03/2026).

[10] D. Spinellis, "Git", *IEEE Software*, vol. 29, no. 3, pp. 100–101, May 2012, ISSN: 1937-4194. DOI: 10.1109/MS.2012.61. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6188603 (visited on 03/03/2026).

[12] A. E. Hassan, "The road ahead for mining software repositories", in *2008 Frontiers of Software Maintenance*, Sep. 2008, pp. 48–57. DOI: 10.1109/FOSM.2008.4659248. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4659248 (visited on 03/11/2026).

[13] K. Le, C. Chua, and R. Wang, "Mining software engineering team project work logs to generate formative assessment", in *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, Dec. 2017, pp. 78–83. DOI: 10.1109/APSECW.2017.19. [Online]. Available: https://ieeexplore.ieee.org/document/8312528 (visited on 11/11/2025).

[14] K. Mierle, K. Laven, S. Roweis, and G. Wilson, "Mining student CVS repositories for performance indicators", *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, 2005, ISSN: 0163-5948. DOI: 10.1145/1082983.1083150. [Online]. Available: https://dl.acm.org/doi/10.1145/1082983.1083150 (visited on 11/11/2025).

[15] G. Robles and J. M. Gonzalez-Barahona, "Mining student repositories to gain learning analytics. an experience report", in *2013 IEEE Global Engineering Education Conference (EDUCON)*, Mar. 2013, pp. 1249–1254. DOI: 10.1109/EduCon.2013.6530267. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6530267 (visited on 11/11/2025).

[16] S. Hamer, C. Quesada-López, A. Martínez, and M. Jenkins, "Measuring students' contributions in software development projects using git metrics", in *2020 XLVI Latin American Computing Conference (CLEI)*, Oct. 2020, pp. 531–540. DOI: 10.1109/CLEI52000.2020.00068. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9458363 (visited on 11/11/2025).

[17] K. Schwaber and J. Sutherland, "The scrum guide", *Scrum Alliance*, vol. 21, no. 19, p. 1, 2020.

[18] "IEEE standard for a software quality metrics methodology", *IEEE Std 1061-1998*, pp. i–, Dec. 1998. DOI: 10.1109/IEEESTD.1998.243394. [Online]. Available: https://ieeexplore.ieee.org/document/749159 (visited on 03/03/2026).

[19] F. A. Farris, "The gini index and measures of inequality", *The American Mathematical Monthly*, vol. 117, no. 10, pp. 851–864, Dec. 1, 2010, _eprint: https://www.tandfonline.com/doi/pdf/10 ISSN: 0002-9890. DOI: 10.4169/000298910X523344. [Online]. Available: https://www.tandfonline.com/doi/abs/10.4169/000298910X523344 (visited on 03/03/2026).

[20] T. McCabe, "A complexity measure", *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976, ISSN: 1939-3520. DOI: 10.1109/TSE.1976.233837. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1702388 (visited on 11/11/2025).

[21] "Cognitive complexity | proceedings of the 2018 international conference on technical debt", in *ACM Conferences*. [Online]. Available: https://dl.acm.org/doi/10.1145/3194164.3194186 (visited on 03/03/2026).

[23] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994, ISSN: 1939-3520. DOI: 10.1109/32.295895. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/295895 (visited on 11/27/2025).

[24] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Assessing the bus factor of git repositories", in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, ISSN: 1534-5351, Mar. 2015, pp. 499–503. DOI: 10.1109/SANER.2015.7081864. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7081864 (visited on 03/03/2026).

[27] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt", *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, Jun. 1, 2013, ISSN: 0164-1212. DOI: 10.1016/j.jss.2012.12.052. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121213000022 (visited on 03/04/2026).

[28] W. X. Zhao, K. Zhou, J. Li, *et al.*, "A survey of large language models", *arXiv preprint arXiv:2303.18223*, vol. 1, no. 2, pp. 1–124, 2023.

[29] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need", *Advances in neural information processing systems*, vol. 30, 2017.

[30] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, and others, "Improving language understanding by generative pre-training", 2018.

[31] M. Geva, R. Schuster, J. Berant, and O. Levy, "Transformer feed-forward layers are key-value memories", in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 5484–5495.

[32] Z. Ji, N. Lee, R. Frieske, *et al.*, "Survey of hallucination in natural language generation", *ACM Comput. Surv.*, vol. 55, no. 12, 248:1–248:38, 2023, ISSN: 0360-0300. DOI: 10.1145/3571730. [Online]. Available: https://dl.acm.org/doi/10.1145/3571730 (visited on 11/13/2025).

[33] Y. Gao, Y. Xiong, X. Gao, *et al.*, *Retrieval-augmented generation for large language models: A survey*, Mar. 27, 2024. DOI: 10.48550/arXiv.2312.10997. arXiv: 2312.10997[cs]. [Online]. Available: http://arxiv.org/abs/2312.10997 (visited on 11/10/2025).

[34] M. Cheng, Y. Luo, J. Ouyang, *et al.*, "A survey on knowledge-oriented retrieval-augmented generation", *arXiv.org*, Mar. 11, 2025. [Online]. Available: https://arxiv.org/abs/2503.10677v2 (visited on 11/11/2025).

[35] W. Fan, Y. Ding, L. Ning, *et al.*, "A survey on RAG meeting LLMs: Towards retrieval-augmented large language models", in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '24, New York, NY, USA: Association for Computing Machinery, Aug. 24, 2024, pp. 6491–6501, ISBN: 979-8-4007-0490-1. DOI: 10.1145/3637528.3671470. [Online]. Available: https://dl.acm.org/doi/10.1145/3637528.3671470 (visited on 11/11/2025).

[36] W. Lim, Z. Li, G. Kim, *et al.*, *MacRAG: Compress, slice, and scale-up for multi-scale adaptive context RAG*, May 20, 2025. DOI: 10.48550/arXiv.2505.06569. arXiv: 2505.06569[cs]. [Online]. Available: http://arxiv.org/abs/2505.06569 (visited on 02/23/2026).

[37] J. Bai, D. Ning, Y. You, and J. Chen, "LoopRAG: A closed-loop multi-agent RAG framework for interactive semantic question answering in smart buildings", *Buildings*, vol. 16, no. 1, p. 196, Jan. 2026, ISSN: 2075-5309. DOI: 10.3390/buildings16010196. [Online]. Available: https://www.mdpi.com/2075-5309/16/1/196 (visited on 02/23/2026).

[38] T. Nguyen, P. Chin, and Y.-W. Tai, *MA-RAG: Multi-agent retrieval-augmented generation via collaborative chain-of-thought reasoning*, Oct. 11, 2025. DOI: 10.48550/arXiv.2505.20096. arXiv: 2505.20096[cs]. [Online]. Available: http://arxiv.org/abs/2505.20096 (visited on 02/23/2026).

[39] M. Dahl, V. Magesh, M. Suzgun, and D. E. Ho, "Large legal fictions: Profiling legal hallucinations in large language models", *Journal of Legal Analysis*, vol. 16, no. 1, pp. 64–93, Jan. 1, 2024, ISSN: 2161-7201. DOI: 10.1093/jla/laae003. [Online]. Available: https://doi.org/10.1093/jla/laae003 (visited on 11/11/2025).

[41] Z. Xi, W. Chen, X. Guo, *et al.*, "The rise and potential of large language model based agents: A survey", *Science China Information Sciences*, vol. 68, no. 2, p. 121 101, Jan. 17, 2025, ISSN: 1869-1919. DOI: 10.1007/s11432-024-4222-0. [Online]. Available: https://doi.org/10.1007/s11432-024-4222-0 (visited on 11/14/2025).

[42] S. Yao, J. Zhao, D. Yu, *et al.*, "ReAct: Synergizing reasoning and acting in language models", presented at the The Eleventh International Conference on Learning Representations, Sep. 29, 2022. [Online]. Available: https://openreview.net/forum?id=WE_vluYUL-X (visited on 11/14/2025).

[43] J. Wei, X. Wang, D. Schuurmans, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models", *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, Dec. 6, 2022. [Online]. Available: https://proceedings.neurips.cc/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html (visited on 03/22/2026).

[44] Z. Wang, Z. Cheng, H. Zhu, D. Fried, and G. Neubig, *What are tools anyway? a survey from the language model perspective*, Mar. 18, 2024. DOI: 10.48550/arXiv.2403.15452. arXiv: 2403.15452[cs]. [Online]. Available: http://arxiv.org/abs/2403.15452 (visited on 11/18/2025).

[45] C.-Y. Hsieh, S.-A. Chen, C.-L. Li, *et al.*, *Tool documentation enables zero-shot tool-usage with large language models*, Aug. 2, 2023. DOI: 10.48550/arXiv.2308.00675. arXiv: 2308.00675[cs]. [Online]. Available: http://arxiv.org/abs/2308.00675 (visited on 11/14/2025).

[46] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language models are few-shot learners*, Jul. 24, 2020. DOI: 10.48550/arXiv.2005.14165. arXiv: 2005.14165[cs]. [Online]. Available: http://arxiv.org/abs/2005.14165 (visited on 11/14/2025).

[47] I. Imtiyaz, M. Parvaz, S. Mandha, F. Farooq, and A. K. Kolankeh, "Assessing RAG: A comprehensive review of evaluation frameworks", in *2025 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, Nov. 2025, pp. 490–495. DOI: 10.1109/ICCIKE67021.2025.11318213. [Online]. Available: https://ieeexplore.ieee.org/document/11318213 (visited on 03/01/2026).

[48] L. Zheng, W.-L. Chiang, Y. Sheng, *et al.*, "Judging LLM-as-a-judge with MT-bench and chatbot arena", *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 595–46 623, Dec. 15, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html (visited on 04/15/2026).

[49] S. Es, J. James, L. Espinosa Anke, and S. Schockaert, "RAGAs: Automated evaluation of retrieval augmented generation", in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, N. Aletras and O. De Clercq, Eds., St. Julians, Malta: Association for Computational Linguistics, Mar. 2024, pp. 150–158. DOI: 10.18653/v1/2024.eacl-demo.16. [Online]. Available: https://aclanthology.org/2024.eacl-demo.16/ (visited on 03/01/2026).

[50] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, *ARES: An automated evaluation framework for retrieval-augmented generation systems*, Mar. 31, 2024. DOI: 10.48550/arXiv.2311.09476. arXiv: 2311.09476[cs]. [Online]. Available: http://arxiv.org/abs/2311.09476 (visited on 03/04/2026).

[53] D. Ru, L. Qiu, X. Hu, *et al.*, *RAGChecker: A fine-grained framework for diagnosing retrieval-augmented generation*, Aug. 17, 2024. DOI: 10.48550/arXiv.2408.08067. arXiv: 2408.08067[cs]. [Online]. Available: http://arxiv.org/abs/2408.08067 (visited on 03/04/2026).

[54] X. Hu, D. Ru, L. Qiu, *et al.*, *RefChecker: Reference-based fine-grained hallucination checker and benchmark for large language models*, May 23, 2024. DOI: 10.48550/arXiv.2405.14486. arXiv: 2405.14486[cs]. [Online]. Available: http://arxiv.org/abs/2405.14486 (visited on 03/04/2026).

[55] J. Maynez, S. Narayan, B. Bohnet, and R. McDonald, "On faithfulness and factuality in abstractive summarization", in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Online: Association for Computational Linguistics, Jul. 2020, pp. 1906–1919. DOI: 10.18653/v1/2020.acl-main.173. [Online]. Available: https://aclanthology.org/2020.acl-main.173/ (visited on 03/03/2026).

[56] A. Stolfo, "Groundedness in retrieval-augmented long-form generation: An empirical study", in *Findings of the Association for Computational Linguistics: NAACL 2024*, K. Duh, H. Gomez, and S. Bethard, Eds., Mexico City, Mexico: Association for Computational Linguistics, Jun. 2024, pp. 1537–1552. DOI: 10.18653/v1/2024.findings-naacl.100. [Online]. Available: https://aclanthology.org/2024.findings-naacl.100/ (visited on 03/04/2026).

[57] Z. Jing, Y. Su, and Y. Han, "When large language models meet vector databases: A survey", in *2025 Conference on Artificial Intelligence x Multimedia (AIxMM)*, Feb. 2025, pp. 7–13. DOI: 10.1109/AIxMM62960.2025.00008. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/11005010 (visited on 03/20/2026).

[58] Y. Wang, Y. Song, T. Zhu, *et al.*, *TrustJudge: Inconsistencies of LLM-as-a-judge and how to alleviate them*, Sep. 26, 2025. DOI: 10.48550/arXiv.2509.21117. arXiv: 2509.21117[cs]. [Online]. Available: http://arxiv.org/abs/2509.21117 (visited on 04/15/2026).

[59] P. Wang, L. Li, L. Chen, *et al.*, "Large language models are not fair evaluators", in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 9440–9450. DOI: 10.18653/v1/2024.acl-long.511. [Online]. Available: https://aclanthology.org/2024.acl-long.511/ (visited on 04/15/2026).

[60] L. Shi, C. Ma, W. Liang, X. Diao, W. Ma, and S. Vosoughi, "Judging the judges: A systematic study of position bias in LLM-as-a-judge", in *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, K. Inui, S. Sakti, H. Wang, *et al.*, Eds., Mumbai, India: The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Dec. 2025, pp. 292–314, ISBN: 979-8-89176-298-5. DOI: 10.18653/v1/2025.ijcnlp-long.18. [Online]. Available: https://aclanthology.org/2025.ijcnlp-long.18/ (visited on 04/15/2026).

[61] C. Romero and S. Ventura, "Educational data mining: A survey from 1995 to 2005", *Expert Systems with Applications*, vol. 33, no. 1, pp. 135–146, Jul. 1, 2007, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2006.04.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417406001266 (visited on 11/11/2025).

[62] S. Hamer, C. Quesada-López, A. Martínez, and M. Jenkins, "Measuring students' source code quality in software development projects through commit-impact analysis", in *Information Technology and Systems*, Rocha, C. Ferrás, P. C. López-López, and T. Guarda, Eds., Cham: Springer International Publishing, 2021, pp. 100–109, ISBN: 978-3-030-68418-1. DOI: 10.1007/978-3-030-68418-1_11.

[63] F. Koetter, M. Kochanowski, M. Kintz, B. Kersjes, I. Bogicevic, and S. Wagner, "Assessing software quality of agile student projects by data-mining software repositories.", in *CSEDU (2)*, 2019, pp. 244–251.

[64] J. Grabner, R. Decker, T. Artner, M. Bernhart, and T. Grechenig, "Combining and visualizing time-oriented data from the software engineering toolset", in *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, Sep. 2018, pp. 76–86. DOI: 10.1109/VISSOFT.2018.00016. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8530133 (visited on 03/11/2026).

[65] S. Schintler, "Gizual data layer: Enabling browser-based exploration of git repositories", Master's Thesis, Graz University of Technology, Austria, Dec. 9, 2024. [Online]. Available: https://ftp.isds.tugraz.at/pub/theses/sschintler-2024-msc.pdf.

[66] A. Steinkellner, "Gizual user interface: Browser-based visualisation for git repositories", Master's Thesis, Graz University of Technology, Austria, Dec. 9, 2024. [Online]. Available: https://ftp.isds.tugraz.at/pub/theses/asteinkellner-2024-msc.pdf.

[67] Z. Tang, W. Wang, Z. Zhou, *et al.*, *LLM/agent-as-data-analyst: A survey*, version: 1, Sep. 28, 2025. DOI: 10.48550/arXiv.2509.23988. arXiv: 2509.23988[cs]. [Online]. Available: http://arxiv.org/abs/2509.23988 (visited on 04/16/2026).

[68] M. Romero-Arjona, S. Barakat, A. B. Sánchez, and S. Segura, *The rise of language models in mining software repositories: A survey*, version: 1, Apr. 1, 2026. DOI: 10.48550/arXiv.2604.00787. arXiv: 2604.00787[cs]. [Online]. Available: http://arxiv.org/abs/2604.00787 (visited on 04/17/2026).

[69] G. Colavito, F. Lanubile, N. Novielli, and L. Quaranta, "Leveraging GPT-like LLMs to automate issue labeling", in *Proceedings of the 21st International Conference on Mining Software Repositories*, ser. MSR '24, New York, NY, USA: Association for Computing Machinery, Jul. 2, 2024, pp. 469–480, ISBN: 979-8-4007-0587-8. DOI: 10.1145/3643991.3644903. [Online]. Available: https://dl.acm.org/doi/10.1145/3643991.3644903 (visited on 04/16/2026).

[70] T. Zhang, I. C. Irsan, F. Thung, D. Han, D. Lo, and L. Jiang, "Automatic pull request title generation", in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ISSN: 2576-3148, Oct. 2022, pp. 71–81. DOI: 10.1109/ICSME55016.2022.00015. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9978234 (visited on 04/17/2026).

[71] J. Doughty, Z. Wan, A. Bompelli, *et al.*, "A comparative study of AI-generated (GPT-4) and human-crafted MCQs in programming education", in *Proceedings of the 26th Australasian Computing Education Conference*, ser. ACE '24, New York, NY, USA: Association for Computing Machinery, Jan. 29, 2024, pp. 114–123, ISBN: 979-8-4007-1619-5. DOI: 10.1145/3636243.3636256. [Online]. Available: https://dl.acm.org/doi/10.1145/3636243.3636256 (visited on 11/11/2025).

[72] T. Phung, J. Cambronero, S. Gulwani, *et al.*, "Generating high-precision feedback for programming syntax errors using large language models", in *arXiv.org*, Jan. 24, 2023. [Online]. Available: https://arxiv.org/abs/2302.04662v2 (visited on 11/11/2025).

[73] M. Chen, J. Tworek, H. Jun, *et al.*, *Evaluating large language models trained on code*, Jul. 14, 2021. DOI: 10.48550/arXiv.2107.03374. arXiv: 2107.03374[cs]. [Online]. Available: http://arxiv.org/abs/2107.03374 (visited on 11/19/2025).

[74] W. Xie, J. Niu, C. J. Xue, and N. Guan, *Grade like a human: Rethinking automated assessment with large language models*, May 30, 2024. DOI: 10.48550/arXiv.2405.19694. arXiv: 2405.19694[cs]. [Online]. Available: http://arxiv.org/abs/2405.19694 (visited on 11/11/2025).

[75] Y. Zhou, M. Zhang, Y.-H. Jiang, X. Gao, N. Liu, and B. Jiang, *A study on educational data analysis and personalized feedback report generation based on tags and ChatGPT*, version: 1, Jan. 12, 2025. DOI: 10.48550/arXiv.2501.06819. arXiv: 2501.06819[cs]. [Online]. Available: http://arxiv.org/abs/2501.06819 (visited on 11/11/2025).

[76] H. Qiu, B. White, A. Ding, *et al.*, "SteLLA: A structured grading system using LLMs with RAG", *arXiv.org*, Jan. 15, 2025. [Online]. Available: https://arxiv.org/abs/2501.09092v1 (visited on 11/11/2025).

[78] Z. Rasheed, M. A. Sami, M. Waseem, *et al.*, "AI-powered code review with LLMs: Early results",

[79] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation", in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 931–940. DOI: 10.1109/ICSE.2013.6606642. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6606642 (visited on 11/24/2025).

[80] D. Singh, V. R. Sekar, K. T. Stolee, and B. Johnson, "Evaluating how static analysis tools can reduce code review effort", in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct. 2017, pp. 101–105. DOI: 10.1109/VLHCC.2017.8103456. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8103456 (visited on 11/24/2025).

[82] H. Snyder, "Literature review as a research methodology: An overview and guidelines", *Journal of Business Research*, vol. 104, pp. 333–339, Nov. 1, 2019, ISSN: 0148-2963. DOI: 10.1016/j.jbusres.2019.07.039. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0148296319304564 (visited on 11/12/2025).

[83] A. Begel and T. Zimmermann, "Analyze this! 145 questions for data scientists in software engineering", in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, New York, NY, USA: Association for Computing Machinery, 2014, pp. 12–23, ISBN: 978-1-4503-2756-5. DOI: 10.1145/2568225.2568233. [Online]. Available: https://dl.acm.org/doi/10.1145/2568225.2568233 (visited on 11/12/2025).

[84] R. P. L. Buse and T. Zimmermann, "Information needs for software development analytics", in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12, Zurich, Switzerland: IEEE Press, Jun. 2, 2012, pp. 987–996, ISBN: 978-1-4673-1067-3. (visited on 11/12/2025).

[85] M. Weiß, "A lightweight and integrated software repository mining and visualisation approach for software engineering education", Accepted: 2022-06-03T06:57:09Z, Thesis, Technische Universität Wien, 2022. DOI: 10.34726/hss.2022.102022. [Online]. Available: https://repositum.tuwien.at/handle/20.500.12708/20308 (visited on 11/12/2025).

[86] M. Jansen, "Grading software engineering group projects based on individual contributions", Thesis, Apr. 1, 2022. [Online]. Available: https://research.ou.nl/en/studentTheses/grading-software-engineering-group-projects-based-on-individual-c/ (visited on 11/12/2025).

[87] A. Joshi, S. Kale, S. Chandel, and D. K. Pal, "Likert scale: Explored and explained", *British journal of applied science & technology*, vol. 7, no. 4, pp. 396–403, 2015.

[88] I. Besrour, J. He, T. Schreieder, and M. Färber, "SQuAI: Scientific question-answering with multi-agent retrieval-augmented generation", in *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, ser. CIKM '25, New York, NY, USA: Association for Computing Machinery, Nov. 10, 2025, pp. 6603–6608, ISBN: 979-8-4007-2040-6. DOI: 10.1145/3746252.3761471. [Online]. Available: https://dl.acm.org/doi/10.1145/3746252.3761471 (visited on 02/23/2026).

[89] Z. Guan, X. Yin, Z. Peng, and C. Ni, *RepoTransAgent: Multi-agent LLM framework for repository-aware code translation*, version: 1, Aug. 25, 2025. DOI: 10.48550/arXiv.2508.17720. arXiv: 2508.17720[cs]. [Online]. Available: http://arxiv.org/abs/2508.17720 (visited on 02/23/2026).

[90] P. Zhou, J. Zhu, Y. Shen, and R. Yu, *Context-adaptive synthesis and compression for enhanced retrieval-augmented generation in complex domains*, Aug. 26, 2025. DOI: 10.48550/arXiv.2508.19357. arXiv: 2508.19357[cs]. [Online]. Available: http://arxiv.org/abs/2508.19357 (visited on 02/23/2026).

[91] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering", *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Apr. 1, 2009, ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8 (visited on 02/25/2026).

[94] M. Mittal and A. Sureka, "Process mining software repositories from student projects in an undergraduate software engineering course", in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014, New York, NY, USA: Association for Computing Machinery, 2014, pp. 344–353, ISBN: 978-1-4503-2768-8. DOI: 10.1145/2591062.2591152. [Online]. Available: https://dl.acm.org/doi/10.1145/2591062.2591152 (visited on 11/11/2025).

[95] A. L. Patton and M. McGill, "Student portfolios and software quality metrics in computer science education", *J. Comput. Sci. Coll.*, vol. 21, no. 4, pp. 42–48, Apr. 1, 2006, ISSN: 1937-4771.

[96] J. J. Sandee and E. Aivaloglou, "GitCanary: A tool for analyzing student contributions in group programming assignments", in *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '20, New York, NY, USA: Association for Computing Machinery, Nov. 22, 2020, pp. 1–2, ISBN: 978-1-4503-8921-1. DOI: 10.1145/3428029.3428563. [Online]. Available: https://dl.acm.org/doi/10.1145/3428029.3428563 (visited on 11/11/2025).

[98] M.-C. Lee, "Software quality factors and software quality metrics to enhance software quality assurance", *British Journal of Applied Science & Technology*, vol. 4, no. 21, pp. 3069–3095, 2014.

[99] T. Mladenova, "Software quality metrics – research, analysis and recommendation", in *2020 International Conference Automatics and Informatics (ICAI)*, Oct. 2020, pp. 1–5. DOI: 10.1109/ICAI50593.2020.9311361. [Online]. Available: https://ieeexplore.ieee.org/document/9311361 (visited on 11/11/2025).

[100] K. Mordal, N. Anquetil, J. Laval, A. Serebrenik, B. Vasilescu, and S. Ducasse, "Software quality metrics aggregation in industry", *Journal of Software: Evolution and Process*, vol. 25, no. 10, pp. 1117–1135, 2013, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sm ISSN: 2047-7481. DOI: 10.1002/smr.1558. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1558 (visited on 11/11/2025).

[101] T. B. Alakus, R. Das, and I. Turkoglu, "An overview of quality metrics used in estimating software faults", in *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, Sep. 2019, pp. 1–6. DOI: 10.1109/IDAP.2019.8875925. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8875925 (visited on 11/11/2025).

[103] H. Barkmann, R. Lincke, and W. Löwe, "Quantitative evaluation of software quality metrics in open-source projects", in *2009 International Conference on Advanced Information Networking and Applications Workshops*, May 2009, pp. 1067–1072. DOI: 10.1109/WAINA.2009.190. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5136793 (visited on 11/11/2025).

[104] F. Alaswad and E. Poovammal, "Software quality prediction using machine learning", *Materials Today: Proceedings*, International Conference on Innovative Technology for Sustainable Development, vol. 62, pp. 4714–4720, Jan. 1, 2022, ISSN: 2214-7853. DOI: 10.1016/j.matpr.2022.03.165. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785322014936 (visited on 11/11/2025).

[105] P. Vytovtov and E. Markov, "Source code quality classification based on software metrics", in *2017 20th Conference of Open Innovations Association (FRUCT)*, Apr. 2017, pp. 505–511. DOI: 10.23919/FRUCT.2017.8071355. [Online].

Available: https://ieeexplore.ieee.org/document/8071355 (visited on 11/11/2025).

[106] W. Chen, H. Yu, G. Fan, Z. Huang, and Y. Liang, "Usage patterns of software product metrics in assessing developers' output: A comprehensive study", *Information and Software Technology*, vol. 189, p. 107 935, Jan. 1, 2026, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2025.107935. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584925002745 (visited on 11/11/2025).

[107] J. Lima, C. Treude, F. F. Filho, and U. Kulesza, "Assessing developer contribution with repository mining-based metrics", in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2015, pp. 536–540. DOI: 10.1109/ICSM.2015.7332509. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7332509 (visited on 11/11/2025).

[108] P. Behnamghader, R. Alfayez, K. Srisopha, and B. Boehm, "Towards better understanding of software quality evolution through commit-impact analysis", in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Jul. 2017, pp. 251–262. DOI: 10.1109/QRS.2017.36. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8009930 (visited on 11/11/2025).

[109] G. Gousios, E. Kalliamvakou, and D. Spinellis, "Measuring developer contribution from software repository data", in *Proceedings of the 2008 international working conference on Mining software repositories*, ser. MSR '08, New York, NY, USA: Association for Computing Machinery, 2008, pp. 129–132, ISBN: 978-1-60558-024-1. DOI: 10.1145/1370750.1370781. [Online]. Available: https://dl.acm.org/doi/10.1145/1370750.1370781 (visited on 11/19/2025).

[110] C. Hundhausen, P. Conrad, O. Adesope, and A. Tariq, "Combining GitHub, chat, and peer evaluation data to assess individual contributions to team software development projects", *ACM Trans. Comput. Educ.*, vol. 23, no. 3, 33:1–33:23, Jul. 22, 2023. DOI: 10.1145/3593592. [Online]. Available: https://dl.acm.org/doi/10.1145/3593592 (visited on 12/01/2025).

[112] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence", *Information*, vol. 11, no. 4, p. 193, Apr. 2020, ISSN: 2078-2489. DOI: 10.3390/info11040193. [Online]. Available: https://www.mdpi.com/2078-2489/11/4/193 (visited on 02/18/2026).

[113] K. Srinath, "Python–the fastest growing programming language", *International Research Journal of Engineering and Technology*, vol. 4, no. 12, pp. 354–357, 2017.

[114] L. Yujian and L. Bo, "A normalized levenshtein distance metric", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1091–1095, Jun. 2007, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2007.1078. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4160958 (visited on 02/19/2026).

[118] P. Ohm, "Broken promises of privacy: Responding to the surprising failure of anonymization", *UCLA l. Rev.*, vol. 57, p. 1701, 2009.

[119] "Datenschutzinformation___Academic AI",

[123] N. F. Liu, K. Lin, J. Hewitt, *et al.*, "Lost in the middle: How language models use long contexts", *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024. DOI: 10.1162/tacl_a_00638. [Online]. Available: https://aclanthology.org/2024.tacl-1.9/ (visited on 02/23/2026).

[124] Z. Huma, *Dynamic planning and tool use in next-gen AI agents*, Jan. 6, 2026. DOI: 10.13140/RG.2.2.17401.74084.

[125] J. Wang and Z. Duan, *Agent AI with LangGraph: A modular framework for enhancing machine translation using large language models*, Dec. 5, 2024. DOI: 10.48550/arXiv.2412.03801. arXiv: 2412.03801[cs]. [Online]. Available: http://arxiv.org/abs/2412.03801 (visited on 02/20/2026).

[126] V. Mavroudis, *LangChain v0.3*, Nov. 8, 2024. DOI: 10.20944/preprints202411.0566.v1. [Online]. Available: https://www.preprints.org/manuscript/202411.0566 (visited on 02/20/2026).

[127] T. Guo, X. Chen, Y. Wang, *et al.*, *Large language model based multi-agents: A survey of progress and challenges*, Apr. 19, 2024. DOI: 10.48550/arXiv.2402.01680. arXiv: 2402.01680[cs]. [Online]. Available: http://arxiv.org/abs/2402.01680 (visited on 02/23/2026).

[128] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier", in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, New York, NY, USA: Association for Computing Machinery, Aug. 13, 2016, pp. 1135–1144, ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939778. [Online]. Available: https://dl.acm.org/doi/10.1145/2939672.2939778 (visited on 01/26/2026).

[129] T. Fei, C. Chen, Y. Pan, M. Zheng, and M. Song, *CodeDelegator: Mitigating context pollution via role separation in code-as-action agents*, version: 1, Jan. 21, 2026. DOI: 10.48550/arXiv.2601.14914. arXiv: 2601.14914[cs]. [Online]. Available: http://arxiv.org/abs/2601.14914 (visited on 02/23/2026).

[130] J. White, Q. Fu, S. Hays, *et al.*, *A prompt pattern catalog to enhance prompt engineering with ChatGPT*, Feb. 21, 2023. DOI: 10.48550/arXiv.2302.11382. arXiv: 2302.11382[cs]. [Online]. Available: http://arxiv.org/abs/2302.11382 (visited on 02/23/2026).

[131] P. Vadlapati and Z. Ali, *iRAT: Replanning and controlled retrieval for robust LLM reasoning*, Jul. 22, 2025.

[132] J. Yang, B. Hou, W. Wei, S. Chang, and Y. Bao, *WebDART: Dynamic decomposition and re-planning for complex web tasks*, Oct. 8, 2025. DOI: 10.48550/arXiv.2510.06587. arXiv: 2510.06587[cs]. [Online]. Available: http://arxiv.org/abs/2510.06587 (visited on 02/23/2026).

[133] J. Chen, Z. Yang, Y. Shen, *et al.*, "SurveyGen-i: Consistent scientific survey generation with evolving plans and memory-guided writing", in *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, K. Inui, S. Sakti, H. Wang, *et al.*, Eds., Mumbai, India: The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Dec. 2025, pp. 3687–3714, ISBN: 979-8-89176-298-5. [Online]. Available: https://aclanthology.org/2025.ijcnlp-long.193/ (visited on 02/23/2026).

[134] V. Paramanayakam, A. Karatzas, I. Anagnostopoulos, and D. Stamoulis, "Less is more: Optimizing function calling for LLM execution on edge devices", in *2025 Design, Automation & Test in Europe Conference (DATE)*, ISSN: 1558-1101, Mar. 2025, pp. 1–7. DOI: 10.23919/DATE64628.2025.10992798. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10992798 (visited on 02/24/2026).

[135] T. K. Kim, "T test as a parametric statistic", *Korean Journal of Anesthesiology*, vol. 68, no. 6, pp. 540–546, Nov. 25, 2015. DOI: 10.4097/kjae.2015.68.6.540. [Online]. Available: https://synapse.koreamed.org/articles/1156170 (visited on 04/21/2026).

Book References

[22] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[40] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach* (Prentice Hall series in artificial intelligence). Upper Saddle River: Prentice Hall, 1995, 932 pp., ISBN: 978-0-13-103805-9 978-0-13-360124-4.

[81] R. J. Wieringa, "The design cycle", in *Design Science Methodology for Information Systems and Software Engineering*, R. J. Wieringa, Ed., Berlin, Heidelberg: Springer, 2014, pp. 27–34, ISBN: 978-3-662-43839-8. DOI: 10.1007/978-3-662-43839-8_3. [Online]. Available: https://doi.org/10.1007/978-3-662-43839-8_3 (visited on 11/12/2025).

[92] R. J. Wieringa, "Treatment validation", in *Design Science Methodology for Information Systems and Software Engineering*, R. J. Wieringa, Ed., Berlin, Heidelberg: Springer, 2014, pp. 59–69, ISBN: 978-3-662-43839-8. DOI: 10.1007/978-3-662-43839-8_7. [Online]. Available: https://doi.org/10.1007/978-3-662-43839-8_7 (visited on 02/25/2026).

[93] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Essential areas in empirical research", in *Experimentation in Software Engineering*, C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Eds., Berlin, Heidelberg: Springer, 2024, pp. 27–49, ISBN: 978-3-662-69306-3. DOI: 10.1007/978-3-662-69306-3_3. [Online]. Available: https://doi.org/10.1007/978-3-662-69306-3_3 (visited on 02/25/2026).

[102] S. D. Conte, H. E. Dunsmore, and Y. E. Shen, *Software engineering metrics and models*. USA: Benjamin-Cummings Publishing Co., Inc., Sep. 1986, ISBN: 978-0-8053-2162-3.

[111] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Surveys", in *Experimentation in Software Engineering*, C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Eds., Berlin, Heidelberg: Springer, 2024, pp. 65–71, ISBN: 978-3-662-69306-3. DOI: 10.1007/978-3-662-69306-3_5. [Online]. Available: https://doi.org/10.1007/978-3-662-69306-3_5 (visited on 12/15/2025).

[136] J. Cohen, *Statistical power analysis for the behavioral sciences*. routledge, 2013.

Online References

[5] "Annex III - high-risk AI systems referred to in article 6(2)", AI Act. (), [Online]. Available: https://ai-act-law.eu/annex/3/ (visited on 02/12/2026).

[11] "What is GitLab and how to use it? | simplilearn", Simplilearn.com. (), [Online]. Available: https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab (visited on 03/03/2026).

[25] "What is mean time to repair (MTTR)? | IBM". (Apr. 27, 2023), [Online]. Available: https://www.ibm.com/think/topics/mttr (visited on 03/04/2026).

[26] "Code Quality, Security & Static Analysis Tool with SonarQube". (), [Online]. Available: https://www.sonarsource.com/de/products/sonarqube/ (visited on 02/18/2026).

[51] "DeepEval by confident AI - the LLM evaluation framework". (), [Online]. Available: https://deepeval.com/ (visited on 03/04/2026).

[52] "API reference - TruLens". (), [Online]. Available: https://www.trulens.org/reference/ (visited on 03/04/2026).

[77] "Autonomous generative AI agents: Under development", Deloitte Insights. (), [Online]. Available: https://www.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2025/autonomous-generative-ai-agents-still-under-development.html (visited on 11/18/2025).

[97] "Assessing the bus factor of git repositories". (), [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7081864?casa_token=gUaYlTww6vMAAAAA:NStu0sNfNEeU3bUmlj2pCrKUV29UgXPf5MrC0Ue9GFoEgyaCSvYIIYflLv (visited on 03/03/2026).

[115] ProgrammingKnowledge. "How to sonarqube setup from scratch and code analysis". (Jan. 29, 2024), [Online]. Available: https://www.youtube.com/watch?v=6vdRvz_LnbQ (visited on 02/19/2026).

[116] "Art. 5 GDPR – principles relating to processing of personal data", General Data Protection Regulation (GDPR). (), [Online]. Available: https://gdpr-info.eu/art-5-gdpr/ (visited on 02/12/2026).

[117] “Gemini apps privacy hub - gemini apps help”. (), [Online]. Available: https://support.google.com/gemini/answer/13594961?hl=en#pn_what_data (visited on 02/12/2026).

[120] “Article 13: Transparency and provision of information to deployers | EU artificial intelligence act”. (), [Online]. Available: https://artificialintelligenceact.eu/article/13/ (visited on 04/20/2026).

[121] “Article 14: Human oversight | EU artificial intelligence act”. (), [Online]. Available: https://artificialintelligenceact.eu/article/14/ (visited on 04/17/2026).

[122] “Article 15: Accuracy, robustness and cybersecurity | EU artificial intelligence act”. (), [Online]. Available: https://artificialintelligenceact.eu/article/15/ (visited on 04/20/2026).

[137] “Section 2: Requirements for high-risk AI systems | EU artificial intelligence act”. (), [Online]. Available: https://artificialintelligenceact.eu/section/3-2/ (visited on 04/20/2026).

[138] “GitLab”, about.gitlab.com. (), [Online]. Available: https://about.gitlab.com/ (visited on 02/18/2026).

Appendices

APPENDIX A

Software Quality Metrics Expert Interviews

SEPR Software Quality Metrics Survey

* Indicates required question

Demographics

1. How long have you been working in software engineering education? *

Mark only one oval.

- < 1 year
- 1-2 years
- 2-5 years
- 5-10 years
- 10-15 years
- > 15 years

2. How many groups are you supervising during a semester? *

Mark only one oval.

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- > 8

Software Quality Metrics

The following metrics are taken from 24 different papers and articles from both industry and education.

3. Are the **Lines of Code (per Student)** relevant? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

4. Is the **Cyclomatic Complexity** relevant? *

= Number of independent paths (using the control flow graph)
https://en.wikipedia.org/wiki/Cyclomatic_complexity

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

5. Is the **Cognitive Complexity** relevant? *

Describes, how difficult the code is to understand.
"less Cognitive Complexity = better Readability"
Factors:

- Function Length
- Nesting of Structural Control Flow
- Sequences of Logical Operators
- Recursion

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

6. Are the **Halstead Metrics** relevant? *

- η_1 = the number of distinct operators
- η_2 = the number of distinct operands
- N_1 = the total number of operators
- N_2 = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated estimated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

7. Are **CK Metrics (Chidamber and Kemerer)** relevant? *

- Weighted Methods per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of Children (NOC)
- Coupling Between Object Classes (CBO)
- Response for a Class (RFC)
- Lack of Cohesion in Methods (LCOM)

https://medium.com/@benkaddourmed54/analyzing-ck-metrics-results-and-quality-assessment-a70ba56534f0

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

8. Is the **Number of Commits** relevant? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

9. Are **Files/Lines added/deleted (per Commit)** relevant? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

10. Is the **Time Spent (per Student)** relevant? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

11. Is the **Number of Bugs** relevant? *

Calculated by static code analysis tools

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

12. Is the **Number of Code Smells** relevant? *

Calculated by static code analysis tools

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

13. Is the **Work Distribution** (of time spent or commits) relevant? *

Either measured in total or with Gini Coefficient (0 = total Equality, 1 = total Inequality)

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

14. Is the **Contribution Frequency** (per Student) relevant? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

15. Is the **Commit Message Quality** relevant? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

16. Is the **Mean Time to Repair (MTTR)** relevant? *

= The average time between when a bug was introduced until it was fixed

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

17. Is the **Documentation Written** relevant? *

How much was written / How was the quality of it?

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

18. Is the **Git Blame** data relevant?

→ The final code ownership

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

19. Is the **Code Duplication** (of the project) relevant?

Calculated by static code analysis tools

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

20. Is **Tech Debt** relevant? *

"Time required to refactor and fix Maintainability Issues".
Calculated by static code analysis tools

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

21. Are **Security Vulnerabilities** relevant? *

Calculated by static code analysis tools

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

Additional Metrics

22. Is the **Test Coverage** relevant? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

23. Is the **Bus Factor** relevant? *

"The minimum number of team members that have to suddenly disappear from a project before the project stalls due to lack of knowledgeable or competent personnel."

SEPR Context: Is the percentage of code owned by one person above e.g. 50%?

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

Skip to question 24

Additional Questions

24. What are additional important metrics, that were not mentioned?

25. When applicable for Metrics: Should they be on a per-file basis? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly relevant

26. Additional Notes

Information Needs during the Assessment

27. What did each student in group XY work on? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

28. Which top 5 issues remained open the longest in group XY? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

29. Did someone in group XY only work on minor fixes and documentation, rather than critical features? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

30. Where is the highest collaboration in group XY's codebase? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

31. Which issues have the most time spent for group XY? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

32. Is there proper testing documentation in group XY's wiki? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

33. Are there Knowledge Silos in the project of group XY? Are these critical features? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

34. How often did group XY merge code in the last week before the deadline (placeholder 31.01.2026)? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

35. Is one student of group XY carrying the team based on code volume and complexity? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

36. Which file or module of group XY was reworked the most? *

Mark only one oval.

	1	2	3	4	5	
Irrel	◯	◯	◯	◯	◯	Highly Relevant

37. What other questions would be relevant?

APPENDIX B

LLM Agent Prompts

```
PLANNER_SYSTEM_PROMPT = """
You are a Senior Technical Project Lead planning a comprehensive analysis of student software repositories.
The user has provided multiple questions or objectives. Your goal is to create a single, unified,
sequential plan that addresses all of them efficiently, but comprehensively.

Guidelines:
1. Flat Structure: Create a flat list of steps. Do not use nested loops, indentations, or multi-line control structures.
   - Bad: 2. For each project found: 3. Get the metadata
   - Good: 2. Retrieve metadata for all projects identified in step 1.
2. Atomic Actions: Each step should be an atomic action, not several.
   - Bad: "Document the specific files modified, commits made, and code changes for each student."
   - Good: "Document the commits made for each student."
3. Implicit Iteration: If you need to analyse multiple items (like students or projects), write a single step for "all" or "each" of them. The system will handle the looping automatically.
4. Efficiency: Do not repeat steps. Fetch common data (student lists, commit history) once at the beginning.
5. Selective Analysis: If a collection (e.g., wiki documents, commits) is large (over 10 items):
   - Step A: Retrieve the list of metadata (filenames/paths).
   - Step B: Identify items from that list relevant to the goal.
   - Step C: Analyse only those specific candidates.

Available Tools: {tools}

Example plan for multiple questions (Who worked on the project? Was there testing?):
1. Retrieve a list of all students of the project.
2. Retrieve commit statistics for all students found in step 1.
3. List all files in the repository to identify testing directories.
4. Read the contents of the test-related wiki documents to evaluate testing documentation.

Output Format:
- Numbered list.
- One step per line.
- The plan must not contain a summary/report as the last step; this will be done later on.
- No introductory text.
"""
```

Listing B.1: Planner Agent Prompt.

```
EXECUTOR_PROMPT = """
You are an Agent Executor. Your task is to execute Step {step_index}
    of the following plan:

Plan:
{plan}

Current Step:
{current_step}

Context (What is known so far):
{analysis_summary}

The schema of the database:
{schema}

Instructions:
1. Choose the single most appropriate available tool to accomplish
    the current step.
2. The result should be a tool call only.
3. Do not provide a textual description of what you are doing.
"""
```

Listing B.2: Executor Agent Prompt.

```
SUMMARISER_PROMPT = """
You are a Data Synthesiser. Summarise the raw tool output into a
    concise, factual insight.

Instructions:
1. Preserve Identifiers: If the tool returns a list of items (
    filenames, student IDs, project names), list them explicitly.
    - Good: "Found 5 text files: file_a.txt, file_b.txt, ..."
2. Time Log Grouping: Group logs by Issue/Task title or ID. Sum total
     hours for each functional area and list in descending order of
    effort.
3. Commit Impact: Highlight commits with significant line changes
    (+/-) or clear feature labels (e.g., [US], [feat], [fix]).
4. No Noise: Remove formatting boilerplate, but keep the data (
    numbers, dates, names).

Original Goal: {original_question}
Current Step: {current_step}
Tool Output: {tool_output}

Output only your summary. This will be added to the permanent
    analysis log.
"""
```

Listing B.3: Summariser Agent Prompt.

```
REPLANNER_PROMPT = """
You are a Plan Refiner. Your job is to strictly maintain the original
    plan while handling errors or step expansions.

Original Goal: {original_question}
Context:
- Just Finished Step: {last_step}
- Findings: {last_summary}
- Raw Tool Output: {last_tool_output}
- Remaining Plan: {remaining_steps}

Available Tools: {tools}

Instructions:
1. Error Recovery: If the step failed but findings suggest a fix (e.g
   ., correct filepath), generate a Retry Step with the corrected
   parameter.
2. Step Expansion: If the next step implies iteration (e.g., "for
   each student", "for all ...") and findings contain the list,
   replace that step with Specific Atomic Steps.
3. Selective Expansion: If the list is large (>10 items), only
   generate steps for items highly likely to contain relevant
   information. For that, compare the item names (filenames, paths,
   IDs) against the 'Original Goal'.
4. Depth Search: If a specific file wasn't found, add a step to
   explore the file structure more deeply.

Constraints:
- Always output the full remaining plan.
- No meta-tags (e.g., [Retry], [Expanded Steps]).
- Output only the list of new remaining steps, one per line.
"""
```

Listing B.4: Replanner Agent Prompt.

```
REPORTER_PROMPT = """
You are an Expert Software Engineer Educator. Write a comprehensive
    markdown report answering the user's question.

User Question: {question}
Collected Evidence: {analysis_summary}

Reporting Guidelines:
1. Evidence-Based Claims: Back every claim with findings. Refer to
    the Step number in parentheses, e.g., '(Step 10)'.
2. Balanced Evaluation: Differentiate between boilerplate/refactoring
     volume and complex feature implementation.
3. Multi-Metric Synthesis: Cross-reference 'Time Logs' with 'Commits'
     to identify significant functional efforts.
4. Data Integrity: Use exact numbers, dates, and identifiers (Issue
    #, MR !) found in the evidence.
5. Structural Clarity:
   - Use headings based on the original question.
   - Organize by 'Functional Area' (e.g., Backend, UI/Admin) rather
      than tool type.
   - Include an "Analysis Methodology" section listing the steps
      carried out.

Tone: Academic, professional, and grounded in data. Avoid over-
    praising.
"""
```

Listing B.5: Reporter Agent Prompt.

APPENDIX C

Available Tools of the RAG System

This appendix lists all tools available to the Executor agent and describes their required inputs and the information they return. If an error occurs or a requested resource cannot be found (e.g., an invalid project key or student name), each tool returns a textual error message indicating the cause of the failure.

All tools return their results as formatted text strings, which are readable both for the LLM and for manual human verification.

---

list_available_projects

Purpose: Retrieves a list of all projects available in the database.

Inputs: None.

Output: A list of project keys representing all projects that can be queried by other tools.

Note: This tool is typically used at the beginning of an interaction when the user has not provided a valid project key or when a potential spelling mistake must be resolved.

---

get_students_of_project

Purpose: Returns the list of students who are members of a given project.

Inputs: Project key.

Output: A list of student names associated with the specified project.

get_project_metadata

Purpose: Provides general metadata about a project.

Inputs: Project key.

Output: Project-level information including:

- Study semester,
- Subject of the project,
- Local paths of the source code and wiki repositories,
- Activity timeline (first and last recorded activity).

The activity timeline distinguishes between different activity types such as time logs, commits, created issues, and merge requests.

---

get_project_code_contribution_statistics

Purpose: Summarises code contribution statistics for each student in the source code repository.

Inputs: Project key.

Output: For each student:

- Number of commits,
- Net lines added over the project duration,
- Number of owned lines at analysis time,
- Average and median values of:
  - Lines of code added per commit,
  - Lines of code deleted per commit,
  - Files touched per commit.

---

get_project_wiki_contribution_statistics

Purpose: Provides contribution statistics for the project's wiki repository.

Inputs: Project key.

Output: The same metrics as **get_project_code_contribution_statistics**, but computed for the wiki repository instead of the source code repository.

---

get_project_workload_distribution_statistics

Purpose: Combines several SQM to describe workload distribution within a project.

Inputs: Project key.

Output:

- Bus factor for frontend and backend components, including responsible students and their percentage of authored code,

- Gini inequality coefficients for:
  - Time spent,
  - Number of commits,
  - Net lines of code.
- per-student statistical overview of:
  - Total time spent,
  - Time spent reviewing merge requests,
  - Number of commits,
  - Net lines of code written.

All code-related metrics exclude initial project setup commits and merge commits to reduce noise from refactoring and conflict resolution.

---

get_project_sonar_metrics

Purpose: Retrieves static code analysis metrics produced by SonarQube.

Inputs: Project key.

Output: For frontend and backend components:

- Cognitive complexity,
- Number of code smells,
- Number of security vulnerabilities,
- Code duplication.

Additionally, backend test coverage is reported.

---

get_project_sonar_issue_details

Purpose: Returns detailed information about issues detected by the SonarQube scan.

Inputs: Project key.

Output: A list of up to 500 detected issues, including:

- Severity (Blocker, Critical, Major),
- Issue type (code smell or security vulnerability),
- Issue message,
- Code location (frontend/backend, filepath, line).

Minor and informational issues are reported only as aggregated counts.

---

get_student_code_commit_history

Purpose: Returns commit history for a specific student in the project's source code repository.

Inputs:

- Project key,

- Student name,
- Optional limit parameter (default: 300 commits).

Output: For each commit:

- Commit date,
- Short commit hash,
- Commit message,
- Number of added and deleted lines,
- Number of modified files,
- Associated merge request (if applicable).

---

get_student_wiki_commit_history

Purpose: Returns commit history for a specific student in the project's wiki repository.

Inputs:

- Project key,
- Student name,
- Optional limit parameter (default: 300 commits).

Output: The same information as **get_student_code_commit_history**, but for the wiki repository.

---

get_student_time_logs

Purpose: Retrieves all tracked time logs for a student.

Inputs:

- Project key,
- Student name.

Output:

- total accumulated time spent,
- For each commit:
  - Log date,
  - Duration,
  - Related issue or merge request ID and title.

---

get_contribution_frequency

Purpose: Provides a temporal overview of project activity.

Inputs:

- Project key,
- Optional student name.

Output:

Weekly statistics of time spent and number of source code commits.

This tool can be applied either to the entire project or to an individual student.

---

get_issue_statistics

Purpose: Summarises issue-related activity in a project.

Inputs: Project key.

Output:

- Total number of issues,
- Number of open and closed issues,
- Average and median discussion intensity (excluding system-generated notes),
- Top five issues with the most comments,
- Distribution of created and assigned issues per student (including unassigned issues).

---

get_merge_request_statistics

Purpose: Summarises merge request activity in a project.

Inputs: Project key.

Output:

- Total number of merge requests,
- Number of open, closed, and merged merge requests,
- Average and median review time,
- Average and median discussion intensity (excluding system-generated notes),
- Top five issues with the most comments,
- Distribution of authored merge requests per student.

---

get_issue_details

Purpose: Returns GitLab issue details ordered by recency.

Inputs:

- Project key,
- Optional limit parameter (default: 200 issues).

Output: For each issue:

- State (open/closed),
- GitLab ID,
- Title,
- Assigned student,
- Number of code commits that reference the issue,
- Labels.

get_merge_request_details

Purpose: Returns GitLab merge request details ordered by recency.

Inputs:

- Project key,
- Optional limit parameter (default: 200 merge requests).

Output: For each merge request:

- State (open/closed/merged),
- GitLab ID,
- Title,
- Description,
- Author,
- Branches (source → target).

---

get_collaboration_interactions

Purpose: Returns an interactions matrix based on student comments on GitLab issues and merge requests. It identifies who commented and reviewed whose work.

Inputs: Project key.

Output: For each student pair with existing interactions, one entry:
"*Author* → *Recipient: number of comments*". Self-comments are excluded.

---

get_wiki_document_list

Purpose: Provides an overview of all existing documents in the wiki repository.

Inputs: Project key.

Output: A list of the filepaths of all wiki documents.

---

read_wiki_file

Purpose: Reads a single wiki file. Supported file types are *.md*, *.txt*, *.docx*, and *.pdf*. This tool allows only reading files within the base directory of the wiki repository.

Inputs:

- Project key,
- Relative path to the wiki file inside the wiki base directory. This matches the result from **get_wiki_document_list**

Output: The content of the specified wiki document.

get_project_branches

Purpose: Returns an overview of all source code repository branches. The list may be incomplete as it includes only the branches available at the time of analysis and excludes branches that were deleted after merging.

Inputs: Project key.

Output: A list of all branches.

---

get_file_change_details

Purpose: Provides an overview of changed files in one commit.

Inputs:

- Project key,
- Commit hash

Output: For each file touched:

- File path,
- Number of added lines,
- Number of deleted lines.

---

get_repository_structure

Purpose: Provides an overview of the hierarchical structure of the code repository. Directories that contain only other directories, with no files, are collapsed.

Inputs:

- Project key,
- Depth (default: 3 directory levels),
- Optional relative path (default: base directory of the code repository)

Output: A hierarchical list of all files inside the given relative path until the given depth.

---

get_file_ownership

Purpose: Returns the ownership information of all files inside the provided path pattern.

Inputs:

- Project key,
- File path pattern. (e.g., *backend/src/main/java/*)

Output: For each file inside the given pattern:

- Name of the student,
- Number of lines that they own.

get_discussion_thread

Purpose: Returns a detailed list of all student comments of a merge request or issue ordered chronologically.

Inputs:

- Project key,
- Entity type ('*issue*' or '*mr*'),
- GitLab ID of the entity.

Output:

- Type (issue or merge request),
- GitLab ID,
- Title,
- State,
- Author,
- Creation Date,
- Number of comments,
- For each comment:
  - Author,
  - Date,
  - Content of the comment.

---

read_source_file

Purpose: Reads the content of the specified source code file. In addition, provides a summary of ownership data. This tool allows only reading files inside the base directory of the source code repository.

Inputs:

- Project key,
- File name or relative path (e.g. *backend/endpoint/UserEndpoint.java*)

Output:

- File path,
- File size,
- Ownership summary,
- Full content of the file.

---

run_custom_sql_query

Purpose: Executes a custom SQL query and returns the result from the database. This tool uses read-only database access.

Inputs: Project key, PostgreSQL query

Output: Result of the SQL query.

APPENDIX D

Example of a generated RAG Report

Analysis Report: Report Example for Appendix

Model: glm-4.7-355b **Temperature:** 0.0

Used input queries

- What did each student of project group 25ws-subj-div-xx do?
- Is one student of group 25ws-subj-div-xx carrying the team based on code volume and complexity?
- Did someone in group 25ws-subj-div-xx only work on minor fixes and documentation, rather than critical features?
- Is there proper testing documentation in the wiki of group 25ws-subj-div-xx?

Everything after this point is LLM-generated.

Project Group Analysis Report: 25ws-subj-div-xx

Analysis Methodology

This report synthesizes data from multiple sources to evaluate the contributions and workload distribution of the project team. The analysis follows these steps:

1. **Identification:** Listing all team members (Step 1).
2. **Quantitative Metrics:** Analyzing code volume (commits, net lines), time logs, and inequality metrics (Gini coefficient) (Steps 2, 3).
3. **Qualitative Assessment:** Reviewing commit histories to identify specific features, complexity, and testing efforts (Steps 6-11).
4. **Documentation Review:** Examining wiki contributions and the quality of testing documentation (Steps 5, 20-28).

---

1. Individual Contribution Analysis

The team consists of six students: Student A, Student B, Student C, Student D, Student E, and Student F. Below is an in-depth analysis of their contributions categorized by functional area.

Student A

- **Core Features:** Student A focused heavily on the **Merchandise & Checkout** module (Feature 8.1), implementing stock validation, mixed order logic, and filtering. They also drove the **News Management** system (Features 3.1, 3.4), building the creation and editing interfaces, and developed **Event Analytics** (Feature 5.1) (Step 6).
- **Order & User Management:** They implemented the order history views with status badges (Feature 6.5) and added email notification support for account creation (Feature 9.1) (Step 6).
- **Testing:** Authored substantial unit and integration tests for order endpoints and checkout logic (Step 6).

- **Workload:** Logged 90.58 hours, with significant time spent on Merchandise (15.25h) and Meetings (16.59h) (Step 12).

Student B

- **Search & UI:** Student B was the primary contributor for **Event Search** (Feature 5.2-5), implementing the search page, venue suggestions, and backend integration. They also led the **Frontend Beautification** efforts (Feature 0.7), updating UI styling across login, registration, and checkout (Step 7).
- **Core Logic:** Implemented the **User Registration** flow (Feature 1.1) and the **PDF Invoice** generation service (Feature 7.1) (Step 7).
- **Loyalty System:** Built the service for loyalty point redemption and integrated it into the checkout (Feature 8.5) (Step 7).
- **Workload:** Logged 95.59 hours, with the largest block dedicated to Event Search (17.84h) and PDF generation (15.50h) (Step 13).

Student C

- **Ticketing Core:** Student C handled the most complex transactional logic, implementing the core **Ticket Buying & Checkout** flow (Feature 6.3), connecting seat selection to the backend. They also implemented **Reservations** (Features 6.4 & 6.7) (Step 8).
- **Testing:** Authored extensive test suites for checkout, ticket buying, and reservations, contributing over 1,600 lines of tests alone (Step 8).
- **Merchandise:** Implemented the backend tests for the merchandise module (Step 8).
- **Workload:** Logged 90.08 hours, focusing heavily on Reservations (12.0h) and Checkout (11.33h) (Step 14).

Student D

- **Data Architecture:** Student D was responsible for the **Core Data Model** (Feature 0.3), creating and modifying Entities, Repositories, and DataGenerators. They performed large-scale refactoring of service implementations (Step 9).
- **Security & Loyalty:** Implemented the **Password Reset** mechanisms (Feature 9.4) and the backend logic for the **Loyalty System** (Feature 8.4) (Step 9).
- **Invoicing:** Built the Checkout Service and DTOs and implemented backend logic for invoices (Step 9).
- **Workload:** Logged 91.67 hours. While their net lines are lower, They spent significant time on foundational tasks like Entity Setup (9.0h) and Meetings (17.67h) (Step 15).

Student E

- **Complex UI/Visualization:** Student E developed the **Seatmap** features (Features 6.1/6.2), implementing complex zoom/pan/selection components and curve rendering algorithms. They also built the interface for **Creating Performances** (Feature 5.6) (Step 10).
- **Testing & QA:** Actively worked to increase test coverage, adding tests for UserService, NewsEndpoint, and MerchandiseService. They handled frequent merge operations and pipeline maintenance (Step 10).
- **Workload:** Logged 99.25 hours, the highest in the group, with deep focus on the Seatmap (20.5h) and Event Creation (19.0h) (Step 16).

Student F

- **Admin & Invoicing:** Student F implemented the **Admin Dashboard** (Feature 9.5) and advanced **Invoicing** features (Feature 7.2), including merged PDF downloads and credit note generation. They handled **Ticket Cancellation** logic (Feature 6.8) (Step 11).
- **Data & Search:** Implemented scalable faker-based data generation (Feature 0.10) and contributed to the **Event Search** backend (Feature 5.2-5) (Step 11).
- **Wiki & Documentation:** Student F was the primary contributor to the Wiki, adding 5,258 net lines (Step 5).
- **Workload:** Logged 102.17 hours, the highest total time, focused on Cancellation (14.0h) and Admin Locking (14.0h) (Step 17).

---

2. Workload Balance: Is one student carrying the team?

No, there is no evidence that a single student is carrying the team. While there are variations in code volume, the overall workload is exceptionally balanced.

- **Time Balance:** The logged hours are tightly clustered between 90.08h (Student C) and 102.17h (Student F). The **Time Inequality Gini Coefficient is 0.03**, which indicates near-perfect equality in time investment (Step 3).
- **Code Volume Distribution:**
  - **Student F** has the highest code volume (8,874 net lines) and commits (144).
  - However, **Student C** achieved 5,100 net lines in only 74 commits, indicating they handled very large, complex features (like the core checkout) rather than small iterative changes.
  - **Student E** logged the most hours (99.25h) for complex UI work (Seatmap), which often yields fewer lines of code but high complexity.
- **Complexity Distribution:**
  - The **Bus Factor** analysis shows the team is not siloed around one person. The Frontend relies on Student E (28%) and Student B (27%), while the Backend relies on Student F (31%) and Student D (25%) (Step 3).
  - **Student D** has the lowest net lines (2,387), but they handled the foundational Data Model and large-scale refactoring (Step 9). This type of work often results in lower "net" lines due to the nature of modifying existing code, but it is critical to the project's architecture.

Conclusion: The team demonstrates a healthy distribution of labor. Student F is the most prolific contributor by volume, but the Gini coefficients (0.16 for commits, 0.2 for net lines) and the balanced time logs suggest a collaborative effort rather than a dependency on a single "hero" developer.

---

3. Assessment of Minor Fixes and Documentation Roles

No student focused exclusively on minor fixes or documentation. Every team member implemented critical features.

- **Student D (Lowest Volume):** Although Student D contributed the fewest net lines (2,387), their work was architectural. They implemented the **Core Data Model** (Entities, Repositories) and **Security** (Password Reset) (Step 9). They also performed large-scale refactoring. This is high-impact work, not minor maintenance.
- **Student B (Lowest Wiki Volume):** Student B contributed only 1 net line to the Wiki (Step 5). However, they implemented critical features like **PDF Invoices**, **Event Search**, and **User Registration** (Step 7).

Their lack of wiki contribution is offset by their feature delivery.

- **Documentation Effort:** Student F contributed significantly to documentation (5,258 wiki lines), but they also led **Ticket Cancellation**, **Admin Management**, and **Data Generation** (Step 11). Documentation was a shared responsibility, not a primary role for any single student to the exclusion of coding.

Conclusion: All students are engaged in feature implementation. The student with the lowest code output (Student D) handled backend architecture, and the student with the lowest wiki output (Student B) handled core business logic.

---

4. Testing Documentation Review

Yes, there is proper and comprehensive testing documentation in the wiki.

- **Test Plan:** The group defined a clear testing strategy in `Documents/Projektauftrag/Testplan.md`. It outlines three levels of testing (Unit, Integration, System) with specific coverage goals (e.g., 70-80% line coverage for Unit tests) (Step 22).
- **Manual System Tests:** The document `Documents/Manuel-System-Tests.md` defines **54 distinct test cases** organized into 9 User Stories. It covers Normal, Error, and Special scenarios with detailed preconditions and expected outcomes (Step 21).
- **Execution & Reporting:** The group documented two full test runs:
  - **Test Run 1 (Jan 13, 2026):** Executed by Student E. 45/54 tests passed. 3 failures were identified (e.g., Account deletion failure, missing error messages) and logged in `Documents/Tests/Report-1.md` (Steps 23, 25).
  - **Test Run 2 (Jan 28, 2026):** Executed by Student F. All 54/54 tests passed. This indicates the team successfully addressed the issues found in the first run. Results are in `Documents/Tests/Report-2.md` (Steps 24, 26).
- **Templates:** The group utilized standardized templates for Test Logs and Reports to ensure consistency (Steps 27, 28).

Conclusion: The testing documentation is exemplary. It moves beyond a simple plan to include evidence of execution, error tracking, and verification of fixes (100% pass rate in the final run).

APPENDIX E

Evaluation Interviews

Evaluation of RAG-based Assessment Reports

* Indicates required question

1. What is your role in the course? *

Mark only one oval.

- ◯ Tutor *Skip to question 2*
- ◯ Study Assistant *Skip to question 21*

Tutor Evaluation

Introduction on how to use the report and the tool outputs/intermediate summaries

2. How clear was the structure of the report? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very clear

3. How useful was the generated report for answering the given questions? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very useful

4. Did the report provide more/less relevant information than you found without the report? *

Mark only one oval.

	1	2	3	4	5	
Muc	◯	◯	◯	◯	◯	Much more

5. How trustworthy did you find the report? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Fully trustworthy

6. How often did you consult the intermediate tool outputs or summaries to verify the report? *

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Very often

7. How helpful were the intermediate outputs for verification?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very helpful

8. How often did you consult other data sources (e.g., Gitlab, Meeting Notes, Binocular,...) to verify the report? *

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Very often

9. How transparent was the reasoning from raw data to final report? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very transparent

10. Did you notice any discrepancies between the generated report and the actual repository data (code, commits, issues, documentation)? *

Mark only one oval.

- ◯ No discrepancies
- ◯ Minor Discrepancies
- ◯ Major Discrepancies

11. What type of discrepancies did you observe? (multiple choices possible)

Tick all that apply.

- ☐ Incorrect metric values
- ☐ Incorrect interpretation of code
- ☐ Missing relevant information
- ☐ Hallucinated or non-existent facts
- ☐ Wrong conclusions
- ☐ Other: _____

12. If differences in the findings occurred, were they justified by information in the report or intermediate outputs?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Fully justified

13. How much additional effort was required to verify or correct the report's statements? *

Mark only one oval.

	1	2	3	4	5	
Non	◯	◯	◯	◯	◯	Very high effort

14. The report helped me detect issues or insights I previously overlooked or would not have found easily without the tool . *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

15. The report reduced my manual search effort. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

16. Using the report reduced my cognitive effort during the assessment task. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

17. Overall, how reliable did you find the report compared to your manual assessment? *

Mark only one oval.

	1	2	3	4	5	
Muc	◯	◯	◯	◯	◯	Much better

18. Would you use such a report (including intermediate outputs) in a real assessment scenario? *

Mark only one oval.

	1	2	3	4	5	
Unlil	◯	◯	◯	◯	◯	Definitely

19. What did you find most useful?

20. What should be added or improved?

Study Assistant Evaluation

Demographics

21. How long have you been working in software engineering education? *

Mark only one oval.

- < 1 year
- 1-2 years
- 2-5 years
- 5-10 years
- 10-15 years
- > 15 years

Study Assistant Evaluation

22. How clear was the structure of the report? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very clear

23. How trustworthy did you find the report? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Fully trustworthy

24. Optional comments on the trustworthyness

_____

25. How complete was the information provided? (Regarding the used questions) *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Fully complete

26. Optional comments on the completeness of information

_____

27. The report provided relevant context for understanding the group's work. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

28. The report effectively cross-referenced multiple data sources (code commits, time logs, documentation, and quality metrics) to establish a comprehensive context. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

29. How transparent was the reasoning from raw data to conclusions? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Fully transparent

30. How useful were the intermediate tool outputs and summaries for understanding or verifying the report?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very useful

31. How much verification (e.g., checking GitLab or intermediate tool outputs) would be necessary before you would feel comfortable using this report for assessment? *

Mark only one oval.

	1	2	3	4	5	
Non	◯	◯	◯	◯	◯	Very much

32. Would you use such a report (including intermediate outputs) in a real assessment scenario? *

Mark only one oval.

	1	2	3	4	5	
Unlil	◯	◯	◯	◯	◯	Definitely

33. What did you find most useful?

34. What should be added or improved?